Introduction to C Programming

(Series 1 – with simple Visual C++ 2012 jump-start)

Copernicus P. Pepito

Dedication

This book is excitedly dedicated to
Queenie lyn Lenterna Tautjo for being simply unbelievable!

Acknowledgement

I would like to express my heartfelt gratitude to Ms. Letty Custodio and Ms. Mildred Villapando of National Bookstore - Purchasing Department whose tandem made this book possible.

I am very thankful also to Madam Zeny Alulod of Cacho Hermanos Publishing, for the heart-to-heart talks and to Ms. Lyn Francisco of National Bookstore – Accounting Department whose inspiring words (based on biblical wisdom) can never be forgotten .

If there is someone who motivated me to be brave in expressing myself through written-words, she was no other than my beloved grandmother, Dr. Roberta Pepito. Her editing-style made me feel very confident about my English-grammar proficiency (eventhough, I'm not good at it - honestly speaking).

Most of the time, my book was written behind an inspiration coming from a "girl-next-door". But in the case of Queenie lyn Lenterna Tautjo, she is a "girl-next-jungle", literally speaking! Well, probably this makes her unbelievable! Until now, I cannot believed in my experience that I had to pass-through a small jungle before reaching the house where you lived. And to think of it that you lived in a faraway place right there at the top of the mountain, made me feel a little bit eerie. I am sure that if I say again that you are beautiful, you won't believe it (as you usually do). But who cares? The most important thing is that I said what I feel. And you know what? It makes me happy.

Lastly, I give thanks to our Almighty God who unceasingly poured out a lot of blessings to me, and whose plan to my life and the lives of my loveones I cannot questioned, and who answered my questions about life, trials, and death through the books written by His servants.

Preface

"In my experience, C has proven to be a pleasant, expressive, and versatile languagefor a wide variety of programs. It is easy to learn, and it wears well as one's experience with it grows."

-Dennis Ritchie

This book is written to help the Filipino students learn how to program in C programming language easily. It contains introductory topics of C, from simple input/output statements up to two-dimensional arrays and string manipulations. Mostly, the C programming class covers these topics. Furthermore, this book is designed as lecture notes, reviewer, and laboratory manual.

Most of the examples presented here are collections of my notes, which I have used for my lecture and laboratory classes in C programming subject.

As a teacher of C language for over 8 years, I felt that I have the responsibility to help the students in different schools all over the Philippines to learn and appreciate this powerful computer language. I wish and pray that may you find this book useful to your present endeavor and would eventually make your "student-life" more enjoyable and full of fun. I myself had enjoyed so much in learning C programming. From the very start, I fell in love with its power and mystery. For me, like girls – C is full of mystery and it takes an uncommon patience and unconditional love to learn and know it well. I hope that you are now ready to discover C as it unfolds its power and spirit. Remember, it takes love to become successful in any endeavors. Would you start loving C now?

Copernicus P. Pepito
Member, PSITE-NCR
Makati, Metro Manila

Note:
PSITE- Philippine Society of Information Technology Educators
NCR- National Capital Region

About the Author

Copernicus P. Pepito is formerly an Assistant Professor of AMA Computer University – Quezon City Campus for 11 years and once a Training Consultant of Mapua IT Center – Makati Campus. He holds 4 degrees: Bachelor of Science in Computer Science (BSCS), Bachelor of Science in Computer Engineering (BSCoE), Master of Science in Computer Science (MSCS), and Master in Business Administration (MBA). Professor Pepito earned 7 international certifications in Information Technology: Cisco Certified Network Associate (CCNA), Cisco Certified Academy Instructor (CCAI), Microsoft Certified Professional (MCP), Microsoft Certified Systems Administrator (MCSA), Microsoft Certified Database Administrator (MCDBA), Microsoft Certified Solution Developer (MCSD), and Microsoft Certified Systems Engineer (MCSE). He is an author of 14 computer books, all of which are published by National Bookstore.

Contents

Chapter 1
Introduction to C Language

"They can because they think they can."
-Virgil

I Believe

I believe (with all my heart) that we are capable of learning new things, and new languages. So if C is new to you, don't worry, for you can learn this programming language like the way you learn Pascal language. Just believe that you can, and eventually you can!

In The Beginning

Everything has its beginning. Now this is the right time and place to start, to start talking about how C started. Well, C is like Pascal. It is a general-purpose programming language and it started to run under Unix environment. C is closely associated with Unix, because the operating system itself and majority of its supporting application programs are written in C. However, this doesn't mean that C could only run under Unix operating system environment. As a matter of fact, C can run on almost any hardware and software platforms available in the world today.

Dr. Dennis Ritchie invented the C programming language at AT&T Bell Laboratories, New Jersey, U.S.A. in early 1970s. Since C is similar to Pascal, a structured-programming language; it provides the fundamental control-flow for well-designed programs. These include statement groupings, decision-making (*if-else*), selecting one of the set of possible cases (*switch/case and if/else if*), and looping statements (*while, for, do-while*).

C is considered as middle-level programming language, because it combined the power of low-level language (such as Assembly language) and the elegance of high-level language like Pascal. This further means that C language can directly manipulate the bits, bytes, and even the computer hardware memory addresses. This makes C powerful, and the best tool for systems programming. Systems programming is aimed in designing and developing **operating systems**, **compilers, interpreters, database software,** and other highly sophisticated software including embedded intelligence programs. Embedded intelligence systems are programs embedded inside an Integrated Circuits (IC) or chips. Like other programming language, C has its own blemishes. Some of its syntax could be designed better and shorter like in the case of "*case* conditional statement". Compared to Pascal, it is longer and cumbersome to use. Well, inspite of that, C is still the choice of many professional programmers and software engineers for designing and developing highly sophisticated software. This is because C language has been proven to be an extremely powerful, expressive, and effective language for a wide variety of programming applications.

"Variables and constants are the basic data
objects manipulated in a program."
-Dennis Ritchie

C Data Types

Before a variable name (identifier) can be used in a C program, it must be declared explicitly, together with its corresponding data type. There are some restrictions in giving a name to variables and constants. Names can be composed of letters and numbers, however the first character must be a letter. Capital (uppercase) letters and small (lower case) letters are treated differently, so capital **A** is not the same with small **a**. When we declare capital **A** as a variable or constant name, capital **A** must be consistently used throughout the program, not in combination of small **a.** Otherwise, it will trigger an error during compilation. Keywords or commands such as **case, int, float, if, do, for,** etc., cannot be used as variable or constant names, because they are considered as reserved words. C programming language is a case-sensitive. This means that all keywords, commands and standard functions of C language must be written in lower case letters.

Remember that it is a good programming practice to choose and use variable and constant names which are related to its intended purpose. For example, the best variable name for **sum** is *sum,* not **s**, and the best constant name of Pi (π) is **Pi.**

Basic Data Types of C

Type	Format	Meaning (Example(s))
int	%d or %i	-a whole number (3,7,+8,-9,1000000)
float	%f	-a number with decimal point (5.8, 9.0, -5.6789, +0.0032)
char	%c	-a single letter, symbol, or number enclosed within two single quotes ('B', 'r', '*', '3')
char	%s	-a string in C is considered as a series of characters ("Ms.","Hi!", "143", "I love")
double	%f	-for bigger or larger numbers with a decimal point (123456789.123)

Every time we issue an input/output command or function in C program, we always format the input/output variable. For example, if we declare **n** as type **int**(integer) such as: **int n;** our input formatting is : `scanf("%d",&n)`. Now if we declare **Area** as type **float**(real) such as: **float Area;** our output formatting is: `printf("The area: %f",Area)`. Since string is considered as just a series of characters in C, therefore its declaration is in character form (char). In declaring string variable **name**, we simply write: **char name[20];.** The number **20** inside the square brackets is the maximum characters a string variable **name** can handle or hold. If the input string value is longer than 20 characters (including the spaces), the first 20 characters are only considered for manipulation

and display. It's like the **name** is being cut-off. The input formatting of string is simply: `scanf("%s", &name)`; and the output formatting is `printf("%s",name).`

To declare a constant name, we use the keyword `#define.` For example, to declare Pi as a constant name, we use this syntax: `#define Pi 3.1416 .` This `#define` declaration follows right after the `#include` declaration. In traditional C programming practice, variable names are written in lower case, while constant names are written in uppercase, or combination of both uppercase and lowercase, like in the case of **Pi** constant name.

In our output statement `printf`, we use control characters or escape sequences. The most popular and widely used is the backslash n (\n). This means a *new line* or *next line*. The other set of control characters are the following:

Control Characters	Meaning
\a	alert(bell)character
\b	backspace
\n	new line/next line
\r	carriage return/Enter key
\t	horizontal tab
\v	vertical tab
\\	backslash
\f	\form feed

Variable Declaration

We have to declare all variables and constants, before we can use them in our main program (main()). When we declare a variable, we specify its data type. For example, to declare a variable with data type *integer, float,* and *character,* we simply write: `int sum;`

```
float area;
char  letter;
```

or we can group similar data type variables in one list such as:

```
int n1,n2,sum,diff;
float area, fah, celsius, divided;
char letter, symbol, initial;
```

We can initialize the variable too, upon declaration. Initializing a variable means we put the first value of that variable. In some cases, the C system compiler would issue a warning message about a variable that was used but need to be initialized first before being executed, manipulated, or processed. This warning message was issued by the C compiler, so that the process or computation is correct or accurate. We have to remember that an uninitialized variable would contain a garbage data which is automatically produced or given by the computer's main memory (RAM). You could notice

this one when you use the *Break/Watch* menu. This sub-menu item in our C system compiler is a very handy tool in monitoring how the variables handled its value while they were being executed. In other words, this is the best debugging tools we can use as programmer in C, Pascal, and some other programming languages. I constantly used this *Break/Wat*ch submenu, when I'm analyzing, debugging, and improvising my program that uses many looping statements. After typing the variable to watch at the pop-up window of B*reak*-W*atch* submenu. I just press **F7** key to trace each line of my program. In that way, I can watch clearly and objectively, how the C compiler executes my program one line at a time. It gives me more ideas how my program behaves and why it behaves that way. And with my careful analysis and examination, I could find out why my program produces such undesired output. It is called "undesired" because that's not what I expected it to produce. This is really very important in programming task.

We have to examine our program seriously, because in programming, "trial-and-error" technique doesn't work at all! Based on my own experience, this "trial-and-error" will only mislead you to believe that programming is a very disappointing task. Programming is not a matching type or a multiple choice examinations which the odd of selecting the right answer is probable through the law of elimination. It is more than that. It is a mind-power game, a mental exercise. It demands a sound mind, a logical thinking capacity, and unwavering courage to try all over and over again once we failed so many times. It's like saying, programming is not fit for the faint-hearted humans, but for the humans whose heart is full of love to learn, no matter what! That is why I borrowed the wisdom of Socrates who once said: "Unexamined life is not worth living." We can apply it in our programming endeavor by saying: "Unexamined program is not worth doing." Would you like to try?

By the way, let us go back with variable monitoring. W. Digjstra, one of the famous authorities and number one advocate of structured programming paradigm once said: " Once a person understands how a variable is used in the program, he has understood the quintessence of programming." So that's it! Know the basics: using variables, and how it is being manipulated or processed within the program is the beginning of wisdom (tooth?).

Basic Operators of C

Arithmetic Operators

Symbols	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (Remainder)

In Pascal language, we use the command **mod** to get the remainder in a particular operation. This can be easily understood through this comparative syntax:

```
In Pascal syntax:  R  :=  N mod 2;
In C syntax:       r  =  n  % 2;
```

Syntax is the rule of writing a correct program in a certain programming language. Once we commit a mistake in syntax, we eventually get an error message when we compile our program . This procedure can be done by pressing *F10* and choosing the *Compile* menu in our Turbo C compiler, or using the short-cut commands the: *Ctrl-F9* keys to be pressed together or simultaneously. It's the same with Turbo Pascal's compilation and running the program *(Alt-F9* and *Ctrl-F9* procedures).

Relational Operators

Symbols	Meaning
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

The difference between Pascal and C syntax of relational operators are the *equal* or *not equal* symbols. In Pascal programming language, we use single = (equal) sign for our equality symbol while in C programming language we use double == (double equal) sign for our equality symbol. This is because the single equal symbol (=) is used as *assignment operator*. That's where we can find the conflict in C application of syntax. In Pascal language, there's no conflict of syntax since its *assignment operator* is this symbol := (colon equal symbol).

Logical Operators

Symbol	Meaning
&&	and
||	or
!	not

Logical AND (&&)Truth Table

X	Y	Z=X*Y
0	0	0
0	1	0
1	0	0
1	1	1

Logical OR (||)Truth Table

X	Y	Z=X+Y
0	0	0
0	1	1
1	0	1
1	1	1

Logical NOT (!) Truth Table

X	$\overline{X}$
0	1

In Pascal programming language, we use the words **and, or,** and **not,** however, in C we have to use symbols for logical *and* : && (two ampersand symbols), for logical *or* : || (two pipe symbols), and for logical *not*: ! (exclamation mark). The pipe symbol (||) can be found below (for old keyboard) or above (for newly designed keyboard) the famous *Enter* key. (Don't say it doesn't exist in your keyboard, man!).

Other Assignment Operators

Symbols	Meaning
+=	plus equal
-=	minus equal
*=	multiply equal
/=	divide equal
%=	modulus equal
--	minus minus (decrement)
++	plus plus (increment)

The application of these combined assignment operators is to compress the form of operation. For example:

`n = n+1`	can be written as `n++` or `++n`
n = n-1	can be written as - -n n- -

Similarly the following equations below can be written in compressed form:

Normal Form	**Compressed Form**
i=i+2	i+=2
p=p*b	p*=b
r=r/4	r/=4
j=j-3	j-=3
n=n%2	n%=2

Technically, the compressed form can be processed by the computer faster than the normal or traditional form because the calculation is performed only once. However, this difference in speed is unnoticeable with small programs. Furthermore, the disadvantage of the compressed format is its inherent unreadability.

Chapter 2

The Basic Input/Output Statements of C

"The only way to learn a new programming language is by writing programs in it."
-Dennis M. Ritchie

Usually our program involves three main operations. The first one is the input operation that uses input functions such as **scanf( ), getch( ), gets( ), getche( ), getchar( )** and others. The second is the process operation. In this part of our program, we can see some equations that are calculated, conditions which are evaluated, and tasks being performed. The third part is the output operation. Here, we use output statements such as **printf( )** function, **puts( ), putch( ), putchar( )** functions and other output statement functions. We could notice that these three operations are interrelated from each other, since whatever data we input, we also process it. And whatever variables, equations, or tasks that we processed, we output them into the screen. As a matter of fact, we can break down our programming tasks into three phases with these three main operations. Like for example, we have to analyze first what are those variables to be used as inputs in a given worded-problem. Then, we will analyze how to process those input variables. And finally, we can think our way on how to output those things (equations, conditions, or tasks) that we processed. To learn this technique, we can jump right in and type our first program that involves these three main operations.

Example 1:

Write a C program that calculates the sum of two input numbers, and display the result.

Algorithm:

Input ⟶ Enter two numbers (n1,n2)
Process ⟶ Compute the sum (sum=n1+n2)
Output ⟶ Display the sum (sum)

Solution:

```
#include <stdio.h>
main()
{
     int sum, n1,n2;
     clrscr();
     printf("\n Enter two nos.");
    scanf("%d%d",&n1,&n2);
    sum=n1+n2;
    printf("\n The sum: %d",sum);
    getch();
```

```
}
```

Explanation:
(A C program by dissection)

The first line in our program is the `#include` statement which means we tell the C compiler to use the standard input/output header file `<stdio.h>` that contains the standard input/output functions such as *printf( ), scanf( ), getch( ), clrscr( );* and much more. The second line is the *main( )* function (program) statement. Every C program should have this statement. The third line is the list of variables declared as integer data type. The fourth line is the clear screen (*clrscr( )*) standard function that tells the computer to clear the entire screen. The fifth line is the *printf( )* output function that commands the computer to display the message on the screen. The \n (backslash n) control character is used to tell the C compiler to occupy the whole line in the screen regardless of how long the message is to be displayed. It further means that the next message will be displayed in the next line or new line. The sixth line is the *scanf()* standard input function which tells the C compiler to scan (read or accept) the values typed from the keyboard and store them into variables. This is the input operation in our program. The two *%ds* are the data type format of the *n1* and n2 data type variables. The variables to be scanned should be preceded with *&* symbol. The seventh line is the calculation of *sum.* We are the one who formulate this equation based on our previous knowledge in basic mathematics. This part is the **process** (part) **operation.** The eight line is our **output part** where we output the value of variable *sum.* The ninth line is an instruction to the computer to pause, in this way we can see our output. The *getch( )* function means **get character** from the keyboard. Actually there is an invinsible program pointer that executes our program line by line. Once this program pointer reaches the last *end* (}) symbol, it will go back to the main menu in our C compiler program editor. We can experiment this by replacing *getch( )* with *delay(3000)*; in your program instead of writing *getch( ).* You could notice how this "pause effect" works without *getch( ).* This *getch*( ) function simply waits user to type a character from the keyboard for it to get that character.

Example 2:

Write a program to calculate the area of a circle and display the result. Use the formula: $A=\pi r^2$ where Pi (π) is approximately equal to 3.1416.

Algorithm:

Input ⟶ Enter the radius (r)
Process ⟶ Compute the Area (A=Pi*r*r)
Output ⟶ Display the Area (A)

Solution:

```
#include <stdio.h>
#define Pi 3.1416
```

```
main()
{
  int r;
  float A;
clrscr();
printf("\n Enter the radius:");
scanf("%d",&r);
A=Pi*r*r;
printf("\n The area:%f",A);
getch();
}
```

Explanation:

We could notice how the constant name *Pi* is declared in C, it is by using the *#define* command. This is the exact syntax: no *equal (=)* symbol between the constant name and its corresponding value. Take note, there is no *semicolon* symbol after the constant value.

We declare variable A (for Area) as **float** because it would eventually contain a value that has a decimal point. This is because of the resulting computation that whatever number multiplied to *Pi* would produce a number with a decimal point. T he *float* data type in C programming language is equivalent to *real* data type in Pascal programming language.

You could notice also that in our output statement *printf("\n The area: %f",A);* we format our output variable **A** with *%f*. This is because we had declared our variable **A** as *float* data type.

Example 3:

Write a program that computes the average of three input quizzes, then display the result.

Algorithm:

Input ⟶ Enter three quizzes (q1,q2,q3)
Process ⟶ Compute the average (ave=(q1+q2+q3/3)
Output ⟶ Display the average (ave)

Solution:

```
#include <stdio.h>
main()
{
   int q1,q2,q3;
   float ave;
   clrscr();
    printf("\n Enter  three quizzes:");
    scanf("%d%d%d",&q1,&q2&q3);
```

```
    ave=(q1+q2+q3)/3;
    printf("\n The average:%f",ave);
     getch();
}
```

Explanation:

In this example, we declare *average* variable *ave* as *float*, because it is used in a division operation. In any rules of programming, a whole(integer) number divided by another whole(integer) number has a possibility of yielding a computation value with a decimal point. Like in the case of *5/2*; the resulting computation is 2.5 and 2.5 is a floating point value (a number with a decimal point). That is why we have to declare a data type *float* for a variable that holds the computed value of an equation that involves a division (/) operation. Based on our previous knowledge in mathematics, we come up with the equation*: ave=(q1+q2+q3)/3*. This means that we need a background in basic mathematics before we can successfully solve a given worded-problem in our computer subject.

Example 4:

Write program that converts the input Fahrenheit degree into its Celsius degree equivalent. Use the formula: C=(5/9) *F-32. Display the result.

Algorithm:

Input ⟶ Enter the Fahrenheit (f)
Process ⟶ Compute the Celsius (C=(5.0/9.0)*F-32.0)
Output ⟶ Display the Celsius (C)

Solution:

```
#include <stdio.h>
main()
{
  float c,f;
  clrscr();
  printf("\n Enter the Fahrenheit:");
  scanf("%f",&f);
  c=(5.0/9.0)*f-32.0;
  prinf("\n The celsius:%f",c);
  getch();
}
```

Explanation:

Since integer division in C language truncates the resulting computed value to zero, that is why it is safe to write the values in floating point form (i.e. 5.0/9.0 or 32.0) and to declare the input variable needed in the equation as *float* data type. This will result to a correct (untruncated) answer. So if you could notice a case like this: a truncated answer (number), probably you declare the input variables needed in computation with an integer (*int)* data type. Try to change it to *float* data type declaration and observe if it rectifies the problem.

Some Things to Remember

In C program, we have to type all commands (*int, float, for, do ,while, if, case,* etc.) and standard functions (*printf, scanf, getch( ),* etc.) in lowercase (small) letters, otherwise, we can get a syntax error message during or upon compilation. This is because C language is a case-sensitive compiler. In other programming languages such as Pascal, we can type our program in all capital letters; in all small letters; or combination of both.
It works just fine. But this programming practice is not applicable to C language. Maybe you had observed that we use constant and variable names in capital letters such as in the case of **A** for *Area*, **C** for *Celsius*, **F** for *Fahrenheit*, and **Pi** as **constant** name. Yes, we can use constant names and variable names in capital letters form or combination of lowercase and uppercase letters such as in the case of **Pi** (capital P and small i*)* again. Take note, that we cannot use PI (all capitals) in some parts of our program when we had already use **Pi**, because they are not equivalent; unlike in Pascal language.

Maybe you had also observed how we format our variables. Actually, the number of formats is dependent to the numbers of variables being scanned or printed. For example, in our solution about the area of a circle, we have only one *%d*, because we have only one scanned variable, the *&r.*

In our example about the average of three quizzes, we have three *%d%d%d* because we have three variables: *&q1, &q2, &q3.* Remember that we use the *%d* format when the variables scanned are declared as integer and *%f* when declared them as **float** data type. The same also when we output the value of these mentioned variable in an output statement *printf( )* function. Like the output of our example about "*sum* of the two input numbers". We format the output variable *sum* as *%d,* and we format the output variable *A* (area) as *%f;* this is because the variable *sum* was declared as integer, while variable *A* (area) was declared as **float**.

We have to remember also that when we scanned our variables, we have to write them preceded with the & (ampersand) symbol, otherwise our program will behave differently; different from what we expected. In contrast, when we print the output values of our variables, we must write the variables without preceding it with ampersand (&) symbol. This too, would produce a different output when we do. Again, different from what we expected. To make these explanation clearer, we will have these examples:

```
scanf("%d%d%d",q1,q2,q3);    /* WRONG STATEMENT  */
printf("\n The area: %f", &A);   /* WRONG STATEMENT */
```

Warning!

These two above statements will not produce syntax errors during compilation and running time. However, your program will behave differently, such as your program will execute so fast without waiting for an input values from the keyboard. In the case of *printf( ),* your program will produce an unusual output value. Some of these values are in ASCII form (or Chinese-like characters) and with letters and numbers combined.

LAB ACTIVITY
TEST 1

1.Create a program to compute the volume of a sphere. Use the formula : $V=(4/3)*\pi r^3$ where π is equal to 3.1416 approximately. The r^3 is the radius. Display the result.

2. Write a program that converts the input Celsius degree into its equivalent Fahrenheit degree. Use the formula: F=(9/5)*C+32.

3. Write a program that converts the input dollar to its peso exchange rate equivalent. Assume that the present exchange rate is 51.50 pesos against the dollar. Then display the peso equivalent exchange rate.

4.Write a program that converts an input inch(es) into its equivalent centimeters. Takenote that one inch is equivalent to 2.54 cms.

5.Write a program that exchanges the value of two variables : *x* and *y*. The output must be : the value of variable *y* will become the value of variable *x*, and vise versa.

6. Design a program to find the circumference of a circle. Use the formula: $C=2\pi r$, where π is approximately equivalent to 3.1416.

7.You can solve the worded-problem number 5 with the use of three variables declaration. Now try to solve it with only two variables declaration. Formulate with an equation that exchanges the value of variable *x* and *y*. The hint is: use 3 equations that involve with plus and minus operations.

8.Write a program that takes as input the purchase price of an item (P), its expected number of years of service (Y) and its expected salvage value (S). Then outputs the yearly depreciation for the item (D). Use the formula: D= (P-S)/Y

9.Determine the most economical quantity to be stocked for each product that a manufacturing company has in its inventory. This quantity, called *economic order quantity (EOQ)* is calculated as follows:

$$EOQ=\sqrt{\frac{2RS}{I}}$$

where:

R= total yearly production requirement
S= set up cost per order
I=inventory carrying cost per unit

10.Write a program to compute the radius of a circle. Derive your formula from the given equation: $A=\pi r^2$, then display the output.

Chapter 3

The Conditional Statements

"The *if-else* statement is used to express decisions"
-Dennis Ritchie

The computer has the capability to make decisions based on a given set of conditions and choices. This can be done through the application of conditional statements in our program.

Simple *if-else* Conditional Statements

In simple *if-else* conditional statement, there are two given choices the computer could make. However, the computer could only choose one of them. Its like having two girlfriends, you could only choose one to marry (if you are a true Christian, not a fake one). The formal syntax of *if-else* is:

```
if (condition)
  statement1;
else
  statement2;
```

Actually, the *else* part is optional which means that it can be included or not in our program; (our program will still run even without it). When the condition of i*f* statement is proven true, statement1 is executed, otherwise statement2 is executed. In the absence of *else* statement, there is nothing to be executed if the **if** conditional statement is evaluated to false.

Example 1:

Write a program that determines if the input age is qualified to vote or not. The qualifying age is 18 years old and above.

Algorithm:

```
Enter age
if  (age>18)
   printf("\n Qualified to vote);
else
   printf("\n Too young!");
```

Solution: (Partial)

```
#include <stdio.h>

main()
```

```
{
  int age;
  clrscr();
  printf("\n Enter age:");
  scanf("%d",&age);
  if (age>=18)
     printf("\n Qualified to vote");
  else
    printf("\n Too young!");
  getch();
}
```

Explanation:

Our program works fine when we enter a reasonable age such as 15, 30, 13, 50, 45, 65 and 20. However, when we enter an impossible age like 501, this value (age) will still be accepted by our program and would display an output (which is supposed not to): "Qualified to vote". Our output message doesn't fit to the given input value. The message should be: "You are already dead, Magellan?" (and you probably add: "Now go back to the grave and drink a *lambanog(*perhaps) with Lapu-lapu, ". Not only this, our program will accept negative values (age) too, such as –1, -9, and even 0. Imagine, we input an age –1, our program would display a message: "Too young!"; when in fact, the right message is: " You are still a fetus, baby?". We know that there is no such age as –1 or 0. The probable logic that we can construct to solve this dilemma could be:

```
if   ((age>=1) && (age<=100))
   if (age>=18)
       printf("\n Qualified to vote");
  else
     printf("\n Too young!");
else
    printf("\n Out-of-Range");
```

However, this correct solution is hard to understand since it applies the **nested**-*if (*an *if* statement within an i*f* statement*).* Most especially if all the statements are not properly indented (written in the same column; the common style of those lazy humans, no not you man, not you, I really mean it).

You really need to learn how this nested *if* works and behaves technically. Otherwise, you will be confused in your analysis. As a matter of fact, I had only discovered this correct solution on the night I wrote this part of the book. And I wrote this at 11:30 in the evening. Take for example this case: Which of the two *ifs,* the *else* statement is partnered? This question is very critical to the exactness of your analysis and accuracy of your program. According to the rule of syntax in C and Pascal programming languages, the nearest *if* statement is the partner of the *else* statement. Without knowing this rule, you would end-up guessing. And that is not a good practice in our field. "Guessing style" has no room in the study of computer science or engineering.

Here in our program, the inner *if* conditional statement would only be executed if the outer *if* conditional statement is evaluated to true, otherwise the last *else* conditional statement will be executed:

```
e/se
  printf("\n Out-Of-Range");
```

If the outer *if* conditional statement is evaluated to true then the inner *if* (conditional statement) is executed. Since this inner *if* is also a conditional statement, therefore it will also be evaluated. If the evaluation results to true, then its associated statement would be executed, otherwise its inner *else* partner would be executed:

```
else
   printf("\n Too young!");
```

Here is the simplified version of this correct solution (using ladderized *if* conditional statements):

```
if  ((age>=18) && (age<=100))
     printf("\n Qualified to vote");
 else if ((age>=1) && (age<=17))
    printf("\n Too young!");
else
  prinf("\n Out-of-Range");
```

This solution is called the ladderized *if-else if* conditional statements; where the computer would choose only one of the given lists of choices. However, this type of conditional statements are our next topic. So let us just stick around to our partial solution (as of the moment) for the sake of simplicity. In this way, we can truly and deeply understand the essence and inner workings of simple *if-else* conditional statements.

Let us have another example that won't require such kind of scrutinization. You could notice that it is very time-consuming to understand this **nested** *if,* as we had experienced earlier.

Example 2:

Write a program that accepts the input magic number. If the input is right, the magic words will be displayed. The magic number is 143 and its corresponding magic words are: "I love you". If the input number is wrong, displays: "Sorry, better luck next time".

Algorithm:

```
Enter a magic number (mn)
if  (mn= =143)
    printf("\n 143 means I love you");
else
    printf("\n Sorry, better luck next time");
```

Solution:

```
include <stdio.h>
main()
{
 int mn;
 clrscr();
 printf("\n Enter a magic number:");
 scanf("%d",&mn);
 if (mn==143)
   printf("\n 143 means I love you");
else
   printf("\n Sorry, better luck next time");
getch();
}
```

Explanation:

In this solution, we use the double equal (= =) symbol for equality operation because the single equal (=) sign is exclusively reserved as an assignment operator in C programming language.

Example 3:

Write a program that determines if the input number is POSITIVE or NEGATIVE. Consider 0 as positive (considering that it contains no negative sign).

Algorithm:

Enter a number (n)
if (n>=0)
  printf("\n Positive no.");
else
  printf("\n Negative no.");

Solution:

```
#include <stdio.h>
main()
{
  int n;
  clrscr();
  printf("\n Enter a number:");
  scanf("%d",&n);
  if (n>=0)
    printf("\n Positive no.");
```

```
  else
    printf("\n Negative no.");
  getch();
}
```

Explanation:

To come up with a correct algorithm, we have to think how and when the number becomes negative. Eventually, we could think that the number becomes negative if it is less than 0. From this idea in logic, we can construct the correct solution. So we can also have this second solution and algorithm:

```
if  (n<0)
    printf("\n Negative number");
else
  printf("\n Positive number");
```

In the same manner, a number that is greater than or equal to zero is a positive number, otherwise that number is negative. Some of my students used the solution below as their answer (which is equally correct):

```
  if  (n<=-1)
      printf("\n Negative number");
 else
    printf("\n Positive number");
```

As you could notice, we can have many solutions in a given particular problem. It depends upon how we think and analyze. Our logic can be different, but we may arrive with the same correct solution.

Example 4:

Write a program that determines if the input number is ODD or EVEN number.

Algorithm:

```
Enter a number (n)
r = n % 2
if   (r= =0)
     printf("\n Even number");
else
    printf("\n Odd number");
```

Solution:

```
#include <stdio.h>
main()
{
   int r,n;
   clrscr();
   printf("\n Enter a number:");
   scanf("%d",&n);
   r= n % 2;
   if (r==0)
      printf("\n Even number");
   else
      printf("\n Odd number");
   getch();
}
```

Explanation:

How would we know that a number is an EVEN number? We would know that it is an EVEN number if it is divisible by 2. It further means that if a number divided by 2 and it yields a remainder of 0, then that number is divisible by 2; an EVEN number. Even if we are given a very large number such as 129788, we can still determine if it is an EVEN number, by considering the last digit of a given large number. In this case, it is 8. The number 8 is divisible by 2, therefore it is an EVEN number.

"[The] sequence of *if* [*else if*] [*else*] statements is the most general way of writing a multi-way decision. The expressions [(conditions)] are evaluated in order; if any expression [(condition)] is true, the statement associated with it is executed, and this terminates the whole chain"

-Dennis Ritchie

Ladderized *If / Else If / Else Conditional Statements*

With the use of ladderized *if / else if / else* conditional statements, the computer can decide to choose one and only one of the given choices. Only those conditional statements and expression(s) that were first proven true, are the ones to be chosen and its associated statement(s) will be executed. The formal syntax of ladderized *else if* is:

```
if (condition1)
     statement1
else if (condition2)
   statement2
```

else if (condition3)
  statement3
else if (condition4)
  statement4
else if$_n$
  statement$_n$

When all conditions were proven and evaluated false, the computer would execute the associated statement of *else.* Like in simple *if-else* conditional statement, the *else* part is optional; it can be included or not. Our purpose why we include this *else* conditional statement (most of the time in our program), is to use it to catch some "impossible condition" or the "none of the above" case. We may use the *else* for error checking, sometimes.

Example 1:

Write a program to assist a teacher in calculating student grades at the end of the semester. It accepts a numerical grade as input, then it will display the character grade as output, based on the given scale:

Range	Grade
90 and above	A
80 – 89	B
70 – 79	C
60 – 69	D
below 60	F

Algorithm:

```
Enter grade (g)
if (g>=90)
    printf("\n A");
else if (g>=80)
    printf("\n B");
else if (g>=70)
    printf("\n C");
else if (g>=60)
    printf("\n D");
else
    printf("\n F");
```

1st Solution: (Partial)

```
#include <stdio.h>
main()
{
  int g;
clrscr();
printf("\n Enter grade:");
scanf("%d",&g);
if(g>=90)
   printf("\n A");
else if (g>=80)
   printf("\n B");
else if (g>=70)
   printf("\n C");
else if (g>=60)
   printf("\n D");
else
   printf("\n E");
getch();
}
```

Explanation:

This example best describe and illustrate how ladderized *else if* conditional statement works technically. Like for example, if we input a numerical grade of 95, the output is "A", since the conditional expression: g>=90 is evaluated to true. Now, how about the remaining conditions such as: g>=80, g>=70, g>=60 they can be evaluated to true too! Yes, they can be evaluated to true, however, the invisible program pointer (that executes our program) will jump and ignore all the remaining conditions below, once there was already a proven true conditional expression above. This is how the ladderized *if / else if / else* conditional statement is designed; to choose one and only one of the given choices. And the first to be proven true will be chosen and its associated statement(s) will be executed. In this case, the expression: g>=90, is proven true first. If we input a grade of 87, our output is: "B". The explanation is the same with the previous one.

Our solution is again, a partial solution since it cannot handle such unusual input values: 501, 555, 777, 1001, -1, 0, or -99. For example if we input a value of 1001, the output is "A". But in reality, there is no such grade as 1001. When we input a grade of
–99, our output is "F", where in reality –99 is not a grade. This case is the same with our example number 1 under the Simple *If-Else* conditional statement. Our solution is incomplete. In real-world programming practice, we call this one a "case of software engineering". Software engineering is a field of computer science that deals with the study of producing quality software. Our partial solution program can be classified as a "no quality" or low quality program, because it contains no error trapping and checking, as well as it cannot withstand with unusual input data. A quality program must be able to anticipate such unusual input data, and can rectify (recover) itself in case an error occurs. They call it: "data validation technique". This technique is applied to preserve and protect the integrity of the data when it served as input and when displayed.

We can come up with the correct solution by using the logical && (AND) operator. In logical && operation, all conditional expressions in a given statement of choices must be evaluated to T (true) so that the associated statement(s) will be executed. For example:

if ((A>B) && (I>5) && (J<=10))
statement

The conditional expressions: A>B, I>5, J<=10 should be evaluated and proven T (true) so that its associated statement will be executed. They can be expressed in the same manner with *else if* conditional statement :

else if ((A>B) && (I>5) && (J<=10)
statement

Please be careful with the number of open and close parentheses. The number of close parentheses must match with the number of open parentheses. In C programming language, the correct syntax is to enclose a conditional expression(s) with an open and close parentheses, even if there is only one conditional expression present. In Pascal programming language if there is only one conditional expression, you can enclose it with the parenthesis or not, your program will still work. However, in C language, it is a pre-requisite to enclose it with parenthesis.

And finally, we have arrived here in our complete working solution, at last amigo!

Algorithm:

```
Enter grade (g)
if   ((g>=90)  &&  (g<=100))
     printf("\n A");
else if (( g>=80)  &&  ((g<=89))
    printf("\n B");
else if ((g>=70)  &&  ((g<=79))
    printf("\n C");
else if ((g>=60)  &&  ((g<=69))
    printf("\n D");
else if ((g>=50)  &&  ((g<=59))
   printf("\n F");
else
    printf("\n Out-Of-Range");
```

Solution: (Complete)

```
#include <stdio.h>
main()
{
  int g;
  clrscr();
  printf("\n Enter grade:");
```

```
  scanf("%d",&g);
  if ((g>=90) &&(g<=100))
     printf("\n A");
  else if ((g>=80)&& (g<=89))
     printf("\n B");
  else if ((g>=70) && (g<=79))
     printf("\n C");
  else if (g>=60)  && (g<=69))
     printf("\n D");
  else if (g>=50) &&  (g<=59))
     printf("\n F");
  else
      printf("\n Out-Of-Range");
getch();
}
```

Explanation:

Here in our solution, we trapped the range of grades such as 90 to 100, 80 to 89, 70 – 79, 60 – 69, and 50 – 59. With the use of logical && (AND), we forced the computer to limit the range of grades from 50 up to 100 only. Fifty(50) since in a percentile grade equivalent, it is equivalent to 0(zero). Any input grade below 50 and beyond 100 is considered an "Out-Of-Range". Therefore in our correct and complete solution, *else* conditional statement is specifically used to catch an "impossible input data". We can say also that this is a data-input error checking technique. In this case, we apply a defensive programming practice, which is essential for producing quality programs (a software engineering case).

Try to analyze deeply, when we input a grade of 96, this value (data) falls within the range of 90 to 100. So our conditional expressions: *((g>90) && (g<=100))* were evaluated and proven T (true), because the variable **g** has a value of 96, which in turn greater than 90 and (&&) less than 100. Remember that in logical && (AND), all conditional expressions must be evaluated and proven T (true), so that its associated statement(s) will be executed. After the computer executes the associated statement: *printf("\n A");* the invisible program pointer will ignore all the remaining conditional statements or choices below, and it will jump into the statement that follows the last conditional statement. In this case, our last conditional statement is *else* and its associated statement is: *printf("\n Out-Of-Range");*. Therefore, the program pointer jumps into the *getch( );* (the statement that follows the last conditional statement).

When we enter a grade of 78, the output is "C" because this value (data) falls within the range of 70 to 79. Again, after the program pointer executes the associated statement: *printf("\n C");* it will jump into the statement that follows the last conditional statement. In this example that statement is *getch( ).*

Example 2:

Write a program that displays an equivalent color once an input letter match its first character. For example **b** for Blue, **r** for Red, and so on. Here are the given criteria:

Letters	Colors
'B' or 'b'	Blue
'R' or 'r'	Red
'G' or 'g'	Green
'Y' or 'y'	Yellow
other letters	" Unknown Color"

Algorithm:

```
Enter a letter (Let)
If  ((Let=='B') || (Let=='b'))
     Printf("\n Blue");
If ((Let=="R') || (Let=='r'))
    Printf("\n Red");
If  ((Let=='G') || (Let=='g'))
    Printf("\n Green");
If  ((Let=='Y') || (Let=='y'))
    Printf("\n Yellow");
Else
   Printf("\n Unknown Color");
```

Solution:

```
#include <stdio.h>
main()
{
  char let;
  clrscr();
  printf("\n Enter a letter:");
  scanf("%c", &let);
  if ((let=='B') || (let=='b'))
     printf("\n Blue");
  else if ((let=='R') || (let=='r'))
     printf("\n Red");
  else if ((let=='G') || (let=='g'))
     printf("\n Green");
  else if ((let=='Y') || (let=='y'))
     printf("\n Yellow");
```

```
  else
     printf("\n Unknown Color");
getch();
}
```

Explanation:

The logical OR in C programming language is a double pipe symbols: ||. This symbol is located above or below the famous Enter key in the keyboard. Now remember that in logical || (OR), at least one conditional expression that is evaluated T (true) will trigger the computer to execute the associated statement.

The logical || (OR) is applied to this kind of program requirement, because the user's input whether uppercase(capital) letter or lowercase(small) letter should be accepted as valid. In programming, the character data small "b" is not equivalent to capital "B". Now, we use *%c* in *scanf( )* function because we declare *let* as char (character) data type.

Example 3:

Write a program to display the high school level of a student, based on its year-entry. For example, the year-entry 1 means the student is a freshman, 2 for sophomore, and so on. Here are the given criteria:

Year-entry number	High – School Level
1	Freshman
2	Sophomore
3	Junior
4	Senior

Other entry no. "Out-of-School"

Algorithm:

```
Enter year-entry number: (n)
If (n==1)
  Printf("\n Freshman");
Else if (n==2)
   Printf("\n Sophomore");
Else if (n==3)
    Printf("\n Junior");
Else if (n==4)
    Printf("\n Senior");
Else
    Printf("\n Out-Of-School");
```

Solution:

```
#include <stdio.h>
main()
{
    int n;
   clrscr();
   printf("\n Enter year-entry number:");
   scanf("%d",&n);
   if (n==1)
        printf("\n Freshmen");
else if (n==2)
      printf("\n Sophomore");
else if (n==3)
      printf("\n Junior");
else if  (n==4)
      printf("\n Senior");
else
      printf("\n Out-of-School");
getch();
}
```

Explanation:

In our examples 2 and 3, we used the double equal (= =) symbol to evaluate the equality between the value stored in the variable and the constant value given. The single equal symbol (=) cannot be used in the conditional expression, because it is exclusively reserved as an assignment operator in C programming language. Unlike in Pascal programming language, its assignment operator is := (colon equal). Moreover, in C programming language, the *then* command is not included. In Pascal, the statement that follows the *else* should not contain a semi-colon (;) , however in C language it should have. Now you know the difference in syntax between C and Pascal programming languages.

Example 4:

Write a "scissors-rock-paper" game, commonly known as "jack n poy" in the Philippines (our beloved country, of course). Two players will input the first equivalent letter. For example, **s** for scissors, **r** for rock, and **p** for paper. Then display the winner and the basis of winning such as: "paper covers rock", "rock breaks scissors", and "scissors cut paper", or "nobody wins". Capital or small letters should be accepted by your program.

Algorithm:

```
Enter player 1: (p1)
Enter player 2: (p2)
If (((p1=='p') || (p1=='P')) && ((p2=='R') || (p2=='r'))) {
   Printf("\n Player 1 wins!");
   Printf("\n Paper covers rock");
}
Else if (((p2=='p') || (p2=='P'))  &&  ((p1=='R')  ||  (p1=='r')))  {
   Printf("\n Player 2 wins!");
   Printf("\n Paper covers rock");
}


Else if  (((p1=='R') || (p1=='r'))  &&  ((p2=='S') || (p2=='s')))  {
   Printf("\n Player 1 wins!");
   Printf("\n Rock breaks scissors");
}

Else if (((p2=='R') || (p2=='r'))  &&  ((p1=='S')  ||  (p1=='s')))  {
   Printf("\n Player 2 wins!");
   Printf("\n Rocks break scissors");
}
Else if  (((p1=='S') || (p1=='s'))  &&  ((p2=='P') || (p2=='p')))  {
   Printf("\n Player 1 wins!");
   Printf("\n Scissors cut paper");
}
Else if (((p2=='S') || (p2=='s'))  &&  ((p1=='P')  ||  (p1=='p')))  {
   Printf("\n Player 2 wins!");
   Printf("\n Scissors cut paper");
}
Else if  (p1==p2)  {
   Printf("\n Nobody wins!");
   Printf("\n Your entered the same letters");
}
Else {
   Printf("\n Unknown input data!");
   Printf("\n Enter only letters : P,S, or R");
}
```

1st Solution:

```
#include <stdio.h>
main()
{
char p1,p2;
clrscr();
```

```
printf("\n Enter player 1 :");
p1=getche();
printf("\n Enter player 2:");
p2=getche();
if (((p1=='p')||(p1=='P')) && ((p2=='R') || (p2=='r'))){
    printf("\n Player 1 wins!");
    printf("\n Paper covers rock");
}
else if (((p2=='p')||(p2=='P'))&&((p1=='R') ||(p1=='r')))  {
    printf("\n Player 2 wins!");
    printf("\n Paper covers rock");
}
else if (((p1=='R')||(p1=='r')) && ((p2=='S')||(p2=='s')))  {
    printf("\n Player 1 wins!");
    printf("\n Rock breaks scissors");
}

else if (((p2=='R')||(p2=='r')) &&((p1=='S')||(p1=='s')))  {
    printf("\n Player 2 wins!");
    printf("\n Rocks break scissors");
}
else if (((p1=='S')||(p1=='s')) && ((p2=='P')||(p2=='p')))  {
    printf("\n Player 1 wins!");
    printf("\n Scissors cut paper");
}
else if (((p2=='S') ||(p2=='s'))&& ((p1=='P')||(p1=='p')))  {
    printf("\n Player 2 wins!");
    printf("\n Scissors cut paper");
}
else if (p1==p2) {
    printf("\n Nobody wins!");
    printf("\n Both of you entered the same letters");
}
else {
    printf("\n Unknown input data!");
    printf("\n Enter only letters : P,S, or R");
}
getch();
}
```

Explanation:

We declare **p1** and **p2** as character (char) data type. In this example, we combine the use of logical || (OR) and logical && (AND) to come up with a shorter solution. Actually, this solution can be accomplished using logical && (AND) alone. However, we need to design a longer program, since we have to separate the small input letters with another conditional statements and expressions. Like for example, to accomplish the first conditional statement, we need to break them into two separate conditional statements such as:

```
If  ((p1=='P') &&  (p2=='R')) {
    Printf("\n Player 1 wins!");
    Printf("\n Paper covers rock");
}
else if ((p1=='p')  &&  (p2='r'))  {
     Printf("\n Player 1 wins!");
      Printf("\n Paper covers rock");
}
```

This further means that we doubled the number of lines in our program, if we would implement this kind of solution. Let us complete and finish this second solution so that you can understand it better.

2nd Solution:

```
#include <stdio.h>
main()
{
char p1,p2;
clrscr();
printf("\n Enter player 1:");
p1=getche();
printf("\n Enter player 2:");
p2=getche();
else if ((p1='P') && (p2='R')) {
   printf("\n Player 1 wins!");
   printf("\n Paper covers rock");
}
else if ((p1='p') && (p2='r')) {
   printf("\n Player 1 wins!");
   printf("\n Paper covers rock");
}

else if ((p2='P') && (p1='R')) {
   printf("\n Player 2 wins!");
   printf("\n Paper covers rock");
}

else if ((p2='p') && (p1='r')) {
   printf("\n Player 2 wins!");
   printf("\n Paper covers rock");
}
else if ((p1='R') && (p2='S')) {
   printf("\n Player 1 wins!");
   printf("\n Rock breaks scissors");
}
```

```
else if ((p1=='r') && (p2='s')) {
   printf("\n Player 1 wins!");
   printf("\n Rock breaks scissors");
}

else if ((p2=='R') && (p1='S')) {
   printf("\n Player 2 wins!");
   printf("\n Rock breaks scissors");
}

else if ((p2=='r') && (p1='s')) {
   printf("\n Player 2 wins!");
   printf("\n Rocks breaks scissors");
}

else if ((p1=='S') && (p2='P')) {
   printf("\n Player 1 wins!");
   printf("\n Scissors cut  paper");
}

else if ((p1=='s') && (p2='p')) {
   printf("\n Player 1 wins!");
   printf("\n Scissors cut paper");
}

else if ((p2=='S') && (p1='P')) {
   printf("\n Player 2 wins!");
   printf("\n Scissors cut paper");
}

else if ((p2=='s') && (p1='p')) {
   printf("\n Player 2 wins!");
   printf("\n Scissors cut paper");
}

else if ((p1==p2)) {
   printf("\n Nobody wins!");
   printf("\n Both of you had input the same letters");
}

else {
    printf("\n Unknown input data !");
    printf("\n Input letter S,P,or R only");
}
getch();
}
```

Explanation:

You could notice that we use the pair of begin ({) and end (}) for each and every associated statements of our conditional statement. We have to enclose these associated statements with begin ({) and (}) symbols so that the computer would know that there are two associated statements to be executed, once the conditional expressions are proven and tested true. Remember that in the absence of these symbols { and }, the computer would assume that there is only one associated statement to be executed, once the condition is evaluated and proven true. Actually, only one associated statement is considered as the default, when the computer cannot find the grouping of compound statements with begin ({) and (}) symbols.

In our first solution, the computer would evaluate first the conditional expressions found at the inner part of the multiple open and close parentheses. And the evaluation will start from left to right. That is why the conditional expressions at the left side connected by the logical || (OR) will be evaluated first. The conditional expressions at the right side connected also by logical || (OR) will be the next to be evaluated. Then finally, each resulting evaluation will be tested by the logical && (AND).

Actually, the evaluation of these conditional expressions has a similarity with the order of evaluation of the mathematical expressions we encountered in our mathematics subject. If we try to think and analyze the whole thing, we would find out that we can simplify our solution. This is by the use of a standard function: *toupper( ).* Let us now have our third solution:

3rd Solution:

```
#include <stdio.h>
main()
{
  char p1,p2;
  char P1,P2;
  clrscr();
  printf("\n Enter player 1:");
  p1=getche();
  printf("\n Enter player 2:");
  p2=getche();
  P1=toupper(p1);
  P2=toupper(p2);
  if ((P1=='P') && (P2=='R')) {
     printf("\n Player 1 wins!");
     printf("\n Paper covers rock");
  }
  else if ((P2=='P')  &&  (P1=='R'))  {
     printf("\n Player 2 wins!");
     printf("\n Paper covers rock");
  }
  else if ((P1=='R')  &&  ((P2='S')) {
     printf("\n Player 1 wins!");
     printf("\n Rock breaks scissors");
}
```

```
 else if ((P2=='R')  &&  ((P1='S')) {
     printf("\n Player 2 wins!");
     printf("\n Rock breaks scissors");
}
else if ((P1=='S')  &&  ((P2=='P'))  {
     printf("\n Player 1 wins!");
     printf("\n Scissors cut paper");
}
else if ((P2=='S')  &&  ((P1=='P'))  {
     printf("\n Player 2 wins!");
     printf("\n Scissors cut paper");
}
else if (P1==P2)
   printf("\n Nobody wins!");
   printf("\n Both of you typed the same letters");
}
else
   printf("\n Unknown input data!");
getch();
}
```

Explanation:

Since *toupper( )* function is belonged to the <ctype.h> **character type** header file, that is why we include it in our program. Forgetting to include this header file would result to a syntax error during compilation. The computer cannot recognize the standard function *toupper( )* without the inclusion of <*ctype.h*> header file in our program.

We add two more variables, the capital P1 and P2. In Pascal programming language syntax, this kind of variable declaration is illegal, since they are treated the same (P1 and p1, as well as P2 and p2). This is because Pascal language is not a case-sensitive programming language. This will result to "Duplicate identifier" compiler error-message during compilation.

The *toupper()* function returns the equivalent upper-case letter of an input value. When an input value or data is already a capital letter, the value will remain the same.

This last example will remind us that there are many solutions in a given particular problem. It's up to us to think the best one and the easiest one to implement.

The Switch/Case Conditional Statement

"The switch[/case] [conditional] statement is a multi-way decision that tests whether an expression matches one of a number of constant integer [or character] values, and branches accordingly. If a case matches the expression value, execution starts at the case."
-Dennis Ritchie

With the *switch/case* conditional statement, we can simplify some of the tasks that are long and cumbersome when we use the ladderized *if / else if / else* conditional statements. In other words, the *switch/case* statement can be the best substitute command and function to write a clearer and concise multi-way decision program compared to the messy *if / else if / else* conditional statements. However, this *switch/case* conditional statements has its own limitation. They cannot test or evaluate floating point and string values or variables. This further means that only those variables that are declared as integer or character are the candidates that can be tested or evaluated by the *switch/case* conditional statement.

The switch statement has an equivalent *else* conditional statement; it is called *default.* It works and accomplishes the same thing. They are also both optional. Meaning, they can be included or not in our program. If they are not needed or required in the problem specs, why include them? But again, remember that in our previous examples, the *else* statement is very helpful to catch the impossible input data and use also extensively for data entry validation. The same thing also with the d*efault* command , we can use it to catch some impossible input-data or to validate our data-entry.

Here is the *switch/case* syntax:

```
switch (var-expression) {
    case const-value: statement1;
     case const-value: statement2;
    case const-value: statement3;
    case const-value: statementn;
    default: statement;
}
```

Now let us convert some of our examples which previously used previously the ladderized *if / else if / else* conditional statements to *switch/case* conditional statements. Let me also write the worded problems again for our easy reference.

Example 1:

Write a program to assist a teacher in calculating student's grade at the end of the semester. It accepts numerical grade as input data then it will display the equivalent character grade as output based on the given scale:

Range	Grade
90-100	'A'

80-89	‘B’
70-79	‘C’
60-69	‘D’
50-59	‘F’
others	‘Out-Of-Range’

Algorithm:

Enter grade (g)
Switch(g) {
Case 90: case 91: case 92: case 93: case 94:
Case 95: case 96: case 97: case 98: case 99:
Case 100: Printf(“\n A”); break;
Case 80: case 81: case 82: case 83: case 84:
Case 85: case 86: case 87: case 88: case 89:
Printf(“\n B”); break;
Case 70: case 71: case 72: case 73: case 74:
Case 75: case 76: case 77: case 78: case 79:
Printf(“\n C”); break;
Case 60: case 61: case 62: case 63: case 64:
Case 65: case 66: case 67: case 68: case 69:
Printf(“\n D”); break;
Case 50: case 51: case 52: case 53: case 54:
Case 55: case 56: case 57: case 58: case 59:
Printf(“\n F”); break;
Default: Printf(“\n Out-Of-Range”); break;
}
getch( );
}

Solution:

```
#include <stdio.h>
{
main()
{
   int g;
  clrscr();
  printf("\n Enter grade:");
  scanf("%d",&g);

switch(g) {
    case 90:  case 91: case 92:  case 93: case 94:
    case 95: case 96:  case 97: case 98: case 99:
    case 100: printf("\n A"); break;
    case 80:  case 81:  case 82: case 83: case 84:
    case 85:  case 86:  case 87:  case 88: case 89:
```

```
                printf("\n B"); break;
    case 70: case 71: case 72: case 73:  case 74:
    case 75: case 76: case 77: case 78:  case 79:
             printf("\n C"); break;
    case 60: case 61: case 62: case 63: case 64:
    case 65: case 66: case 67: case 68: case 69:
             printf("\n D"); break;
    case 50: case 51: case 52: case 53: case 54:
    case 55: case 56: case 57: case 58: case 59:
             printf("\n F"); break;
    default:  printf("\n Out-Of-Range"); break;
}
getch();
}
```

Explanation:

It seems that our equivalent **switch/case** conditional statement does not improve the readability of our program. Nor it shortens our program listing. Yes, we could say that why it is not like in Pascal programming language, where in we can write the range of 90 to 100 with the simple syntax: *90..100 : writeln('A');*. Honestly speaking, I do not know the answer why Dr. Dennis M. Ritchie had designed the *switch/case* in C programming language by requiring us to individually specify each constant value explicitly. Maybe this is what he meant when he said, "C, like any language has its blemishes. Some of the operators have the wrong precedence; some parts of the syntax could be better". What a brave confession by a genius computer scientist!

Can you confess your own blemishes? Or what is known as weaknesses? I myself can do! My weakness is my opposite sex (oops…censored). Every time I'm nearer to them, I almost give in. Give in to what? My innocence. Just kidding.

So much for the joke, let us now go back to our discussion. When we enter the value or data in the keyboard, the computer would scan this value and store it to the variable **g.** This is really how the *scanf( )* function works. The *switch (g)* command holds the value of **g** for testing, evaluation, or comparison with the given constant values of each case or group of cases.

Once the match is found (means the value of variable **g** is the same with the constant value found) then the program pointer will search the associated statement and execute it. For example, if we enter the value of 94, the program pointer will execute the associated statement *printf("\n A");* since this constant value can be found at the first group of *case* labels. When the value entered does not match to any constant values listed, the associated statement of *default* is executed instead. If the *default* is not included (since it is optional), then nothing will be executed.

You could observe a that all the associated statements are followed by the *break* command. This *break* command will trigger the program pointer to break out (when it is countered) from the whole *switch/case* (statement) block. Meaning, it jumps to the statement that follows the end (}) symbol of *switch/case* statement. This makes our *switch/case* solution to act and execute like the way *if / else if / else* do. It's like ignoring all the remaining conditional statements and expressions below when there is already a true-evaluation at the top. In short, the *break* statement causes an immediate exit from the *switch/case* (statement) block.

" The **break** statement causes an immediate exit from the **switch.** Because **cases** serve just as labels, after the

code for one **case** is done, execution falls through to the next unless you take explicit action to escape."
- Dennis Ritchie

Example 2:

Write a program that displays an equivalent color once an input letter matches its first character. For example, b for Blue, r for Red, and so on. Here is the given criteria: (The letters serve as an input data, and the color as output information).

Letters	Colors
'B' or 'b'	Blue
'R' or 'r'	Red
'G' or 'g'	Green
'Y' or 'y'	Yellow
other letters	Unknown-Color

Algorithm:

Enter a letter (let);
Switch(let) {
Case 'B': case 'b':
Printf("\n Blue"); break;
Case 'R': case 'r':
Printf("\n Red"); break;
Case 'G': case 'g':
Printf("\n Green"); break;
Case 'Y': case 'y':
Printf("\n Unknown Color"); break;
Default: Printf("\n Unknown Color"); break;
}

Solution:

```
#include <stdio.h>
main()
{
char let;
clrscr();
printf("\n Enter a letter:");
scanf("%c",let);
/* if the scanf( ) function won't work well, use the  */
/* getche() function instead; for example:let=getche() */
```

```
switch(let) {
case 'B': case 'b':
   printf("\n Blue"); break;
case 'R': case 'r':
   printf("\n Red"); break;
case 'G': case 'g':
   printf("\Green"); break;
case 'Y': case 'y':
   printf("\n Yellow"); break;
default: printf("\n Unknown Color");
}
getch();
}
```

Explanation:

When we enter the letter **b**, the computer would execute the associated statement: *printf("\n Blue");*. Then the program pointer goes to the next statement which happen to be the *break* command. This will trigger the program pointer to break-out from the whole *switch/case* (statement) block. In this example, it jumps into the *getch( );* function, the statement that follows the end (}) symbol of the *switch/case* statement. It is just like ignoring all the *case* labels below, and acts like the ladderized *if / else if / else* conditional statements.

If we enter the value **G**, the program pointer will simply evaluates the first and second group of *case* labels as false, since the value **G** doesn't match this constant values. Then the third group of *case* label is evaluated as true therefore the associated statement is executed: *printf("\n Green");*

Example 3:

Write a program to display the high school level of a student, based on its year-entry number. For example, the year-entry 1 means the student is a freshman, 2 for sophomore, and so on. Here is the given criteria:

Year-entry number	High-School Level
1	"Freshman"
2	"Sophomore"
3	"Junior"
4	"Senior"
other entry no.	"Out-Of-School"

Algorithm:

Enter year-entry number (n)
Switch(n) {

```
   Case 1: printf("\n Freshman"); break;
   Case 2: printf("\n Sophomore");break;
   Case 3: printf("\n Junior"); break;
   Case 4: printf("\n Senior"); break;
  Default: printf("\n Out-Of-School");break;
}
```

Solution:

```
#include <stdio.h>
main()
{
  int n;
  clrscr();
  printf("\n Enter year-entry number");
  scanf("%d",&n);
  switch(n) {
   case 1: printf("\n Freshman"); break;
   case 2: printf("\n Sophomore");break;
   case 3: printf("\n Junior"); break;
   case 4: printf("\n Senior"); break;
   default: printf("\n Out-Of-School");break;
}
getch();
}
```

Explanation:

In this example, when the value of the variable at the *switch/case* statement matches with the constant value of the list of *case* labels, associated statement is executed. If no match is found in all *case* labels, then the associated statement of *default* command is executed instead.

In example number 4, the *switch/case* equivalent solution is very difficult to construct, since it uses two variables, the *p1* and *p2*. This further means that there are some situations or cases where the *if / else if / else* ladderized conditional statements are easier to use and implement compared to the *switch/case* statement, and vise-versa.

LAB ACTIVITY
TEST 2

1. Write a program that determines if the input letter is a VOWEL or CONSONANT. The vowels are : A E I O U. Your program must be able to handle a capital or small input letter.

1st Solution: Use the ladderized *if / else if / else* conditional statements
2nd Solution: Use the *switch/case* conditional statement

2. Write a program that accepts dates written in numerical form and then output them as a complete form. For example, the input:

2 26 1986

should produce the output:

February 26, 1986

1st Solution: Use the ladderized *if / else if / else* conditional statements
2nd Solution: Use the *switch/case* conditional statement

3. Write a program that examines the value of a variable called **temp.** Then display the following messages, depending on the value assigned to **temp.**

Temperature	Message
Less than 0	ICE
Between 0 and 100	WATER
Exceeds 100	STEAM

4. Write a program for the Air Force to label an aircraft as military or civilian. The program is to be given the plane's observed speed in km/h and its estimated length in meters. For Planes traveling in excess of 1100 km/h, and longer than 52 meters, you should label them as "civilian" aircraft, and shorter such as 500 km/h and 20 meters as "military" aircraft. For planes traveling at more slower speeds, you will issue an "It's a bird" message.

5. Write a program that determines the class of the Ship depending on its class ID (identifier). Here are the criteria. The class ID serves as the input data and the Ship class as the output information.

Class ID	Ship Class
B or b	Battleship
C or c	Cruiser
D or d	Destroyer
F or f	Frigate

1st Solution: Use the ladderized *if / else if / else* conditional statements

2nd Solution: Use the *switch/case* conditional statement

6. The National Earthquake Information Center has the following criteria to determine the earthquake's damage. Here are the given richter scale criteria and their corresponding characterization. The richter scale serves as the input data and the characterization as output information.

Richter Numbers (n)	Characterization
N>5.0	Little or no damage
5.0>=n<5.5.	Some damage
5.5>=n<6.5	Serious damage
6.5>=n<7.5	Disaster
higher	Catastrophe

7. Write a program that accepts a number and outputs its equivalent in words.

Sample input/output dialogue:

Enter a number: 1380

one thousand three hundred eighty

Take note that the maximum input number is 3000.

8. Write a program that accepts an ordinary number and outputs its equivalent Roman numerals. The ordinary numbers and their equivalent Roman numerals are given below:

Ordinary Numbers	Roman Numerals
1	I
5	V
10	X
50	L
100	C
500	D
1000	M

Sample input/output dialogue:

Enter a number: 2968 ← Input number
MMCMLXVIII ← Output number

Or

Enter a number: 734
DCCXXXIV

Note that the maximum input number is 3000.

9. Write a program that computes and assesses the tuition fee of the students in one trimester, based on the given mode of payment below:

Plan (key)	Discount (-) or Interest (+)
Cash (1)	10 % discount
Two-Installment (2)	5 % discount
Three-Installment(3)	10 % interest

The target-user must use the key in selecting or choosing the mode of payment. The first input data is the tuition fee, and the second input data is the mode of payment.

Sample input/output dialogue:

Enter tuition fee: 20,000 ← Input data
(Press 1 for Cash, 2 for Two-Installment, 3 for Three-Installment)
Enter mode of payment: 2 ← Second input data

Your total tuition fee is: 21,000 ← Output data

10. Write a program that accepts an input grade in percentile form and output its grade equivalent; based on the given range of percentile and grade equivalent table below:

Range	Grade
98-100	1.00
95-97	1.25
92-94	1.50
89-91	1.75
85-88	2.00
82-84	2.25
80-81	2.50
77-79	2.75
75-76	3.00
other grades	“Out-of-Range”

Chapter 4

Looping Statements

"The **while** and **for** loops test the termination condition at the top. By contrast, the third loop in C, the **do-while**, tests at the bottom after making each pass through the loop body; the body is always executed at least once."
-Dennis Ritchie

The looping statement is a program instruction that repeats some statement or sequence of statements in a specified number of times. Looping statement allows a set of instructions to be performed all over again until a certain condition is reached, met, or proven and tested as false.

The three looping statements of C Programming Language are:

1. for loop
2. while loop
3. do-while loop

The syntax of the *for* loop statement is:

```
for ( init; condition; inc/dec) {
        statement(s)
}
```

The three parts of the *for* loop are the initialization (*init)* part , the *condition* part, and the increment (*inc)* or decrement (*dec)* part. The sample of incrementation formula is i++ or ++i, while the decrementation formula is --i or i-- .

The syntax of the *while* loop statement is:

```
initialization;
while (condition)  {
   statements;
   inc/dec;
}
```

statements within the body of the loop

The syntax of the *do-while* loop statement:

```
initialization;
do  {
statements;
inc/dec;
}  while (condition);
```

Incrementation, Decrementation, and Accumulator Formulas

These three formulas are oftenly used in looping statements operation:

1. Incrementation Formula:
   Examples:
   i++ or ++i
   ++n or n++

2. Decrementation Formula:
   Examples:
   i-- or --i
   --n or n--

3. Accumulator Formula:
   Examples:
   acc=acc+n
   sum=sum+n
   total=total+n

The variable initialization (*init)* specifies the first value where the loop starts. The conditional part specifies conditions to be evaluated or tested. If the condition is **false**, then no action is taken and the program pointer proceeds to the next statement after the loop. If the conditional expression is evaluated to be **true**, then all the statements within the body of the loop are executed. This looping process (iteration) is repeated as long as the conditional expression remains true. After each iteration, the incrementation/decrementation (*inc/dec)* part is executed, then the conditional expression is reevaluated. In a simple *for* loop statement, the initialization part is executed only once. This happens during the first time the program pointer points to the loop. But in nested loop, the inner loop's initialization part could be reexecuted many times depending on the condition set at the outer loop. This will happen every time the program pointer reevaluates the condition of the outer loop to T (true).

For the sake of simplicity and easy understanding about looping statement, we will focus our discussion on the simple looping statement construct.

Example 1:

Write a program that generates the given sequence nos.

Sequence nos.

1
2
3

4
5

Algorithm:

N=1
While (N<=5) {
  Printf("\n %d",N);
  N++;
}

1st Solution: using *while* loop statement

```
#include <stdio.h>
main()
{
   int n;
   clrscr();
   printf("\n While Loop Example - Incrementation");
   n=1;
   while (n<=5){
     printf("\n  %d",n);
     delay(3000);
     n++;
  }

getch();
}
```

Explanation:

Our initial value for *n* is 1. The program pointer evaluates the conditional expression: *n<=5.* Since *n* has an initial value of 1, therefore the condition is evaluated to be **true.** Once the condition is evaluated to true, all the statements within the body of the loop are executed. In this example, we have the:

p*rintf("\n %d", n);*
delay(3000);
n++;

The program pointer iterates(loop) all over again until the condition is evaluated and tested to **false.** This is the only time the loop process stops executing the statements within the body of the loop.

2nd Solution: using *do-while* loop statement

```
#include <stdio.h>
main()
{
   int n;
   clrscr();
   printf("\n Do-While Loop Example - Incrementation");
   n=1;
   do {
     printf("\n  %d",n);
     delay(3000);
     n++;
  } while(n<=5);
getch();
}
```

Explanation:

The *while* and *do-while* conditional statements almost act and behave at the same manner. However, their construction is different. First the *while* loop is designed with the conditional-expression positioned at the top of the loop, while the *do-while* has a conditional expression placed at the bottom of the loop. The difference between the two has a tremendous effect and impact, since there are cases or situations that they will act and behave differently (or produces different result or output).

Like for example, if we change the initial value of variable **n** by 6(so now we have n=6;), in *while* loop, there is no output, since the program pointer evaluated the conditional expression to **false.** The program pointer will simply jump into the statement that follows the looping statement. Usually this statement can be found right after the }
(end) symbol of the looping statement. In this example, it is the *getch( );* so the program pointer jumps into the *getch( ).*

Now when it comes to the *do-while* loop, the output is 6 since the statements within the body of the loop are executed first before the program pointer can reach to the conditional expression. Therefore, the *printf("\n %d",n); delay(3000);* and *n++* are executed first before the program pointer be able to evaluate the conditional expression: *(n<=5).*

Eventhough the conditional expression *n<=5* is evaluated to false (that causes the termination of the loop, it is still to no avail or help at all since the statements within the body of the loop are executed first, before the conditional-expression is evaluated.

3rd Solution: using *for* loop statement

```
#include <stdio.h>
main()
{
```

```
  int n;
  clrscr();
  printf("\n For Loop Example –Incrementation");
  for (n=1; n<=5; n++) {
     printf("\n  %d",n);
     delay(3000);
   }
 getch();
}
```

Explanation:

The difference between the *for* loop and other looping statements (*while and do-while*) is that in the *for* loop statement; the initialization, condition, and incrementation/ decrementation parts are placed on the same line. Unlike in *while* and *do-while* loops, the initialization part is placed before the loop, while the incrementation/decrementation part is placed within the body of the loop.

This arrangement on the same line of the *for* loop makes our analysis of how it works, confusing. For example, which of the three expressions is the first to be executed and how many times it is executed. Well, obviously, our answer is the initialization part. That is correct !. However, the question "how many times" is very hard to answer, since in looping operation the possibility of repeated execution is a norm. Now, remember that in an ordinary *for* loop statement (meaning not a nested *for* loop), the initialization part is executed only once throughout the looping operation and it happened during the first time the program pointer points to the *for* loop statement.

How about the second expression to be executed? The answer for this one is the **conditional** expression. Remember also that the initialization is executed first, then the conditional expression follows to be evaluated (whether T -true or F-false). If the evaluation is tested or proven true, the statements within the body of the loop are executed. After executing all the statements, the program pointer loops again and will execute this time the incrementation/decrementation part, then the conditional expression is evaluated again. If the condition is proven or tested true, then all the statements within the body of the loop are reexecuted. This looping process (the execution of incrementation/decrementation and reevaluation of condition) are repeated or iterated until the conditional expression is evaluated to **false**.

How about the *delay(3000)* standard function? What is this for? We included this *delay(3000);* to animate the looping or iteration process. Without it, we can see the sequence 1 to5 in an instant flash (as though there is no iteration process that happens, a one by one display of numbers). The value 3000 inside the parentheses of the *delay( )* function means tthe computer would delay or pause the execution for about 3000 milliseconds or microseconds before proceeding to the next iteration process.

What if we are given a task to display the sequence number in horizontal form such as this display:

1 2 3 4 5

Our answer is simply to omit the "\n" control character. This control character "\n" means a new line or next line. The computer would display each generated sequence number in the same line, since there is no control character "\n" that instructs the computer to print the output on the next line after the other.

Now what if each generated number has a comma such as this display:

1, 2, 3, 4, 5,

We simply add a comma after the *%d (%d,)*. This will print a comma after each generated number. Remember that the value of the variable is printed wherever the *%d* is located, positioned or written.

Our first example features the incrementation formula *n++*. This formula is a C-style. It has an equivalent traditional formula: *n=n+1* and this formula is handy in our programming task. In real-life programming job, this formula is applied in so many business application systems such as automatic generation of ID numbers, Employee numbers, and other tasks that demands incrementation operation. (You will realize it once you finish your course in Computer Science or Engineering). The opposite of incrementation formula is decrementation. In incrementation, it increases by 1 (plus 1) while in decrementation it decreases by 1 (minus 1). Actually, we can specify whether we want to increment by 1, by 2, or by 3, and so on. This can be accomplished by using the combined assignment operator: **+=**. If we want to increment by 2, we will use *n+=2* or *n-=2* for decrement by 2. We can also use the traditional formula: n=n+2 for incrementation by 2, or n=n-2 for decrementation by 2. Let us go back to our previous example; when the program pointer loops again, our increment/decrement part will be executed. Since we use the **n++**, therefore it increments(increases) by 1. The initial (first) value of **n** is 1, that is why the first looping process will yield a value of 2 for variable **n.** Then the program pointer goes to the conditional expression: **n<= 5;** and evaluates it. If the condition is tested true, the statements within the body of the loop will be reexecuted . After executing the statements, the program pointer loops(iterates) again because the condition was still evaluated to true. Then, the incrementation expression is executed again by adding 1. This time the value of variable **n** which is 2 will become 3. After executing the increment expression, the program pointer will reevaluate the condition. If the condition is still evaluated to true, then the statements within the body of the loop will be reexecuted. This iteration (looping) process will be repeated all over and over again until the conditional expression: (**n<=5**) is proven false.

Example 2:

Write a program that generates the given sequence numbers:

5
4
3
2
1

Algorithm:

```
For (n=5; n>=1; n--) {
  Printf("\n  %d",n);
  Delay(3000);
}
```

1st Solution:

```
#include <stdio.h>
main()
{
int n;
clrscr();
printf("\n For Loop Example - Decrementation");
for (n=5; n>=1; n--) {
  printf("\n  %d",n);
  delay(3000);
}
getch();
}
```

Explanation:

Here in our first solution in example number 2, we initialize the variable **n** with **n=5.** Then the program pointer evaluates the conditional expression: **n>=1.** Since **n** has a value of 5, the condition is evaluated to true, therefore all the associated statements within the body of the loop are executed: *printf( ); delay( );*. After executing the statements, the program pointer loops again (since the condition is still true) and will execute the decrementation expression: **n--;** thus, the value of variable **n** is decreased by 1. This time the value of variable **n** becomes 4 and then the conditional expression with *n=1* is reevaluated. Since the value of **n** is 4 which is greater than 1, therefore the condition is evaluated to true once again. This will cause for the reexecution of all the statements within the body of the loop. After executing the statements, the program pointer loops (iterates) again, thus the decrementation is reexecuted. This time the value of variable **n** becomes 3. Then the conditional expression is reevaluated, and proven true again, thus causes the reexecution of all the statements within the body of the loop. This iteration process will be repeated by the program pointer all over and over again until such time the condition is evaluated to false. In other words, it will be evaluated to false only when the value of variable **n** becomes 0 (zero).

2nd Solution: (Example 2: using *while* loop statement)

```
#include <stdio.h>
main()
{
  int n;
  clrscr();
  printf("\n While Loop Example- Decrementation");
  n=5;
  while (n>=1){
    printf("\n %d",n);
    delay(3000);
```

```
    n--;
  }
  getch();
}
```

Explanation:

In our *while* loop solution, we placed the initialization part(n=5) before the loop statement. This enables the program pointer to know the first value of **n** when it evaluates the conditional expression: **n>=1**. Since **n** has a value of 5, therefore, the condition is proven and tested true. If the condition is evaluated to true then all the statements within the body of the loop are executed.

We could notice that the decrementation formula: **n--** is found inside the body of the loop. This means that the formula is among the statements to be executed. Furthermore, the decrementation formula is considered as one of the statements within the body of the loop.

The *delay( )* function will pause the execution for about 3000 milliseconds/microseconds. The invisible program pointer will loop again since the condition was still true. And the whole iteration process will be repeated until the condition is proven or evaluated to false. When this happens, the program pointer will execute the next statement. This statement can be found after the end (}) symbol of the looping statement. Not the last end (}) symbol of course! In this example, it is the *getch( )* function, because it is the statement that follows the loop's end symbol.

3rd Solution:

```
#include <stdio.h>
main()
{
  int n;
  clrscr();
  printf("\n Do While Loop Example – Decrementation");
  n=5;
  do {
    printf("\n %d",n);
    delay(3000);
    n--;
  } while (n>=1);
getch();
}
```

Explanation:

Like the *while* loop, the *do-while* loop has an initialization variable placed before it. Moreover, the decrementation formula is also found within the body of the loop. Their big difference is, with the *do-while* loop statement, the conditional expression is located at the bottom. This means that all the statements within its body will be executed at least once, regardless of any result to the evaluation of the conditional expression (whether true or false), since the program pointer is free to enter within the body of the loop in the first time of execution.

Example 3:

Write a program that calculates the sum of the given sequence numbers:
1 + 2 + 3 + 4 + 5

Algorithm:

```
For (I=1; I<=5; I++) {
   Printf("\n  %d",I);
   Sum =  Sum + I;
}
Printf("\n The sum: %d",Sum);
```

1st Solution: (using *for* loop statement)

```
#include <stdio.h>
main()
{
  int sum,i;
  clrscr();
  printf("\n For Loop Example - Accumulator");
  sum=0;
  for( i=1; i<=5; i++) {
   printf("\n %d",i);
   sum=sum+i;
  }
  printf("\n The sum:%d",sum);
getch();
}
```

Explanation:

In this example, we apply the accumulator formula **sum:=sum+ i**. Like the incrementation and decrementation formulas, it is useful and applied to the actual programming job in the company. The accumulator formula accumulates the value while the program pointer iterates and executes the statements within the body of the loop. The incrementation/decrementation and accumulator formulas are found mostly inside the body of the loop. That is why they are repeatedly executed many times. Their nature is always like this; you could notice that everytime the program pointer executes the
accumulator formula it stores or saves the computed value to the accumulator variable. The value is always updated everytime the formula is reexecuted. This further means that the old value stored in the accumulator variable (in this example it's the **sum**) is overwritten by the current (new) value. This updating process (of value) is repeated until such time the program stops executing the formula. This will happen usually when the conditional expression is evaluated to be **false.**

2nd Solution: (Example 3: using while loop)

```
#include <stdio.h>
main()
{
  int sum,i;
  clrscr();
  printf("\n While Loop Example-Accumulator");
  i=1;
  sum=0;
  while (i<=5) {
    printf("\n %d ",i);
    sum=sum+i;
    i++;
}
printf("\n The sum: %d",sum);
getch();
}
```

Explanation:

The *delay( )* standard function is no longer needed, since our concerned is only the accumulated value of variable **sum**. Here, we can see the different arrangement and places of the initialization part, conditional part, and incrementation part. We write and place the incrementation formula below the accumulator formula so that it will be executed after the accumulator. This arrangement really matters since the program pointer executes each statement from top to bottom. The output is wrong and different from what we expect, if we didn't place each part properly.

3rd Solution:

```
#include <stdio.h>
main()
{
  int sum,i;
  clrscr();
  printf("\n Do-While Loop - Accumulator");
  i = 1;
    sum=0;
    do {
      printf("\n %d",i);
      sum=sum+i;
      i++;
    } while (i<=5);
 printf("\n The sum: %d",sum);
 getch();
}
```

Explanation:

We initialize the accumulator variable: **sum=0**. This makes the program pointer to know the first value of the variable. The initial values of the variables used in the looping statements are highly required. They are needed for the computer to compute the equations and execute the whole program correctly.

Example 4:

Write a program that computes the factorial value of *n* (as input) and displays it.
For example:

Enter a number: 5 (or any no.) ← Input statement
The factorial value: 120 ← Output

Algorithm:

Enter a number: (n)
F=1;
For (I=0; I>=1; I--) {
  F = F * I;
}
Printf("\n The factorial value: %d",F);

1st Solution: (Using *for* loop)

```
#include <stdio.h>
main()
{
  int n,f,i;
  clrscr();
  printf("\n Enter a number:");
  scanf("%d",&n);
  f=1;
  for(i=n; i>=1; i--) {
    f = f * i;
}
printf("\n The factorial value:  %d",f);
getch();
}
```

Explanation:

We initialize the variable **f** (for factorial) with the value of 1 so that the computer would know its first value. To compute the factorial of **n** (n!) is to multiply the product of **n** to its preceding number all over again until the iteration process done by variable **i** reaches the value 1. For example, the factorial value of 5 is 120. This is how it is computed:

5! = 5*4*3*2*1
= 120

This way of computation is best suited with the use of decrementation formula **i--** since the iteration process of variable **I** is to decrease by 1 each time the loop operation is being performed. Take note also that the factorial value can be computed using the other way such as:

5! = 1*2*3*4*5
= 120

Since the product of the value is multiplied to its succeeding iteration (**i)** number until the value of **n** is reached, then the incrementation formula **i++** is best suited to this kind of solution or computation.

2nd Solution:

```
#include <stdio.>
main()
{
  int n,f,i;
  clrscr();
  printf("\n Enter a number:");
  scanf("%d",&n);
  f=1;
```

```
  i=n;
  while (i>=1) {
    f=f*i;
    i--;
  }
  printf("\n The factorial value: %d",f);
getch();
}
```

Explanation:

Here in *while* loop solution, we place the initialization part before the loop statement, then we place the decrementation formula inside the body of the loop. The conditional expression part is placed at the top of the loop.

3rd Solution:

```
#include <stdio.h>

main()
{

int n,f,i;
clrscr();
printf("\n Enter a number:");
scanf("%d",&n);
f=1;
i=n;

do {
  f=f*i;
  i--;
} while (i>=1);
printf("\n The factorial value: %d",f);
getch();
}
```

Explanation:

Like in while loop solution, we place the initialization part before the loop statement, then we place also the decrementation formula inside the body of the loop. Their big difference is that in *do-while* loop, the conditional expression is placed at the bottom of the loop. Moreover, the statements within the body of the loop are always executed at least once, regardless of any condition (whether true or false evaluation results).

Program Simulation Example:

Topic : Looping Statement

Example 1:

Determine the output of the following C programs:

```
#include <stdio.h>
main()
{
 int i=0; x=0;
clrscr();
for (i=1; i<=10; i*=2)  {
   ++x;
   printf(" %d",x);
}
getch();
}
```

OUTPUT (ANSWER):

1 2 3 4

Solution and Analysis:

Variable Monitoring Table:

Loop	x=x+1 (++x)	i=i*2 (i*=2)	i Computed
1st	1	i= 1 * 2	2
2nd	2	i= 2 * 2	4
3rd	3	i= 4 * 2	8
4th	4	i= 8 * 2	16 (False)
5th	5 (loop is terminated)		

Explanation:

The initial value of variables **i** and **x** is zero respectively. However, within the loop's conditional expression, the value of variable **i** was reinitialized to 1. Well, this reinitialization is important, so that the iteration process can be terminated. Otherwise, this will cause an infinite or

endless loop. Remember that the looping process is controlled with the third expression which is in this case an equation form: i*=2;

This C style equation syntax is equivalent to the traditional syntax i=i*2; this will result to 0 computed value if the initialization value of variable **i** remains zero (i=0*2).

When the loop process progresses, we could notice that the value of variable **i** will be tested or evaluated against the constant value of 10. You could notice that the looping process is terminated (stopped) at the 5th loop. This is because the variable **i** has already reached a value of 16 which is greater than 10.

Example 2:

```
#include <stdio.h>
main()
{
  int i=0, x=0;
  clrscr();
  while (i<20) {
    if (i%5==0){
      x+=i;
      printf("   %d",x);
     }
    i++;
  }
   getch();
}
```

OUTPUT (ANSWER):

0 5 15 30

Solution and Analysis:

Variable Monitoring Table:

Loop	I	x=x+i (x+=i)	x
1st	0	x=0+0	0
2nd	1		
3rd	2		
4th	3		
5th	4		
6th	5	x=0+5	5
7th	6		

8th	7		
9th	8		
10th	9		
11th	10	x=5+1015	
12th	11		
13th	12		
14th	13		
15th	14		
16th	15	x=15+15	30
17th	16		
18th	17		
19th	18		
20th	19		

Explanation:

The initial value of variable **i** is 0 as well as the value of variable **x**. Since the value of variable **i** is less than 20**,** therefore the conditional expression of *while* loop is evaluated to .T. (true). This will trigger the invisible program pointer to execute the statements within the body of the loop.

However, within the body of the loop there is an *if* conditional statement so the computer will test (evaluate) again if the conditional expression is **true**. In this case, since variable **i** has a value of 0, the equation(expression) i **% 5 = =0** is evaluated to .T. (true). Remember that 0 **%** 5 (zero modulus five) is equal to 0 (zero). Therefore, the remainder

of 0 over 5 is 0. This causes the program pointer to execute the statements within the body of the *if* conditional statement. In this case we have two lines:

x+=i;
printf("%d",x);

The x+=i is a C style equation syntax which means *x=x+i.* Since the value of variable **x** is 0 and the value of variable **i** is 0, that is why the resulting computed value is 0. Thus, our first output(answer) is 0. Now you could notice that only if the iteration process of variable **i** reaches to 5, 10, and 15 will the expression **i % 5 ==0** becomes true (divisible by 5); this further means that the conditional expression of **if** statement can become .T. (true) which triggers the program pointer to execute the *x+=i* and *printf(" %d",x)* statements. Take note also that when the iteration process of variable reaches to 20, the conditional expression of *while* loop which is **i<20** will be evaluated to .F. (false). This causes the termination of the looping process; thus eventually ends our program.

LAB ACTIVITY
TEST 3

1.Write a program that calculates and produces these two columns sequence numbers using the three looping statements:

Sequence nos. Squared

1 1
2 4
3 9
4 16
5 25

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

2.Write a program which produces the given sequence nos. (in alternate arrangement) using the three looping statements:

1, 5, 2, 4, 3, 3, 4, 2, 5, 1,

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

3. Write a program which produces the given sequence numbers (in alternate arrangement and reverse order of the problem no. 2) using the three looping statements:

5, 1, 4, 2, 3, 3, 2, 4, 1, 5,

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

4.Write a program to calculate the factorial value of the input number **n!** Use the incrementation formula (i++) for your solution instead of decrementation formula (i--)
which we had already use and applied in our previous example. Apply the three looping statements for your solutions:

Sample input/output dialogue:

Enter a no. 4
Factorial value : 24

(The computation is : 1*2*3*4=24)

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

5. Write a program that generates and displays the Fibonacci sequence numbers of **n** (as input). In Fibonacci, the current third number is the sum of two previous numbers. Apply three solutions using the three loop statements:

Sample input/output dialogue:

Enter a no. 9
Fibonacci series: 1 1 2 3 5 8 13 21 34

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

6. Write a program that calculates the **power** value of the input base number and exponent number. Apply three solutions using the three looping statements:

Sample input/output dialogue:

Enter base no. 5 ← Input data
Enter exponent no. 3 ← Second input data
Power value: 125 ← Output value

(The computation is : $5^3 = 5 * 5 * 5 = 125$)

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

7. Write a program to scan a number **n** and then output the sum of the squares from 1 to n. Thus , if the input is 4, the output should be 30 because : $1^2 + 2^2 + 3^2 + 4^2$
$1 + 4 + 9 + 16$
$= 30$

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

8. Write a program to caluculate the sum of the sequence no. from 1 to **n.** Thus if the input is 6, the output should be 21 because: (Apply the three looping statements in your solutions)

$$1 + 2 + 3 + 4 + 5 + 6 = 21$$

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

9. Write a program that reverses the input number n. Formulate an equation to come up with the answer: (Apply the three loop statements in your solutions)

Sample input/output dialogue:

Enter a number: 1238 ← Input data
Reverse number: 8321 ← Output value

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

10. Write a program to scan a number **n** and then output the sum of the **powers** from 1 to **n.** Thus, if the input is 4, the output should be 288 because:

$$1^1 + 2^2 + 3^3 + 4^4$$
$$1 + 4 + 27 + 256 = 288$$

1st Solution – using *for* loop statement
2nd Solution – using *while* loop statement
3rd Solution- using *do while* loop statement

Chapter 5

Functions as Subprograms

"Functions break large computing tasks into smaller ones and enable people to build on what others have done instead of starting over from scratch."

-Brian W. Kernighan

A **function** is a subprogram(subroutine) that performs a specific task. Usually, we call or invoke this program(function) from our main program: *main( ).* Mostly, we use functions to break up a large program into simpler and manageable parts (modules). In other words, a function-oriented program is clearer to understand, easier to debug, and easier to maintain and modify.

In reality, there are two types of functions. The first one is the **standard function** and the second one is the **programmer-defined function**. The standard functions are predefined functions or built-in functions of the C system compiler (be it Borland's Turbo C or Microsoft C). These kinds of functions are numerous; in fact, majority of the statements we had used and applied in our C programs were standard functions. We could easily know them since they are the C statements with an open and close parentheses: ( ).

So what are they? They are the *main( ), clrscr( ), printf( ), scanf( ), getch( ),* and a lot more. Truly, function is the building block of C programming language. If we may use an analogy, functions are the hollow-blocks and C is the high-rise building. Without hollow-blocks , probably there is no building (You know that it is so difficult if not impossible to construct a building using rocks or stones, isn't it?).

A function can return a value (which it does usually) or not. When we want a function to simply perform a task, without returning a value, then we can explicitly specify the data type as *void* before the function name. By convention, C's parameter passing is **call by value**. All the examples that we will study here are called by value. We have to take note that majority of the actual application of functions are called by value, that is why we focus into it.

Here is the syntax of a function:

```
 datatype functioname (parameter variables)
{
 variable declarations;
statements;
return
}
```

Example 1:

Write a function-oriented program that performs a printing task for a long line above the word : "OFFICIAL RECEIPT" and below it. Call or invoke this function from the main program two times.

Algorithm:

```
Main( )
{
  lines( );
  printf("\n OFFICIAL RECEIPT");
 lines( );
}

void lines(void)
{
 printf("\n----------------------------------------");
}
```

Solution:

```
#include <stdio.h>
void lines(void);
main()
{
clrscr();
lines();
printf("\n OFFICIAL RECEIPT");
lines();
getch();
}

/* Here is now our function */

void lines(void)
{
 printf("\n ----------------------");
 return 0;
}
```

Explanation:

Here in our program, we can see the function heading written above the *main*( ) function (main program). This function heading tells the C compiler that there is a subprogram called *lines( )* below the main program.

When the program pointer encounters the function name (in this example it is the *lines( )*), it directly jumps into that subprogram, and will execute or perform all the statements within the body of the function. In this example, there are two times the function name was called by the main program or twice encountered by the program pointer within the main program's body. This further means that the program pointer jumped two times to perform the task which is done by the function (subprogram).

This particular function example returns no value, thus *void* data type is explicitly written which precedes the function name. Furthermore, since there is no value to be passed into the parameter variable, *void* is explicitly written inside the function's parentheses. The explicit declaration of data type *void* is a good programming practice, since it tells the C compiler in advance about the non-returning of value and no value to be passed whatsoever into the parameter variable.

We can shortened the *printf*( ) statement code for the *lines( )* function and animate it by using loop statements and *delay( )* function. For example to shorten the code and animate it, we need these program fragments:

```
for (i=1; i<=20; i++) {
   printf("-");
   delay(3000);
}
```

This fragment tells the computer to generate the symbol "-" twenty times and delay the displaying process (each time it generates) for about 3000 microseconds.

Example 2:

Write a function-oriented program that emulates the *sqr( )* function of Pascal programming language. Display the square and square root values of the input **n.**

Algorithm:

```
Main( )
{
Printf("\n Enter a number:");
Printf("\n The square value: %d",sqr(n));
Printf("\n The square root value: %f",sqr(n));
}


int sqr(int num)
{
  int s;
  s=num*num;
  return s;
}
```

Solution:

```
#include <stdio.h>
#include <math.h>
int sqr(int num);
main()
{
  int n;
  clrscr();
  printf("\n Enter a number:");
  scanf("%d",&n);
  printf("\n The square value: %d",sqr(n));
  printf("\n The square root value: %f",sqrt(n));
  getch();
}

int sqr(int num)
{
   int s;
   s=num*num;
   return s;
}
```

Explanation:

In this example, we applied the two types of function, the programmer-defined function and the pre-defined (standard) function. First, we construct the *sqr( )* programmer-defined function, then it was called from our main program: *main( ).* Every time the program pointer encounters the function name *sqr( ),* it will jump into that subprogram and executes all the statements within the body of the function. After executing all the statements, the program pointer goes back to the main program (or calling program) and proceeds to the next statement.

In the case of *sqrt( )* standard function, the program pointer jumps into the *math.h* (math header file) and search for *sqrt( )* subroutine. After finding and executing the whole *sqrt( )* function, the program pointer jumps back to the calling main program and proceed to the next statement. Furthermore, the math header file contains all the mathematical functions such as the *sin( ), cos( ), tan( ), cotan( ),* and a lot lot more. When you plan to use one of them, don't forget to include the math header file in your program.
The syntax is *#include* <*math.h*>.

Passing no Value or Parameter Passing! (Pass by Value)

Our example 1 was passing no value, however in our example 2, the value of the argument variable **n** was passed into the parameter variable **num** of *sqr( )* function. In reality, pass by value is passing only a copy of the original value. Therefore, the original value of argument variable **n** is not changed nor affected by whatever happens inside the body of the function.

The copied value received by the parameter variable **num** will only take effect within the body of the function. Once the program pointer goes back to the calling main program, the current active value is the original value of argument variable **n.** What happens to the value of **num**? The value is simply discarded after the function was executed.

Example 3:

Write a function-oriented program that computes the area of a rectangle. Use the formula: Area=Length*Width.

Algorithm:

```
Main( )
{
Enter the length (le)
Enter the width (wi)
Printf("\n The area of a rectangle: %d",rect(le,wi));
}

int rect(int l, int w)
{
 int a;
 a=l*w;
return a;
}
```

Solution:

```
#include <stdio.h>
int rect(int l, int w);
main()
{
int le, wi;
clrscr();
printf("\n Enter the length ");
scanf("%d",&le);
```

```
printf("\n Enter the width ");
scanf("%d",&wi);
printf("\n The area of a rectangle: %d",rect(le,wi));
getch();
}

int rect(int l, int w)
{
 int a;
 a = l * w;
return a;
}
```

Explanation:

In this example, we passed two values into the two parameter variables: **l** and **w**. The two values came from the argument variables (of the main program): **le** and **wi**. This example demonstrates the syntax on how to pass multiple values to parameter variables and how they are declared.

The *return* command simply returned the value of a variable specified. When the function returned no value, you can simply write *return 0* to tell the computer explicitly that the function returns nothing .

Example 4:

Write a function-oriented program that computes the factorial value of an input **n**! Then display the computed factorial value on the screen.

Algorithm:

```
Main( )
{
Enter a number (n)
Printf("\n The factorial value:%d",facto(n));
}

int facto(int no)
{
 int f,I;
f=1;
for ( I=n; I>=1; I--) {
  f= f*I;
}
return f;
}
```

Solution:

```
#include <stdio.h>
int facto(int no);
main()
{
  int n;
  clrscr();
  printf("\n Enter a number");
  scanf("%d",&n);
  printf("\n The factorial value:%d",facto(n));

  getch();
}

int facto(int no)
{
 int f,i;
f=1;
for (i=no; i>=1; i--) {
  f = f * i;
}
return f;
}
```

Explanation:

Here in our example, we passed a value to the parameter variable **no**. We could notice that in our previous examples, the parameter variable is always included in the equation. The value of variable **no** is simply used to initialize the loop statement and nothing more. This further means that not all values passed are used for computation process inside the body of the function. The passed value can be used for some other purposes such as controlling how many times the loop should be iterated (repeated).

Function Using Pointers
(Pass By Reference)

Call by value is the first way a function (subprogram) can have arguments passed to it. In this method, the value of argument (variable) is passed to the parameter of the function. However, only a copy of this value is passed, that is why changes made to the parameter (variable) of the function have no effect to the original value of the argument variable. The value or data of the argument variable remains the same. Argument variables are declared within the main program (main( )).

Call by reference is the second way a function(subroutine) can have arguments(variables) passed to it. In this method, the memory location (address) of an argument variable is copied into the parameter variable. Within the body of the function(subprogram), the memory address is used to access the actual argument variable applied in the call. That is why changes made to the parameter variable can affect its value. The original value of the argument variable is replaced by a new value. Call by reference is simply means a reference to the memory address.

Functions and Pointers: Call by Reference (A SUMMARY)

"When an expression is passed as an argument to a function, a copy of the value of the expression is made, and it is this copy that is passed to the function. This mechanism is known as **call -by –value** and is strictly adhered to in C programming language. Suppose that **v** is a variable and **f( )** is a function. Then a function call such as **f(v)** cannot change the value of variable **v** in the calling environment, because only a copy of the value of variable **v** is passed to **f( ).** In other programming languages, however, such a function call can change the value of variable **v** in the calling environment. The mechanism that accomplishes this -is known as **call-by-reference**. To get the effect of call-by-reference in C, we must use pointers in the parameter list in the function definition, and pass addresses of variables as arguments in the function call."

-Herbert Schildt, Author
Turbo C/C++
The Complete Reference

"Pointers are used in programs to access and manipulate memory location (address). We have already seen the use of addresses as arguments to *scanf*( ). A function call such as *scanf("%d",&v)* causes an appropriate value to be stored at a particular address (location) in the computer's main memory (RAM). If **v** is a variable, then *&v* is the address or location in memory of its stored value."

-Herbert Schildt, Author
Turbo C/C++
The Complete Reference

Example:

This example demonstrates how the values of the argument variables that are passed as memory address have been changed or affected by the operation of the function call by reference.

```
#include <stdio.h>
int swap(int *x, int *y, int t);
main
{
int x,y,t;
clrscr();
x=10;
y=20;
t=30;
printf("\n The original value of x: %d",x);
printf("\n The original value of y: %d",y);
printf("\n The original value of t: %d",t);
printf("\nTheir values after calling the function and");
swap(&x,&y,t);
printf("\ntheir effect when passed by reference/pointers");
printf("\n x+1 = %d",x+1);
printf("\n y+1 = %d",y+1);
printf("\n t+1 = %d",t+1);
getch();
}

int swap(int *x, int *y, int t)
{
t = *x;
*x = *y;
*y = t;
}
```

Sample Run and Output:

The original value of x: 10
The original value of y: 20
The original value of t: 30
Their values after calling the function and
their effect when passed by reference/pointers
x+1= 21
y+1= 11
t+1= 31

Explanation:

To declare a pointer variable, we precede it with the asterisk symbol (*). When we passed it as argument variable, we precede it with the ampersand symbol (&). When a call is passed by reference (using pointer), the original value is affected by the changes made which happened within the body of the called function (subroutine). However, if a call is passed by value, the original value is not affected by the changes made that happened within the body of the function (subprogram).

If we try to analyze the output: x + 1 = 21 and t + 1 = 31, we could notice that it has confirmed the explanation above. The original value of **x** is 10, however it was affected by the changes during the function call. So it becomes 20. This value 20 of **x** will take effect throughout the whole operation at the main program as well as to the subprogram.

The value of variable **t** should be the same with the value of variable x, since there is an expression t = *x; within the body of the function. However, the variable **t** is passed by value, that is why its original value which is 30 remains the same and unaffected by what happened within the body of the function.

Global and Local Variables

The **local variable** is a variable that is declared within a subprogram (function), while the **global variable** is declared for the use of the entire program (including the subprogram). This means that a local variable can only be seen and use within a function (subroutine). The global variables on the other hand, are variables that can be seen and used by any main program (function) or subprograms. This makes them different in actual application.

Let us now have an example that demonstrates how these global and local variables work in action. In addition, we will use a famous example to illustrate their technical inner workings:

```
#include <stdio.h>
void localswap(int x, int y);

int a,b,t;  /* globally declared variables */
main()
{
   clrscr();
   a = 1;
   b = 4;
   t = 3;
   printf("\n %d  %d  %d",a,b,t);
   localswap(a,b);
   printf("\n %d   %d  %d",a,b,t);
   getch();
 }


void localswap(int x, int y)
{
  int t;       /* locally declared variable */
```

```
    t = x;
    x = y;
    y = t;
    printf("\n %d  %d  %d",x,y,t);
}
```

The Output:

```
1  4  3
4  1  1
1  4  3
```

Explanation:

The program above has two variables called **t.** The first one was declared as global variable and the other one was declared as local variable. The global variable can be found at the upper-most part of our program, usually before the word *main( ),* while the local variable can be found within the body of the function (subprogram).

Even though these two variables have the same name, they are not related to each other. When the computer executes our main program, the value of the global **t** will be saved for future use or will be used for present processing demand. When the computer executes the function (subprogram), the value of the local variable is the one being used for processing. After the execution of the function, the global variable will be again reactivated for use.

The local variable which has the same name with the global variable will only take effect when the computer executes again the function where it belongs. Any changes of values made to the local variable have no effect on the global variable of the same name.

You could notice that the last *printf* command in our program produces an output: 1 4 3, because after the swapping function is executed, it produces an output: 4 1 1 the original(old) values of variables **a b** and **t** were retained. In other words, any changes made to the local variable have no effect on the global variable, even if they have the same name. Furthermore, since locally declared variables cannot be seen and known by other subprograms (functions), we don't need to be bothered remembering them and still we can reuse the same names to other subprograms (functions).

Program Simulation Example:

Topic : Functions as Subprograms

Example 1:

Determine the output of the following C programs :

```
#include <stdio.h>
int funct(int x);
main()
```

```
{
   int c;
   clrscr();
   for (c=1; c<=5; c++) {
       printf("\n  %d",funct(c));
}
getch();
}

int funct(int x)
{
   int y;
   y=x*x;
   return y;
}
```

OUTPUT (ANSWER):

1 4 9 16 25

Solution and Analysis:

Variable Monitoring Table:

Loop	c=c+1 (c++)	y=x*x
1st	1	1
2nd	2	4
3rd	3	9
4th	4	16
5th	5	25

In this example, the function **funct** was called 5 times by the main program because it is written inside the body of the *for* loop statement (with a conditional expression: **c<=5**). Now since variable **c** is initialize to 1, we could expect that the loop will be iterated 5 times, since our third loop expression is c++ which means: **c=c+1**. You could notice that every time the loop iterates, the value of variable **c** is passed to the function. The function processes this passed value of variable **c** after the parameter variable **x** receives it. The process or computation takes place at the equation: **y=x*x**. Since the value of argument variable **c** is passed to the parameter variable **x**, the program pointer will simply hand it down to the statement within the body of the function and process it. After processing the passed value, the function returns the processed(computed) value by using the command *return.* We could observe that in the first loop, the value of argument variable **c** is 1. So, when this is passed to the parameter variable **x**; x now has a value of 1.

In the equation within the body of the function: **y = x*x**; this will result to a computed value: 1. Now on the second loop, the value of variable **c** will increase from value 1 to value 2 . This will again passed to parameter variable **x** which in turn causes to return a value of 4 since **y=2*2.**

Remember that this process will be repeated until the fifth loop. Eventually, the last output value is 25. Did you notice that the function(subprogram) is simply squaring the value of variable **c**? This is because the equation within its body is **y=x*x**.

Example 2:

Determine the output of the following C programs:

```
#include <stdio.h>
int subprog(int c);
/* globally declare variable y */
int y;
main()
{
    int c;
   clrscr();
   c=1;
   y=0;
   while (c<=5){
     printf("   %d",subprog(c));
     c++;
   }
   getch();
}

int subprog(int x)
{

   y+=x;
   return y;
}
```

OUTPUT (ANSWER):

1 3 6 10 15

Solution and Analysis:

Variable Monitoring Table

Loop	c=c+1 (++c)	y=y+x (y+=x)
1st	1	1
2nd	2	3
3rd	3	6

4th	4	10
5th	5	15

Explanation:

On the first loop, the value of variable **c** which is 1 is passed to the parameter variable **x**. Now variable x has a value of 1. Since the initial value of the local variable **y** (within the main( ) function) is 0, therefore the equation *y=y+x* yields a value of 1. On the second loop, the value of variable **c** is increased by 1 due to the equation of the third expression of the loop which is **++c**(means c=c+1). This results to the value of 2 of variable **c** which in turn passed into the parameter variable **x.** Since the previous computed value of global variable **y** yields to 1, therefore the equation *y=y +x* is equivalent to *y=1+2* that resulted to the computed value of 3. Now as you could observe that while the value of variable **c** will increase by 1 again as the loop iterates (repeats), this value will be passed to the parameter variable **x** and will be added to the currently computed value of global variable **y**. In this case, the value of argument variable **c** is 3 which is passed to the parameter variable **x** ,while the current value of variable **y** is 3, therefore the computed value of the equation *y=y+x* is now 6.

When we try to analyze this program deeper, we could predict that the next value of argument variable **c** is 4 which becomes the value of parameter variable x (after parameter passing). We can conclude that since the current value held of local variable **y** is 6 (and 6 plus 4 is 10), therefore the next output is 10. And you know that the next iterated value of argument variable **c** is 5 which will be passed to the parameter variable **x**. And since the global variable **y** has a current value of 10, so *y=y+x* (which is y=10+5) will yield a computed value of 15. We have to remember also that the global variable **y** should be declared above the main( ) function so that our subprogram(s) could see it and use it for manipulation or calculation.

LAB ACTIVITY
TEST 4

1.Write a function-oriented program to calculate the area of a circle. Use the formula: $A=\pi r^2$ where Pi(π) is equal to 3.1416 (approximately).

2.Write a function-oriented program that converts the input inches into its equivalent centimeters. One inch is equal to 2.54 cms. Display the converted centimeters value.

3.Write a function-oriented program to convert an input Fahrenheit into Celsius degree. Use the formula C=(5.0/9.0)* F-32.0 . Display the Celsius degree.

4. Write a function-oriented program to convert an input Celsius into Fahrenheit degree. Use the formula F=(9.0/5.0)*C+32.0 . Display the Fahrenheit degree.

5.Write a function-oriented program that generates the Fibonacci series numbers of **n** (as input) and display them. In Fibonacci, the current third number is the sum of two previous numbers.

6. Write a function-oriented program that calculates the power value of the input base number and exponent number. Then display the power value.

Sample input/output dialogue:

Enter base number: 5 ←——— Input data
Enter exponent number: 3 ←——— Second input data
Power value: 125 ←——— Output

(The computation is $5^3 = 5 * 5 * 5 = 125$)

7. Write a function-oriented program that calculates the sum of the squares from 1 to **n.** Thus, if the input is 3, the output should be 14 because:

$$1^2 + 2^2 + 3^2$$
$$1 + 4 + 9 = 14$$

8. Write a function-oriented program that calculates the sum of the sequence number from 1 to **n**. Thus, if the input is 5, the output should be 15 because:

$$1 + 2 + 3 + 4 + 5 = 15$$

9. Write a function-oriented program to convert the input dollar(s) into its equivalent peso. Assume that one dollar is equivalent to 51.60 pesos.

10.Write a function-oriented program that scans a number **n** and then output the sum of the powers from 1 to **n**. Thus, if the input is 3, the output should be 14 because:

$$1^1 + 2^2 + 3^3 = 1 + 4 + 9 = 14$$

Chapter 6

Arrays

"*Programs,* after all, are concrete formulations of abstract *algorithms* based on particular representations and structures of data."
-Niklaus Wirth

An **array** is a special type of variable which can contain or hold one or more values of the same data type with reference to only one variable name. In other words, the array variable has a common name identifier and can hold many values at the same time, provided that they have the same data type.

An array variable can be distinguished through a pair of square brackets and on specified number inside the square brackets: [ ] . The number inside the square brackets is called an **index** or **element**. In the case of **enumerated arrays,** we can distinguish its declaration through a pair of curly brackets: { }.

Here is the syntax of One-dimensional arrays:

datatype arrayname[index];

Example:

int ar[5]; or *float arrayvar[10];*

Graphical Representation Analogy

10	**20**	**50**	**80**	**30**

This example illustrates that the array variable **ar[5]** is an integer data type and we can store a maximum of 5 values into it. These values to be stored or manipulated must be all integer data (whole numbers).

Here is the individual value of array variable **ar[5]**.

```
ar[0] = 10;
ar[1] = 20;
ar[2] = 50;
ar[3] = 80;
ar[4] = 30;
```

Other examples:

```
char name[4];
float area[3];
```

Two-Dimensional Arrays syntax:

Data type arrayname[arow][acol];

Example:

int score[2][3];

Graphical Representation Analogy:

Column

0	**1**	**2**		
10	**80**	**30**	**Row**	**0**
40	**50**	**90**		**1**

Our two-dimensional array *score[r][c]* is an integer data type and it can hold only an integer data with a maximum of 6 values **([2] x [3]**). The row 2 and a maximum of 6 is simply called *2 by 3* **(2 multiplied by 3).** Using these graphical representations, we can understand the array easily. We have to think that the index-number inside the square brackets are part of the variable name. It is safe to say that array variable *ar[0]* is equivalent to ordinary variable **a0** , where **0**(zero) is part of a variable name. It is also the same with the **score[0][0]** which is equivalent to ordinary variable **score00**.
Here is the individual value of array variable **score[2][3]**.

```
score[0][0] = 10
score[0][1] = 80
score[0][2] = 30
score[1][0] = 40
score[1][1] = 50
score[1][2] = 90
```

The fist index-number of two dimensional arrays specifies the maximum *row(r),* while the second index-number specifies the maximum *column(c).* The value of each array is in between the intersection of row and column.

Example 1:

Write a program using one-dimensional array that loads or stores the 5 values into an array variable. The values are the resulting computation from a simple equation. Then display the stored values.

Algorithm:

Int no[5];

/* this first loop stores the 5 values */
for (I=0; I<5; I++)
   no[I] = I + 10;

/* this second loop displays the previously stored 5 values */
for (I=0; I<5; I++)
  printf(" %d ", no[I]);

Solution:

```
#include <stdio.h>
main()
{
int no[5];
int i;
clrscr();
printf("\n One-Dimensional Array");
/* this first loop stores the 5 values */
for (i=0; i<5; i++)
   no[i]= i + 10;

/*this second loop displays the previously  */
stored 5 values */

for (i=0; i<5; i++)
  printf(" %d",no[i]);
getch();
}
```

Explanation:

We use the ordinary variable **i** to control the loop and to manipulate the individual value. That is the main reason why we put variable **i** inside the square brackets (**[i]).** Every time the loop iterates, the value of **i** increases by 1. This process causes the array variable's index –number to change. With this application of loop statement in manipulating each value of an array, we can make our program shorter and more compact. Imagine if we are trying to manipulate an array variable with an index-number of **100**? Manual manipulation of this case needs probably 100 statements instead of two statements with the use of *for* loop statement.

Example 2:

Write a program using two-dimensional array that loads and stores 6 values into an array variables. The values are the resulting computation from a simple equation (given within the algorithm). Then display the previously stored values.

Algorithm:

Int num[2][3];

/* loads the values */
for (R=0; R<2; R++)
  for (C=0; C<3; C++)
    num[r][c] = (R*4)+(C+19);

/* displays the previously loaded values */

for (R=0; R<2; R++)
  for (C=0; C<3; C++)
    printf(" %d ",num[R][C]);

Solution:

```
#include <stdio.h>
main()
{
  int r,c;
  int num[2][3];
  clrscr();
  printf("\n Two Dimensional Array");
  /* loads the values */
  for (r=0; r<2; r++)
    for (c=0; c<3; c++)
        num[r][c] = (r*4)+c+19;

  /* displays the previously loaded values */
  for (r=0; r<2; r++) {
     for (c=0; c<3; c++)
        printf(" %d",num[r][c]);
     printf("\n");
  }
getch();
}
```

Explanation:

In two-dimensional array, we use the nested *for* loop to manipulate each value of an array variable. In a nested *for* loop statement (or any loop statement), the inner loop is the first to be satisfied. This further means that the program pointer will iterate all over and over again within the body of the inner loop until its conditional expression is proven and evaluated to **false**. When the evaluation becomes false, the program pointer will iterate to the outer loop, and if its conditional expression is proven true again, the initialization part of the inner loop will be reexecuted. This reexecution will reinitialize the value of variable **c** to 0. Then, the iteration process within the body of the loop is repeated until it is satisfied. This long process of inner and then outer loop iterations will stop only when the conditional expression of outer loop is evaluated and proven false.

Example 3:

Write a program using one dimensional array that determines the highest value among the five input values from the keyboard and prints the difference of each value from the highest.

Sample Input Data:

Enter five numbers: 10 20 15 7 8
The

Algorithm:

```
   Int n[5];
  Int high:
/* enter five values */
Enter five numbers ( n[5] )
For  (I=0; I<5; I++) {
     Scanf("%d",&n[I]);
}
high=0;
/*  determines the highest  */
for (I=0; I<5; I++) {
 if (high <n[I])
     high = n[I];
}

/* to display the difference */
for (I=0; I<5; I++) {
    printf("\n The difference: %d",high-n[I]);
}
```

Solution:

```
#include <stdio.h>
main()
{
int n[5];
int high, i;
clrscr();
/* enter the five values */
printf("\n Enter five numbers:");
for (i=0; i<5; i++) {
   scanf("%d ",&n[i]);
}
high = 0;
/* determine the highest */
for (i=0; i<5; i++) {
   if (high<n[i])
      high=n[i];
}
printf("\n The highest is: %d", high);
printf("\n");
printf("\n Their difference from the highest are:");
for (i=0; i<5; i++) {
  printf("\n %d difference: %d", n[i], high-n[i]);
}
getch();
}
```

Explanation:

Our first *for* loop statements is designed to accept the five input values from the keyboard. These five values are stored at the array variable **n[ ]**. By using a loop statement, we need only one *scanf( )* statement to accomplish this task. Without the loop, it requires us to have five *scanf( )* statement to scan the five values inputted from the keyboard.

What if we are required to accept 50 values from the keyboard? Without the array variable and looping statement, we need 50 individual variables to store each value and probably more than five *scanf( )* function to scan them; assuming, that we compressed 10 variables in one *scanf*( ) . What a messy code. Isn't it?

Our second *for* loop statement is designed to determine if the current value of ordinary variable **high** is less than the current value of array variable **n[i]**. If the value of variable **high** is less than the value of array-variable **n[i],** the corresponding value of **n[i]** is stored into the variable **high**. Whatever the previous value of variable **high** is simply overwritten by the new current value of array-variable **n[i]**.

The third loop will simply print each value of the array variable **n[i]** together with its difference from the highest value.

Example 4:

Write a program using two-dimensional arrays that determines the even numbers among the twelve input values from the keyboard and prints the list of these **Even** numbers.

Sample input data:

Enter twelve numbers:
8 9 7 10 25 30 69 101 1001 798 10111 100023

Algorithm:

Int nos[3][4]

/* to scan twelve numbers */

Enter twelve numbers
For (r=0; r<3; r++)
    For (c=0; c<4; c++)
        Scanf("%d",&nos[r][c]);
/* to determine and print the **Even** numbers */
For (r=0; r<3; r++)
    For (c=0; c<4; c++) {
        Mod=nos[r][c] % 2;
        If (Mod = = 0)
            Printf(" %d ", nos[r][c]);
    }

Solution:

```
#include <stdio.h>
main()
{
  int nos[3][4];
  int r,c, mod;
  clrscr();

  /* this loop scans the twelve nos */
  printf("\n Enter twelve numbers:");

  for (r=0; r<3; r++)
     for (c=0; c<4; c++)
        scanf("%d ", &nos[r][c]);

  /* this loop determines and prints the Even numbers */
  printf("\n Here is the list of Even number");
```

```
  for (r=0; r<3; r++)
     for (c=0; c<4; c++) {
        mod = nos[r][c] % 2;
        if (mod==0)
           printf(" %d", nos[r][c]);
     }
 getch();
}
```

Explanation:

The first loop accepts the 12 values from the keyboard. The second loop determines and prints the **Even** numbers. Here we use the variable *mod* to hold the remainder which yields from the calculation of the equation. The % symbol is the modulus (remainder) operator of C programming language. If the remainder is 0 then the number is an Even number, since it is divisible by 2.

Program Simulation Example:

Topic : Arrays

Example 1:

Determine the output of the given C program below:

```
#include <stdio.h>
main()
{
   int a, b=0;
   int c[10]={1,2,3,4,5,6,7,8,9,0};
   clrscr();
   for (a=0; a<10; a++)
      if ((c[a]%2)==0)
         b += c[a];
   printf("%d ",b);
   getch();
}
```

OUTPUT (ANSWER):

20

Solution and Analysis:

Variable Monitoring Table

a=a+1 (a++)	b=b+c[a] (b+=c[a])
0	0
1	0
2	2
3	2
4	6
5	6
6	12
7	12
8	20
9	20

Explanation:

The output value 20 is the sum of the array elements whose values are **even**. You could notice that only if the conditional expression (*c[a] % 2==0*) is evaluated to true will the equation *b+=c[a]* be executed. This is not noticeable in our solution where the output numbers are in pairs (evenly or divisible by 2). This further means that there is no change in the value of variable **b** if the iteration number of variable **a** is not an even number.

Example 2:

Determine the output of the given C program below:

```
#include <stdio.h>

main()
{
   int a, b=0;
   int c[10] = {1,2,3,4,5,6,7,8,9,0}
   clrscr();
   for (a=0; a<10; a++)  {
      if ((c[a]%2)== 0)
          b += c[a];
   printf(" %d \n",b);
   }
   getch();
}
```

OUTPUT (ANSWER) :

0
2
6

12
20

Solution and Analysis:

a=a+1 (a++)	b+=c[a]	(b=b+c[a])	b
0	0		0
1	0		
2	2		2
3	2		
4	4		4
5	4		
6	6		6
7	6		
8	12		12
9	12		
10	20		20
11	20		

Explanation:

The program above is almost similar to program number 1. The only difference is that we include a begin ({) symbol and end (}) symbol to our *for* loop statement. Plus, we include the backslash (\n) within our *printf*( ) function. This will affect our output greatly. Since our *printf*( ) function is written inside the body of the *for* loop statement (or whatever looping statement is being used) it will be performed or executed one or more times as long as the conditional expression (***a<10)*** of the loop statement is evaluated to true. Unlike in our first example, the *printf( )* function will only be executed if the conditional expression (**a<10**) of the loop statement is finally evaluated to false. In this particular case, the *printf( )* function will only be executed once (and wait until such time the loop iteration process ended or evaluated to false).

Example 3:

Determine the output of the given C program below:

```
#include <stdio.h>
main()
{
     int ar[5];
     int i;
     int sum=0;
     clrscr();
     /* initializing the array variable */
     for (i=0; i<5; i++)  {
        ar[i] = i + 3;
```

```
    }
   /* displaying the value of array variable */
   for (i=0; i<5; i++)  {
      printf("\n ",ar[i]);
   }
   printf("\n \n ");
/* computing the sum of the array variable */
for (i=0; i<5; i++) {
    sum += ar[i];
}
printf("\n \n The sum: %d",sum);
getc();
}
```

OUTPUT (ANSWER):

3
4
5
6
The sum is: 18

Solution and Analysis:

i= i + 1 (i++) ar[i]= i+3 sum = sum + ar[i] (sum+=ar[i])

3	0	ar[0] 3 = 0 + 3
4	1	ar[1] 4 = 1 + 3
5	2	ar[2] 5 = 2 + 3
6	3	ar[3] 6 = 3 + 3
\0		ar[4]

In this example, we have three for loop statements that are used to manipulate the array variable. The first one is to initialize the array variable. This means we store an initial values to each array element. On the first loop iteration, the array variable **ar[i]** contains the value 3 since equation ar[i] = i + 3 generates a value of 3 because the iterationnumber of
variable **i** is 0. So 0+3 is equal to 3 which is stored at array variable **ar[i]**. The next time the loop iterates, the iteration value of ordinary variable **i** is 1. This makes the value of array variable **ar[i]** to become 4 (1+3). Take note that the element (index) number of array variable **ar[i]** will increase by 1 every time the loop iterates. This element (index) number is controlled by the ordinary variable **i** which is placed inside the square brackets.

You could also notice that whenever we want to use an array variable, we will use a *for* loop statement to manipulate each value of an array variable. This is because the *for* loop construct is best suited to control and manipulate an array variable since its element (index) number is sequential in

nature (or in index form). In other words, it is arranged from 0, 1, 2, 3, 4, 5… and so on (sequential order). This maybe the reason why the element number inside the square brackets is also called **index** number, because its
number refers to a particular part of an array variable and serves as a point of reference for manipulation and searching purposes.

Remember that whatever we store or initialize at the first *for* loop statement, it will be simply displayed in the second *for* loop statement. That is basically what the second *for* loop statement is doing. In the third loop, as the loop iteration progresses, it accumulates the value of variable **sum** while the loop is in iteration process. And it only stops accumulating the value when the conditional expression **i<5** is evaluated to false.

LAB ACTIVITY
TEST 5

1.Write a program using one-dimensional array that calculates the sum and average of the five input values from the keyboard and prints the calculated sum and average.

2. Write a program using one-dimensional array that determines the lowest value among the five input values typed from the keyboard and prints the difference of each value from the lowest.

3.Write a program using one-dimensional array that accept five input values from the keyboard. Then it should also accept a number to search. This number is to be searched if it is among the five input values. If it is found, display the message “Searched number is found!”, otherwise display “Search number is lost!”.

4.Write a program using two-dimensional arrays that determines the ODD numbers among the 12 input values typed from the keyboard and prints the list of these ODD numbers.

Sample input/output dialogue:

Enter twelve numbers:

8 9 10 7 25 30 69 101 798 10111 10023

Odd numbers are:

9 7 25 69 101 10111 10023

5.Write a program using two dimensional arrays that searches a number and display the number of times it occurs on the list of 12 input values.

Sample input/output dialogue:

Enter twelve numbers:

15 20 13 30 35 40 16 18 20 18 20

Enter a number to search: 20
Occurrence(s) : 3

6. Write a program using two-dimensional arrays that calculates the sum and average of the twelve input values which the user would type from the keyboard and prints the calculated sum and average.

7. Write a program using two-dimensional arrays that determines the highest and lowest of the 12 input values.

Sample input/output dialogue:

Enter twelve numbers:

13 15 20 13 35 40 16 18 20 18 20 19

The highest is: 40
The lowest is: 13

8. Write a program using two-dimensional arrays that lists the Odd numbers and Even numbers separately in a given 12 input values.

Sample input/out dialogue:

Enter twelve numbers:
13 15 20 13 35 40 16 18 20 18 20 19

Odd numbers are:
13 15 13 35 19

Even numbers are:
20 40 16 18 20 18 20

9. Write a program using two-dimensional arrays that computes the sum of data in rows and sum of data in columns of the 3x3 (three by three) array variable n[3[3].

Sample input/output dialogue:

```
 5   9   8 = 22
 3   8   2 = 13
 4   3   9 = 16
---------------------
12  20  19
```

10. Write a program using one-dimensional array that searches a number if it is found on the list of the given 5 input numbers and locate its exact location in the list.

Sample input/output dialogue:

Enter a list of numbers:
5 4 8 2 6

Enter a number to be searched: 2
2 found in location 4

Chapter 7
Strings

"I consider it essential that programs are shown in final form
with sufficient attention to details, for in programming,
the devil hides in the details."
-Niklaus Wirth

In C programming language, **string** is considered as a sequence or series of characters and treated as a single data item. A string data is enclosed within the double quotes. It includes letters, symbols, digits, and control characters. **Strings** are used in programs to store and process text data manipulation.

Since string is a sequence or an array of characters, it is declared as *char* (character) data type and an array variable. Turbo C has a library routines for string manipulations called <*string.h*>. Every time we want to use some of these string functions, we are required to include this string function library to make our program runs properly.

Example of string data:

"I love you Tara!"
"Tara B. Williams"
"Tara, 143"
"$ 143.44"
"Ms."
"14344"

The Basic String Functions of Turbo C Language

1. *getch( )* – a string input function. The function reads text from the keyboard until the Enter key is pressed.

Syntax: gets(string_var);

2. *puts( )* – a string output function. This function displays the string value (which is stored from the string variable).

Syntax: puts(string_var);

3. *strlen( )* – this function returns the length of the string.

Syntax: strlen(string);

4. *strcpy( )* – this string function copies the content of string2 to string1. The str1 and str2 can be a variable or a string data (value).

Syntax: strcpy(str1, str2);

5. *strcat( )* – this string function concatenates the strings. It appends (add/join) string2 to the end of string1.

Syntax: strcat(str1,str2);

6. *strlwr( )* – this string function converts all the uppercase letters in string to lowercase.

Syntax: strlwr(str);

7. *strupr*( ) – this string function converts all lowercase letters in string to uppercase.

Syntax: strupr(str);

8. *strrev*( ) - this string function reverses all the characters in the string.

Syntax: strrev(str);

9. *strcmp*( ) – this string function compares two strings. If string1 >string2, the function returns a positive value; if string1<string2, the function returns negative value; if string1= =string2, the function returns a zero (0) value.

Syntax: strcmp(str1,str2);

10. strcmpi( ) – this string function compares two strings and ignores whether an a uppercase or lowercase letters are being compared. Lowercase and uppercase letters are treated equal or the same.

Syntax: strcmpi(str1,str2);

11.*strncpy*( ) – this string function copies only a portion (size) of strings into string1.

Syntax: strncpy(str1,str2,size);

12. *toupper*( ) - this string function converts an input lowercase letter into its uppercase equivalent.

Syntax: toupper(vletter);

13. *tolower*( ) – this string function converts an input uppercase letter into its lowercase equivalent.

Syntax: tolower(vletter);

Example 1:

This program demonstrates how to copy a string data into the variable.

Algorithm:

strcpy(gn,"Tara");
strcpy(fn,"Williams");
puts(gn);
puts(fn);

Solution:

```
#include <stdio.h>
#include <string.h>

main()
{
   char gn[10], fn[10];
   clrscr();
   strcpy(gn,"Tara");
  strcpy(fn,"Williams");
   puts(gn);
   puts(fn);
   getch();
}
```

Explanation:

This *strcpy( )* example demonstrates the syntax on how we can copy the string into the variable and how we can display them. In our first string copy (*strcpy*( )) function, we copy the string data: "Tara" into the variable **gn**, and in our second string copy function, we copy the string value: "Williams" into the variable **fn**. We use the *puts( )* function to display the copied string data.

Example 2:

This program demonstrates how to copy a string data coming from the keyboard input using the *gets( )* function and prints them.

```
#include <stdio.h>
#include <string.h>
```

```
main()
{
char gn[10], fn[10];
char cgn[10], cfn[10];

clrscr();
printf("\n Enter your given name:");
gets(gn);
printf("\n enter your family name:");
gets(fn);
strcpy(cgn,gn);
strcpy(cfn,fn);
puts(cgn);
puts(cfn);
printf("%s  %s",gn,fn);
getch( );
}
```

Explanation:

We declare four string variables: **gn[10], fn[10], cgn[10], cfn[10]**. The *gets( )* function is used to get the string that is being inputted from the keyboard. The string value of variable **gn** is copied into the variable **cgn**, while the value of **fn** is copied into the variable **cfn**. Therefore when we put the string (*puts( )*) of **cgn** to the screen, the value is the same with the value of **gn**. This case is also the same with the **cfn** and **fn** string variables.

We use the *printf( )* function to display the value of string variables **gn** and **fn** at the same line. We used the data type format specifier: *%s* to display a string value.

Example 3: *strcpy( )*

This program demonstrates how the value of string1 is overwritten by the value of string2.

```
#include <stdio.h>
#include <string.h>
main()
{
char str1[10], str2[10];
clrscr();
strcpy(str1,"I love");
strcpy(str2,"Tara");
strcpy(str1,str2);
puts(str1);
puts(str2);
getch();
}
```

Explanation:

In our third *strcpy( ),* the value of **str2** is copied into **str1**: *strcpy(***str1,str2***).* The string value of **str1** which is "I love" was overwritten by a new value coming (copied) from **str2** that contains "Tara". This is the reason why our output is: **Tara** for the string function *puts(***str1***)* and *puts(***str2**).

Example 4: *strlwr( ), strupr( ), strrev( ), strlen( )*

This program demonstrates how the value of strings are being converted into lowercase or uppercase and how it is being reversed. Then it computes the length of the string.

```
#include <stdio.h>
#include <string.h>
main()
{
  char str1[15], str2[15];
  int length1,length2;
   clrscr();
   strcpy(str1,"Tara");
   strcpy(str2,"Williams");
   length1=strlen(str1);
   length2=strlen(str2);
   printf("\n The length of string 1 is:%d",length1);
   printf("\n The length of string 2 is:%d",length2);
   strcat(str1,str2);
   printf("\n The value of string1 is: %s",str1);
   printf("\n The value of string2 is: %s",str2);
   length1=strlen(str1);
   printf("\n The new length of string1 is: %d",length1);
   length2=strlen(str2);
   printf("\n The new and old length of ");
   printf("  string2 is: %d",length2);
   strlwr(str1);
   strupr(str2);
   printf("\n The string1 in all lowercase: %s",str1);
   printf("\n The string2 in all uppercase: %s",str2);
   strrev(str2);

   printf("\n The string2 in reverse order: %s",str2);
   getch();
   }
```

Explanation:

Here is the output of this program:

The length of string 1 is: 4
The length of string 2 is: 8
The value of string 2 is: TaraWilliams
The value of string1 is: Williams
The new length of string1 is: 12
The new length of string2 is: 8
The string1 in all lowercase: tarawilliams
The string2 in all uppercase: WILLIAMS
The string2 in reverse order: SMAILLIW

The original length of string1 is 4 because "Tara" is a four character string, while the original length of string2 is 8 because "Williams" is an eight character string. The new length of string1 is 12 since the value of string2 is concatenated(added or joined) into string1. In concatenation process (*concat( )*), only the value of string1 is affected, thus its value increases while the value of string2 remains the same.

Example 5: *strcmp( )*

This program demonstrates how the string compare function (*strcmp( )*) works.

```
#include <stdio.h>
#include <string.h>
main()
{
   char str[10];
   int c;
   clrscr();
   printf("\n Enter your password:");
   gets(str);
   c=strcmp(str,"Tara");
   if (c==0)
      puts("Welcome to the system!");
   else
      puts("Intruder detected! Police, police!");
   getch();
}
```

Explanation:

When I type "Tara" on the keyboard, this string value is stored into the string variable **str**. The *gets( )* function accomplishes this task. Since the value of **str** is the same with the value of string 2 which is "Tara", thus the string compare function *strcmp( )* returns the value 0 (zero) and stored it into the variable **c**. If the variable **c** has a value of 0 (or equal to 0), then our output statement *puts(*"Welcome *to* the system*");* is executed.

If the value of variable **c** is not zero (0) or other than zero then the associated statement of conditional *else* statement will be executed, instead. Now remember that our input password must start with a capital letter (in this case: Tara has a capital letter T), otherwise string compare function would return a non-zero value. For example, if we type the word "tara" (all in lowercase) will cause the program to display the message: "Intruder detected! Police police!", since "tara" and "Tara" string values are not equal. Capital letter "T" is not equal to lowercase "t" in *strncmp*( ) function syntax-rule.

Example 6: *strcmpi( )*

This program demonstrates how *strcmpi*( ) works and how it differs from the *strcmp*( ). The *strcmpi* ( ) ignores the lowercase or uppercase difference between the two compared values. Capital letters and lowercase letters are treated equally and the same.

```
#include <stdio.h>
#include <string.h>
main()
{
   char str[10];
   int c;
   clrscr();
   printf("\n Enter your password:");
   gets(str);
   c=strcmp(str,"Tara");
   if (c==0)
      puts("\n Welcome to the system!");
   else
      puts("\n Intruder detected! Police, police!");
   getch();
}
```

Explanation:

In this example, whether we type "Tara" (with a capital letter) or "tara" (all in small letters), our output is the same: "Welcome to the system!" because they are treated equal and the same in *strcmpi( )* function. In other words, the difference in lowercase and uppercase is simply ignored. Most of the login name and password are designed and programmed this way. You can type them in all capital

letters, all small letters, or combination of both. Well, in some network operating system, this case is not applicable, instead it demands an exact letters like in our previous example.

Example 7: *strncpy*( )

This program demonstrates how *strncpy*( ) function works technically. This string function copies only a portion (a number of characters) of the given string.

```
#include <stdio.h>
#include <string.h>
main()
{
   char target[15], source[15];
   clrscr();
   strcpy(target,"Taramylove!");
   strncpy(source,target,7);
   puts(source);
   getch();
}
```

Explanation:

Here is the output of this program:

Taramyl

Here in our program, we specify the number of characters to be copied only. We specify **7**. Therefore our output is only: **Taramyl**. The counting of characters starts from left to right. In this example, it starts from **T** and ends with **l** (the number 7 character).

Program Simulation Example:

Topic : Strings

Example 1:

Determine the output of the given C program below:

```
#include <stdio.h>
#include <string.h>
main()
{
char str1[20], str2[20];
int lengthst;
clrscr();
printf("\n Enter string 1:");
gets(str1);
lengthst = strlen(str1);
strcpy(str2,str1);
printf("\n %s",str2);
strlwr(str1);
puts(str1);
printf("\n %d",lengthst);
getch();
}
```

Solution and Analysis:

Suppose we enter the string:

I love Tara Williams

Our OUTPUT (ANSWER):

```
I love Tara Williams
i love tara williams
20
```

Explanation:

The first output is: *I love Tara Williams* produced by the output statement *printf("\n %s", str2).* We could notice that we copied the input string value from **str1** to **str2** using the string function **strcpy( )**. This is the reason why our **str1** and **str2** string variables have the same string value after this string function **strcpy( )** statement was executed.

We have the second output: *i love tara williams* in all small letters caused by the string functions: **puts( )** and **strlwr( )**. And the last output is 20 triggered by the string function **strlen( )**.

Example 2:

Determine the output of the given C program below:

```
#include <stdio.h>
#include <string.h>
main()
{
  char str1[15], str2[15];
  int length1,length2;
  clrscr();
  strcpy(str1,"My true north");
  strcpy(str2,"Tara my love");
  length1=strlen(str1);
  length2=strlen(str2);
  printf("\n The length of string 1 is: %d",length1);
  printf("\n The length of string 2 is: %d",length2);
  strcat(str1,str2);
  length1=strlen(str1);
  length2=strlen(str2);
  printf("\n The new length of str1 is: %d",length1);
  printf("\n The new length of str2 is: %d",length2);
  printf("\n%s",str1);
  strupr(str2);
  printf("\n%s",str2);
  strrev(str1);
  printf("\n  %s",str1);
  getch();
}
```

Solution and Analysis:

OUTPUT (ANSWER):

The length of string 1 is: 13
The length of string 2 is: 12
The new length of string 1 is: 25
The new length of string 2 is: 12
My true northTara my love
TARA MY LOVE
htron eurt yM

Explanation:

Our first line output is **13** caused by our string function: strlen(str1). The message: *My true north* is equivalent to 13 characters (including the spaces between words). Our second line output is **12** since our second message (stored at the **str2** string variable) is equivalent to 12 characters (including the spaces between words). Our third line output is **25** since we concatenated the string values of string variable 1 (str1) and string variable 2 (str2).

Our fourth line output is **12** since string variable 2 (str2) remains 12 characters. Only string variable 1 (str1) is affected to the concatenation operation, based on the syntax-rule of *concat*( ) string functions. Our fifth line output is : *My true north Tara my love* generated by the concatenation operation: **concat(str1,str2)**. Our sixth line output is an all upper case message triggered by the **strupr( )** string function. And the last output is in reverse order caused by the **strrev( )** function.

LAB ACTIVITY
TEST 6

1.Write a program using standard string functions that accepts a price of an item and display its coded value. The base of the key is:

X	C	O	M	P	U	T	E	R	S
0	1	2	3	4	5	6	7	8	9

Sample input/output dialogue:

Enter price: 489.50
Coded value: PRS.UX

2.Write a program using string functions that accepts a coded value of an item and display its equivalent tag price.
The base of the key is:

0	1	2	3	4	5	6	7	8	9
X	C	O	M	P	U	T	E	R	S

Sample input/output dialogue:

Enter coded value: TR.XX
Tag price: 68.00

3. Write a program using string function that determines if the input word is a palindrome. A palindrome is a word that produces the same word when it is reversed.

Sample input/output dialogue:

Enter a word: AMA
Reversed: AMA
“It is a palindrome”

Enter a word: STI
Reversed: ITS
“It is not a palindrome”

4.Write a simple encryption program using string functions which apply the substitution method. Here is the given Substitution Table.

Substitution Table:

A	*
E	$
I	/
O	+
U	-

Sample input/output dialogue:

Enter message: meet me at 9:00 a.m. in the park
Encrypted message: m$$t m$ ***t 9:00 *.m. /n th$ p***rk

5.Write a simple decryption program using string functions which will apply the Substitution Method. Here is the given Substitution Table.

Substitution Table:

*	A
$	E
/	I
+	O
-	U

Encrypted message: m$$t m$ ***t 9:00 *.m. /n th$ p***rk
Decrypted message: meet me at 9:00 a.m. in the park

6. Write a program using string functions that will accept the name of the country as input value and will display the corresponding capital. Here is the list of the countries and their capitals.

COUNTRY	CAPITAL
Canada	Ottawa
United States	Washington D.C.
U.S.S.R.	Moscow
Italy	Rome
Philippines	Manila

7. Write a program using string functions that will accept the name of the capital as input value and will display the corresponding country.

CAPITALS	COUNTRIES
Ottawa	Canada
Washington D.C.	United States
Moscow	Russia
Rome	Italy

Manila Philippines

8. Write a program that will accept the currency value and the name of the country and will display the equivalent in U.S. dollar, based on the given list:

COUNTRY	CURRENCY	U.S. DOLLAR EQUIVALENT
British	Pound	0.6 U.S. dollar
Canadian	Dollar	1.3 U.S. dollar
Japanese	Yen	140 U.S. dollar
German	Mark	1.7 U.S. dollar
Philippines	Peso	53 U.S. dollar

9. Write a program that takes nouns and forms their plurals on the basis of these rules:

a.If a noun ends in “y”, remove the”y” and add “ies”
b. If a noun ends in “s”, “ch” or “sh”, add “es”
c. In all other cases, just add “s”

10.Write a program that takes data, a word at a time and reverses the words of the line.

Sample input/output dialogue:

Input string value: birds and bees
Reversed: bees and birds

Chapter 8

Introduction to Visual C++ 2012

The Examples will Run also in Visual C++ 2008 and Visual C++ 2010

All the program examples in this book are coded using Microsoft Visual C++ 2012 Express Edition. However, the main reference books that I have used are based on Visual C++ 2010. With this reason, backward compatibility is not a problem. You can use either the new Visual C++ 2012 or the older versions which are Visual C++ 2008 and 2010 releases. You can use even the much more older Visual C++ 2005 too.
All program examples in this book will run successfully on any editions of Visual Studio whether you are using Express Edition, Professional Edition, Premium Edition, or Ultimate Edition.

Visual Studio Express Is a Free Edition

The Express Edition is configured to be used by students, developers, hobbyists and consultants for testing and experimentation purposes. This means that you cannot use the programs you develop in Express Edition for commercial use and implementation due to its inherent limitation. You have to use the licensed edition (Professional, Premium, or Ultimate edition), when you plan to develop a business application system that will be used by companies or government agencies. Whatever you learn in Visual Basic Express Edition is applicable to the licensed edition. So, the Express Edition is a good way to start mastering the language. So, what are you waiting for? Grab your copy now? Copy of what? Copy of the Microsoft Visual Studio 2012 Express Edition and this book too. This is just a suggestion, okay? Don't get offended, buddy.

A Brief History and Background of C++

The C++ is a superset of C programming language. This language was invented by Dr. Bjarne Stroustrup of AT&T Bell Laboratories, in 1985. This is the standard Object Oriented Programming (OOP) language, meaning, the other newly invented programming languages that are OOP language in nature adheres to its design philosophy.

The computer scientists had observed that the Structured Programming languages like Pascal and C can no longer meet the present demand of computing. More and more users wanted more graphics; programmers and developers wanted more powerful language that can make larger programs easier to develop, manage, and maintain. This is the very reason why C++ was created: to come up with a programming language that makes larger programming tasks easier and faster to design and construct.

Common Elements of the Visual C++ IDE

The Integrated Development Environment (IDE) is the workspace where we construct together all the components of our application system such as the place where we design our forms and controls as well as the place where we develop our code or programs.

We just simply discuss here the common elements of the Visual C++ IDE which we commonly used in our programming task.

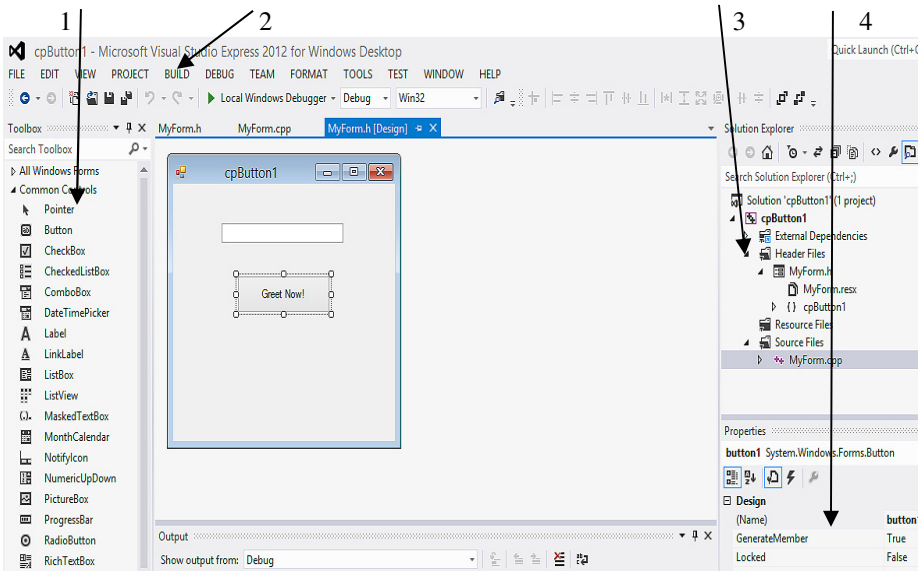


Figure 1.1 Visual C++ Integrated Development Environment (IDE) Start Page

Here are the meaning of the arrows and legends that we point to:

1 - ToolBox
2 – Menu Bar (Menu System)
3 –Solution Explorer
4 –Properties Window

The Start Page

We can use the Start Page to select from recent projects. We can also select a New Project from here by clicking the New Project icon at the Toolbar or as a menu item at the File menu.

Note:
The Recent Projects are simply the list of recent projects that we have designed and developed. There is no other special meaning about them.

The New Project Dialog

When we create a new project, we select this menu item from the File menu or alternatively, we click the New Project icon (the first icon) from the Toolbar. We will be presented with different Visual Studio installed templates such as Windows Form Application, Class Library, Console Application, and more. In this Series 1 book, we focus more on the Windows Forms Application. So, we select always the Windows Forms Application template.

Menu Bar

The menu bar (menu system) is actually like the menus we have seen in other Windows application software such as MS Word, MS Excel, and PowerPoint. It is the line of text that lies across the top of the Visual C++ window. This menu will give us access to many features within the integrated development environment (IDE). Some of the menu items that we can see at the Menu bar are File, Edit, View, Tools and more.
In the **File** menu, we will work with the actual files that make our application system. Here we can open or save the projects or programs that we have created.
In the **Edit** menu, we can perform the standard Clipboard options such as cut, copy, and paste. In the View menu, we can view various component and tools. We can view the Class View, Error List, Output, Properties Window as well as other utilities that help make our program development time more productive, easy, and fast.

Toolbars

Below the menu bar are list of icons which are called toolbars. They provide us a quick access to commonly used menu items (as an alternate way to search for a command), we simply click an icon (which is an equivalent to that menu-bar command we need, and presto, it serves our wishes. Plus, we can dock this list of icons beneath the menu bar or above it, or dock it sideways.

Toolbox

The toolbox contains the tools or objects (controls) that we might want to place (drag and drop) on a Form control in our application system on which we presently design and develop. An example of these objects are Buttons control and Text boxes control.

Solution Explorer

The Solution Explorer tracks the item in our projects. If we want to add new items on our project, we simply use the menu items in our Project menu, such as Add Windows Form. Furthermore, the Solution Explorer sees things in terms of files or folders. Like for example, the References folder holds the currently referenced items such as *namespaces* in a project. Now if we want to set the properties of the various items in our project, we will simply click the Properties folder.

Properties Window

The properties window is docked right under the Solution Explorer window. It provides us the lists of the property settings from the currently selected form or other control objects. A **property** in Object-Oriented Programming (OOP) term is a characteristic of an object such as its size, caption, or color. The characteristic is not limited to the appearance of an object (control) but also about the way it behaves.

Output Window

The Output Window echoes the result when we build and run our programs. This is where the debugging results occur. On this Output Window, we can see the list of errors our program generated.

Context Menus

The context menus contain shortcuts to our frequently performed commands. To open the context menu is easy, we simply click the right mouse button on the control or object we are currently using. The specific list of shortcuts available from context menu depends on the part of the environment where we click the right mouse button.

Form Designer

The form designer is our drawing board where we design graphically the layout of the form and controls of our application system. In short, this is where we draw our graphical-user interface (GUI). Here we can add controls, pictures, or other graphics to a form to design the look we want. We have all the powers to make our GUI to look funny or elegant in the eyes of the beholder (our target users). Imagine how powerful we are? It doesn't take a professional artist to do that; but a clear mind and a heart with a good taste of beauty as an art.

Code Designer

The code designer window serves as a program editor where we can write our program (code) behind each form and control (object) we design or where we can place the code module we develop in our application system. We can use the tabs at the top center of the IDE to switch between the Form Designer such as the **Form1**.**h**[**Design**] and the Code designer such as **Form1**.**h**.

Note:
The *.cpp* of the Form1 means *c plus plus*.

Class View

The Class View window which is located next to the Solution Explorer, is actually another tab that sits beside the Solution Explorer, presents solutions and projects in terms of the classes they contain, and the members of these classes. Furthermore, the class view window provides us an easy way of jumping to a member of the class that we want to access quickly. We simply find it in the Class View window, and double-click it to bring it up in the code designer.

The IntelliSense

The IntelliSense is a box that pops-up as we write the code into our Code Designer, and lists all possible options and even completing our typing of code for us. This is useful when we cannot remember what built-in Visual C++ method accepts, because it will display those arguments as we type in the call to that method.
Moreover, the IntelliSense offers syntax tips which display the syntax of the statement we are typing. This is great, if we know what statement we have to apply but cannot recall its exact syntax.

Component Trays

The **Component Trays** are one of the most useful components of Visual C++ IDE, most especially when our application system requires us to put a timer, a menustrip, a toolstrip, or contextmenustrip in our program. These Timer, MenuStrip, ToolStrip, ContextMenuStrip and ToolTips controls will appear in the component tray which we can see at the bottom part of our Form during the time we design our program.

Program Conventions Used

Here in this book, the code to be embedded in the procedure are written in bold letters. For example:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
  this->textBox1->Text="Hello World!";
}
```

The above code demonstrates that the statement:

this->textBox1->Text="Hello World!";

is the code to be embedded (typed) within the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
.........
.........
}
```

In short, the statements you are instructed to type or to embed within the method appears in bold letters. In most situations, the statements which are not in bold letters are compiler system-generated code, so there is no need to type them. In the above example, they are:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {


}
```

Explaining Properties, Methods, and Events briefly

Properties can be considered as an object's attributes, methods as its actions, and events as its responses. For example, a button's attributes are its caption, color, and size. An object such as a button can perform methods and respond to events. This event can be a click by the mouse or a key-press from the keyboard. The performed method could be to show a form object or hide it instead.

Tip:
The *caption* refers to the *Text* property of a particular control.

Warning!
The Sample Output may differ from the given Figure in our example. The author is trying his very best to provide you with the exact sample output. However, due to time constraints, the output is but just nearer to the given figure.

Tip:
A *method* is simply a *function* that is a member of a class. In structured programming era, a *function* is a subprogram that performs a specific task. Function is also commonly called a *subroutine*.

Tip:
During the high time of Pascal programming language (it earns the title of being the structured programming flagship), there is another type of subprogram that looks and acts almost like a *function*. It is called *procedure*. A *procedure* is a type of subprogram that performs a specific task, and after doing the task, it returns no value. The Pascal creator argues that this is how procedure differs from a function, because a function will always return a value. It could be a valuable value returned, or in other case it returned a null value (in case of no value to return to). The declaration of a function that returns no value will always contain a *void* keyword.

Specific Solutions *Steps for Visual Studio 2012 and Beyond?

We try to be backward compatible with previous versions of Visual Studios (VS):VS 2002, VS2003, VS2005, VS2008, and VS2010. Now, in the new VS 2012, we have a new ways (new steps) of accomplishing a Windows Forms Application. That's because, the new VS2012 is by default a text-based (console-based) UI (user-interface), not in GUI (graphical-user-interface) programming set-up like we usually do in Visual Basic or Visual C# programming languages. As a matter of fact, we can no longer choose the Windows Forms Application new project template in VS2012 version upon creating a project. Because it is no longer an option, nor can be found in the given options. This is the biggest difference between the new VS2012 and the older VS versions. That is why in trying to make our solution steps to be compatible with older VS versions, we put an asterisk symbol (*) to specific solution *steps that are applicable in VS2012 only. Here they are:
The following *Steps are specifics for VS2012 version (or beyond?) only:

1. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
2. *Name the new application system **projectName**.
3. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
4. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace projectName;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
```

```
Application::Run(gcnew MyForm());
return ;
```

5. *At the Menu bar again, select Project menu, then select the *projectName Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

In other words, all the remaining steps in our solution is applicable to the older Visual Studios (VS) versions. Well, with some minor modification of steps in our solution. For example, we can directly create our Windows Forms Application right at the step of creating our project because we can choose the Windows Forms Application right away.
So, instead of having this step in VS2012:

Name the new application system:* ***projectName.*

We have this step in older VSs (VS2002,VS2003,VS2005, VS2008 and VS2010):

Create a new Windows Forms Application and name the new application system: ***projectName.***

Do you understand? I wish you do.

Chapter 9

Designing Basic Controls or Objects

"When people ask me what is the best way to learn programming or how to use a specific language, I tell them to experiment - write as many small programs as you can, as each one will teach you more - and each one will add another trick to your tool bag."
- Alan Cooper
(Father of Visual Basic)

Controls or objects are the primary medium of the user's interaction with the application system on which we design and develop. Mostly, we use controls to get the user input and to display the corresponding output. The other controls provide us an access to some application system and process data or information as though the remote application system is part of our program. The basic controls that we can use in our application system are buttons, text box, label, check boxes, radio buttons, list box, and combo box. By clicking or typing something on these controls, the user can accomplish their tasks.

Using Button and Text box

The function of a Button is to carry out a command or action (event) as the user clicks it, while the text box provides an area to input or display text. We can use the Button to extract simple responses from the user or to invoke special functions on the form. The text boxes are commonly used to display string or numeric data and to accept user input or for entering data.

Example 1:

Design and develop an application system that when the user clicks the Greet Now! button, the message "Hello World!" will be displayed at the text box. Follow the given figure below in designing and developing the application system.

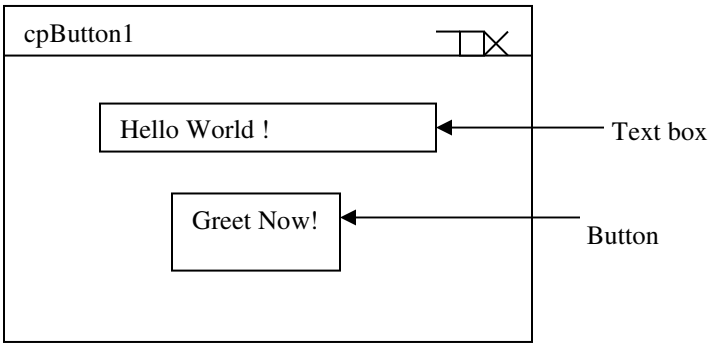


Figure 2.1 Button and Text box design

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpButton1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpButton1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **cpButton1**.
7. Click, drag and drop a *text box* from the Toolbox to add it to the form. Position it properly and adjust its width appropriately. Next, click, drag and drop a *button* from the Toolbox to add it into the Form, then set its Text property to **Greet Now!**
8. Double-click the *button* on the Form to display the button1_Click event method. Embed the following lines (in bold only) into the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
   this->textBox1->Text="Hello World!";

}
```

9. *At the Menu bar again, select Project menu, then select the *cpButton1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

10. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Note:
CLR means Common Language Runtime.

Sample Output:

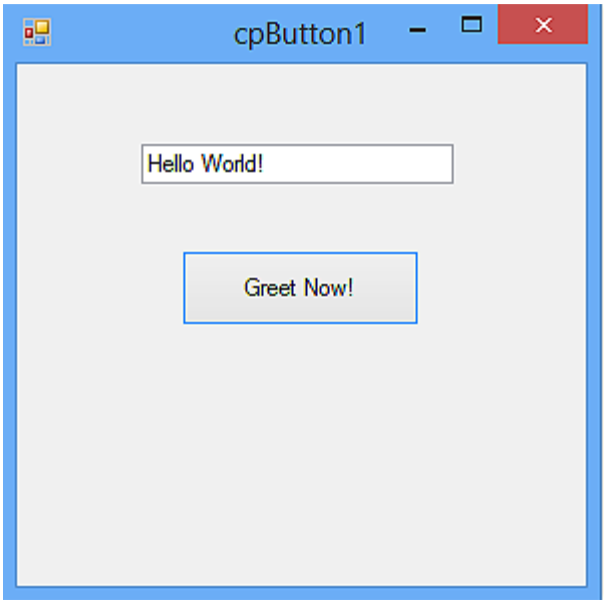


Figure 2.1a Button and Text box Output

Note:
The *this* keyword refers to the current instance of the class. It is also used as a modifier of the first parameter of an extension method. Moreover, the application of *this* keyword is useful when you want to qualify members hidden by similar names.

Explanation:

We usually use the *button* to get simple responses from the user such as clicking it to do some action . We will notice here that the embedded code (within the method):

```
this->textBox1->Text="Hello World!";
```

is simply changing the *Text* property of the object named *textBox1* to display "Hello World!" at the Text box control. Text box is commonly used for accepting input or for entering data, but in this example, it is used to display the data. In the next chapter, we will use text boxes for accepting input data. As of the moment, we will focus on how to output the data on the text box control. The syntax for our example takes the format of

object->property

where *textBox1* is the object (control) and *Text* is the property. We can use this syntax to change property settings for any form or control in response to events that occurs while our application system is running. The specific event here is a click event (generated by clicking the *button*).

Example 2:

Design and develop an application system that when the user clicks the Greet Now! *button*, the message "Hello World!" will be displayed at the *message box*. Follow the given figure below in designing and developing the application system.

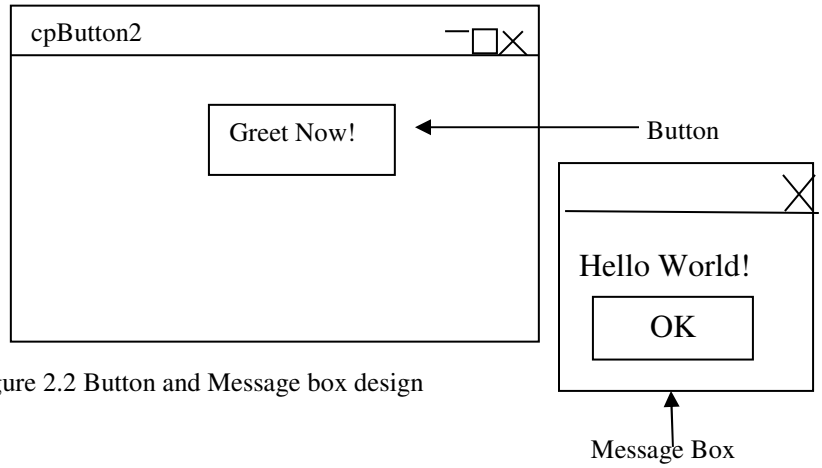


Figure 2.2 Button and Message box design

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpButton2**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpButton2;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
```

```
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **cpButton2**.
7. Click, drag and drop a *button* from the Toolbox to add it into the Form, then set its Text property to **Greet Now!**
8. Double-click the *button* on the Form to display the button1_Click event method. Embed the following lines (in bold only) into the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
  MessageBox::Show("Hello World!");
}
```

9. *At the Menu bar again, select Project menu, then select the *cpButton2 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows (/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

10. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

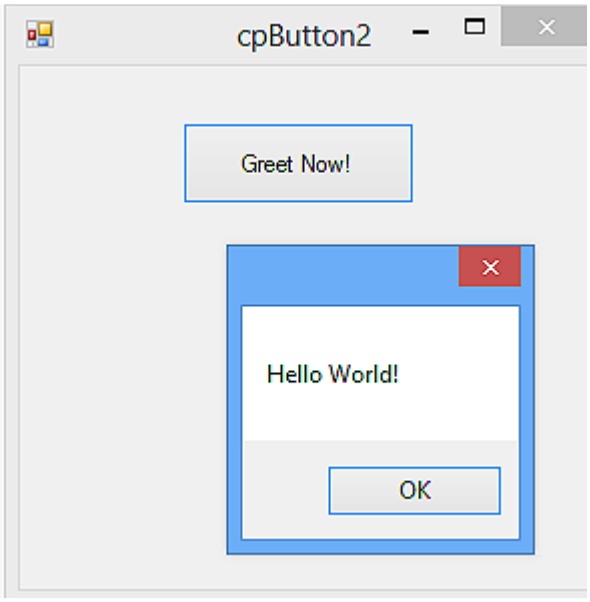


Figure 2.2a Button and Message box Output

Explanation:

We usually use the *button* to get simple responses from the user such as clicking it to do some action . We will notice here that the embedded code (within the method):

MessageBox::Show("Hello World!");

is simply to display the message "***Hello World!***" at the Message box control.
A *message box* is used for displaying simple messages to the user.

Using Check Boxes, Radio buttons and Message Box

Check boxes are valid as single controls, however they are not mutually exclusive. Meaning, the user can check as many check boxes as they want, unlike in Radio buttons, the user can only select one option at a time. Check boxes are applicable to the situation where you can select a lot of items because you want them all, such as you can order a pizza and choose all the ingredients you want to put on it as long as you can afford to pay. While in radio buttons, you can only choose one and only one. The radio buttons is applicable to this situation - marrying a girl. In the Christian world, no matter how many girlfriends you have at the same time or in the same place (shame on you brother!), you can only choose one among them to marry, unless you change your religion into something else (better?) that allows polygamous marriages (which I like – joke only).

Example 3:

Design and develop a simple Check box and Text box application that when the user clicks one of the three check boxes, it will indicate in the text box on which check box the user had clicked. For example if Check box 2 was clicked by the user, it will display "Check box 2 is clicked!" at the text box. It will do the same with Check box 1 and Check box 3. Follow the given figure below in designing and developing the application system.

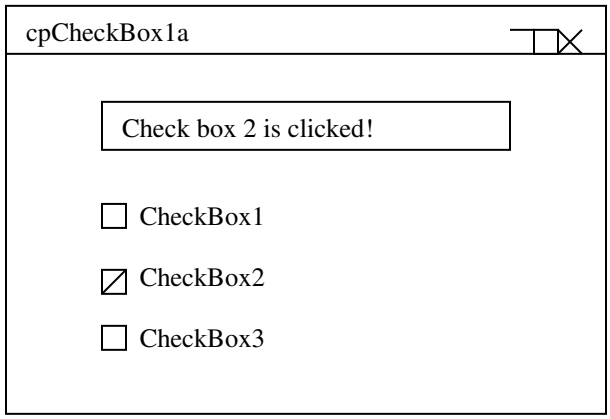


Figure 2.3 Check box and Text box design

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpCheckBox1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpCheckBox1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **cpCheckBox1a**.
7. Click, drag and drop three (3) *Check boxes* from the Toolbox to add them into the Form, then set their respective Text property to **CheckBox1**, **CheckBox2**, and **CheckBox3!** Finally, click, drag and drop a *Text box* from the Toolbox to add it into the Form.
8. Double-click the *Check Box 1* on the Form to display the checkBox1_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void checkBox1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
   this->textBox1->Text="Check box 1 clicked!";
}
```

9. Double-click the *Check Box 2* on the Form to display the checkBox2_CheckedChanged event method. Embed the following lines (in bold only) into the method:

```
private: System::Void checkBox2_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
   this->textBox1->Text="Check box 2 clicked!";
}
```

10. Double-click the *Check Box 3* on the Form to display the checkBox3_CheckedChanged event method. Embed the following lines (in bold only) into the method:

```
private: System::Void checkBox3_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  this->textBox1->Text="Check box 3 clicked!";
}
```

11. *At the Menu bar again, select Project menu, then select the *cpCheckBox1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows (/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.
12. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

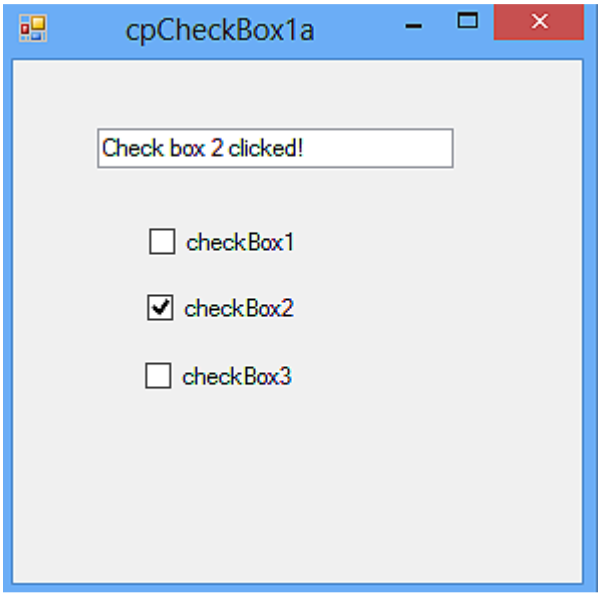


Figure 2.3a Check box and Text box Output

Explanation:

Although a check box control is rather similar to a radio button which we will describe in the upcoming examples, there are two fundamental differences between them. You can select many choices in check boxes, while in radio button you are only allowed to select one option at a time. You can observe that when you click the check box 1 (Check 1) then you click the check box two (Check 2), these two check boxes have the check mark on both of them. And even when you

click the check box three (Check 3), it will also contain the check mark. In radio button, this will not be the case, only one radio button will contain the bullet or the big dot at a time, because once you click the next radio button, the next radio button will contain the bullet as though it transfers from the previous radio button.
In this example, we will notice here that the embedded code (within the method)

```
this->textBox1->Text="Check box 1 clicked!";
```

is simply changing the Text property of the object named *textBox1* to display "Check box 1 is clicked!" at the Text box control. The same thing happen to other check boxes if we click them. The syntax for our example takes the format of *object->property,* where *textBox1* is the object (control) and *Text* is the property. We can use this syntax to change property settings for any form or control in response to events that occurs while our application system is running. The specific event here is the click event which is generated by clicking the check boxes.

Example 4:

Design and develop a simple Check box and Message box application that when the user clicks one of the three check boxes, it will indicate in the Message box on which check box the user had clicked. For example if Check box 2 was clicked by the user, it will display "Check box 2 is clicked!" at the Message box. It will do the same with Check box 1 and Check box 3. Follow the given figure below in designing and developing the application system.

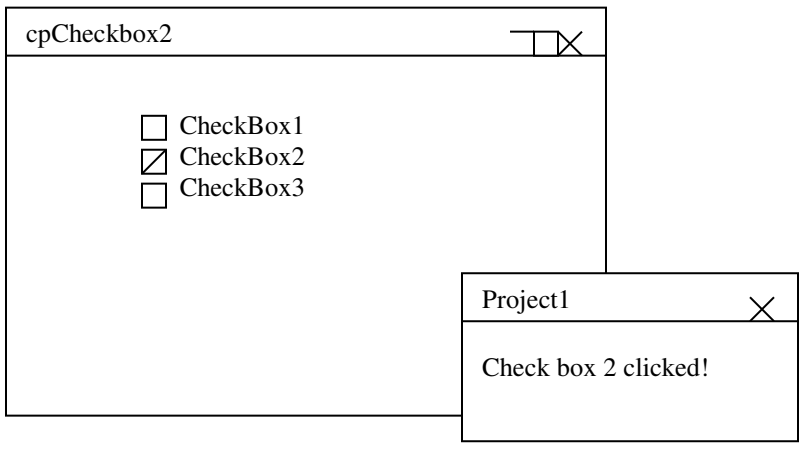


Figure 2.4 Check box and Message box design

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpCheckBox2**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpCheckBox2;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **cpCheckBox2**.
7. Click, drag and drop three (3) *Check boxes* from the Toolbox to add them into the Form, then set their respective Text property to **CheckBox1**, **CheckBox2**, and **CheckBox3!**
8. Double-click the *Check Box 1* on the Form to display the checkBox1_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void checkBox1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  MessageBox::Show("Check box 1 clicked!");
}
```

9. Double-click the *Check Box 2* on the Form to display the checkBox2_CheckedChanged event method. Embed the following lines (in bold only) into the method:

```
private: System::Void checkBox2_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  MessageBox::Show("Check box 2 clicked!");
}
```

10. Double-click the *Check Box 3* on the Form to display the checkBox3_CheckedChanged event method. Embed the following lines (in bold only) into the method:

```
private: System::Void checkBox3_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  MessageBox::Show("Check box 3 clicked!");
}
```

11. *At the Menu bar again, select Project menu, then select the *cpCheckBox2 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

12. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

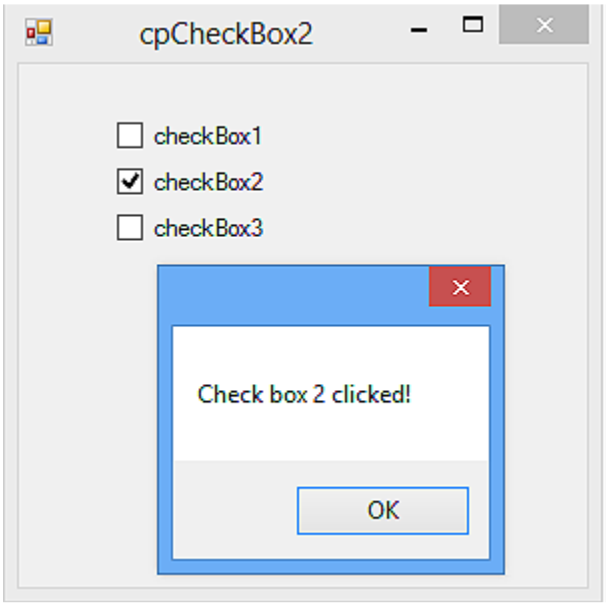


Figure 2.4a Check box and Message box Output

Explanation:

A message box is used for displaying simple messages to the user.
We will notice here that the embedded code (within the procedure)

MessageBox::Show("Check box 1 clicked!");

is simply to display the message “Check box 1 is clicked!” at the Message box control. The same thing happened to other check boxes if we click them. The Message box control is a pop-up dialog box that displays the message that we would like to convey to the user.

Example 5:

Design and develop a simple Text box and Radio buttons application that when the user clicks one of the three radio buttons, it will indicate in the Text box on which radio button the user had clicked. For example, if radio button 2 was clicked by the user, it will display “Radio button 2 is clicked!” at the Text box. It will do the same with Radio button 1 and Radio button 3. Follow the given figure below in designing and developing the application system.

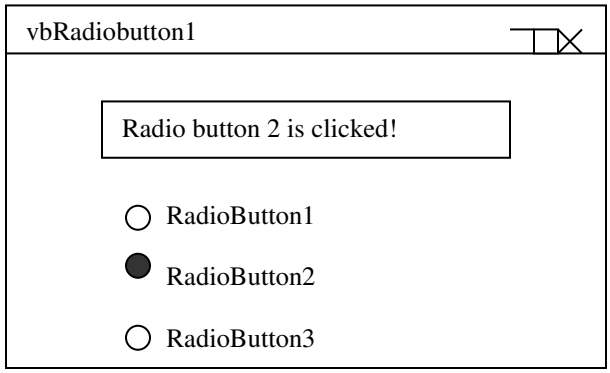


Figure 2.5 Text box and Radio buttons design

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpRadioButton1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpRadioButton1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
```

```
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **vbRadioButton1**.
7. Click, drag and drop three (3) *Radio buttons* from the Toolbox to add them into the Form, then set their respective Text property to **RadioButton1**, **RadioButton2**, and **RadioButton3.** Finally, click, drag and drop a *Text box* from the Toolbox to add it into the Form.
8. Double-click the *Radio button 1* on the Form to display the radioButton1_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  this->textBox1->Text="Radio button 1 clicked!";
}
```

9. Double-click the *Radio button 2* on the Form to display the radioButton2_CheckedChanged event method. Embed the following lines (in bold only) into the method:

```
private: System::Void radioButton2_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  this->textBox1->Text="Radio button 2 clicked!";
}
```

10. Double-click the *Radio button 3* on the Form to display the radioButton3_CheckedChanged event method. Embed the following lines (in bold only) into the method:

```
private: System::Void radioButton3_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  this->textBox1->Text="Radio button 3 clicked!";
}
```

11. *At the Menu bar again, select Project menu, then select the *cpRadioButton1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

12. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

Figure 2.5a Text box and Radio buttons Output

Explanation:

Now you will see the effect of using the radio button. As I have said, you can only select one option at a time. Once you click the next radio button, the bullet or the big black dot will transfer from the previous button to the currently clicked radio button.
We will notice here that the embedded code (within the method):

```
this->textBox1->Text="Radio button 1 clicked!";
```

is simply changing the *Text* property of the object named *textBox1* to display "Radio button 1 clicked!" at the Text box control. The same thing happened to other radio buttons if we click them. The syntax for our example takes the format of *object->property,* where *textBox1* is the object (control) and *Text* is the property. We can use this syntax to change property settings for any form or control in response to events that occur while our application system is running. The specific event here is the click event which is generated by clicking the radio buttons.

Example 6:

Design and develop a simple Message box and Radio buttons application that when the user clicks one of the three radio buttons, it will indicate in the Message box on which radio button the user had clicked. For example if radio button 2 was clicked by the user, it will display "Radio button 2 is clicked!" at the Message box. It will do the same with Radio button 1 and Radio button 3. Follow the given figure below in designing and developing the application system.

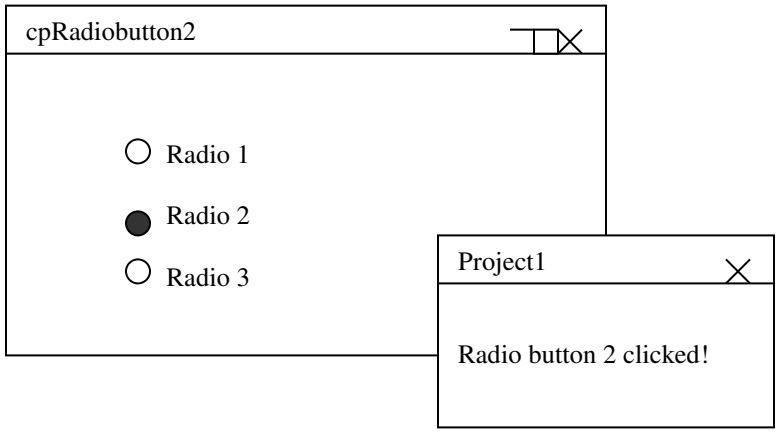


Figure 2.6 Message box and Radio buttons design

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpRadioButton2**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpRadioButton2;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **vbRadioButton2**.
7. Click, drag and drop three (3) *Radio buttons* from the Toolbox to add them into the Form, then set their respective Text property to **RadioButton1**, **RadioButton2** and **RadioButton3 .**
8. Double-click the *Radio 1* on the Form to display the radioButton1_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  if (radioButton1->Checked)
     MessageBox::Show("Radio button 1 clicked!");
}
```

9. Double-click the *Radio button 2* on the Form to display the radioButton2_CheckedChanged event method. Embed the following lines (in bold only) into the method:

```
private: System::Void radioButton2_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
   if (radioButton2->Checked)
     MessageBox::Show("Radio button 2 clicked!");
}
```

10. Double-click the *Radio button 3* on the Form to display the radioButton3_CheckedChanged event method. Embed the following lines (in bold only) into the method:

```
private: System::Void radioButton3_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
   if (radioButton3->Checked)
     MessageBox::Show("Radio button 3 clicked!");
}
```

11. *At the Menu bar again, select Project menu, then select the *cpRadioButton2 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

12. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

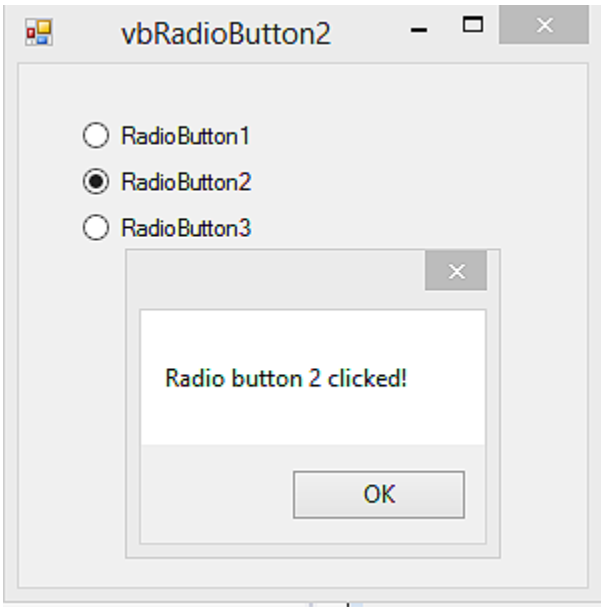


Figure 2.6a Message box and Radio buttons design output

Explanation:

You will noticed that the Radio button 1 is by default, the first control to be selected. It's because in Radio button controls, one of its control object must be selected, and only one of the controls must be selected exclusively. Unlike in Check boxes control, two or more check boxes can be selected simultaneously.
A message box is used for displaying simple messages to the user. We will notice here that the embedded code (within the method):

```
if (radioButton1->Checked)
  MessageBox::Show("Radio button 1 clicked!");
```

is simply to display the message "Radio button 1 clicked!" at the Message box control. The same thing happened to other radio buttons if we click them. The Message box control is a pop-up dialog box that displays the message that we would like to convey to the user.
The most noticeable part in our code is that we put the MessageBox method under the *if* conditional statement. This is due primarily to the fact that we have to check first if a particular radio button is being checked (selected), and if it is, we have to trigger the MessageBox to show the message. The MessageBox method needs to be triggered to display or show something, if we are using a radio button control, since this control has to be pre-selected first, as its default selection.
You can experiment based on this example, what would be the effect if you wouldn't apply the *if* conditional statement. You will see why we really need to put the conditional statement, after the experimentation. Because I am pretty sure, there is a side-effect that would come up with your simple solution.

Example 7:

Design and develop a simple Text box and Radio buttons application that when the user clicks one of the four radio buttons, it will indicate in the Text box on which radio button the user had clicked. For example if radio button 2 (Second Year) was clicked by the user, it will display "Sophomore!" at the Text box. It will do the same with Radio button 1, Radio button 3, and Radio button 4. The high school level of First Year is "Freshman", for Second Year is "Sophomore", for Third Year is "Junior" while for Fourth Year is "Senior". Follow the given figure below in designing and developing the application system.

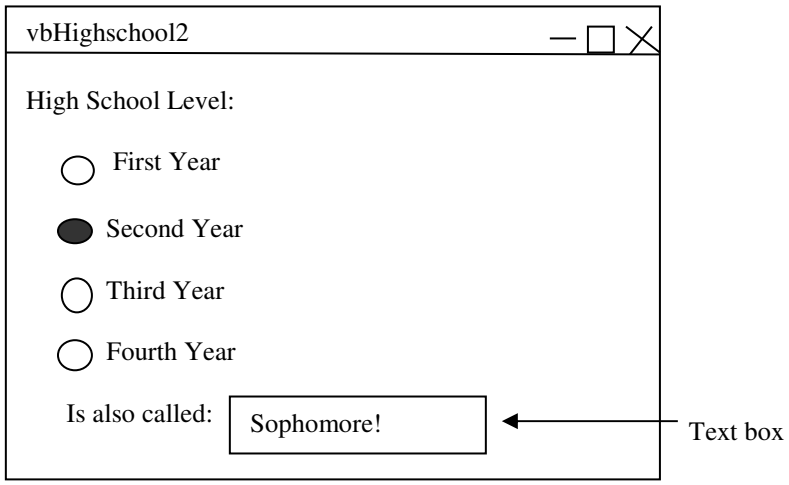


Figure 2.7 Radio buttons and Text box design 2

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpHighSchool2**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpHighSchool2;
```

```
[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **vbHighSchool2.**
7. Click, drag and drop a Label from the Toolbox to add it into the Form, and set its Text property to **High School Level:**.
8. Click, drag and drop four (4) *Radio buttons* from the Toolbox to add them into the Form, then set their respective Text property to **First Year**, **Second Year**, **Third Year**, and **Fourth Year**.
9. Finally, click, drag and drop a *Text box* from the Toolbox to add it into the Form.
10. Double-click the *First Year* radio button on the Form to display the radioButton1_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "Freshman!";
}
```

11. Double-click the *Second Year* radio button on the Form to display the radioButton2_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton2_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "Sophomore!";
}
```

12. Double-click the *Third Year* radio button on the Form to display the radioButton3_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton3_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "Junior!";
}
```

13. Double-click the *Fourth Year* radio button on the Form to display the radioButton4_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton4_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "Senior!";
}
```

14. *At the Menu bar again, select Project menu, then select the *cpHighSchool2 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

15. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

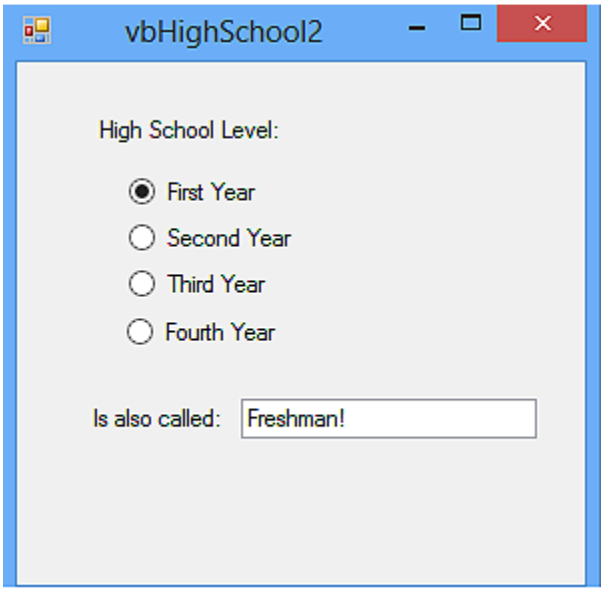


Figure 2.7a Radio buttons and Text box design 2 output

Explanation:

Now you will see the effect of using the radio button. As I have said, you can only select one option at a time. Once you click the next radio button, the bullet or the big black dot will transfer from the previous button to the currently clicked radio button.
We will notice here that the embedded code (within the method):

```
textBox1->Text = "Freshman!";
```

is simply changing the *Text* property of the object named *textBox1* to display "Freshman!" at the Text box control. The same thing happened to other radio buttons if we click them. The syntax for our example takes the format of *object->property,* where *textBox1* is the object (control) and *Text* is the property. We can use this syntax to change property settings for any form or control in response to events that occur while our

application system is running. The specific event here is the click event which is generated by clicking the radio buttons.
You will notice in our example, that there will always be a default selection in Radio button control. And that default selection is the Radio button 1 (in this example, it is the *First Year* radio button). This is how Radio button is specifically designed.

Hiding and Disabling Controls

We will tackle some program examples that demonstrate on an actual basis how to hide or disable a control. Actually, it is very easy. We just simply set the *Enabled* or *Visible* property of a particular control to *false* at the design time. Or alternatively, we can set it at the run-time, by issuing this syntax:

Objectname->Enabled = false;

Or

Objectname->Visible = false;

Note:
When we talk about *design time* it means that we are modifying our program as we design it on the Form where we can set the Control's property through the Property Window. In other words, as we design the controls on our Form while we invoke the Form Designer. Now, how about *run- time*? Run-time refers to the coding we do that sets the property of a particular Control right on our program. In other words, as we open (invoke) the Code Designer, we hard-coded the setting of properties.

Example 8:

Design and develop an application system that disables or enables a Button, and displays its feedback to the Textbox. When the user clicks the *Try to Click Me!* Button, the feedback that says "Yes! I was Enabled ! Thank you!" should be displayed at the Text box. Now when the user clicks the *Disable Button!* Radio button, the Button control should be disabled (grayed), and the feedback that says "You Disabled the Button!" is displayed at the Text box. Plus, the Try to Click Me! Button is unclickable. Now when the user clicks the *Enable Button!* Radio button, the Try to Click Me! Button should be restored to its enabled state, and the feedback that says "You Enabled the Button!" should be displayed at the Text box. Follow the given figure below in designing and developing the application system.

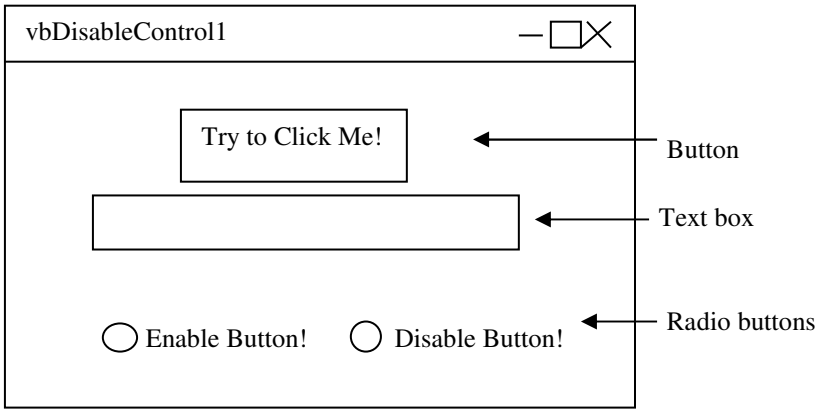


Figure 2.8 Disabling and Enabling Controls

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpDisableControl1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpDisableControl1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **vbDisableControl1**.
7. Click, drag and drop a Button from the Toolbox to add it into the Form, and set its Text property to **Try to Click Me!**.
8. Click, drag and drop a Text box from the Toolbox to add it into the Form.
9. Click, drag and drop two (2) Radio buttons from the Toolbox to add them into the Form, then set their respective Text property to **Enable Button!**, and **Disable Button**!.

10. Double-click the Try to Click Me! Button on the Form to display the button1_Click event method. Embed the following lines (in bold only) to the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "Yes! I was Enabled!, thank you!";
}
```

11. Double-click the *Enable Button!* radio button on the Form to display the radioButton1_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
   button1->Enabled = true;
   textBox1->Text = "You Enabled the Button!";
 }
```

12. Double-click the *Disable Button!* radio button on the Form to display the radioButton2_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton2_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
   button1->Enabled = false;
   textBox1->Text = "You Disabled  the Button!";
}
```

13. *At the Menu bar again, select Project menu, then select the *cpDisableControl1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

14. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

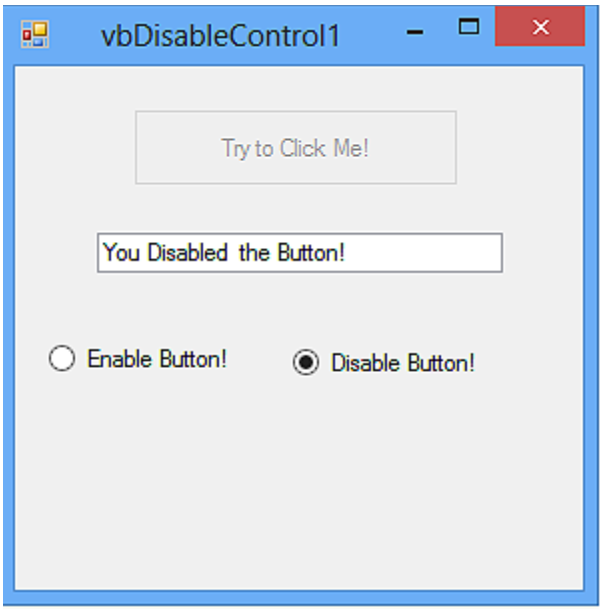


Figure 2.8a Disabling and Enabling Controls output

Explanation:

We can see here in the example above how easy it was to enable or disable a control. We just simply set a particular control's Enabled property to *true* (for enabling) and *false* (for disabling) it. Take for example, when the user clicks the Enable radio button (radioButton1), we enabled it through the following code:

```
private: System::Void radioButton1_CheckedChanged(System::Object^  sender, System::EventArgs^  e) {
   button1->Enabled = true;  ←
   textBox1->Text = "You Enabled the Button!";
 }
```

Now, when the user clicks the Disable radio button (radioButton2), we disabled it through the following code:

```
private: System::Void radioButton2_CheckedChanged(System::Object^  sender, System::EventArgs^  e) {
   button1->Enabled = false;  ←
   textBox1->Text = "You Disabled  the Button!";
}
```

Yes buddy, as simple as this code. It is very easy isn't it? Okay, let us now go to how to hide or show a control in the succedding example.

Example 9:

Design and develop an application system that shows or hides Radio button 1. Follow the given figure below in designing and developing the application system.

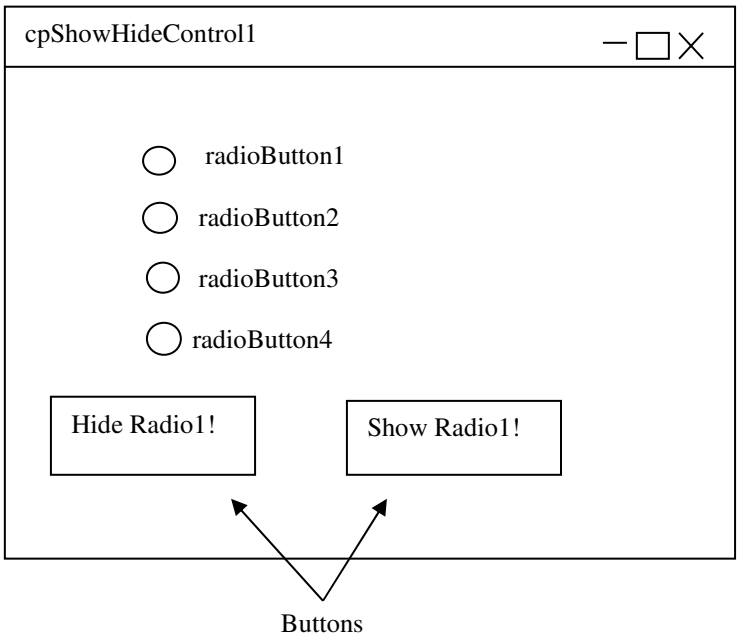


Figure 2.9a Showing and Hiding Controls

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpShowHideControl1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpShowHideControl1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
```

```
}
```

6. Set the Form's Text property to **vbShowHideControl1**.
7. Click, drag and drop four (4) Radio buttons from the Toolbox to add them into the Form.
8. Click, drag and drop two (2) Buttons from the Toolbox to add them to the Form, then set their respective Text property to **Hide Radio1!**, and **Show Radio1!**.
9. Double-click the *Hide Radio1*! Button on the Form to display the button1_Click event method. Embed the following lines (in bold only) to the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
  radioButton1->Visible = false;
}
```

10. Double-click the *Show Radio1*! Button on the Form to display the button2_Click event method. Embed the following lines (in bold only) to the method:

```
private: System::Void button2_Click(System::Object^  sender,
System::EventArgs^  e) {
  radioButton1->Visible = true;
}
```

11. *At the Menu bar again, select Project menu, then select the *cpShowHideControl1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

12. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

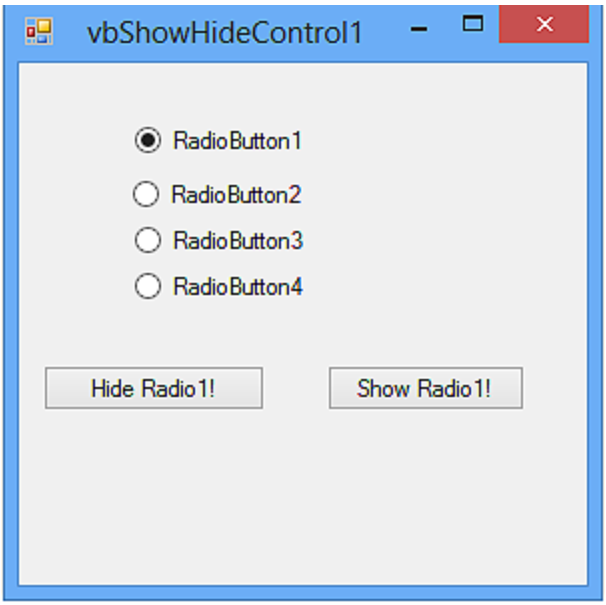


Figure 2.9a Showing and Hiding Controls output

Explanation:

We can see here in the example above how easy it was to hide or show a control. We just simply set a particular control's Visible property to *true* (for showing) and *false* (for hiding) it. Take for example, when the user clicks the Hide radio button (radioButton1), we hide it through the following code:

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
  radioButton1->Visible = false;
}
```

Now, when the user clicks the Show radio button (radioButton2), we showed it through the following code:

```
private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e) {
  radioButton1->Visible = true;
}
```

You could notice that the preceding codes are similar to the enabling and disabling a particular control which was demonstrated in our previous example.

Tweaking the High School Level Example a little bit

Now the question is, what if we want to design an empty list of radio buttons? This is in the case of our High School Level example where we saw that during the time we run our program, the default selection will always be the First Year radio button, thus its corresponding displayed message is "Freshman!". What we want is to be able to display all the list of radio buttons as empty (unselected), so that there is no message displayed at the Text box, by default. Well, this is possible, however, we will apply a trick on our program. Like any tricks, it works sometimes, and in most cases, it won't. To work around this programming task dilemma, we have to hide the first radio button which contain the default selection. In that way, the user was tricked to see the remaining list of radio buttons as empty (or unselected). Very clever, isn't it. But no, don't practice tricks, I warned you. It is but just a temporary success, and eventually a permanent failure. Remember that only truth lasts. And it lasts forever…like diamonds. Putting tricks on our programs will unexpectedly produce side effects, and these side-effects could be disastrous to our application system in the long run. Some technical people call this one "quick fix".

For the sake of showing you that we can turn around this programming task using tricks, let us do it now. But again, this is just for demonstration purposes only.

Example 10:

Design and develop a simple Text box and Radio buttons application that when the user clicks one of the four radio buttons, it will indicate in the Text box on which radio button the user had clicked. For example if radio button 2 (Second Year) was clicked by the user, it will display "Sophomore!" at the Text box. It will do the same with Radio button 1, Radio button 3, and Radio button 4. The high school level of First Year is "Freshman", for Second Year is "Sophomore", for Third Year is "Junior" while for Fourth Year is "Senior". Follow the given figure below in designing and developing the application system.

Now, your implementation will differ from cpHighSchool1, because here we have to display the list of Radio buttons as empty (unselected). Then we add two Buttons that can enable the Radio button 1 to reappear or to hide again.

Follow the given figure below in designing and developing the application system.

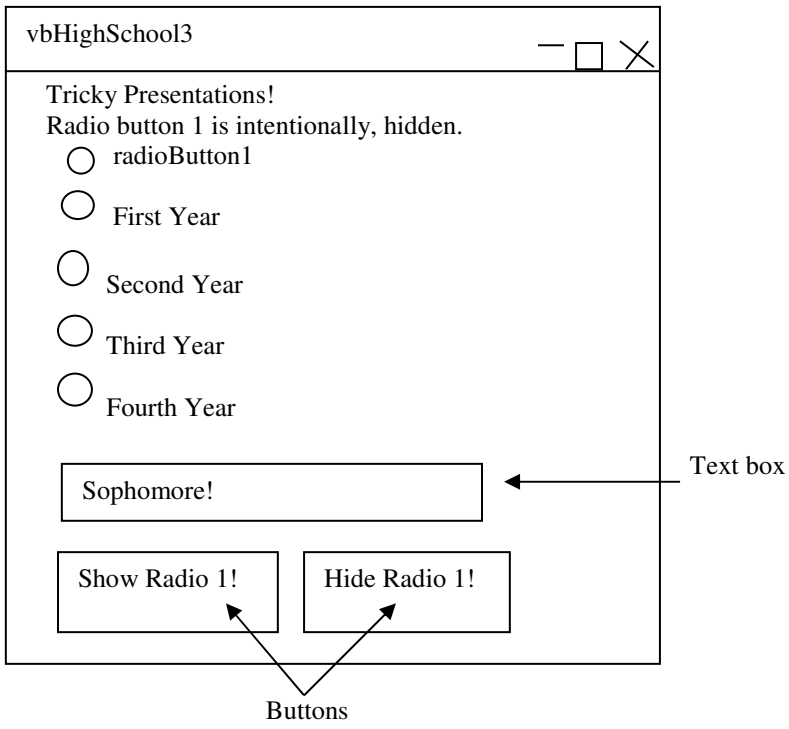


Figure 2.9 Hiding some controls to trick up the application

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpHighSchool3**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpHighSchool3;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **vbHighSchool3**.
7. Click, drag and drop five (5) Radio buttons from the Toolbox to add them into the Form, then set their respective Text property to **First Year** for the second radio button, **Second Year** for the third radio button, **Third Year** for the fourth radio button, and **Fourth Year** for the fifth radio button. We just leave Radio button 1 with its default caption.
8. Click, drag and drop a Text box from the Toolbox to add it into the Form.
9. Click, drag and drop two (2) Buttons from the Toolbox to add them into the Form, then set their respective Text property to **Show Radio1!**, and **Hide Radio1!**.
10. Double-click now the Form to display the Form1_Load event method. Embed the following lines (in bold only) to the method:

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^
e) {
  radioButton1->Visible = false;
  radioButton1->Checked = true;
}
```

11. Double-click the Radio button 1 on the Form to display the radioButton1_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton1_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "";
}
```

12. Double-click the *First Year* Radio button on the Form to display the radioButton2_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton2_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "Freshman!";
}
```

13. Double-click the *Second Year* Radio button on the Form to display the radioButton3_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton3_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "Sophomore!";
}
```

14. Double-click the *Third Year* Radio button on the Form to display the radioButton3_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton4_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "Junior!";
```

```
}
```

15. Double-click the *Fourth Year* Radio button on the Form to display the radioButton3_CheckedChanged event method. Embed the following lines (in bold only) to the method:

```
private: System::Void radioButton5_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
  textBox1->Text = "Senior!";
}
```

16. Double-click the *Show Radio1*! Button on the Form to display the button1_Click event method. Embed the following lines (in bold only) to the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
   radioButton1->Visible = true;
   textBox1->Text = "I just hide for a purpose!";
}
```

17. Double-click the *Hide Radio1*! Button on the Form to display the button2_Click event method. Embed the following lines (in bold only) to the method:

```
private: System::Void button2_Click(System::Object^  sender,
System::EventArgs^  e) {
  radioButton1->Visible = false;
  textBox1->Text = "";
}
```

18. *At the Menu bar again, select Project menu, then select the *cpHighSchool3 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

19. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

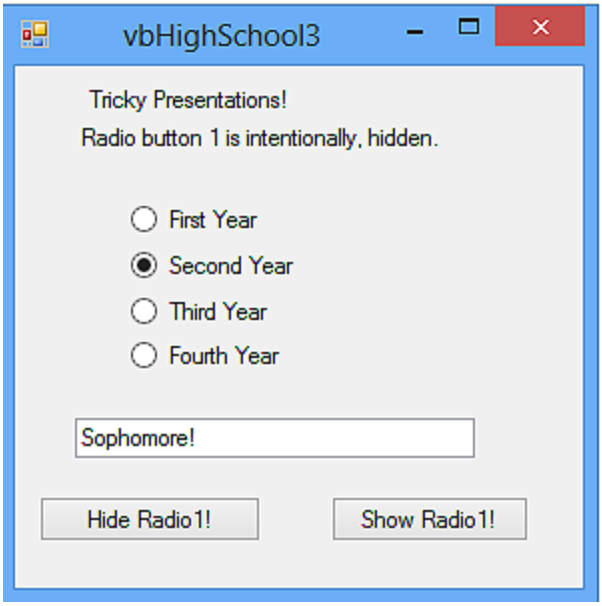


Figure 2.10a Hiding some controls to trick up the application output

Explanation:

The first time we run our application system, we need to hide right away the first radio button so that it won't appear. This is the trick that we are doing to our control, so that it appears as though there is no pre-selection happened in our radio buttons control. We already know that in radio buttons control, there is always a pre-selection of radio button. And it is always the first radio button to be selected by default.
To hide the first button from appearing and selected when we run our application program, we need to set the code this way:

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
  radioButton1->Visible = false;  ←
  radioButton1->Checked = true;  ←
}
```

In the code above, the Visible property of radio button 1 (radioButton1) is set to *false* to hide the control, while its Checked property is set to true so that the selection will always be at radio button 1. This is the main reason why the remaining radio buttons are empty (or no selection appears).
Once the Button 1 (Show radio 1) is clicked by the user, we have to empty the text box from any text message on it. We did it through the following code:

```
private: System::Void radioButton1_CheckedChanged(System::Object^  sender, System::EventArgs^  e) {
  textBox1->Text = " ";
}
```

The code above will ensure that when the user had already selected other radio buttons such as the radio button 2, radio button 3, and others, the displayed text message as the result of the following selections will be erased from the text box, once the user clicks the Button 1 (Show Radio 1).
When the user clicks the Button2 (Hide Radio 1), we have to hide the radio button 1 with the following code:

```
private: System::Void button2_Click(System::Object^  sender, System::EventArgs^  e) {
  radioButton1->Visible = false;
  textBox1->Text = "";
}
```

And at the same time, we have to empty the text box, so that the user is free again to select any of the radio buttons presented and will display the corresponding output text for the respected selection. For example, when the user clicks the First Year radio button, the output text:

```
  "Freshman!"
```

will be displayed at the text box. We did it by using the following code:

```
private: System::Void radioButton2_CheckedChanged(System::Object^  sender, System::EventArgs^  e) {
  textBox1->Text = "Freshman!";
}
```

This will also produce the same effect to other remaining selections of radio buttons.

Example 11:

Design and develop an application system that when the user points the mouse-pointer at any control on the Form, it will display the tooltip of what kind of control it is pointing to. Follow the given figure below in designing and developing the application system.

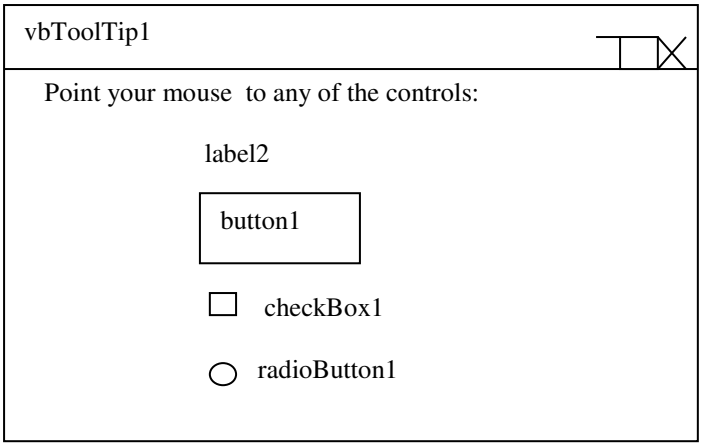


Figure 2.11 Simple ToolTip program

Note:
When the user points his or her mouse on the Form, the tooltip should be, “This is a Form!”, now if the user points the mouse on the Label1, the tooltip should be, “This is a Label!”. Now apply the same respective tooltip to other controls, too.

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpToolTip1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpToolTip1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form’s Text property to **vbToolTip1**.
7. Click, drag and drop a Label, a Button, a Check Box, and a Radio Button from the Toolbox to add them into the form. Then, click, drag and drop five ToolTips into the Form. The *toolTip1*, *toolTip2*, *toolTip3*, *toolTip4*, and *toolTip5* controls should appear at the bottom of the Form. Why the tooltip is up to 5? Because, we will include the Form as one of the controls to be assigned with a *toolTip* control, okay?
8. Double-click now the Form to display the Form1_Load event method. Embed the following lines (in bold only) to the method:

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^
e) {
   toolTip1->SetToolTip(this, "This is a Form!");
   toolTip2->SetToolTip(label2, "This is a Label!");
```

```
    toolTip3->SetToolTip(button1, "This is a Button!");
    toolTip4->SetToolTip(checkBox1, "This is a Check Box!");
    toolTip5->SetToolTip(radioButton1, "This is a Radio Button!");
}
```

9. *At the Menu bar again, select Project menu, then select the *cpToolTip1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

10. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

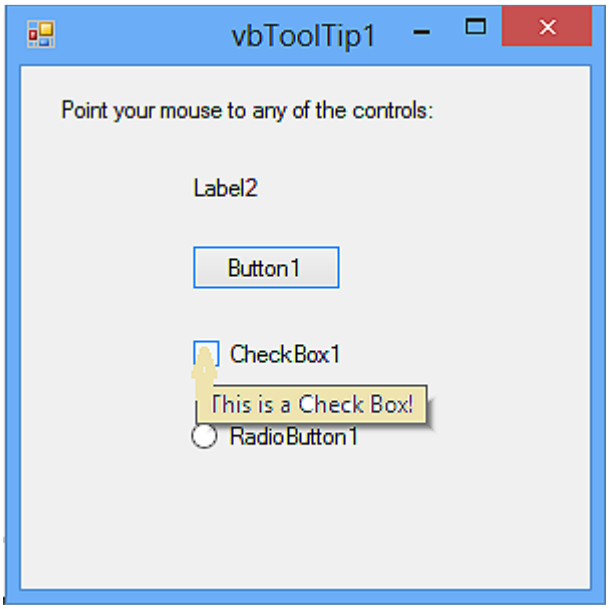


Figure 2.11a Simple ToolTip program output

Explanation:

See, it is easy to program a simple ToolTip application, isn't it? Here, in this code below:

```
private: System::Void Form1_Load(System::Object^  sender, System::EventArgs^  e) {
  toolTip1->SetToolTip(this, "This is a Form!");  ←
  toolTip2->SetToolTip(label2, "This is a Label!");
  toolTip3->SetToolTip(button1, "This is a Button!");
  toolTip4->SetToolTip(checkBox1, "This is a Check Box!");
```

```
  toolTip5->SetToolTip(radioButton1, "This is a Radio Button!");
}
```

the *this* of the *ToolTip1.SetToolTip* object and method parameter means the *Form* control. The other tooltips parameters are self-explanatory. Agree?

Example 12:

Design and develop an application system that will echo the message typed at the Text box to the Label with bigger font size. Follow the given figure below in designing and developing the application system.

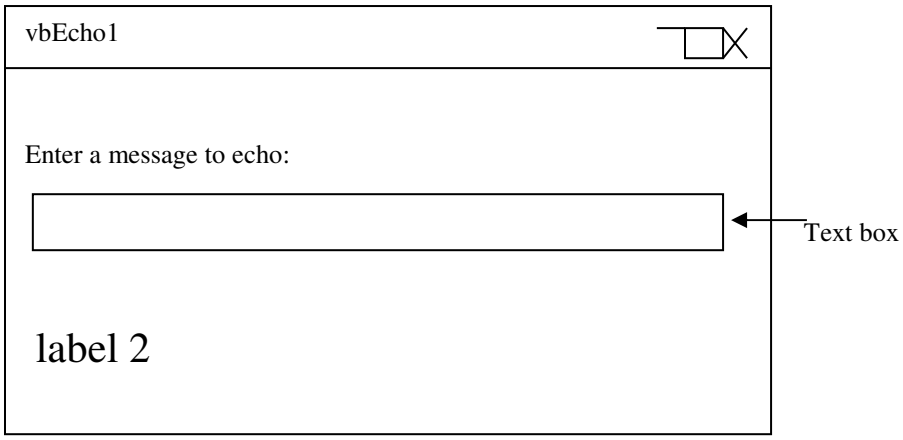


Figure 2.12 A Simple Echo program

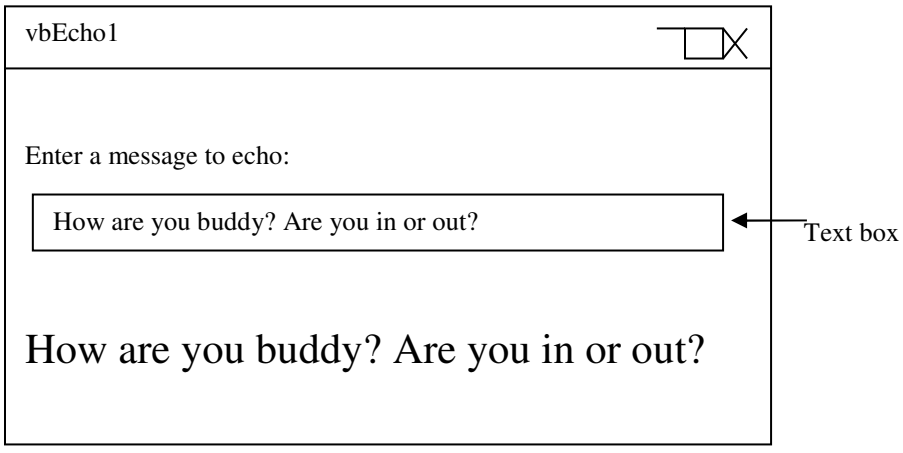


Figure 2.12a A Simple Echo program

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpEcho1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpEcho1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **vbEcho1**.
7. Click, drag and drop a Label from the Toolbox, then set its Text property as **Enter a message to echo:**.
8. Click, drag and drop a *text box* from the Toolbox to add it into the form. Position it properly and adjust its width appropriately.
9. Next, click, drag and drop another Label from the Toolbox, then set its Font size property to **16**.
10. This time, double-click the Text box to display the TextBox1_TextChanged event method. Embed the following lines (in bold only) into the method:

```
private: System::Void textBox1_TextChanged(System::Object^  sender,
System::EventArgs^  e) {
  label2->Text = textBox1->Text;
}
```

11. *At the Menu bar again, select Project menu, then select the *cpEcho1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

12. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Sample Output:

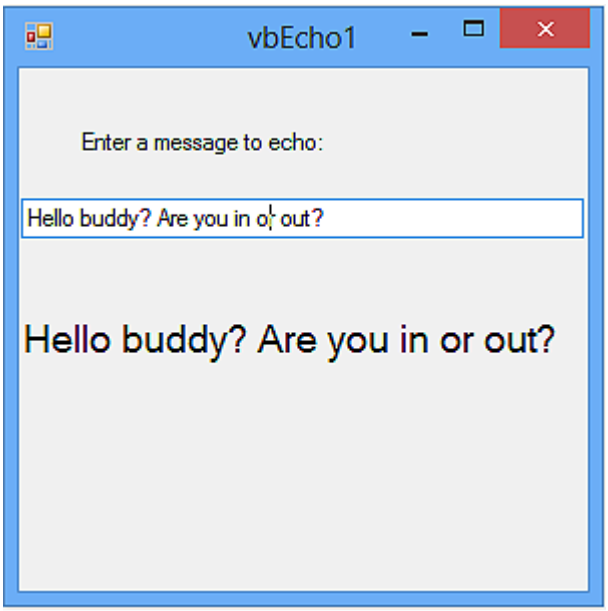


Figure 2.12b A Simple Echo program output

Explanation:

Echoing a message typed at the text box control is very simple and easy. We just simply used the textBox1 control method and its *Text* property respectively, to be stored at *Label2.Text* control. With this, we can echo the message we typed at text box into the label control. We can accomplish this task by the following code:

```
private: System::Void textBox1_TextChanged(System::Object^  sender, System::EventArgs^  e) {
  label2->Text = textBox1->Text;
}
```

That's all for now, buddy.

Event-Driven Programming

In the preceding examples above, we learned how event-driven application system works. An event is an action recognized by a control or form. An event-driven application system executes Visual C# program in response to an event. For example, if the user clicks a button (a check box, or radio button), the button's Click event method is executed. When we want a control such as a button or check box to respond to an event, we write a program in the event

method for that event. As though we are writing a small program behind it, which is really the case.

Object and Classes Defined

Object is simply a combination of program and data that can be treated as a unit. Like for example, a Form or a Button is an object. Inside (or behind) a button is a program and data which we can use to manipulate itself. Its data such as the caption (text property), size, or color can be modified to suit our needs. Now if we want to click it to perform a particular task, we need its Click program to accomplish the task. These program and data are encapsulated within an object, thus eliminates conflict to other code in other object or module. Classes are the blueprint or template of an object. It is just like saying, "If a floor is an object, then its floor-plan is the class. Without the
floor-plan, the floor can be impossibly built reliably and accurately. The same goes to an object, without the class, it is impossible to create it successfully and elegantly.
The object in Visual C# is defined by a class. A class defines the characteristics of an object - for instance, its size and shape. The class is used to create an object. The controls which are also called objects in the Toolbox represent classes. When we design a control (object) onto the form such as clicking, dragging, and dropping a button or text box, we are really creating a copy or an instance of the control class. In this case the control class is the button class or text box class. All objects (controls) are created as identical copies of their respective classes.

To Comment or Not to Comment
(That is the Question!)

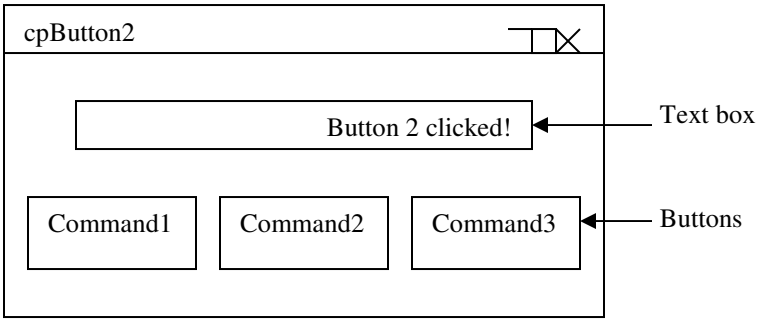
Making a comment or putting one or two in your code, makes your program more readable and comprehensible to other programmers or developers. In other words, your program is a programmer-friendly if it contains comment or comments most especially to some part of your program which are hard to understand or decipher. Well, you will learn later on in your programming career that a comment can help you to remember easily what your code is doing once you have not read it for a week or month. This is but just a simple thoughtfulness that can give you and your fellow programmers an easy way to understand programs without resorting to serious and deep code analysis. You will also be surprised later on in your life as a programmer, that you too can hardly understand your own program if it contains no comments on it. Try not doing it (not putting a comment in your program), and you will be sorry after all. Take my word seriously, buddy; I really mean it.
As you read through the examples in this book, you will often come across with the comment symbol (//) of Visual C++ language (other programming languages have other comment symbols to use). This symbol tells the Visual C++ system-compiler to ignore the words that follow it. These words are remarks placed in the code for the benefit of the other programmer or developer who might examine or use your code later.

A Friendly Reminder for the Instructors

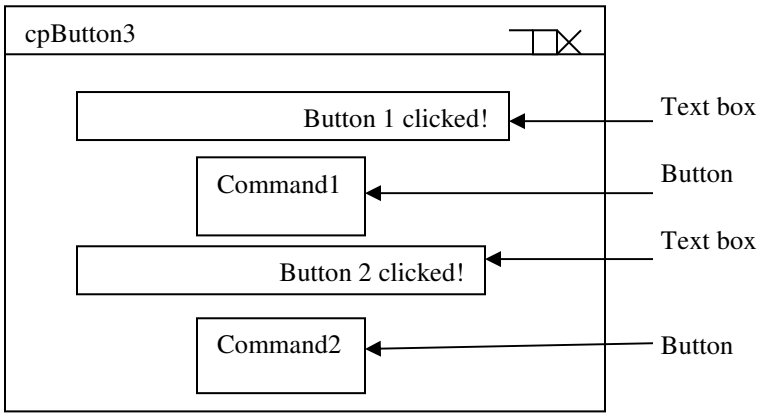
Since programming endeavor is a creative art as well as an analytical science, I would like to suggest that all the laboratory activity tests must be an open notes or open books, or even an open Internet research. In this way, the students won't spend too much time memorizing the commands and functions that they will use in solving the problems presented. The memory capacity is so precious enough to be wasted only for the things that need not to be memorized such as commands and functions or methods. I myself cannot memorize commands and functions or methods with accuracy most of the time. It is my belief that if the students are given enough resources such as notes, books, and Internet connection, they can spend more time in analyzing and solving the problems than cramming their minds with command syntax memorization. In the actual practice of our profession, no boss in his right mind who would instruct his or her programmers and developers to design and develop a real-world application system without the aid of notes, books, manuals or Internet connection. So why not emulate that kind of treatment and consideration to our future programmers or developers, the way the real boss do in real-world?

LAB ACTIVITY
TEST 7

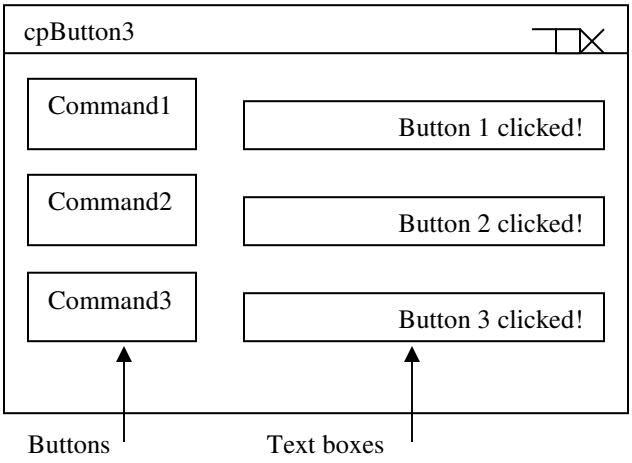
1. Design and develop an application system that when the user clicks the Button 1, the message "Button 1 clicked!" will be displayed at the Text box. When the user clicks the Button 2, the message "Button 2 clicked!" will be displayed at the Text box, and "Button 3 clicked!" will be displayed when the user clicks the Button 3. Follow the given figure below in designing and developing the application system.



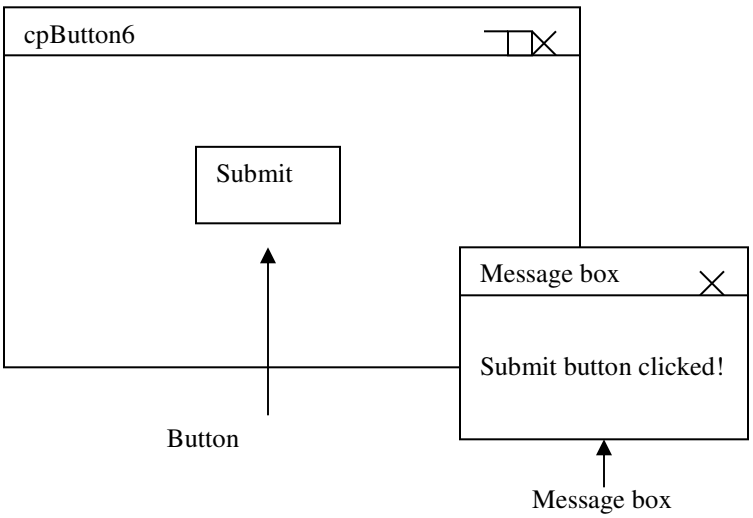
2. Design and develop an application system that when the user clicks the Button 1, the message "Button 1 clicked!" will be displayed at the Text box 1. When the user clicks the Button 2, the message "Button 2 clicked!" will be displayed at the Text box 2. Follow the given figure below in designing and developing the application system.



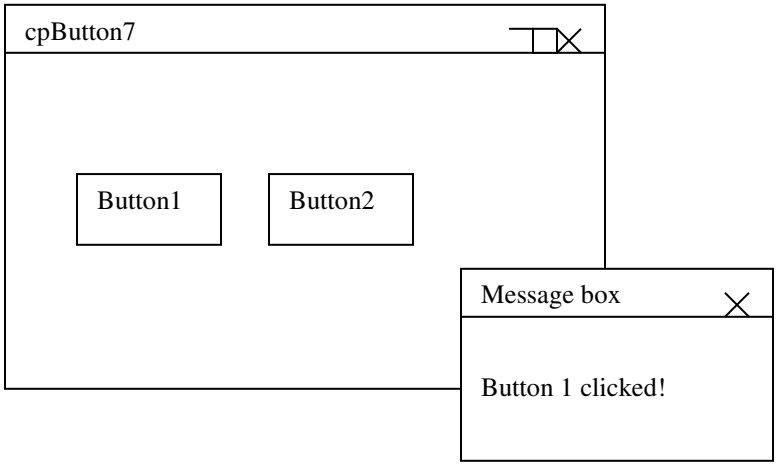
3. Design and develop an application system that when the user clicks the Button 1, the message "Button 1 clicked!" will be displayed at the Text box 1. When the user clicks the Button 2, the message "Button 2 clicked!" will be displayed at the Text box 2, and "Button 3 clicked!" will be displayed at Text box 3 when the user clicks the Button 3. Follow the given figure below in designing and developing the application system.



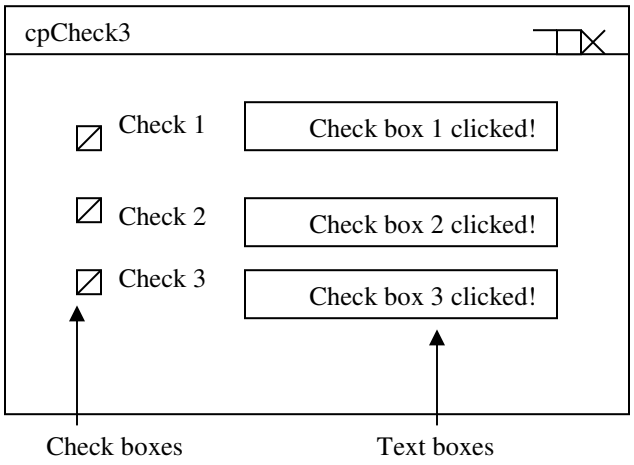
4. Design and develop an application system that when the user clicks the Button with the caption "Submit", the message "Submit button 1 clicked!" will be displayed at the Message box . Follow the given figure below in designing and developing the application system.



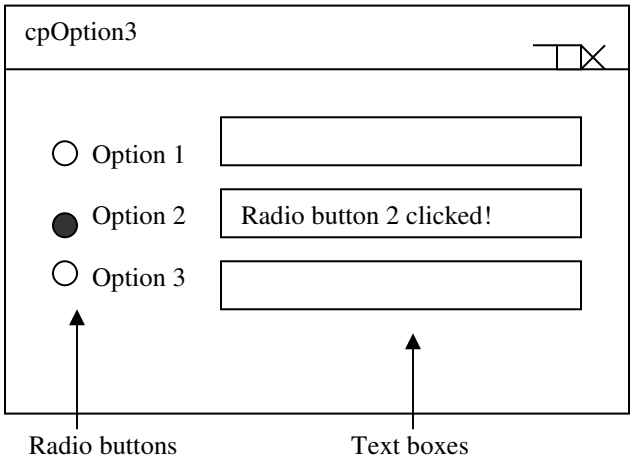
5.Design and develop an application system that when the user clicks the Button 1, the message "Button 1 clicked!" will be displayed at the Message box. When the user clicks the Button 2, the message "Button 2 clicked!" will be displayed at the Message box. Follow the given figure below in designing and developing the application system.



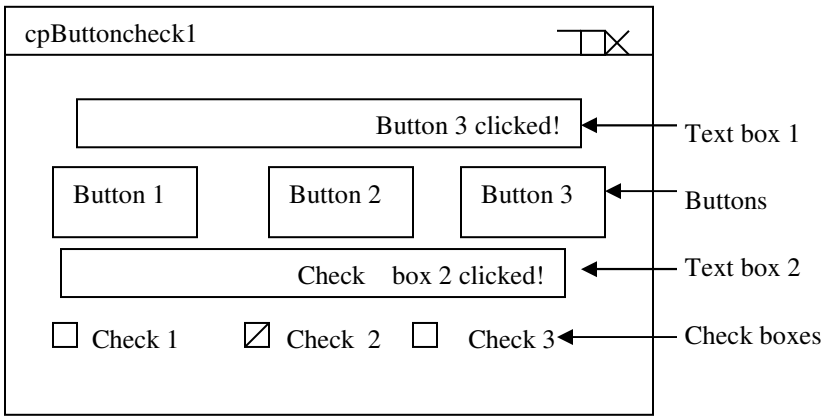
6.Design and develop an application system that when the user clicks the Check box 1, the message “Check box 1 clicked!” will be displayed at the Text box 1. When the user clicks the Check box 2, the message “Check box 2 clicked!” will be displayed at the Text box 2 and when the user clicks the Check box 3, the message “Check box 3 clicked!” will be displayed at the Text box 3. Follow the given figure below in designing and developing the application system.



7.Design and develop an application system that when the user clicks the Radio button 2, the message “Radio button 2 clicked!” will be displayed at the text box 2. When the user clicks the Radio button 1, the message “Radio button 1 clicked!” will be displayed at the text box 1 and when the user clicks the Radio button 3, the message “Radio button 3 clicked!” will be displayed at the text box. Follow the given figure below in designing and developing the application system.

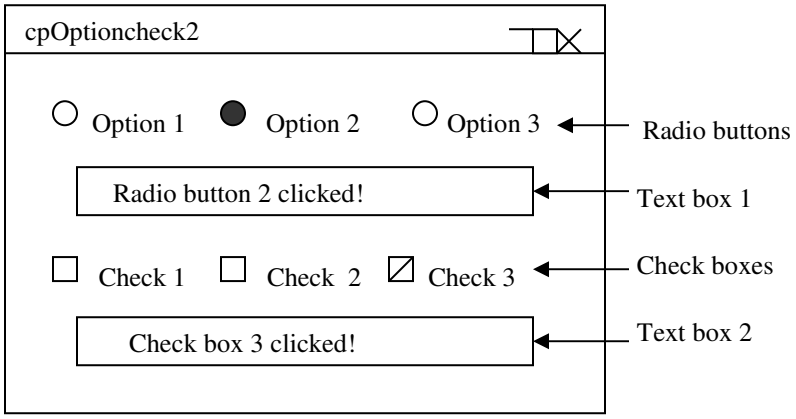


8.Design and develop an application system that when the user clicks the Button 1, the message "Button 1 clicked!" will be displayed at the text box 1. When the user clicks the Button 2, the message "Button 2 clicked!" will be displayed at the text box 1 and when the user clicks the Button 3, the message "Button 3 clicked!" will be displayed at the text box 1.
When the user clicks the Check box 1, the message "Check box 1 clicked!" will be displayed at the text box 2. When the user clicks the Check box 2, the message "Check box 2 clicked!" will be displayed at text box 2, and when the user clicks the Check box 3, the message "Check box 3 clicked!" will be displayed at the text box 2. Follow the given figure below in designing and developing the application system.

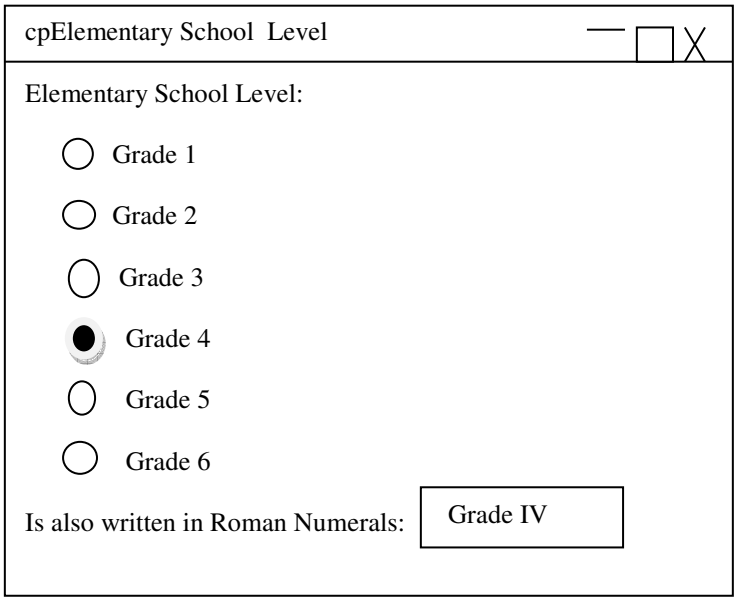


9. Design and develop an application system that when the user clicks the Option 1, the message "Radio button 1 clicked!" will be displayed at the text box 1. When the user clicks the Radio button 2, the message "Radio button 2 clicked!" will be displayed at the text box 1 and when the user clicks the Radio button 3, the message "Radio button 3 clicked!" will be displayed at the text box 1.

When the user clicks the Check box 2, the message "Check box 2 clicked!" is displayed at the text box 2. When the user clicks the Check box 1, the message "Check box 1 clicked!" will be displayed at the text box 2 and when the user clicks the Check box 3, the message "Check box 3 clicked!" will be displayed at the Text box 2. Follow the given figure below in designing and developing the application system.



10.Design and develop an application system that when the user clicks the Radio button 1 (Grade 1), the message "Grade I" will be displayed at the text box 1. When the user clicks the Radio button 2 (Grade 2), the message "Grade II" will be displayed at the text box 2 and when the user clicks the Radio button 3 (Grade 3), the message "Grade III" will be displayed at the text box 3 and when the user clicks the Radio button 4 (Grade 4), the message "Grade IV" will be displayed at the text box and so on and so forth. In other words, the Elementary school level is just written into its equivalent Roman numeral form. Follow the given figure below in designing and developing the application system.



Chapter 10

Using Controls with Input/Output Functions

The Legend of ToInt??( ) and ToString( ) Methods

In calculating and displaying data in our Form, we need to know how they can be manipulated using the two most important methods: Numeric Value method (ToInt??( ) or ToDouble( )) and String method (ToString( )). These two common methods simply convert our data from numeric to string data or vice versa. The ToInt??( ) method converts the displayed data into numeric, while the ToString( ) method converts the displayed data into string data.
The ToInt??( ) method is applied when we want to calculate the numeric data displayed in text boxes. Now if we want to display the computed data into the text box, we need to convert it into a string data using the ToString( ) method.
We need to apply the ToInt??( ) method so that the numbers that are being entered in the text box can be converted to numeric data for computation purposes in our equation or formula, otherwise these numbers are treated as string which cannot be calculated or processed in the equation. We need also to convert the numeric data into its equivalent string data representation using the ToString( ) method for the proper display of the numbers into the text box control. Without doing so, it will result to an undesired output.
Okay, let us have our example about this one. The best example of this application is the simple calculator. In our simple calculator, we will design a program where a user can input two numbers then our program will summed them up. Let us start with a program that will add two numbers.

Tip:
A *method* is simply a *function* that is a member of a class.

Arithmetic Operators

+	Addition
_	Subtraction
*	Multiplication
/	Division
%	Modulus

The above arithmetic operators are handy for our next examples on this chapter. So remember their symbol so that we can easily understand the succeeding examples and successfully solve the upcoming Lab Activity Test. Are you ready now? Yes, you are!

Note:
We use the ToDouble( ) function if we want to get the input value that is in decimal format such as 54.60 or as big as 100000.4567. The ToInt16( ) is only limited to integer value which is a whole number such as 10 or 32000 (but not more than 32,000 something; in case of big whole number - you have to use the ToInt32( ) function instead).

Example 1:

Design and develop a simple application program that calculates the sum of two input numbers. Follow the given figure below in designing and developing the application program:

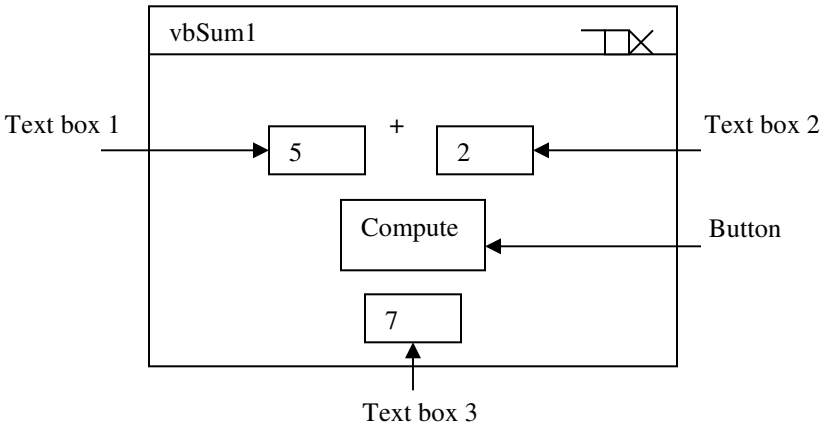


Figure 3.1 Sum of two input numbers

Note:
When the user enters two numbers in two text boxes, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at the third box (Text box 3).

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpSum1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpSum1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **vSum1**.
7. Click, drag and drop a Button from the Toolbox to add it into the Form, and set its Text property to **Compute**.
8. Click, drag and drop three Text boxes from the Toolbox to add them into the Form.
9. Double-click now the Compute button on the Form to display the button1_Click event method. Embed the following lines (in bold only) to the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
   Int16 num1,num2,sum;
   sum = 0;
   num1= Convert::ToInt16(textBox1->Text);
   num2= Convert::ToInt16(textBox2->Text);
   sum = num1 + num2;
   textBox3->Text = sum.ToString();
}
```

10. *At the Menu bar again, select Project menu, then select the *cpSum1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

11. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Try to enter any numbers at the two text boxes. Then click the Compute button to see the result. Is the result okay? I hope so.

Sample Output:

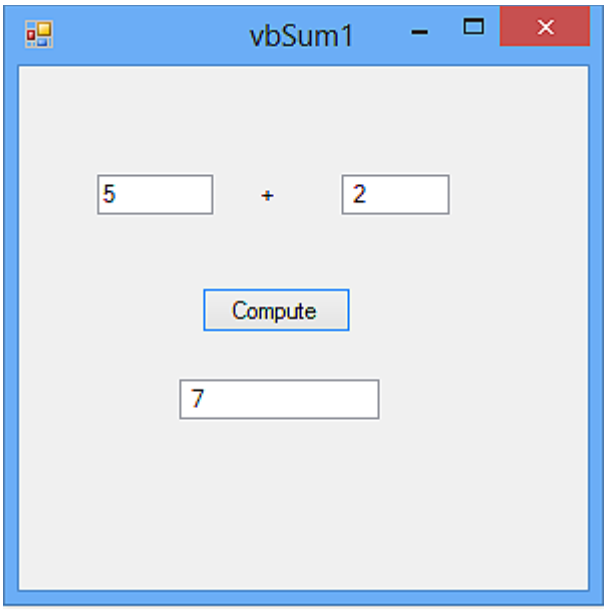


Figure 3.1a Sum of two input numbers output

Explanation:

First, we declare all the variables we used in our program. For the sake of simplicity, we declared all variables as integer data types. This means that our variables are capable only of handling smaller whole numbers that will not exceed the 32,000 (the comma symbol (,) must not be included in entering numeric values, otherwise an error will surely occur) something values. We declare our variables this way:

```
Int16 num1,num2,sum;
  sum = 0;  ←
```

As well as initializing the variable with a value of zero (0), in the case of variable *sum*.
Any character (whether alphabetic, numeric, or special symbol), when it is entered at the text box is considered and formatted as a text. That is why we need to convert an input data into its equivalent numerical data by using the *Convert.ToInt16* syntax. We did it this way:

```
  num1= Convert::ToInt16(textBox1->Text);
  num2= Convert::ToInt16(textBox2->Text);
```

After the conversion, the variable no1 and no2 will now contain the converted numerical data. This is the time that we can now compute their values. So we have this simple adding equation:

```
  sum = num1 + num2;
```

Since we cannot display a numeric data directly to the text box control, that is why we have to convert the numeric data into its text (or string) equivalent using the ToString( ) syntax, such as what we have here in the following code:

```
textBox3->Text = sum.ToString( );
```

So, that is our discussion to this example.

Note:

The ***ToInt16*** means To Integer 16 bit. So the text data entered will be converted into integer which is 16-bit wide. Some ToInt??( ) versions are ToInt32( ) and ToInt64( ) functions which are used mainly in very large numbers of integer data such as in the range of 100,000 (hundreds to millions) to 10,0000000 (the comma symbol is not included in inputting the numeric data, otherwise, an error will occur) . The
ToInt32( ) function is a 32-bit wide integer, while the ToInt64( ) function is a 64-bit wide integer.
The ***ToString( )*** function is our syntax to use in converting a numeric data into its equivalent string data so that we can display the data to the text box control (or other controls that are used for displaying data).

Example 2:

Design and develop a simple application system that computes the area of a circle. Use the formula: $A = \pi r^2$, where Pi (π) is approximately equivalent to 3.1416. Follow the given figure below in designing and developing the application system.

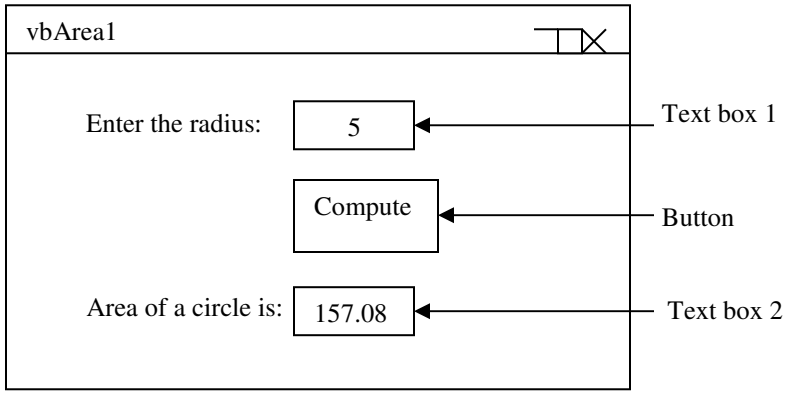


Figure 3.2 Compute the area of a circle

Note:

When the user enters the value of the radius in text box 1, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at text box 2.

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpArea1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpArea1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **vbArea1**.
7. Click, drag and drop a Button from the Toolbox to add it into the Form, and set its Text property to **Compute**.
8. Click, drag and drop two Text boxes from the Toolbox to add them into the Form. Then click two Labels to label these two text boxes properly, based on the given figure above.
9. Double-click now the Compute button on the Form to display the button1_Click event method. Embed the following lines (in bold only) to the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
  Double Area,Pi;
  Int16 r;
  Area = 0;
  Pi = 3.1416;
  r = Convert::ToInt16(textBox1->Text);
  Area = 2*Pi*r*r;
  textBox2->Text = Area.ToString();
}
```

10. *At the Menu bar again, select Project menu, then select the *cpArea1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

11. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Now try to enter any value at the text box of the radius. Then click the Compute button to see the result. Is the result okay?

Sample Output:

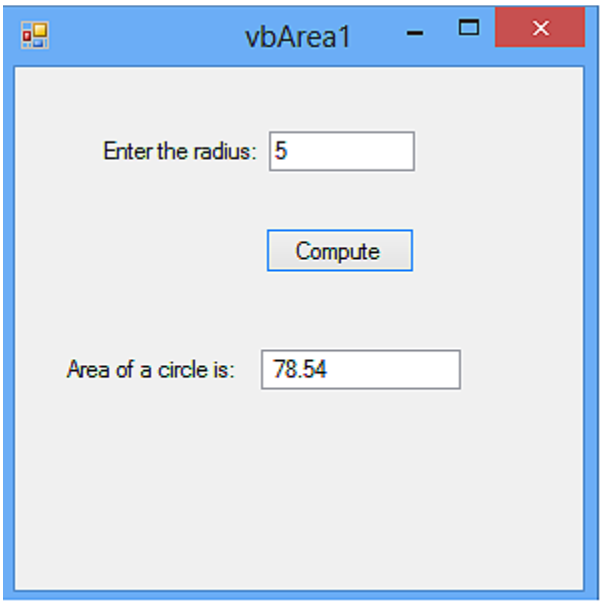


Figure 3.2a Compute the area of a circle

Explanation:

First, we declared all the variables we used in our program. We did that with the following code:

```
Double Area,Pi;
Int16 r;
Area = 0;
Pi = 3.1416;
```

In this example, we declared a constant which name is **Pi** . This is one of the ways on how we can declare a constant value in Visual C++. We could noticed also that the variable *Area* is declared as *double* data type. Variable *Area* will eventually contain a computed value with decimal point upon processing the equation. Since a variable would hold a value with a decimal point, it must be declared as *double*. This is the main reason why we declared variable *Area* as *double* data type. This correct variable data type declaration also ensures the accuracy of the calculation of our equation.

We initialize the variable *Area* with a value of zero(0) to ensure that the resulting calculation in our equation is correct. Forgetting to do so, will result to undesirable output.

In this code:

```
                .
                .
                .

r = Convert::ToInt16(textBox1->Text);
Area = 2*Pi*r*r;
textBox2->Text = Area.ToString( );
```

You will notice also that we convert the input value at text box 1 into its equivalent numeric value using the ToInt16( ) function to properly calculate it in our equation. We can see the conversion syntax on the following code:

```
r = Convert::ToInt16(textBox1->Text);
```

When we input a number in our text box, it is treated as a string data. Therefore, converting it to numeric data is needed, otherwise it cannot be properly calculated in the equation, which in turn results to undesirable output or wrong calculation.
After the conversion, we put the converted numeric integer value at variable *r* so that we can use this converted numeric value in the succedding equation which is:

```
Area = 2*Pi*r*r;
```

This is the proper way on how to calculate the area of a circle in programming syntax form. Then finally, we convert the computed value that is stored in *Area* variable into a string value before we can display it at the text box 2 successfully, with this code:

```
textBox2->Text = Area.ToString( );
```

This is needed to display the value properly. Forgetting to do so, will result to undesirable output.

Example 3:

Design and develop a simple application system that computes the average of three input quizzes. Then display the result. Follow the given figure below in designing and developing the application system.

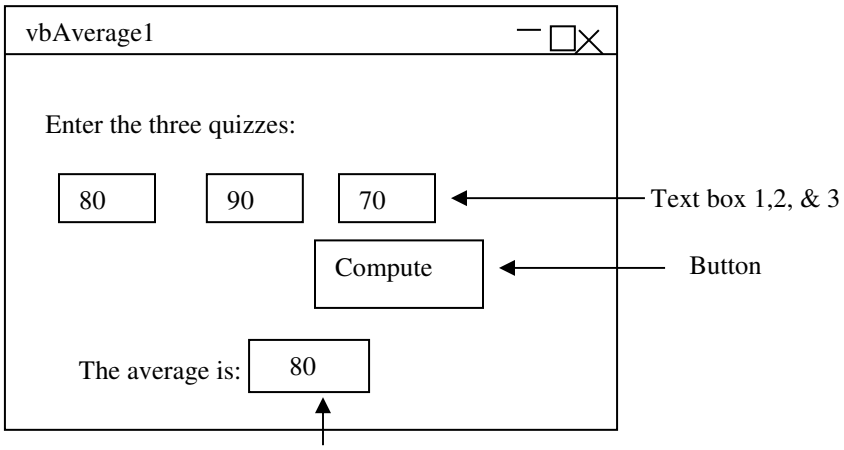


Figure 3.3. Compute the average quiz

Note:
When the user enters the score of three quizzes in text box 1, 2 and 3, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at text box 4.

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpAverage1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpAverage1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **cpAverage1**.
7. Click, drag and drop a Button from the Toolbox to add it into the Form, and set its Text property to **Compute**.
8. Click, drag and drop four Text boxes from the Toolbox to add them into the Form. Then click two Labels to label the text boxes properly, based on the given specification above.
9. Double-click now the Compute button on the Form to display the button1_Click event method. Embed the following lines (in bold only) to the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
  Int16 Quiz1, Quiz2, Quiz3;
  float Ave;
  Ave = 0;
  Quiz1 = Convert::ToInt16(textBox1->Text);
  Quiz2 = Convert::ToInt16(textBox2->Text);
  Quiz3 = Convert::ToInt16(textBox3->Text);
  Ave = (Quiz1 + Quiz2 + Quiz3)/3;
   textBox4->Text = Ave.ToString();
}
```

10. *At the Menu bar again, select Project menu, then select the *cpAverage1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

11. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Now try to enter three quizzes score at the text boxes. Then click the Compute button to see the result. Is the result okay?

Sample Output:

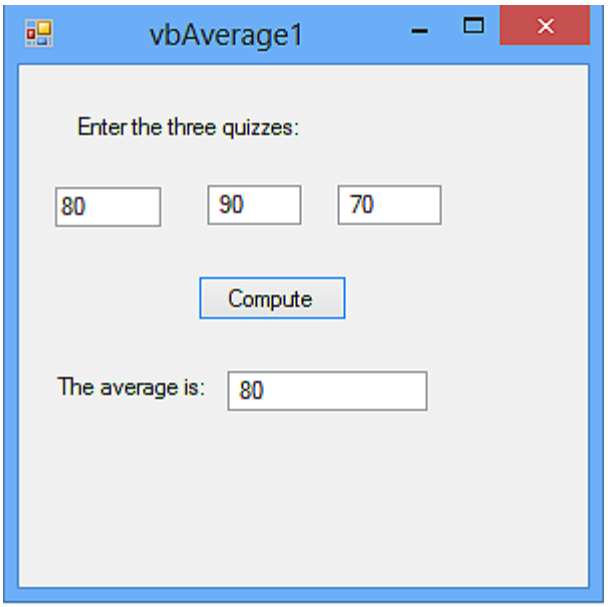


Figure 3.3a Compute the average quiz output

Explanation:

First, we declared all the variables we used in our program. We did that with the following code:

```
Int16 Quiz1, Quiz2, Quiz3;
float Ave;
Ave = 0;
.
.
.
```

Our variable *Ave* (for average) was declared as *float* data type because it is used for a division operation. The syntax rule in Visual C++ and other programming languages is that any variable which hold the resulting computation of the equation which involves a division operation must be declared as *float* (or *double* for larger floating-point numeric data) data type. The main reason for this is that even if we divide two whole (integer) numbers, there is still a big possibility that it yields a value of type *float* (a number with a decimal point). Take for example, if we divide N1 / N2 (D=N1/N2) where N1=7 and N2=3, this division operation will yield a value of 2.3334 which is a *float* data type. Since variable D will hold the resulting computed value with decimal point, therefore variable D must be declared as *float* data type. This is also the same case with the variable *Ave* that we are using here in our program.

This correct variable data type declaration also ensures the accuracy of the calculation of our equation, because the numerical data after the decimal point will also be displayed in our output. So if your output data does not contain any number after the decimal point, even if you believe that it must, then, the problem probably lies with the way you declared the variable with its corresponding data type. You might be declaring that particular variable with an integer data type. Changing it to *float* data type will solve the problem.

Tip:
If you encounter a problem where your output numeric data does not produce a number with a decimal point, then probably you declared the variable that holds that output numeric data as *integer*. Changing its declaration to *float* or *double* data type would solve the problem. Try it!

In this code:

```
    .
    .
    .
   Quiz1 = Convert::ToInt16(textBox1->Text);
  Quiz2 = Convert::ToInt16(textBox2->Text);
  Quiz3 = Convert::ToInt16(textBox3->Text);
  Ave = (Quiz1 + Quiz2 + Quiz3)/3;
  textBox4->Text = Ave.ToString( );
```

You will notice also that we convert the input value at text box 1, text box 2, and text box 3 into its equivalent numeric value using the ToInt16( ) function to properly calculate them in our equation. We can see the conversion syntax on the following code:

```
Quiz1 = Convert::ToInt16(textBox1->Text);
Quiz2 = Convert::ToInt16(textBox2->Text);
Quiz3 = Convert::ToInt16(textBox3->Text);
```

When we input a number in our text box, it is treated as a string data. Therefore, converting it to numeric data is needed, otherwise it cannot be properly calculated in the equation, which in turn results to undesirable output or wrong calculation.
After the conversion, we put the converted numeric integer values at the variables Quiz1, Quiz2, and Quiz3 so that we can use these converted numeric values in the succedding equation which is:

```
Ave = (Quiz1 + Quiz2 + Quiz3)/3;
```

Then finally, we convert the computed value that is stored in *Ave* variable into a string value before we can display it at the text box 4 successfully, with this code:

```
textBox4->Text = Ave.ToString( );
```

This is needed to display the value properly. Forgetting to do so, will result to undesirable output.

Example 4:

Design and develop a simple application system that converts the input value of Celsius into its equivalent Fahrenheit degree. Use the formula: F= (9/5) * C + 32. Follow the given figure below in designing and developing the application system.

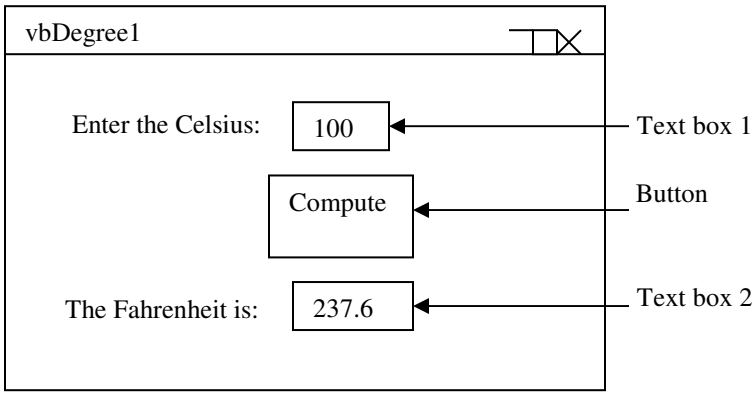


Figure 3.4 Convert Celsius to Fahrenheit degree

Note:
When the user enters the value of the Celsius in text box 1, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at text box 2.

Solution:

1. Start Microsoft Visual Studio Express 2012 for Windows Desktop by clicking the *Start* tab at your Windows 8 operating system. Now Start a New Project. Select the Visual C++ template.
2. *Under the Visual C++ template, select the CLR option. Finally, select the CLR Empty Project new project template.
3. *Name the new application system **cpDegree1**. Now click the OK button.
4. *At the Menu Bar, select the Project menu, then select the Add New Item submenu. Now, select the UI option and click the Add button. Can you now see the MyForm control? If yes, then you are successful!
5. *At the Solution Explorer windows, double-click the *MyForm.cpp*. Add the following template:

```
using namespace cpDegree1;

[STAThreadAttribute]
void main(array<System::String ^> ^args)
{
  Application::EnableVisualStyles();
  Application::SetCompatibleTextRenderingDefault(false);
  Application::Run(gcnew MyForm());
  return ;
}
```

6. Set the Form's Text property to **vbDegree1**.
7. Click, drag and drop a Button from the Toolbox to add it into the Form, and set its Text property to **Compute**.
8. Click, drag and drop two Text boxes from the Toolbox to add them into the Form. Then click two Labels to label the two text boxes properly, based on the given specification above.
9. Double-click now the Compute button on the Form to display the button1_Click event method. Embed the following lines (in bold only) to the method:

```
private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
  Int16 Celsius;
  double Fahrenheit;
  Fahrenheit = 0;
  Celsius = Convert::ToInt16(textBox1->Text);
  Fahrenheit = (9.0 / 5.0) * (Celsius + 32);
  textBox2->Text = Fahrenheit.ToString();
}
```

10. *At the Menu bar again, select Project menu, then select the *cpDegree1 Properties*. Select the Configuration Properties option, then select the Linker option. This time select the System option, then set the SubSystem to **Windows(/SUBSYSTEM:WINDOWS).** Still at the Linker option, select the Advanced option then set the EntryPoint to **main**. After that, click the Apply button. Finally, click the OK button.

11. Build and execute the application system by clicking the Local Windows Debugger icon (or press F5) at the Toolbar which color is a green arrow.

Now try to enter any value at the text box of the Celsius. Then click the Compute button to see the result.

Sample Output:

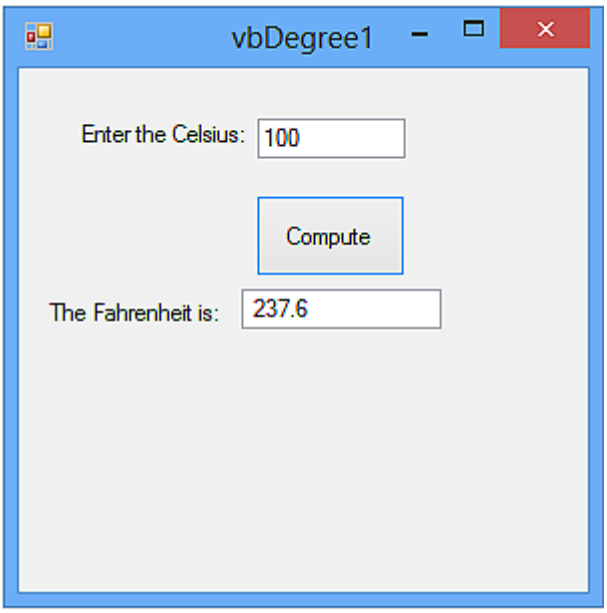


Figure 3.4a Convert Celsius to Fahrenheit degree output

Explanation:

First, we declared all the variables we used in our program. We did that with the following code:

```
 Int16 Celsius;
 double Fahrenheit;
 Fahrenheit = 0;
.
.
.
```

We could notice that the variable *Fahrenheit* is declared as *double* data type. Variable *Fahrenheit* will eventually contain a computed value with decimal point upon processing the equation. Since a variable would hold a value with a decimal point, it must be declared as double data type. This correct variable data type declaration also ensures the accuracy of the calculation of our equation.

In this code:

```
.
.
.
Celsius = Convert::ToInt16(textBox1->Text);
Fahrenheit = (9.0 / 5.0) * (Celsius + 32);
textBox2->Text = Fahrenheit.ToString( );
```

we write the given equation Fahrenheit = (9/5) * (txtCelsius+32) to convert the input Celsius into its equivalent Fahrenheit degree. Then we convert the computed value that is stored in Fahrenheit variable into a string value before we can display it at the text box 2 successfully with this code:

---

textBox2->Text = Fahrenheit.ToString( );

This is needed to display the value properly. Forgetting to do so, will result to undesirable output. You will notice also that we convert the input value at text box 1 into its equivalent numeric value using the ToInt16( ) function to properly calculate it in our equation. We can see the conversion syntax on the following code:

Celsius = Convert::ToInt16(textBox1->Text);

When we input a number in our text box, it is treated as a string data. Therefore, converting it to numeric data is needed, otherwise it cannot be properly calculated in the equation, which in turn results to undesirable output or wrong calculation.

Variable and Constant Declarations

As what we have learned in our previous examples, variables and constants played an important role in our program. Understanding how to use them and how they work is critical to our programming endeavor and crucial to our career as systems programmer or solutions developer. So let us start to learn about them now.

The declaration part in our program consists usually with the list of variables. In some instances, we have declared also the constant names within the program module. **Variables** are memory location (address) which we can assign a specific name. For example, we can name our variables as *Sum*, *Num1*, *Area*, or *Ave*. We can assign or store a value or data into a variable. To declare a variable is to tell the program about it in advance. A variable name must begin with a letter which does not contain an embedded period and must be unique within the same scope (in a form or method).

Constants look like a variable, however unlike the variable, the value or data it holds remain fixed while the application system is running or executing. One of the purposes why we use constants in our program is to make coding and debugging easier. By putting the constant value at the declaration part in our program (usually placed at the top of the module), we can easily modify or debug it once it causes some errors. Here are some examples of the constant declarations:

const float conPi = 3.1416

const float conInterestRate = 0.03

const int conHoursWork=8

Basic Data Types of Visual C++

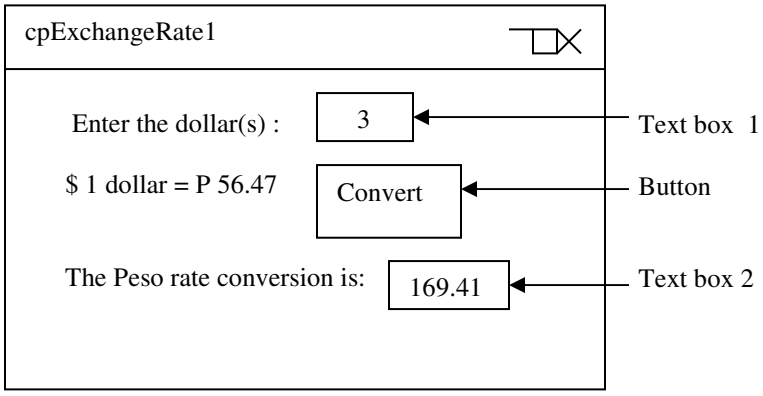
Data Type	Purpose
int	An integer data type is a whole number. Examples: 9 143 32000 +123 -4567
long	A bigger integer - means a wider range than an *int*. Examples: 10,123,456,789 1,000,000,000,321
float	A number with fractional part or decimal point. Examples: 3.1416 2.54 +193.20 -1,000.003
double	A bigger float – means a wider range than float. Examples: 1143441000.5121 1000000000.151314
String	A sequence of one or more characters that are enclosed within double quotation marks. Examples: “a” “Z” “*” “Ma.” “Bianca” “I love you very much, Bianca.”
char	Usually a single value enclosed with single quote. Example: ‘a’ ‘1’ ‘*’ ‘Z’ ‘0’ ‘!’

bool	A *boolean* data type holds *true* or *false* values.
	Examples:
	chkOnionsChili->Checked = true;
	chkTomatoPineapple->Checked = false;

byte	A small integer data type with a range of 0 to 255 Example: 6 254 100 +20 -99

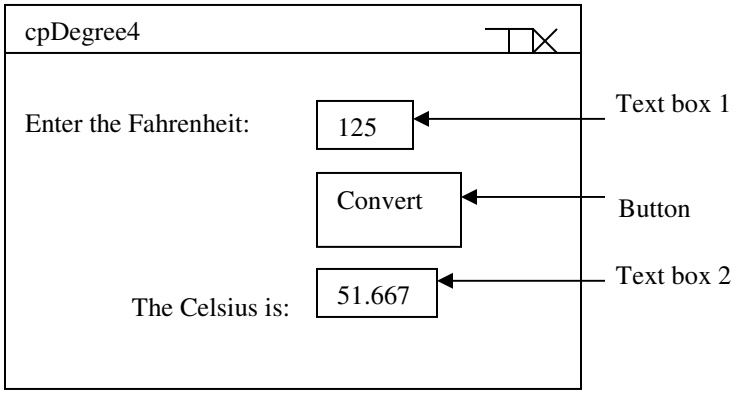
LAB ACTIVITY
TEST 8

1. Design and develop a simple application system that converts the input dollar(s) into its equivalent Peso rate. Follow the given figure below in designing and developing the application system.



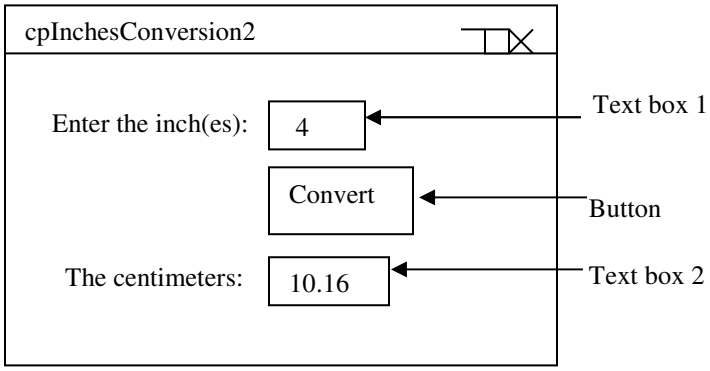
Note:
When the user enters the value of the dollar(s) in text box 1, the user should click the Button (with a Convert caption) before the resulting computed value will be displayed at text box 2.

2. Design and develop a simple application system that converts the input Fahrenheit into its equivalent Celsius degree. Use the formula: C = (5/9) * F - 32. Follow the given figure below in designing and developing the application system.



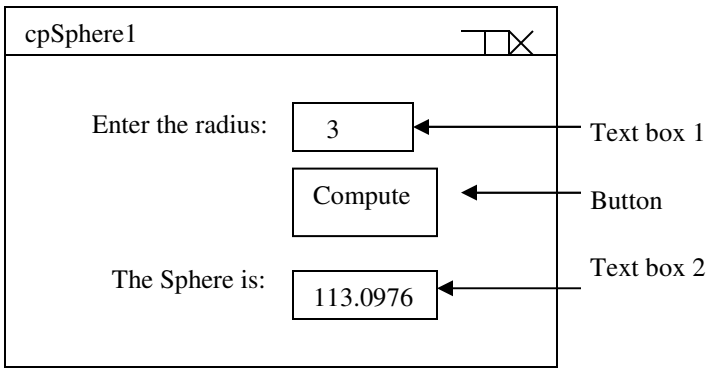
Note:
When the user enters the value of the Fahrenheit in text box 1, the user should click the Button (with a Convert caption) before the resulting computed value will be displayed at text box 2.

3. Design and develop a simple application system that converts the input inch(es) into its equivalent centimeters. One inch is equivalent to 2.54 cms. Follow the given figure below in designing and developing the application system.



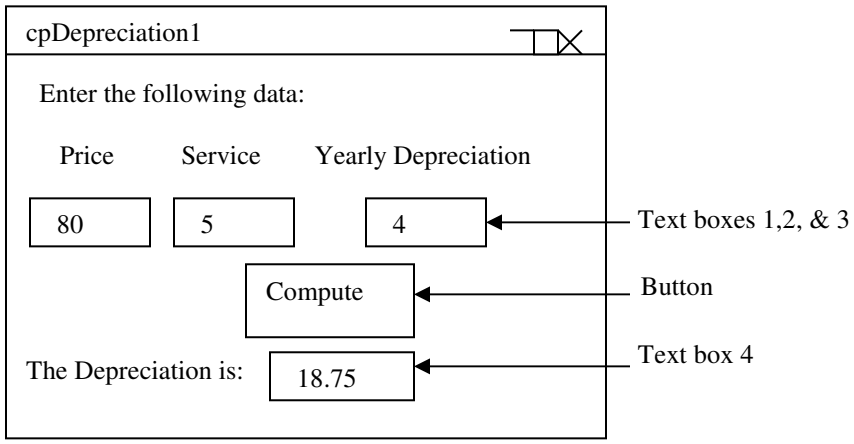
Note:
When the user enters the value of the inch(es) in text box 1, the user should click the Button (with a Convert caption) before the resulting computed value will be displayed at text box 2.

4. Design and develop a simple application system that computes the volume of a sphere. Use the formula: $V = 4/3\ \pi r^3$, where Pi (π) is approximately equivalent to 3.1416. Follow the given figure below in designing and developing the application system.



Note:
When the user enters the value of the radius in text box 1, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at text box 2.

5.Design and develop a simple application system that computes the Depreciation cost of the item (D). Takes as input the purchase Price of an item (P), its expected number of years of Service (S), and Yearly depreciation for the item (Y).
Use the formula: D=P-S /Y. Follow the given figure below in designing and developing the application system.



Note:
After the user enters the purchase Price of an item (P), its expected number of years of Service (S), and Yearly depreciation for the item (Y) at text boxes 1,2 & 3, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at text box 4.

6.Design and develop a simple application system that determines the most economical quantity to be stocked for each product that a manufacturing company has in its inventory. This quantity called *economic order quantity* (EOQ) is calculated as follows:

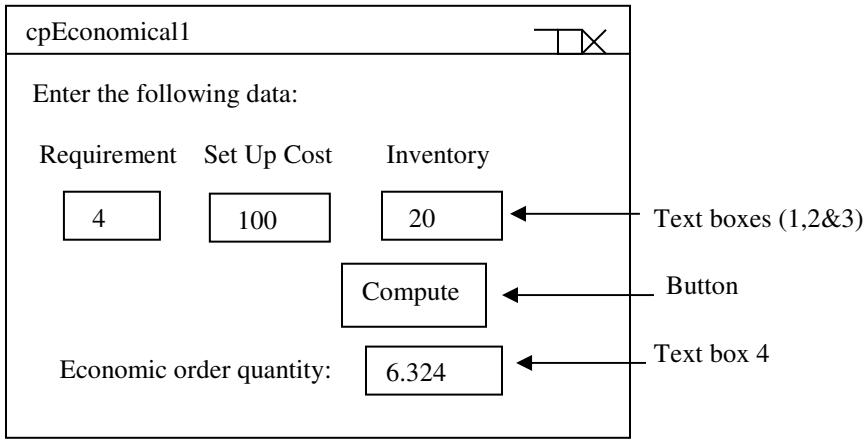
EOQ = sqrt(2RS/I)

where:
R = total yearly production Requirement
S = Set Up Cost per order
I = Inventory Carrying Cost per unit

Hint:
Use the `Convert::ToString(System::Math::Sqrt(intNum));` mathematical function to get the square root value.

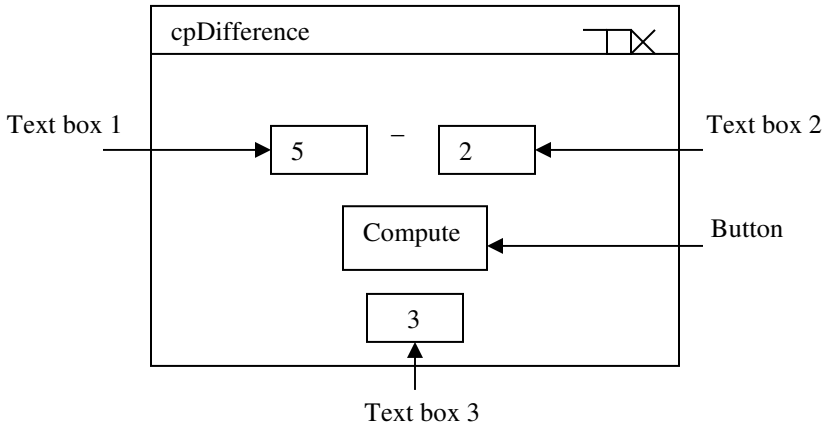
where *intNum* is the number to square root.

Follow the given figure below in designing and developing the application system.



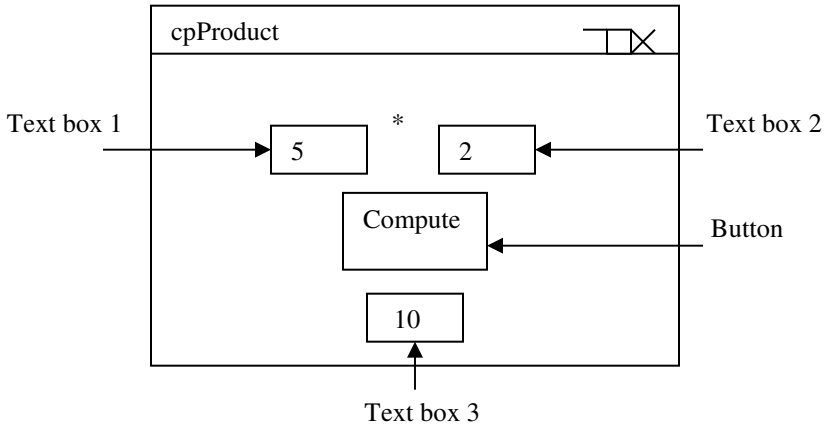
Note:
After the user enters the total year production Requirement (R), its Set up cost per order (S), and Inventory carrying cost per unit (I) at text boxes 1,2 & 3 respectively, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at text box 4.

7.Design and develop a simple application system that calculates the difference of two input numbers. Follow the given figure below in designing and developing the application system.



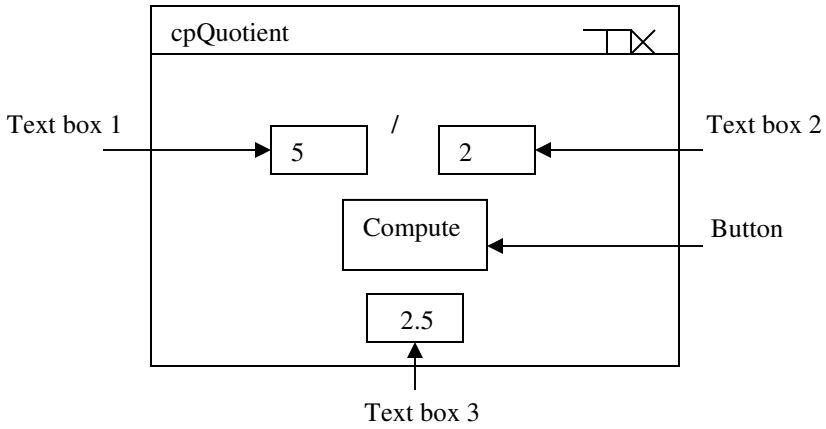
Note:
When the user enters two numbers in two text boxes, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at the third box (Text box 3).

8.Design and develop a simple application system that calculates the product of two input numbers. Follow the given figure below in designing and developing the application system:



Note:
When the user enters two numbers in two text boxes, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at the third box (Text box 3).

9.Design and develop a simple application system that calculates the quotient of two input numbers. Follow the given figure below in designing and developing the application system:



Note:
When the user enters two numbers in two text boxes, the user should click the Button (with a Compute caption) before the resulting computed value will be displayed at the third box (Text box 3).

APPENDIX A
The Basic Mathematical Functions of Turbo C

The mathematical functions have three categories. They are the following:

- Trigonometric functions
- Hyperbolic functions
- Exponential and logarithmic functions

1. *acos( )* – this math function returns the arc-cosine value of **n.**

Syntax: *acos*(n)

2. *asin( )* – this math function returns the arc-sine value of **n.**

Syntax: *asin*(n)

3. *atan( )* – this math function returns the arc-tangent value of **n**.

Syntax: *atan(*n)

4. *atan2( )* – this math function returns the arc-tangent of **y/x**.

Syntax: *atan2(y/x)*

5. *cos*( ) – this math function returns the value of cosine.

Syntax: *cos(* n *)*

6. *cosh( )*- this math function returns the hyperbolic cosine value of **n.**

Syntax: *cosh(* n *)*

7. *exp( )* – this math function returns the value of the natural logarithm **e** raised to the argument power.

Syntax: *exp(* n *)*

8. *fabs( )* – this math function returns the absolute value of **n.**

Syntax: *fabs(* n *)*

9. *fmod( )* – this math function returns the remainder of **x/y**.

Syntax: *fmod(* x/y *)*

10. *log( )* – this math function returns the natural logarithm of **n.**

Syntax: *log(* n *)*

11. *log10( )* – this math function returns the base10 logarithm for **n.**

Syntax: *log10(* n *)*

12. *pow( )* – this math function returns the base (b) raised to the exp (e) .

Syntax: *pow*(b,e)

13. *pow10*( ) – this math function returns 10 raised to the power **n.**

Syntax: *pow10(*n *)*

14. *sin ( )* – this math function returns the sine value of **n.**

Syntax: *sin*(n)

15. *sinh ( )* – this math function returns the hyperbolic sine of **n.**

Syntax: *sinh(*n *)*

16. *sqrt*( ) – this math function returns the square root value of **n.**

Syntax: *sqrt*(n)

17. *tan( )* – this math function returns the tangent value of **n.**

Syntax: *tan*(n)

18. *tanh*( ) – this math function returns the hyperbolic tangent of **n.**

Syntax: *tanh(* n *)*

Example 1:

This program computes and prints the value of **n** for the mathematical functions *asin( ), acos( ), atan( )*, *cos*( ), *sin*( ), *acos*( ), *tanh*( ), *cosh*( ), *sinh*( ), in one-tenth increments, of the numbers –1 up to 1.

```
#include <stdio.h>
#include <math.h>
main()
{
double n;
clrscr();
n=-1.0;
do {
   printf("arc sine %1f of %1f \n ",n,asin(n));
   n += 0.1;
   delay(9000);
} while (n<=1.0);
printf("\n");
n=-1.0;
do {
   printf("arc cosine of %1f is %f \n",n,acos(n));
   n +=0.1;
   delay(9000);
} while (n<=1.0);
printf("\n");
n= -1.0;
do {
   printf("arc tangent of %1f is %f \n",n,atan(n));
   n +=0.1;
   delay(9000);
} while (n<=1.0);
printf("\n");
n= -1.0;

do {
   printf("cosine of %1f is %f \n",n,cos(n));
   n +=0.1;
   delay(9000);
} while (n<=1.0);
printf("\n");
n= -1.0;
```

```
do {
   printf("hyperbolic cosine of %1f is %f \n",n,cosh(n));
   n +=0.1;
   delay(9000);
} while (n<=1.0);
printf("\n");
n= -1.0;

do {
   printf("sine of %1f is %f \n",n,sin(n));
   n +=0.1;
   delay(9000);
} while (n<=1.0);
printf("\n");
n= -1.0;
do {
   printf("hyperbolic sine of %1f is %f \n",n,sinh(n));
   n +=0.1;
   delay(9000);
} while (n<=1.0);
printf("\n");
n= -1.0;
do {
   printf("tangent of %1f is %f \n",n,tan(n));
   n +=0.1;
   delay(9000);
} while (n<=1.0);
printf("\nThe End!");
getch();
}
```

Example 2:

This program computes and prints the value of **n** for the mathematical functions *atan2( ), exp( ),log( ),log10( ), fabs( ), fmod( ), pow( ), pow10( )* and *sqrt( ).*

```
#include <stdio.h>
#include <math.h>
main()
{
double n,x,y,b,e;
clrscr();
y=-1.0;
do {
   printf("atan2 of %1.1f is %1f\n",y,atan2(y,1.0));
   y +=0.1;
   delay(9000);
}  while (y<=1.0);
y=1.0;
do {
   printf("logarithm of %1.1f is %1f\n",y,log(y));
   y +=0.1;
   delay(9000);
}  while (y<=11.0);
y=1.0;

do {
   printf("base10 log of %1.1f is %1f\n",y,log(y));
   y +=0.1;
   delay(9000);
}  while (y<=11.0);
printf("natural logarithm e to the first:%1f\n",exp(1.0));
x=9.0;
y=2.0;
printf("modulus or remainder of x/y is:%1f\n",fmod(x,y));
b=9.0;
e=2.0;
printf("\nbase %f raised to %f is : %1f\n",b,e,pow(b,e));
printf("\nbase10 % raised to %f is: %1f\n",e,pow10(e));
printf("\n square root of %f is: %f\n",b,sqrt(b));
printf("\n The End!");
getch();
}
```

ANSWER KEY

Solutions to the Odd Numbers of Chapter 2 (Simple Input/Output Statements):

Answer for Number 1:

Create a program to compute the volume of a sphere. Use the formula: $V= (4/3)\pi r^3$ where Pi (π) is approximately 3.1416. The r^3 stands for the radius. Display the result.

```
#include <stdio.h>
#define Pi 3.1416
main()
{
float V,r;
clrscr();
printf("\n Enter radius:");
scanf("%f",&r);
V=(4.0/3.0) * Pi * (r*r*r);
printf("\n Volume: %8.4f",V);
getch();
}
```

Answer for Number 3:

Write a program that converts the input dollar to its peso equivalent. Then display the peso equivalent. One dollar is approximately 51.60 pesos..

```
#include <stdio.h>
main()
{
  int dollar;
  float peso;
  clrscr();
  printf("\n Enter dollars:");
  scanf("%d",&dollar);
  /* assume that one dollar is equavalent to 51.60 pesos*/
  peso = 51.60 * dollar;
  printf("\n Peso rate conversion: %8.2f",peso);
  getch();
}
```

Answer for Number 5:

Write a program that exchanges the value of two variables: x and y. The output must be: the value of variable y will become the value of variable x, and the value of variable x will become the value of variable y.

```
#include <stdio.h>
main()
{
   int x,y,t;
   clrscr();
   printf("\n Enter the value of x: ");
   scanf("%d",&x);
   printf("\n Enter the value of y: ");
   scanf("%d",&y);
   t=x;
   x=y;
   y=t;
   printf("\n The new value of x: %d",x);
   printf("\n The new value of y: %d",y);
   getch();
}
```

Answer for Number 7:

You can solve the worded-problem number 5 with the use of three variables declaration. Now, try to solve it with only two variables declaration. Come up with an equations that exchange the value of x and y. The hint is: 3 equations that involves with plus and minus operations.

```
#include <stdio.h>
main()
{
   int x,y;
   clrscr();
   printf("\n Enter the value of x:");
   scanf("%d",&x);
   printf("\n Enter the value of y:");
   scanf("%d",&y);
   x = x + y;
   y = x - y;
   x = x - y;
   printf("\n The new value of x: %d",x);
   printf("\n The new value of y: %d", y);
   getch();
}
```

Answer for Number 9:

Determine the most economical quantity to be stocked for each product that a manufacturing company has in its inventory. This quantity, called *economic order quantity (EOQ)* is calculated as follows:

$$EOQ = \sqrt{\frac{2RS}{I}}$$

where:

R= total yearly production requirement
S= set up cost per order
I=inventory carrying cost per unit

```
#include <stdio.h>
#include <math.h>

main()
{
   float eoq,r,s,i;
   clrscr();
   printf("\n Enter total yearly production requirement:");
   scanf("%f",&r);
   printf("\n Enter set up cost per order:");
   scanf("%f",&s);
   printf("\n Enter inventory carrying cost per unit:");
   scanf("%f",&i);
   eoq = sqrt((2*r*s)/i);
   printf("\n The most economical quantity: %8.2f",eoq);
   getch();
}
```

Solutions to the Odd Numbers of Chapter 3 (Conditional Statements):

Answers for Number 1:

Write a program that determines if the input letter is a VOWEL or CONSONANT. The vowels are : A E I O U while the consonants are the remaining letters of the alphabet. Your program must be able to handle a capital or small input letter.

a.) First Solution using: **if/else** conditional statement

```
#include <stdio.h>
#include <ctype.h>
```

```
#include <string.h>
main()
{
  char let,clet;
  clrscr();
  printf("\n Enter a letter:");
  let=getche();
  clet = toupper(let);
  if(clet=='A' || clet=='E' || clet=='I' || clet=='O' ||clet =='U')
     printf("\n It's a VOWEL!");
  else
     printf("\n It's a CONSONANT!");
  getch();
}
```

b.) Second Solution: using **switch/case** conditional statements

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <string.h>
main()
{
 char let, clet;
 clrscr();
 printf("\n Enter a letter:");
 let=getche();
 clet = toupper(let);
 switch(clet) {
 case 'A': case 'E': case 'I': case 'O': case 'U':
    printf("\n It's a VOWEL"); break;
 default: printf("\n It's a CONSONANT");
}
getch();
}
```

Answer for Number 3:

Write a program that examines the value of variable called temp. Then display the following messages, depending on the value assigned to temp.

Temperature	Message
Less than 0	ICE
Between 0 and 100	WATER
Exceeds 100	STEAM

```
#include <stdio.h>
main()
{
int temp;
 clrscr();
 printf("\n Enter temperature:");
 scanf("%d",&temp);
 if (temp<0)
    printf("\n ICE");
 else if ((temp>=0) && (temp<=100))
    printf("\n WATER");
 else if (temp>100)
    printf("\n STEAM");
 getch();
}
```

Answers for Number 5:

Write a program that determines the Class of the Ship depending on its class ID. Here is the criteria. The Class ID serves as the input data and the Ship Class as output information.

Class ID	Ship Class
B or b	Battle Ship
C or c	Cruiser
D or d	Destroyer
F or f	Frigate

a.) First Solution: using **if/else if/else** conditional statements

```
#include <stdio.h>
main()
{
   char classid;
   clrscr();
   printf("\n Enter a class identification:");
   scanf("%c",&classid);
/* you can use classid = getche()  too */
   if ((classid='B') || (classid='b'))
       printf("\n Battleship");
   else if ((classid='c') || (classid='C'))
       printf("\n Cruiser ");
   else if ((classid='D') || (classid='d'))
       printf("\n Destroyer");
   else if ((classid='F') || (classid='f'))
```

```
        printf("\n Frigate");
    getch();
}
```

b.) Second Solution: using **switch/case** conditional statements

```
#include <stdio.h>
main()
{
 classid;
 clrscr();
 printf("\n Enter class identification:");
 classid=getche();
 switch (classid) {
 case 'B': case 'b':
     printf("\n Battleship"); break;
 case 'C': case 'c':
     printf("\n Cruiser"); break;
 case 'D': case 'd':
     printf("\n Destroyer"); break;
 case 'F': case 'f':
     printf("\n Frigate"); break;
}
getch();
}
```

Answer for Number 7:

Write a program that accepts a number and output its equivalent in words (maximum input number is 3000). For example:

Enter a number: 1380

one thousand three hundred eighty

```
#include <stdio.h>
 main()
{
   int n,r,t,h,e,o;
   clrscr();
   printf("\n Enter a number:");
   scanf("%d",&n);
   t=n/1000;
   n=n%1000;
   h=n/100;
   n= n%100;
/* if the input number is eleven, twelve… nineteen */
```

```
/* this code will be executed */
  if ((n>10)  && (n<20)) {
      e=0;
      o=0;
      r=n/10;
  }
  else {
     e=n/10;
     n=n%10;
     o=n;
 }
 switch (t) {
 case 1: printf("one thousand "); break;
 case 2: printf("two thousand "); break;
 case 3: printf("three thousand ");break;
}
switch (h)  {
case 1: printf("one hundred "); break;
case 2: printf("two hundred "); break;
case 3: printf("three hundred "); break;
case 4: printf("four hundred "); break;
case 5: printf("five hundred "); break;
case 6: printf("six hundred "); break;
case 7: printf("seven hundred "); break;
case 8: printf("eight hundred "); break;
case 9: printf("nine hundred "); break;
}
switch (r) {
case 1: printf("eleven "); break;
case 2: printf("twelve "); break;
case 3: printf("thirteen "); break;
case 4: printf("fourteen "); break;
case 5: printf("fifteen "); break;
case 6: printf("sixteen "); break;
case 7: printf("seventeen "); break;
case 8: printf("eighteen "); break;
case 9: printf("nineteen "); break;
}
switch (e) {
case 1: printf("ten "); break;
case 2: printf("twenty "); break;
case 3: printf("thirty "); break;
case 4: printf("forty "); break;
case 5: printf("fifty "); break;
case 6: printf("sixty "); break;
case 7: printf("seventy "); break;
case 8: printf("eighty "); break;
case 9: printf("ninety "); break;
}
```

```
switch (o) {
case 1: printf("one "); break;
case 2: printf("two "); break;
case 3: printf("three "); break;
case 4: printf("four "); break;
case 5: printf("five "); break;
case 6: printf("six "); break;
case 7: printf("seven "); break;
case 8: printf("eight "); break;
case 9: printf("nine "); break;
}
getch();
}
```

Answers for Number 9:

Write a program that computes and assess the tuition fee of the students in one trimester based on the given mode of payment plans below:

Plans (key) – Discount/Interest +

Cash (1) 10% Discount
Two-Installment (2) 5% Interest
Three-Installment (3) 10% Interest

The user must use the key in selecting or choosing mode of payment. The first input data is the tuition fee, and the second input data is the mode of payment. For example:

Main Menu
(1) Cash
(2) Two-Installment
(3) Three-Installment

Enter tuition fee: 20000
Press 1 for Cash, Press 2 for Two Installment,
Press 3 for Three Installment

Enter mod of payments: 2

This is a TUITION FEE ASSESSMENT PROGRAM

Your total tuition fee is: 21000

a.) First Solution using: **switch/case** conditional statements

```
#include <stdio.h>
main()
{
char plans;
float tfee, disint, atfee;
clrscr();
printf("\n Enter your tuition fee:");
scanf("%f",&tfee);
printf("\n Main Menu");
printf("\n (1) Cash ");
printf("\n (2) Two-Installment");
printf("\n (3) Three-Installment");
printf("\n Press 1 for Cash, 2 for Two-Installment");
printf("\n Press 3 for Three-Installment");
scanf("%d",&plans);
printf("\n\n Enter mode of payment:");
switch(plans) {
case 1: disint = tfee * 0.10;
          atfee = tfee + disint;
          break;
case 2: disint = tfee * 0.05;
          atfee = tfee-disint;
          break;

case 3: disint = tfee *0.10;
          atfee = tfee-disint;
          break;
}
printf("\n This is a TUITION FEE ASSESSMENT PROGRAM");
printf("\n Your total tuition fee is: %8.2f",atfee);
getch();
}
```

b.) Second Solution using **if/else if** conditional statements

```
#include <stdio.h>
main()
{
int plans;
float tfee, disint,atfee;
clrscr();
printf("\n Enter your tuition fee:");
scanf("%f",&tfee);
printf("\n Main Menu");
printf("\n (1) Cash ");
printf("\n (2) Two-Installment");
printf("\n (3) Three-Installment");
printf("\n Press 1 for Cash, 2 for Two-Installment");
printf("\n Press 3 for Three-Installment");
```

```
scanf("%d",&plans);
if (plans==1) {
   disint = tfee * 0.10;
   atfee = tfee-disint;
}
else if (plans==2) {
    disint = tfee * 0.05;
    atfee = tfee-disint;
}
else if (plans==3) {
    disint = tfee *0.10;
    atfee = tfee-disint;
}
printf("\n This is a TUITION FEE ASSESSMENT PROGRAM");
printf("\n Your total tuition fee is: %8.2f",atfee);
getch();
}
```

Solutions to the Odd Numbers of Chapter 4 (Looping Statements)

Answers for Number 1:
Write a program that calculates and produces these two columns sequence numbers using the three looping statements:

N	Squared
1	1
2	4
3	9
4	16
5	25

1st Solution: Using *for* loop statement
2nd Solution: Using *while* loop statement
3rd Solution: Using *do-while* loop statement

a.)First Solution using: **for** loop statement

```
#include <stdio.h>
main()
{
int n;
clrscr();
printf("\n The number and its square equivalent");
printf("\n  N     Squared ");
for (n=1; n<=5; n++){
   printf("\n %d    %d",n,n*n);
```

```
   delay(11000);
}
getch();
}
```

b.) Second Solution using: **while** loop statement

```
#include <stdio.h>
main()
{
int n;
clrscr();
printf("\n The number and its square equivalent");
printf("n  N     Squared ");
n=1;
while (n<=5) {
   printf("\n %d    %d",n,n*n);

   n++;
   delay(1000);
}
getch();
}
```

c.)Third Solution using: **do**-**while** loop statement

```
#include <stdio.h>
main()
{
int n;
clrscr();
printf("\n The number and its square equivalent");
printf("\n  N     Squared ");
n=1;
do {
   printf("\n %d    %d",n,n*n);
   n++;
   delay(1000);
} while (n<=5);
getch();
}
```

Answers for Number 3:

Write a program which produces the given sequence numbers (in alternate arrangement and a reverse order of the problem number 2) using the three looping statements:

5, 1, 4, 2, 3, 3, 2, 4, 1, 5,

1st Solution: Using *for* loop statement
2nd Solution: Using *while* loop statement
3rd Solution: Using *do-while loop* statement

a.) First Solution using: **for** loop statement

```
#include <stdio.h>
main()
{
int i,j;
clrscr();
j=5;
for (i=1; i<=5; i++) {
   printf("%d, %d,",j,i);
   j--;
   delay(11000);
}
getch();
}
```

b.) Second Solution using: **while** loop statement

```
#include <stdio.h>
main()
{
int i,j;
clrscr();
j=5;
i=1;
i=1;
while (i<=5) {
  printf("%d, %d,",j,i);
  j--;
  i++;
  delay(11000);
}
getch();
}
```

c. Third Solution using: **do**-**while** loop statement

```
#include <stdio.h>
main()
{
int i,j;
```

```
clrscr();
j=5;
i=1;
i=1;
do {
  printf("%d, %d,",j,i);
  j--;
  i++;
  delay(11000);
} while (i<=5);
getch();
}
```

Answers for Number 5:

Write a program that will generate the Fibonacci sequence numbers of **n** (as input) and displays them (series). In Fibonacci, the current third number is the sum of two previous numbers. Apply three solutions using the three loop statements.

Example:

Enter a number: 9
Fibonacci series:

1 1 2 3 5 8 13 21 34

a.) First Solution using: **for** loop statement

```
#include <stdio.h>
main()
{
long int p, p1,p2;
int i,n;
clrscr();
printf("\n Enter a number:");
scanf("%d",&n);
p1=1;
p2=1;
p=0;
for (i=1; i<=n; i++) {
   printf("%d ",p);
if ( i>=2) {
   p=p1+p2;
   p2=p1;
   p1=p;
}
delay(11000);
}
```

```
getch();
}
```

b.) Second Solution using: **while** loop statement

```
#include <stdio.h>
main()
{
long int p, p1,p2;
int i,n;
clrscr();

printf("\n Enter a number:");
scanf("%d",&n);
p1=1;
p2=1;
p=0;
i=1;
while (i<=n) {
  printf("%d ",p);
  if ( i>=2) {
     p=p1+p2;
     p2=p1;
     p1=p;
  }
  i++;
  delay(11000);
}
getch();
}
```

c.)Third Solution using: **do**-**while** loop statement

```
#include <stdio.h>
main()
{
long int p, p1,p2;
int i,n;
clrscr();
printf("\n Enter a number:");
scanf("%d",&n);
p1=1;
p2=1;
p=0;
i=1;
do {
   printf("%d ",p);
   if ( i>=2) {
```

```
      p=p1+p2;
      p2=p1;
      p1=p;
    }
    i++;
    delay(11000);
} while (i<=n);
getch();
}
```

Answers for Number 7:

Write a program to scan a number **n** and then output the sum of the squares from 1 to **n**. Thus, if the input is 4, the output should be 30 because:

$$1^2 + 2^2 + 3^2 + 4^2 = 1 + 4 + 9 + 16 = 30$$

a.) First Solution using: **for** loop statement

```
#include <stdio.h>
main()
{
   int i,n;
   long int s,p;
   clrscr();
   printf("\n Enter a number:");
   scanf("%d",&n);
   s=1;
   p=0;
   for (i=1; i<=n; i++) {
      s= i*i;
      p=p+s;
   }
    printf("  %d",p);
   getch();
}
```

b.) Second Solution using: **while** loop statement

```
#include <stdio.h>
main()
{
   int i,n;
   long int s,p;
   clrscr();
   printf("\n Enter a number:");
   scanf("%d",&n);
```

```
   s=1;
   p=1;
   i=1;
   while (i<=n) {
     s= i*i;
     p=p+s;
     i++;
   }
   printf("  %d",p);
   getch();
}
```

c.) Third Solution using: **do**-**while** loop statement

```
#include <stdio.h>
main()
{
   int i,n;
   long int s,p;
   clrscr();
   printf("\n Enter a number:");
   scanf("%d",&n);
   s=1;
   p=1;
   i=1;
   do {

     s=i*i;
     p=p+s;
     i++;
   } while(i<=n);
   printf("  %d",p);
   getch();
}
```

Answers for Number 9:
Write a program that reverses the input number of **n.** Formulate an equation to come up with the answer.

Sample input/output dialogue:

Enter a number: 1238
Reversed: 8321

a.) First Solution using: **while** loop statement

```
#include <stdio.h>
main()
{
int n,r;
clrscr();
printf("\n Enter a number:");
scanf("%d",&n);

while (n!=0) {
 r= n%10;
 printf("%d",r);
 n=n/10;
}
getch();
}
```

b.) Second Solution using: **do**-**while** loop statement

```
#include <stdio.h>
main()
{
int n,r;
clrscr();
printf("\n Enter a number:");
scanf("%d",&n);
do {
 r= n%10;
 printf("%d",r);
 n=n/10;
} while (n!=0);
getch();
}
```

c.) Third Solution using: **for** loop statement

```
#include <stdio.h>
main()
{
int n,r;
clrscr();
printf("\n Enter a number:");
scanf("%d",&n);
for (;n!=0;) {
 r= n%10;
 printf("%d",r);
 n=n/10;
```

```
}
getch();
}
```

Solution to Odd Numbers of Chapter 5 (Functions)

Answer for Number 1:

Write a function-oriented program to calculate the area of a circle. Use the formula:
$A = \pi r^2$ where Pi (π) is equal to 3.1416 approximately.

```
#include <stdio.h>
#define Pi 3.1416
float areac(int r);
main()
{
  int rad;
  clrscr();
  printf("\n Enter radius:");
  scanf("%d",&rad);
  printf("\n The area of a circle: %8.2f",areac(rad));
  getch();
}

float areac(int r)
{
float area;
area=Pi*r*r;
return area;
}
```

Answer for Number 3:

Write a function-oriented program to convert an input Fahrenheit into Celsius degree. Use the formula : C = (5.0/9.0) * F- 32.0. Display the Celsius degree.

```
#include <stdio.h>
float fahcel(float f);
main()
{
  float fah;
  clrscr();
  printf("\n Enter Fahrenheit:");
  scanf("%f",&fah);
  printf("\n Celcius degree equivalent:%8.2f",fahcel(fah));
  getch();
```

```
}
```

```
float fahcel(float f)
{
  float cel;
  cel = (5.0/9.0) * f – 32.0;
  return cel;
}
```

Answer for Number 5:

Write a function-oriented program that generates the Fibonacci series numbers of **n** (as input) and display them (series). In Fibonacci, the current third number is the sum of two previous numbers. Example:

Enter a number: 8
Fibonacci series:

1 1 2 3 5 8 13 21

```
#include <stdio.h>
long fibo(int no);
main()
{
  int n;
  clrscr();
  printf("\n Enter a number:");
  scanf("%d",&n);
  fibo(n);
  getch();
}

long fibo(int no)
{
 long int i,p,p1,p2;
 p1=1;
 p2=1;
 p=0;
 for (i=1; i<=no; i++) {
    printf("%d  ",p);
    if (i>=2) {
      p= p1+p2;
```

```
      p2=p1;
      p1=p;
    }

   delay(1000);
 }
 return 0;
}
```

Answer for Number 7:

Write a function-oriented program that calculates the sum of the squares from 1 to **n**. Thus, if the input is 3, the output should be 14 because: $1^2 + 2^2 + 3^2 = 1 + 4 + 9 = 14$

```
#include <stdio.h>
long int sqrsum (int no);
main()
{
  int n;
  clrscr();
  printf("\n Enter a number:");
  scanf("%d",&n);
  printf("%d \n",sqrsum(n));
  getch();
}

long int sqrsum(int no)
{
  long int i,s,p;
  i=1;
  s=1;
  p=0;
  while (i<=n) {
    s=i*i;
    p=p+s;
    i++;
  }
  return p;
}
```

Answer for Number 9:

Write a function-oriented program to convert the input dollar(s) into its equivalent peso. Assume that one dollar is equivalent to 51.60 pesos.

```
#include <stdio.h>
float doltopes(int d);
main()
{
  int dol;
  clrscr();
  printf("\n Enter dollar:");
  scanf("%d",&dol);
  printf("\n The peso conversion is: %8.2f",doltopes(dol));
  getch();
}

float doltopes(int d)
{
 float dtop;
 /* assuming that one dollar is equal to 51.60 */
 dtop=51.60 * d;
 return dtop;
}
```

Solution to the Odd Numbers of Chapter 6 (Arrays)

Answer for Number 1:

Write a program using one-dimensional array that calculates the sum and average of the five input values from the keyboard and prints the calculated sum and average.

```
#include <stdio.h>
main()
{
  int i,n[5];
  float sum,ave;
  clrscr();
  printf("\n Enter five numbers:\n");
  for (i=0; i<5; i++) {
    scanf("%d",&n[i]);
  }
  sum=0;
  for (i=0; i<5; i++) {
     sum = sum+ n[i];
  }
```

```
  ave=sum/5;


  printf("\n The sum: %8.2f",sum);
  printf("\n The average:%8.2f",ave);
  getch();
}
```

Answer for Number 3:

Write a program using one-dimensional array that accept five input values from the keyboard. Then it should also accept a number to search. This number is to be searched if it is among the five input values. If it is found, display the message "Searched number is found!", otherwise display "Search number is lost!".

Sample input/output dialogue:

Enter five numbers:
10 15 20 7 8
Enter a number to search: 7
Search number is found!

```
#include <stdio.h>
main()
{
  int i,n[5],nsearch,found;
  clrscr();
  printf("\n Enter five numbers:");
  for (i=0;i<5; i++) {
      scanf("%d",&n[i]);
  }
  printf("\n Enter a number to search:");
  scanf("%d",&nsearch);
  for (i=0; i<5; i++)
     if (nsearch==n[i])
         found=1;
     if (found==1)
        printf("\n Search number is found!");
     else
        printf("\n Search number is lost!");
  getch();
}
```

Answer for Number 5:

Write a program using two-dimensional array that searches a number and display the number of times it occurs on the list of 12 input values.

Sample input/output dialogue:

Enter twelve values:

13 15 20 13 30 35 40 16 18 20 18 20

Enter a number to search: 20
Occurrences: 3

```
#include <stdio.h>
main()
{
  int n[12], nsearch, occur, i;
  clrscr();
  printf("\n Enter twelve numbers:\n\n");
  for (i=0; i<12; i++ ) {
     scanf("%d",&n[i]);
  }
  printf("\n Enter a number to search:");
  scanf("%d",&nsearch);
  occur=0;
  for (i=0; i<12; i++) {
     if (nsearch==n[i])
        occur = occur + 1;
  }
  printf("\n Occurrence(s): %d",occur);
  getch();
}
```

Answer for Number 7:

Write a program using two dimensional arrays that determines the highest and lowest of the 12 input values. For example:

Enter twelve numbers:

13 15 20 13 35 40 16 18 20 18 20 14

highest: 40
lowest: 13

```
#include <stdio.h>
main()
{
  int n[3][4];
  int high, low,i,j;
  clrscr();
  printf("\n Enter twelve numbers:\n\n");
  for (i=0; i<3;i++)
    for (j=0; j<4; j++)
       scanf("%d",&n[i][j]);
  high=n[0][0];
  /* this loop determines the highest */
  for (i=0; i<3; i++)
     for (j=0; j<4;j++) {
        if (high<n[i][j])
           high = n[i][j];
     }
 /* this loop determines the lowest */
 low=n[0][0];
 for (i=0; i<3; i++)
   for (j=0; j<4;j++) {
      if (low>n[i][j])
         low=n[i][j];
   }
 printf("\n highest: %d",high);
 printf("\n lowest: %d",low);
 getch();
}
```

Answer for Number 9:

9. Write a program using two-dimensional arrays that computes the sum of data in rows and sum of data in columns of the 3x3 (three by three) array variable n[3[3].

Sample input/output dialogue:

```
 5   9   8 = 22
 3   8   2 = 13
 4   3   9 = 16
--------------------
12  20  19
```

```
#include <stdio.h>
main()
{
  int n[3][3], i,j,rsum,csum;
```

```
  int c=1, r=3;
  clrscr();
  csum=0;
  rsum=0;
  printf("\n Enter nine numbers:\n");
  /* accepting 9 numbers and compute the sum of rows */
  for (i=0; i<3; i++) {
     c=1;
     for (j=0; j<3;j++) {
       scanf("%d",&n[i][j]);
     rsum = rsum+ n[i][j];
     c++;
     }
     gotoxy(c+7,r); printf("%d",rsum);
     rsum=0;
     r++;
  }
  /* computes the sum of the columns */
  r++;
  c=0;
  printf("\n-----------------");
  for (j=0; j<3; j++) {
    csum=0; c+=3;
    for (i=0; i<3; i++) {
       csum=csum+n[i][j];
    }
    gotoxy(c-2,r); printf("%d",csum);
  }
  getch();
}
```

Solutions to the Odd Numbers of Chapter 7 (Strings)

Answer for Number 1:

Write a program using string functions that accepts a price of an item and displays its coded value. The base of the key is: X C O M P U T E R S

0 1 2 3 4 5 6 7 8 9

Example:

Enter price: 489.50
Coded value: PRS.UX

```
#include <stdio.h>
#include <string.h>
```

```
main()
{
  char sprice[10];
  int slength,I;
  clrscr();
  printf("\n Enter price:");
  gets(sprice);
  slength=strlen(sprice);
  printf("\n Coded value:");
  for (I=0; I<slength; I++) {
    switch(sprice[I])  {
      case '0': printf("X"); break;
      case '1': printf("C"); break;
      case '2': printf("O"); break;
      case '3': printf("M"); break;
      case '4': printf("P"); break;
      case '5': printf("U"); break;
      case '6': printf("T"); break;
      case '7': printf("E"); break;
      case '8': printf("R"); break;
      case '9': printf("S"); break;
      case '.': printf("."); break;
     }
  }
  getch();
}
```

Answer for Number 3:

Write a program using string functions that determines if the input word is a palindrome. A palindrome is a word that produces the same word when it is reversed. Example:

Enter a word : ama
Reversed: ama
It is a palindrome

Or

Enter a word: hello
Reversed: olleh
Not a palindrome

```
#include <stdio.h>
#include <string.h>
main()
{
char w1[10], w2[10], wr[10];
```

```
int same;
clrscr();
printf("\n Enter a word:");
gets(w1);
strcpy(w2,w1);
strcpy(wr,strrev(w2));
same=strcmpi(w1,wr);
if (same==0) {
   printf("\n Reversed:%s",wr);
   printf("\n It is a Palindrome");
}
else {
   printf("\n Reversed: %s",wr);
   printf("\n Not a Palindrome");
}
getch();
}
```

Answer for Number 5:

Write a simple decryption program using string functions which apply the substitution method. Here is the given substitution table:

Substitution Table:

*	A
$	E
/	I
+	O
-	U

Example:

Encrypted message: m$$t m$ ***t 9:00 *.m. /n th$ p***rk
Decrypted message: meet me at 9:00 a.m. in the park

```
#include <stdio.h>
#include <string.h>
main()
{
char message[80];
int nlength,i;
clrscr();
printf("\n Enter Encrypted message:");
gets(message);
nlength = strlen(message);
```

```
for (i=0; i<nlength; i++) {
   switch(message[i]) {
   case `*': message[i]='a'; break;
   case `$': message[i]='e'; break;
   case `/': message[i]='i'; break;
   case `+': message[i]='o'; break;
   case  `-`: message[i]='u'; break;
    default: message[i]=message[i];
    }
 }
 printf("\n Decrypted: %s",message);
 getch();
}
```

Answer for Number 7:

Write a program using string functions that will accept the name of the capital as input value and will display the corresponding country.

CAPITALS	COUNTRIES
Ottawa	Canada
Washington D.C.	United States
Moscow	U.S.S.R.
Rome	Italy
Manila	Philippines

```
#include <stdio.h>
#include <string.h>
main()
{
 char ot[20], wa[20], mo[20], ro[20], ma[20], capital[20];
 clrscr();
 printf("\n Enter the capital city:");
gets(capital);
strupr(capital);
strcpy(ot,"OTTAWA");
strcpy(wa,"WASHINGTON D.C.");
strcpy(mo,"MOSCOW");
strcpy(ro,"ROME");
strcpy(ma,"MANILA");
printf("\n Country is:");
if (strcmp(ot,capital)==0)
   printf("  Canada");
else if (strcmp(wa,capital )==0)
  printf("  United States");
else if (strcmp(mo,capital)==0)
```

```
  printf("  U.S.S.R.");
else if (strcmp(ro,capital)==0)
  printf("  Italy");
else if (strcmp(ma,capital)==0)
  printf("  Philippines");
getch();
}
```

Answer for Number 9:

Write a program that takes nouns and forms their plurals on the basis of these rules:

a.) If noun ends in "y", remove the "y" and add "ies".
b.) If noun ends in "s", "ch", or "sh", add "es".
c.) In all other cases, just add "s".

```
#include <stdio.h>
#include <string.h>
main()
{
char noun1[15], noun2[15]=" ";
int lnoun=0, lnoun1=0, lnoun2=0,n=0;
clrscr();
printf("\n Enter a noun:");
gets(noun1);

strlwr(noun1); lnoun=strlen(noun1);
lnoun1=lnoun-1;
lnoun2=lnoun-2;
if (( noun1[lnoun1]=='s') || (noun1[lnoun1]=='h'
&& noun1[lnoun2]=='c') || (noun1[lnoun1]=='h'
&& noun1[lnoun2]=='s')) {
   strcat(noun1,"es");
   printf("\n %s",noun1);
}
else if (noun1[lnoun1]=='y') {
  n=lnoun-1;
  strncpy(noun2,noun1,n);
  strcat(noun2,"ies");
  printf("\n %s",noun2);
}
else {
   strcat(noun1,"s");
   printf("\n %s",noun1);
}
getch();
}
```