Programming

Mazen Nehme

Programming Art

Anyone can write code that a computer can understand. Good programmers write code that humans can understand~

Mazen Nehme

Programming

Programming Art:

Anyone can write code that a computer can understand. Good programmers write code that humans can understand~

☙ Course Objectives:

The main objective of this course is to introduce students to the basic concepts of programming using the C++ language and prepare them to write simple and complex programs.

Upon completion of this course, students should be able to:

1. Understand problem-solving techniques and apply it to programming.
2. Write clear, elementary C++ programs.
3. Code with C++ data types, arithmetic, increment, decrement, assignment, relational, equality and logical operators.
4. Code C++ control structures (if, if/else, switch, for, while, do/while).
5. Understand and manipulate arrays.
6. Write user-defined function definitions.
7. Pass arrays to functions and pointers.
8. Use recursions.
9. Manipulate strings functions and pointers basics.

☙ Evaluation Plan:

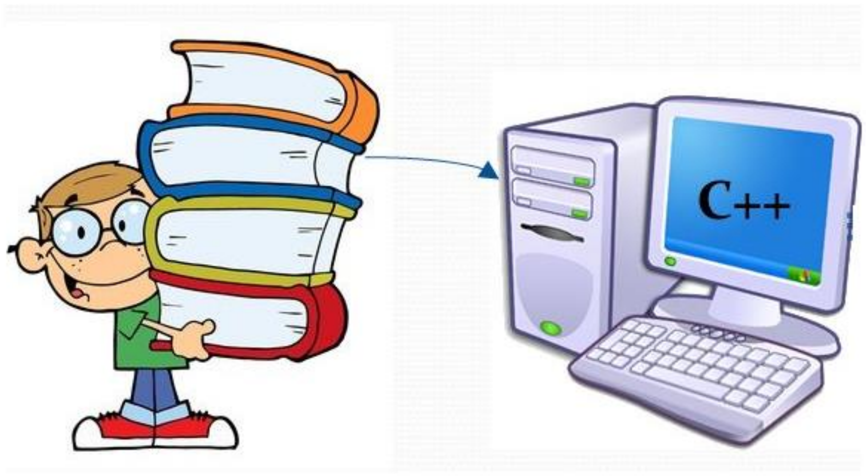
Students will be evaluated in this course using a combination of assessment methods, including:

- Class assessment (40%).
- Final Exam (60%).

Table of Contents:

Chapter 1

Introduction to Programming



Objectives:

In this chapter, we will introduce programming languages, their generations, and how a program can be executed by a computer. Basically, the following topics are discussed:

- What is programming? What is an algorithm?
- What are the different generations of programming languages?
- The C++ language.
- Writing and compiling your first C++ program.

Introduction to Programming

1.1- What is Computer Programming?

Computer programming is simply writing computer programs to solve specific problems using one of the programming languages such as: C++, Java, C#, VB.net…

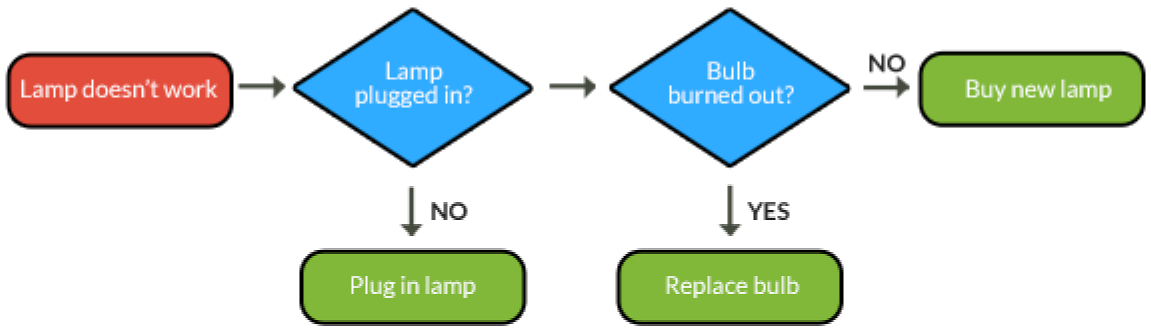
A **computer program** is a set of instructions or programming statements that tell the computer what to do, which information to identify and access, how to process it, and what equipment to use... You can only see it written on a paper, or displayed on screen. A program is usually stored on a hard disk, a compact disc (CD), or a DVD.

A program requires an **Author** to write it (**Programmer**), a **Processor** to execute it (**CPU**), and it must operates on certain **Objects** (**Data**).

1.2- What is an Algorithm?

An **algorithm** is a procedure or logical steps for solving a specific problem. The word derives from the name of the mathematician, ***Mohammed al-Khwarizmi***, who lived from about 780 to 850 in Baghdad. Al-Khwarizmi's work is the likely source for the word algebra as well. In programming, it could be easier to write the algorithm first before transforming it into an executable program using a specific programming language.

Let's take a look at the Lamp algorithm shown in the figure:



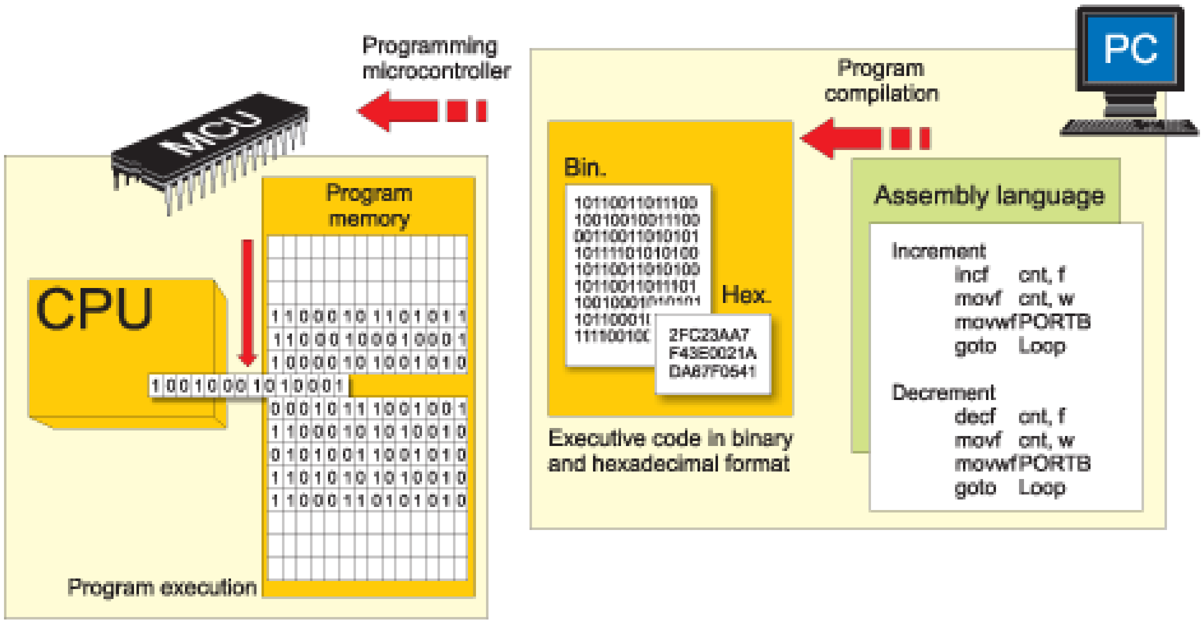
1.3- Generations of Programming Languages:

1.3.1- Machine Language:

A computer can directly understand only its own **Machine Language**. Programs written in any other type of language must be translated into machine language before they can be executed by the computer. The machine language uses the binary system (0,1) to represent data and instructions of programs. The first programs were written in binary form (0,1) which was a difficult task and charged with errors.

1.3.2- Assembly Language:

As computers became more popular, instead of using the machine language, programmers began using the **Assembly Language** which is English-like abbreviations to represent data and instructions of programs. A program written in Assembly Language required translation into Machine Language via a translator program called: **The Assembler.**



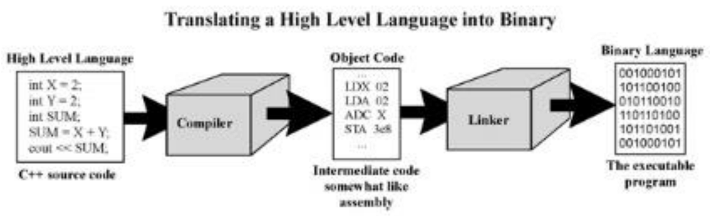
```
Increment
      incf    cnt, f
      movf    cnt, w
      movwf   PORTB
      goto    Loop

Decrement
      decf    cnt, f
      movf    cnt, w
      movwf   PORTB
      goto    Loop
```

```
2FC23AA7
F43E0021A
DA67F0541
```

1.3.3- High Level Languages:

High-level languages (such as: **C** and **C++**) were developed to simplify and to speed up the programming process, in which a single statement could be written to accomplish a complete task. These languages allow to write instructions that look almost like every day English and math formulas. For example, now you can write: **Average= Sum/number**.

Also, programs written in high level language must be translated into machine-language by special programs called ‘**COMPILERS**’ such as: Microsoft Visual Studio and Borland turbo C compilers.

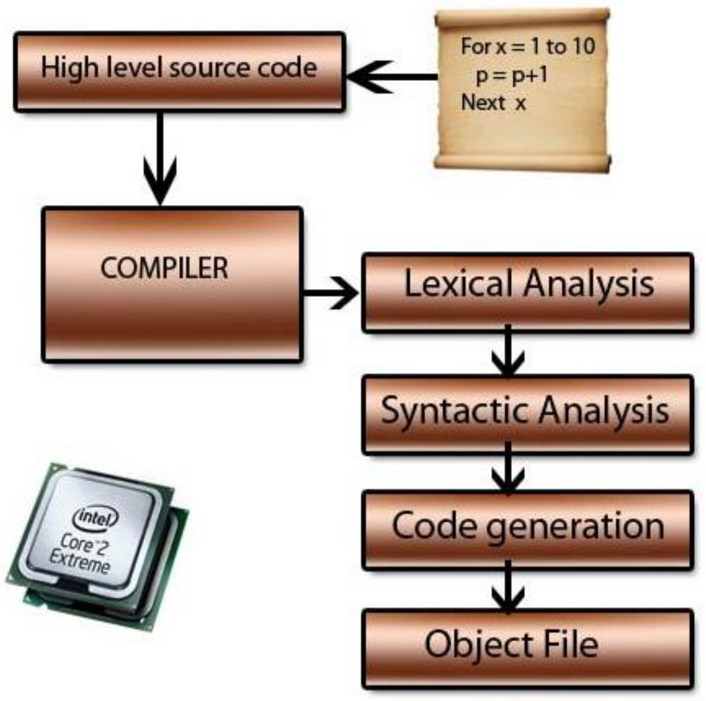


```
int X = 2;
int Y = 2;
int SUM;
SUM = X + Y;
cout << SUM;
```

```
...
LDX  02
LDA  02
ADC  X
STA  3e8
...
```

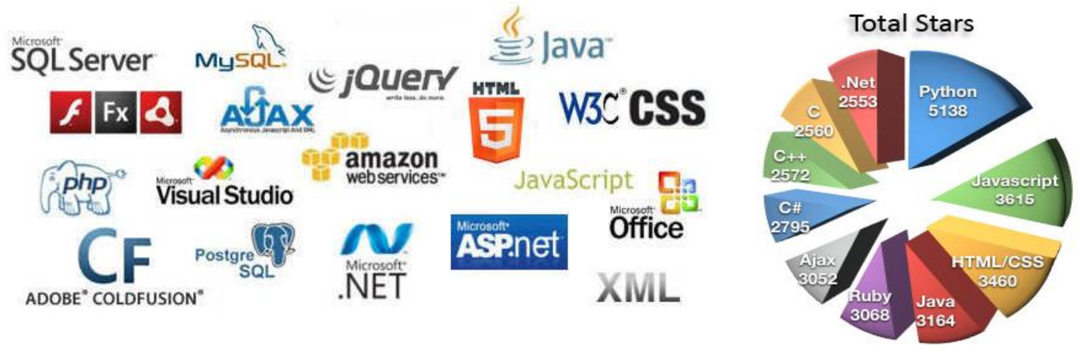
```
001000101
101100100
010110010
110110100
101101001
001000101
```

A **compiler** is a program that translates the source code for another program into executable code. The source code is typically in a high-level programming language.



1.3.4- Current Programming Languages Trends:

Programming language evolution continues, in both industry and research. Some of the current trends are shown in the figure below:



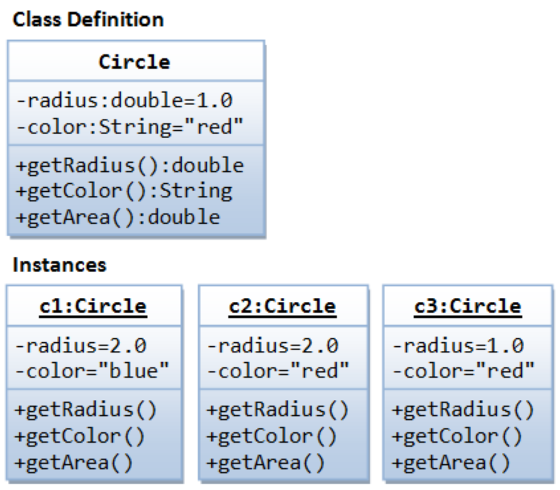
1.4- What is C++ Language?

C++ is a general purpose object oriented programming (OOP) language, developed by "**Bjarne Stroustrup"**, and is an extension of "**C language"**. Initially, the language was called 'C with classes' as it had all properties of C language with an additional concept of 'classes'. However, it was renamed to C++ in 1983.

With C++ you can write different types of applications such as "calculations, displaying high speed graphics in a game or video, and controlling electronic devices", with also facilities for low-level memory manipulation. C++ has also been found useful in many other contexts, including desktop applications, servers (e.g. e-commerce, web search or SQL servers), and entertainment software (e.g. Games).

LIVE Q&A
with the Father of C++
BJARNE STROUSTRUP
C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows away your whole leg.

The main highlight of C++ is a collection of pre-defined classes, which are data types that can be instantiated multiple times. A ***Class is*** a concept of data structures, they can contain data members and functions. An ***object*** is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable. The concept of object programming will be studied in further advanced levels.

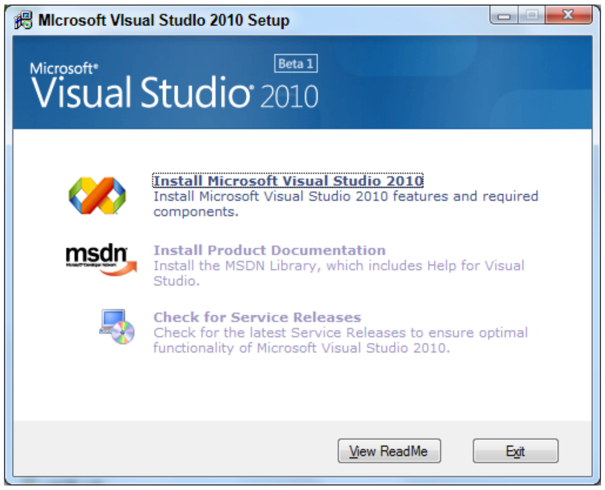


LAB Task:

Installing your C++ compiler

There are simply too many C++ compilers. But it is recommended to choose a compiler that conforms to the ISO standards. I will list down some of popular compilers that you can use for writing your C++ programs:

1. **Visual Studio Express :** **https://www.visualstudio.com/en-US/products/visual-studio-express-vs**


Microsoft Visual Studio 2010 Setup
Microsoft® Visual Studio® 2010
Beta 1
View ReadMe
Exit

Install Microsoft Visual Studio 2010
Install Microsoft Visual Studio 2010 features and required components.

Install Product Documentation
Install the MSDN Library, which includes Help for Visual Studio.

Check for Service Releases
Check for the latest Service Releases to ensure optimal functionality of Microsoft Visual Studio 2010.

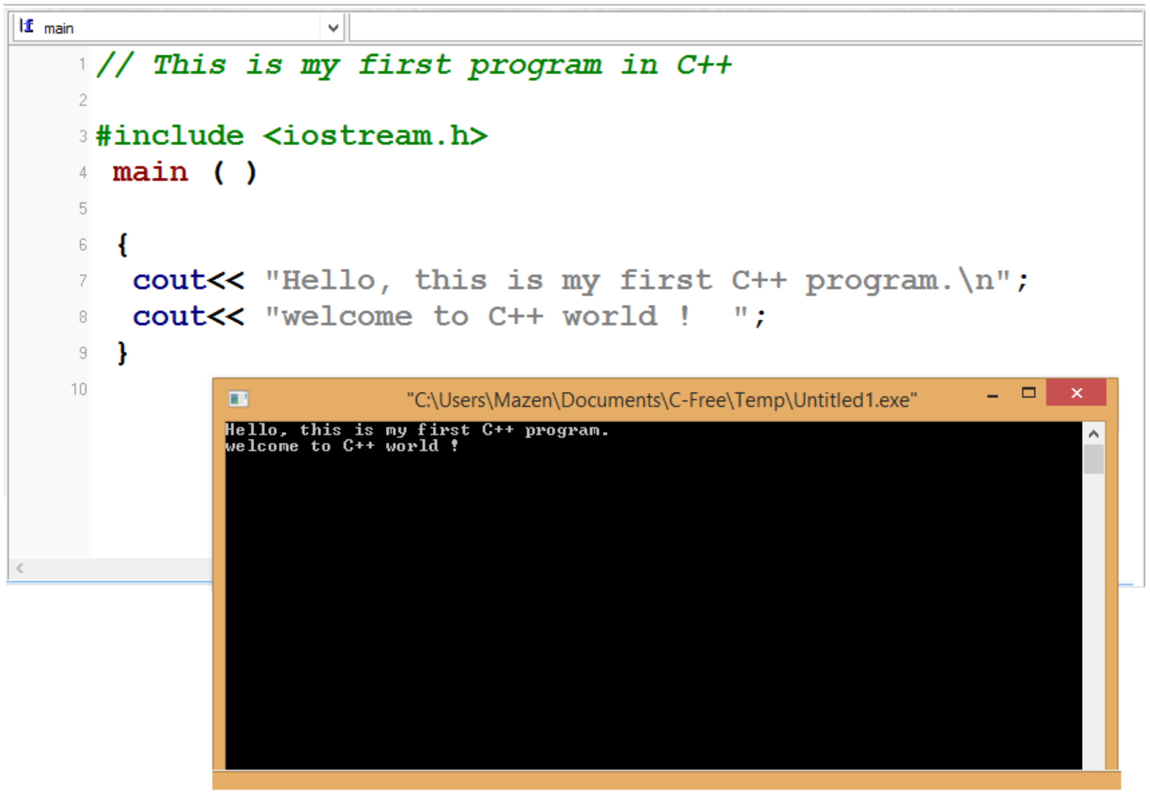
2. **Borland turbo C++ :** **http://www.softpedia.com/get/Programming/Coding-languages-Compilers/TurboCplusplus-for-Windows-7.shtml**

3. **C Free compiler :** **http://www.programarts.com/cfree_en/download.htm**

4. **C++ online Compiler :** **http://webcompiler.cloudapp.net/**

LAB Task:

Writing Your First C++ Program:



```
// This is my first program in C++

#include <iostream.h>
 main ( )

 {
  cout<< "Hello, this is my first C++ program.\n";
  cout<< "welcome to C++ world !   ";
 }
```

"C:\Users\Mazen\Documents\C-Free\Temp\Untitled1.exe"

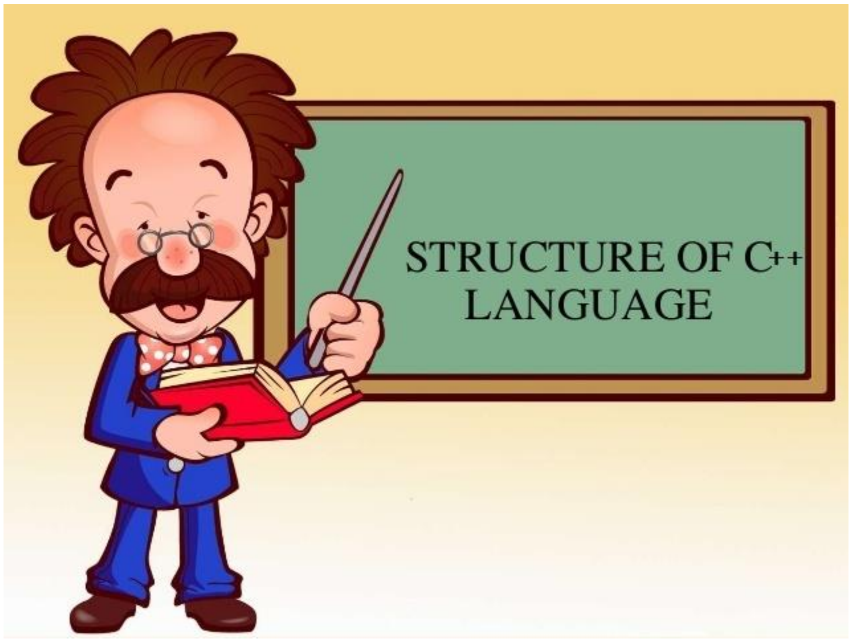
```
Hello, this is my first C++ program.
welcome to C++ world !
```

Compile and Execute your C++ Program:

1. Open your C++ editor and write the code as above.
2. Save your file with **.cpp** extension as: **hello.cpp**

Chapter 2

Structure of a C++ Program



Objectives:

In this chapter, we will explain the basic elements of a C++ program including the following:

- Structure of a C++ program including its main elements
  - The #include directive and the header files
  - The main function
  - The C++ Standard Library
  - Variables and Constants
- The escape character
- The Output Statement: cout <<
- The Input Statement: cin>>

Structure of a C++ Program

2.1- Essential elements of a C++ Program:

A program is made up of sentences that contain basically variables, constants and instructions that we call statements. Often, it is necessary to include a set of libraries and classes in your program. These libraries and classes contain a set of predefined functions that you can use in your program.

Let us look at a simple program that displays the words:

Welcome to C++ Programming

```
1  /* This is a simple program that will display text on
   Two lines  */
2  #include <iostream.h>
3  int main( )    // main( ) is where program execution begins.
4  {
5    cout<< "Welcome to C++ \nProgramming";    // Displays on screen
7    return 0;                  6
   }
```

1. **The comments:** Comments can be used to describe the program objective, the programmer name or role of a variable. This will help the readers to better understand your source code. However, comments will be ignored by the compiler.

   - You can use: /* */ to write comments on multiple lines within your program. The comments must be inserted between the two asterisks.

   - You can also use // to write a single-line comment Single-line comments begin with // and stop at the end of the line.

2. **The '#include' directive and the 'header files':** System header files (such as: iostream.h, math.h, conio.h…) declare the interfaces to parts of the operating system and libraries. You include them in your program to supply the definitions and declarations you need to invoke system calls and libraries.

    - For this program, the header **<iostream.h>** is needed for the execution of **'cout<<'** statement.

    - Header files are included using the preprocessing directive: **#include**.

    - You can use the header files in two ways (according to your compiler):

    #include<iostream.h> //traditional C++

    Typically used with Turbo C++ and C Free compilers

    OR:

    #include<iostream> // ANSI Standard
    using namespace std;

    Typically used with Visual Studio compiler. The line **using namespace std**; tells the compiler to use the **std namespace**. **Namespaces** are a relatively recent addition to C++.

3. **main( ):** is the main function where program execution begins. Each program include only one main function.

4. **{ }:** indicates the beginning and the end of the main function. They can be used also to indicate begin/end of other programming structures.

5. **cout <<:** cout is used for displaying data on the screen.
    - Example of displaying a text: **cout<<"Weclome to C++";**
    - Example of displaying the value of a variable: **cout<<A;**

6. **\n or endl :** move to a new line.

7. **return 0**: terminates **main( )** function and causes it to return the value 0 to the calling process. Note that you can use **'void main'** instead of **'int main'**. Void means that the function has no return value, and thus you must not use the 'return ( )' statement in this case.

2.2- The C++ Standard Library:

C++ is standardized by the **International Organization for Standardization (ISO)**, with the latest standard version published by ISO in December 2014 as ISO/IEC 14882:2014 (informally known as C++14).

The C++ standard consists of two parts: the **core language** and the **C++ Standard Library** that includes vectors, lists, maps, algorithms, sets, queues, stacks, arrays, input/output facilities (iostream, for reading from and writing to the console and files), time utilities...

For now, you will need to use the following header files of the C++ library:

- ✓ **<iostream.h>** provides C++ **input and output** fundamentals.
- ✓ **<conio.h>** is used mostly to provide console **input/output**.
- ✓ **<math.h>** declares a set of functions to compute common **mathematical** and Trigonometric operations.
- ✓ **<string.h>** defines several **functions** to manipulate C **strings** (set of characters) and arrays.

2.3- Variables and Constants:

A variable is a memory location that we can use to store a value for later processing. Each variable in C++ has a specific type, which determines the size and the range of values that can be stored in the variable. To use a variable:

1. **First, you must declare it as follow:**

```
Syntax:      datatype   variable name;
               int          A            ;
```

2. **Then, you can assign it a value as follow:**
   - int A = 5 ;
   - float B= 3.5;
   - char x = ‘a’;   char n = ‘1’ ;
   - char name [4] = “troy”;
   - char name []= “Mazen”;

Data Types:

Data types represent the basic storage units supported by most systems. There are following basic types of variable in C++:

Type	Bits	Range
int	16 or 2 bytes	-32768 to -32767
unsigned int	16	0 to 65535
signed int	16	-31768 to 32767
short int	16	-31768 to 32767
unsigned short int	16	0 to 65535
signed short int	16	-32768 to -32767
long int	32 or 4 bytes	-2147483648 to 2147483647
unsigned long int	32	-2147483648 to 2147483647
signed long int	32	0 to 4294967295
float	32 or 4 bytes	3.4E-38 to 3.4E+38
double	64 or 8 bytes	1.7E-308 to 1.7E+308
long double	80	3.4E-4932 to 3.4E+4932
char	8 or 1 byte	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127

- Use ***int*** and ***long*** to declare variables that will contain integers.
- Use ***float*** and ***double*** to declare variables that will contain floating numbers.
- Use ***char*** to declare variables that will contain characters or strings.

Boolean variables: bool variable

There are two Boolean literals and they are part of standard C++ keywords:

- A value of **1** representing **true**.
- A value of **0** representing **false**.

Example:

```
bool bValue = true;                              Outputs:
cout << bValue << endl;    ____________________     1
cout << !bValue << endl;   ____________________     0

bool bValue2 = false;
cout << bValue2 << endl;   ____________________     0
cout << !bValue2 << endl;  ____________________     1
```

✑ Naming a Variable:

The following are all the characters you can use to make a valid variable name:

- **Characters:** **A** through **Z** and **a** through **z**.
- **Digits 0 through 9**: which can be used in any position except the first of a variable name.
- **The underscore character:** _

☝ Now, let's see what you cannot use in variable naming:

- A variable name cannot contain any **C arithmetic signs**.
- A variable name cannot contain any **dots (.)**
- A variable name cannot contain any apostrophes **(‘) or (“)**
- A variable name cannot contain any other special symbols such as: ***, @, #, ?**...
- The C++ language reserves certain **keywords** that have special meanings to the language. Never use the **C++ keywords**, as variable names or function names in your program. For example, the following are C++ keywords:

asm	continue	float	new	signed	try
auto	default	for	operator	sizeof	typedef
break	delete	friend	private	static	union
case	do	goto	protected	struct	unsigned
catch	double	if	public	switch	virtual
char	Else	Inline	register	template	void
class	enum	int	return	this	volatile
const	extern	long	short	throw	while

Variables Sizes:

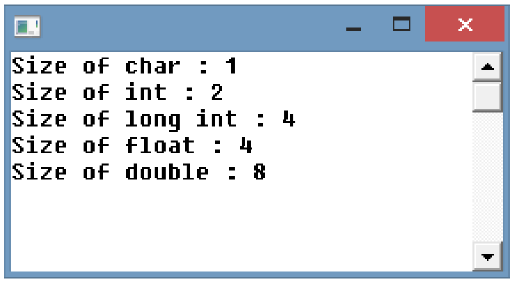
The sizes of variables might be different, depending on the compiler and the computer you are using. You can use the **sizeof ( ) function** to produce the correct size of various data types on your computer.

Exercise 1: Write a program to display the sizes of char, int, long, float and double on your computer.

```
//Solution – Exercise 1: Using sizeof( )
#include <iostream.h>
void main( )
{
  cout << "Size of char : " << sizeof(char) << endl;
  cout << "Size of int : " << sizeof(int) << endl;
  cout << "Size of long int : " << sizeof(long int) << endl;
  cout << "Size of float : " << sizeof(float) << endl;
  cout << "Size of double : " << sizeof(double) << endl;
  }
```

This example uses **endl,** which inserts a new-line character after every line. We are also using **sizeof()** function to get size of various data types.

When the above code is compiled and executed, it produces the following **result** which can vary from machine to machine:



```
Size of char : 1
Size of int : 2
Size of long int : 4
Size of float : 4
Size of double : 8
```

Defining Constants:

There are two simple ways in C++ to define constants:

- Using **#define** preprocessor → **#define variable_name value**
- Using **const** keyword → **const** type variable = value;

Exercise 2: Write a C++ program that define the length as 10 and the width as 5 and calculates the area of a surface.

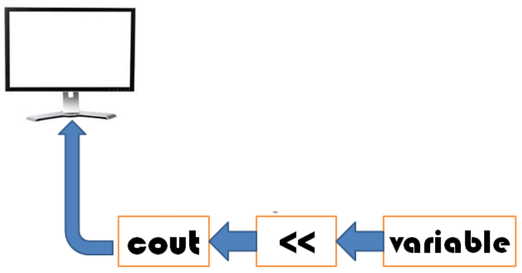
```
//Solution – Exercise 2: using #define
#include <iostream.h>
#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'
void main( )
{
   int area;
   area = LENGTH * WIDTH;
   cout << area;
   cout << NEWLINE;
 }
```

```
//Solution – Exercise 2:  using const
#include <iostream.h>
 void main( )
{
   const int  LENGTH = 10;
   const int  WIDTH  = 5;
   const char NEWLINE = '\n';
   int area;
   area = LENGTH * WIDTH;
   cout << area;
   cout << NEWLINE;
   }
```

- When the above code is executed, it produces the following result: **50**
- Note that it is a good programming practice to define constants in **CAPITALS**.

2.4- The Output Statement: cout <<

cout<< is used to send values out to the screen.



- **Syntax:**

```
cout<<  "text" ;
```

✓**Meaning:** Output/Display the message within the quotes to the screen

```
cout <<  variable_name ;
```

✓ **Meaning:** Meaning: Output/Display the value of the variable <variable_name> to the screen

```
cout << endl ;
```

✓**Meaning:** Display a new line

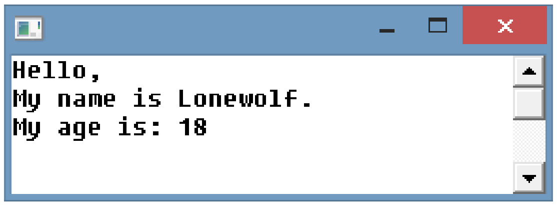
✓ **Examples:**

```
cout << a;
cout << b << c;
cout    << "Hello"      << variable_name <<     " Hello"      << endl  ;
```

➥ **Example:**

```
//Using the cout<< function to display a text and a variable
#include <iostream.h>
 void main( )
{
 int age = 18;
 cout<<"Hello, \nMy name is Lonewolf." <<endl;
 cout<<"My age is: " <<age;
 }
```

- When the above code is executed, it produces the following result:



```
Hello,
My name is Lonewolf.
My age is: 18
```

2.5- The escape character: \

There are certain characters in C++ when they are preceded by a backslash they will have special meaning and they are used to represent like newline (\n) or tab (\t). Here, you have a list of some of such escape sequence codes:

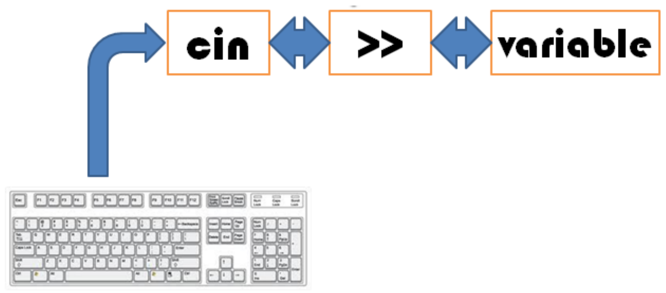
Escape sequence	Result
\n	New line (line feed)
\a	Beep
\t	Horizontal tab
\v	Vertical tab
\r	Carriage return (line home)
\0	Null
\”	Quotation mark “
\’	Apostrophe ‘
\\	Backslash \

2.6- The Input Statement: cin >>

cin>> is used to extract Input from the User using the Keyboard

- **Syntax:**

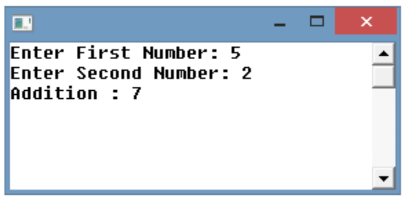
```
cin >>  variable_name;
```



Exercise 3: **Write a C++ program that takes in two numbers and makes the addition.**

```
//Solution – Exercise 3: This program takes in two numbers and makes the addition
#include<iostream.h>
void main( )
{
   int n1, n2 // declaring two variables n1 and n2
   cout<<"Enter First Number: ";
   cin>>n1;        //accept first number
   cout<<"Enter Second Number: ";
   cin>>n2;        //accept first number
   cout<<"Addition: ";
   cout<<number1+number2;       //Display Addition
}
```

- When the above code is executed, it produces the following result:

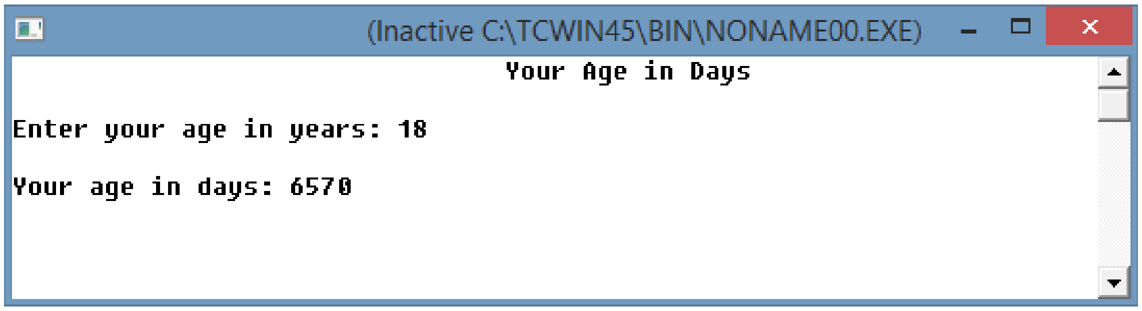


```
Enter First Number: 5
Enter Second Number: 2
Addition : 7
```

Exercise 4: Write a C++ program that takes in your age in years and displays it in days. Note that 1 year = 365 day.

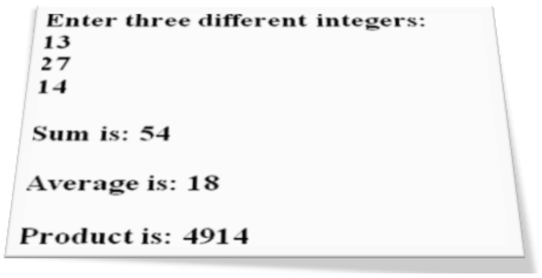
//Solution – Exercise 4

- Your program should produces the following result:


(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

```
                                 Your Age in Days

Enter your age in years: 18

Your age in days: 6570
```

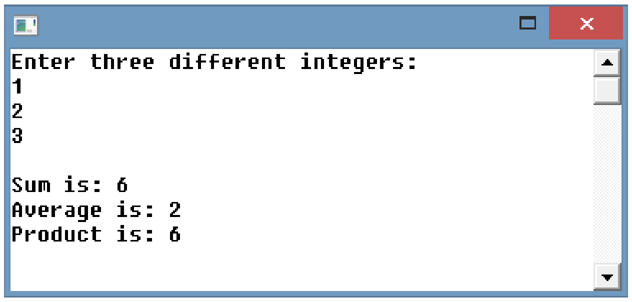
Exercise 5: Write a C++ program that accepts 3 different integers from the user, and display the sum, the average, and the product of these numbers. The output screen should be like:



```
Enter three different integers:
13
27
14

Sum is: 54

Average is: 18

Product is: 4914
```

```
//Solution – Exercise 5
#include<iostream.h>
void main( )
{       int a, b, c; // three integers
        int s= 0, p =1; // sum is initialized to 0 and product to 1
        float v; //average

        cout<<"Enter three different integers:\n";
        cin>>a>>b>>c;
        // you can write it as: cin>> a; cin>> b; cin>>c;
        s=a+b+c;
        p=a*b*c;
        v= s/ 3; // or: v = (a+b+c)/3;

        cout<<"\nSum is: " <<s <<endl;
        cout<<"Average is: " <<v <<endl;
        cout<<"Product is: " <<p;}
```

- When the above code is executed, it produces the following result:



```
Enter three different integers:
1
2
3

Sum is: 6
Average is: 2
Product is: 6
```

Chapter 3

Decision Making in C++

Objectives:

In programming you will need to direct your program according to different conditions related to program activities. In this chapter we will explain how to use the condition statements in C++ including the following:

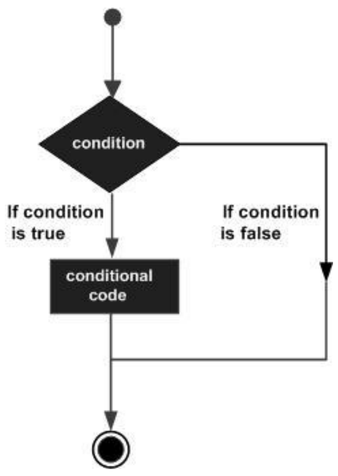
- The if statement
- The if else statement
- Nested if
- The switch statement
- Practical exercises

C++ Decision Statements

3.1- C++ Decision Making:

Decision making require that the programmer specify one or more conditions to be evaluated by the program and directs program execution depending on the result of that evaluation. The figure on the right, is the general form of a typical decision making structure.

C++ programming language provides following types of decision making statements:

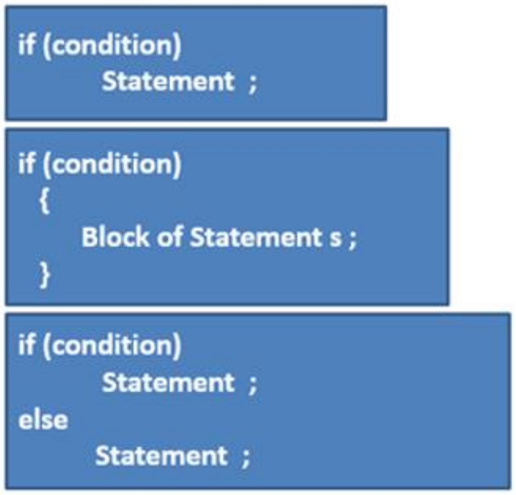


Statement	Short description
if statement	An if statement consists of a boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested if statements	You can use one if or else if statement inside another if or else if statement.
nested switch statements	You can use one swicth statement inside another switch statement.

3.1.1- The if - else Statement:

In C++ the ***if*** statement is used to test a Boolean expression (condition). If the condition is true then the block of statements within the ***if structure*** gets executed. If boolean expression evaluates to false, then the first set of code after the end of the if statement will be executed.

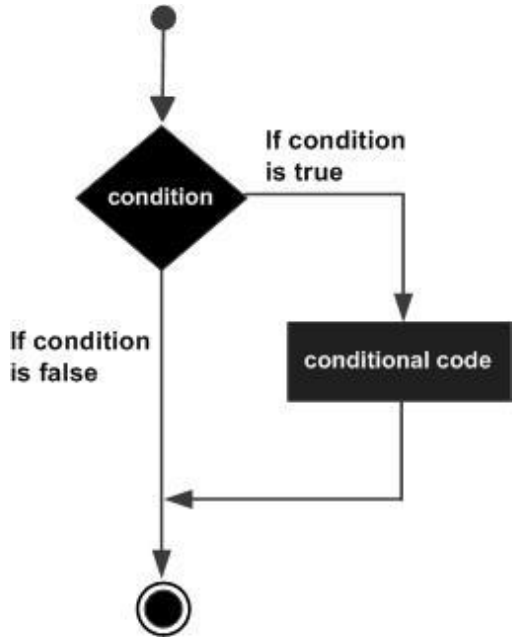
Syntax:



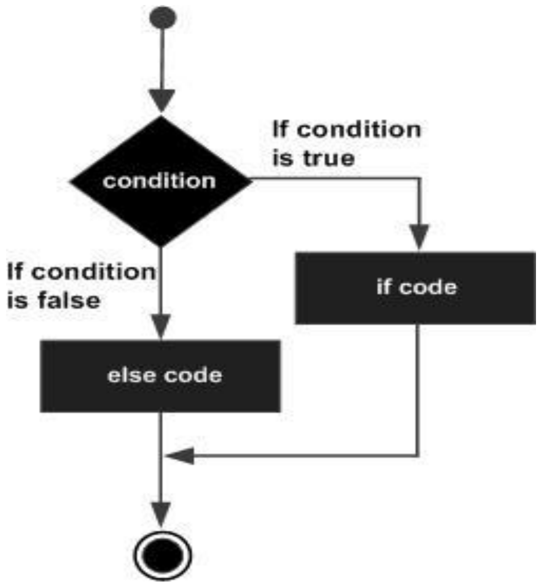
```
if (condition)
        Statement ;
```

```
if (condition)
  {
        Block of Statement s ;
  }
```

```
if (condition)
        Statement ;
else
        Statement ;
```



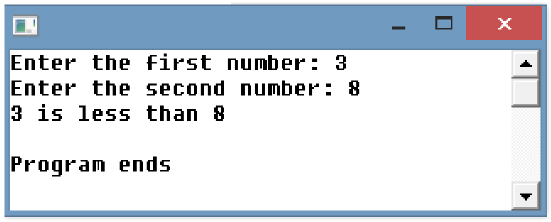
Optionally, the ***If structure*** is sometimes followed with the ***else block***. When the Boolean expression evaluates to false then the ***else*** block gets executed.



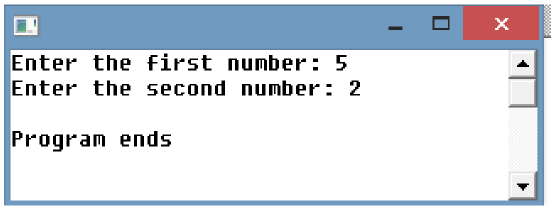
Exercise 6: Write a C++ program that takes in two numbers and check if the first is less than the second.

```
//Solution – Exercise 6: a simple if condition

#include <iostream.h>
int main ( )
{
 int a , b;
 cout<<"Enter the first number: ";
cin>>a;
cout<<"Enter the second number: ";
cin>>b;

// check the boolean condition
   if( a < b )
        cout << a <<" is less than " <<b << endl;

cout<<"\nProgram ends";
return 0;
}
```

- When the above code is executed, it produces the following results:



```
Enter the first number: 3
Enter the second number: 8
3 is less than 8

Program ends
```

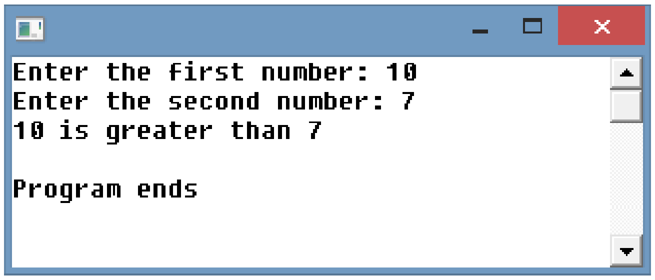


```
Enter the first number: 5
Enter the second number: 2

Program ends
```

Exercise 7: Write a C++ program that takes in two numbers and determine if the first is greater than or less than the first.

```
//Solution – Exercise 7: simple if .. else condition

#include <iostream.h>
int main ( )
{
 int a , b;
 cout<<"Enter the first number: ";
cin>>a;
cout<<"Enter the second number: ";
cin>>b;

// check the boolean condition
   if( a < b )
        cout << a <<" is less than " <<b << endl;
   else
       cout<< a <<" is greater than " <<b <<endl;

cout<<"\nProgram ends";
return 0;
}
```

- When the above code is executed, it produces the following results:



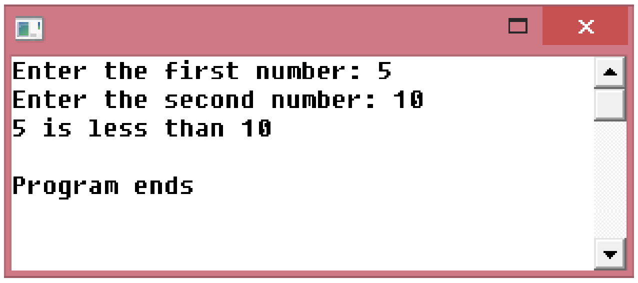
```
Enter the first number: 10
Enter the second number: 7
10 is greater than 7

Program ends
```

Exercise 8: Write a C++ program that takes in two numbers and makes the comparison. Now, we are going to test if the first number is greater than or less than or equal to the second.

//Solution – Exercise 8: Comparing two numbers

```
#include <iostream.h>
int main ( )
{
 int a , b;
 cout<<"Enter the first number: ";
cin>>a;
cout<<"Enter the second number: ";
cin>>b;

// check the boolean condition
   if( a < b )
        cout << a <<" is less than " <<b << endl;
   else
       if( a > b )
            cout<< a <<" is greater than " <<b <<endl;
       else
               cout <<"Both numbers are equals" <<endl;
cout<<"\nProgram ends";
return 0;
}
```

- When the above code is executed, it produces the following results:



```
Enter the first number: 5
Enter the second number: 10
5 is less than 10

Program ends
```

Exercise 9: Write a C++ program that determine the SIGN (+ , - , 0) of a number.

Exercise 10: Write a C++ program that takes in an integer and determine if it is ODD or EVEN.

Exercise 11: Write a C++ program that takes in two numbers and determine if the second is a multiple of the first.

3.1.2- The C++ Operators:

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ provides the following types of operators:

Arithmetic Operators:

Arithmetic Operators		
Operator	Function	Use
+	unary plus	+ expr
-	unary minus	- expr
*	multiplication	expr * expr
/	division	expr / expr
%	remainder	expr % expr
+	addition	expr + expr
-	subtraction	expr - expr

Rules of operator precedence:

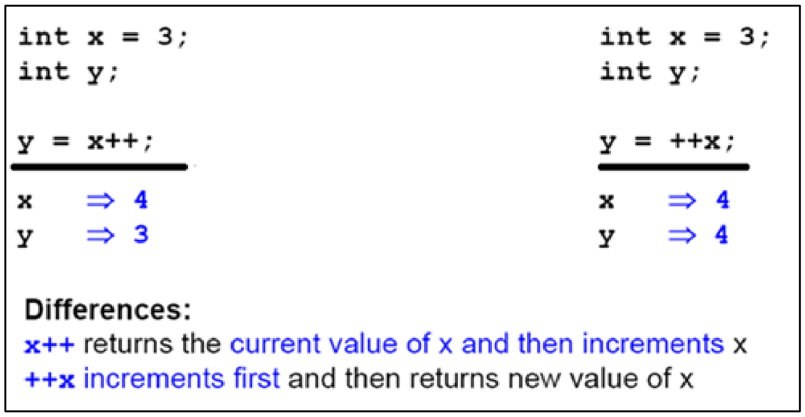
Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
*, /, or %	Multiplication,Division, Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ or –	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

Example: Find the average of three variables **a**, **b** and **c**

- Do not use: `a + b + c / 3`
- Use: `(a + b + c ) / 3`

Increment and Decrement Operators:

In C++ there is a short hand for increment and decrement operations: x++ and x--
Note the difference between Prefix and Postfix increment or decrement:



```
int x = 3;                    int x = 3;
int y;                        int y;

y = x++;                      y = ++x;

x   ⇒ 4                       x   ⇒ 4
y   ⇒ 3                       y   ⇒ 4
```

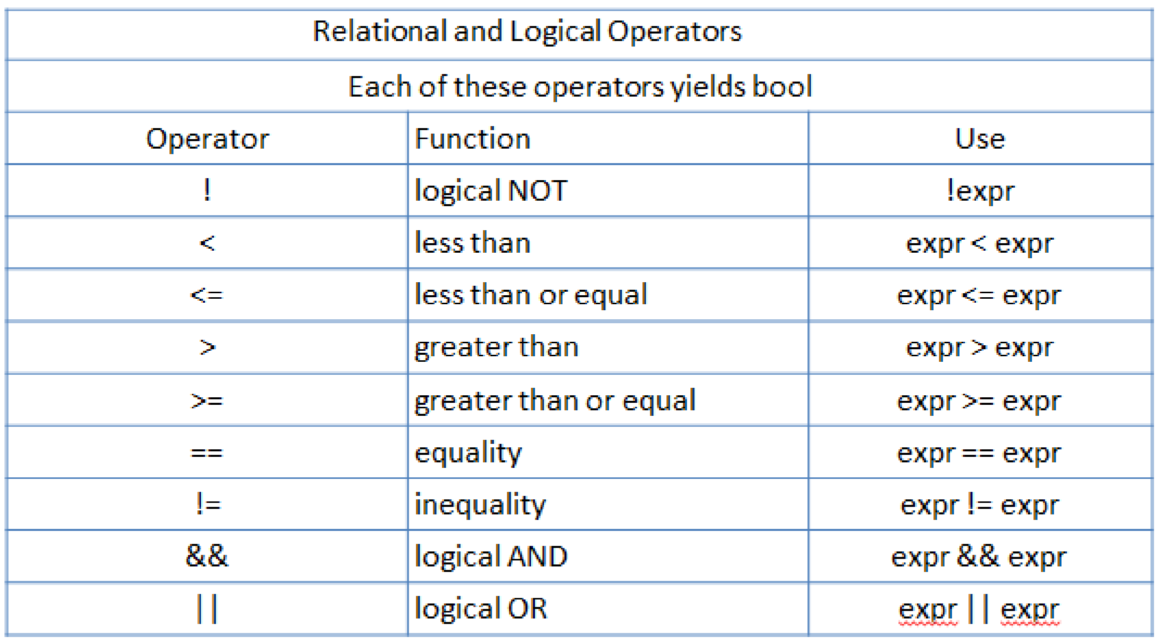
Differences:
x++ returns the current value of x and then increments x
++x increments first and then returns new value of x

Assignment Operators:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A

%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2
*	Pointer operator * is pointer to a variable. For example *var; will pointer to a variable var.	

Relational and Logical Operators:



Relational and Logical Operators		
Each of these operators yields bool		
Operator	Function	Use
!	logical NOT	!expr
<	less than	expr < expr
<=	less than or equal	expr <= expr
>	greater than	expr > expr
>=	greater than or equal	expr >= expr
==	equality	expr == expr
!=	inequality	expr != expr
&&	logical AND	expr && expr
||	logical OR	expr || expr

Logical Operators:

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	**(A && B) is false.**
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	**(A || B) is true.**
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	**!(A && B) is true.**

Operators Precedence in C++:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Category	Operator	Associativity
Postfix	**() ++ - -**	**Left to right**
Unary	**+ - ! ++ & sizeof**	**Right to left**
Multiplicative	*** / %**	**Left to right**
Additive	**+ -**	**Left to right**

Relational	< <= > >=	**Left to right**
Equality	== !=	**Left to right**
Logical AND	&&	**Left to right**
Logical OR	||	**Left to right**
Assignment	= += -= *= /= %= > >= < <=	**Right to left**
Comma	,	**Left to right**

Exercise 12: **Write a C++ program that calculates the net salary of an employee. Given the following information: The amount of sales (AS), The Tax Rate (TR), and The Salary (S).**

Consider the following:

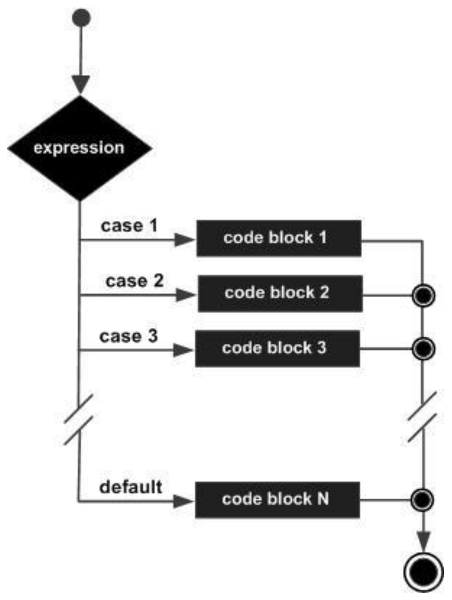
- If the amount of sales is greater than 10000, the employee gets a commission of 4% of the sales amount. Otherwise, he will get only 2%.
- Gross Salary = Salary + commission.
- Tax = Gross salary * Tax Rate.
- Net Salary = Gross salary – Tax

3.1.3- The switch statement:

The switch statement is another condition statement. You can use it instead of multiple if conditions in some cases. Typically, a **switch** statement allows a variable to be tested against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

The syntax for a switch statement in C++:

```
switch (variable){
   case constant-expression  :
     statement(s);
     break; //optional

   case constant-expression  :
     statement(s);
     break; //optional
  ---
    default : //optional
     statement(s);
}
```



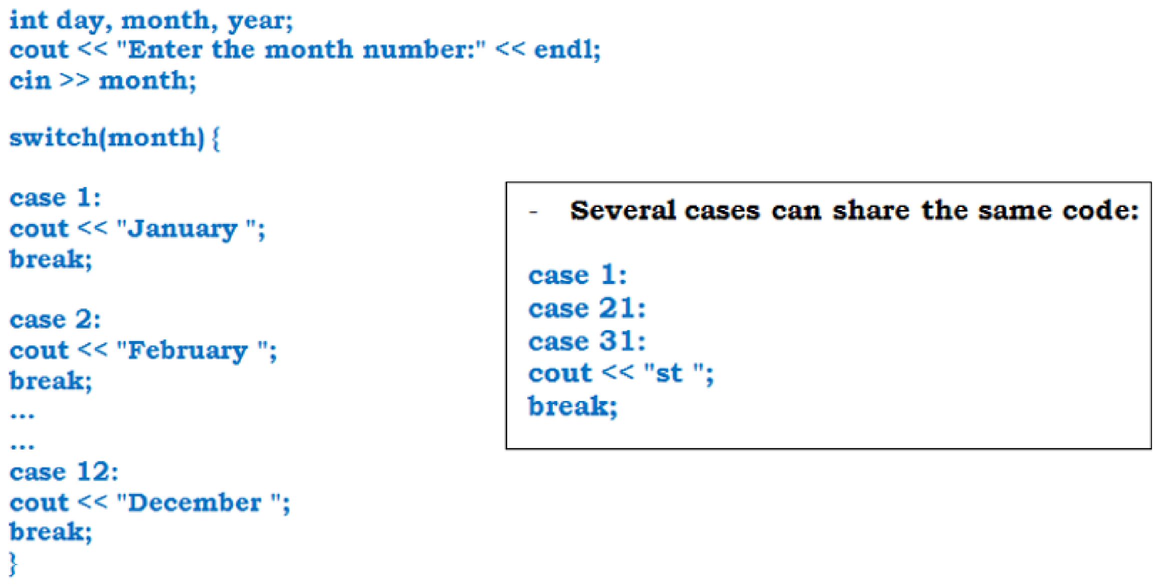
The following rules apply to a switch statement:

- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The constant-expression for a case must be the same data type as the variable in the switch.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- If one of the case values matches the expression, execution jumps to those statements and continues to the end of the switch block, unless a break statement is encountered.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- If nothing matches, the execution moves to the optional default statement.
- If there is no default and there is no matching value, execution falls through the switch statement and the statement ends.

Examples:

```
switch(1) {
   case 1 : cout << '1'; // displays "1",
   case 2 : cout << '2'; // then displays "2"
}
```

```
switch(1) {
   case 1 : cout << '1'; // displays "1"
          break;      // and exits the switch
   case 2 : cout << '2';
        break;
}
```



```
int day, month, year;
cout << "Enter the month number:" << endl;
cin >> month;

switch(month) {

case 1:
cout << "January ";
break;

case 2:
cout << "February ";
break;
...
...
case 12:
cout << "December ";
break;
}
```

- **Several cases can share the same code:**

```
case 1:
case 21:
case 31:
cout << "st ";
break;
```

Example:

```cpp
#include <iostream>
using namespace std;

int main ()
{
   // local variable declaration:
   char grade = 'D';

   switch(grade)
   {
   case 'A' :
      cout << "Excellent!" << endl;
      break;
   case 'B' :
   case 'C' :
      cout << "Well done" << endl;
      break;
   case 'D' :
      cout << "You passed" << endl;
      break;
   case 'F' :
      cout << "Better try again" << endl;
      break;
   default :
      cout << "Invalid grade" << endl;
   }
   cout << "Your grade is " << grade << endl;

   return 0;
}
```

This would produce the following result:

```
You passed
Your grade is D
```

switch example	if-else equivalent

`switch (x) {`
`  case 1:`
`    cout << "x is 1";`
`    break;`
`  case 2:`
`    cout << "x is 2";`
`    break;`
`  default:`
`    cout << "value of x unknown";`
`  }`
	`if (x == 1) {`
	`  cout << "x is 1";`
	`}`
	`else if (x == 2) {`
	`  cout << "x is 2";`
	`}`
	`else {`
	`  cout << "value of x unknown";`
	`}`

Exercise 13: Write a C++ program that calculates the surface of a rectangle or the volume of a circle, depending on the user input.

- An input of 'R' or 'r' means, calculate the surface of a rectangle:
  Surface of rectangle = Length *Width

- An input of 'C' or 'c' means, calculate the volume of a circle:
  Volume of Circle = (4π Rad^3)/3. (Rad: is the radius of circle and π =3.14)

Exercise 14: Write a C++ program that solve a quadratic equation of the form: $ax^2 + bx + c = 0$, using delta = b2 – 4ac.

First, the user must input 3 values for: a, b, and c. Then program will calculate delta:
D = b2 – 4ac ; and then display the roots as follow:

- if delta is < 0 ➔ no roots -- no solution
- if delta = 0 ➔ we have two equal roots: x1 = x2 = -b/2a
- if delta >0 ➔ we have two distinct roots:
  x1 = (-b – sqrt (D))/2a and x2 = (-b + sqrt (D))/2a

The Output should be as follow:

```
Delta

Enter the values of a , b and c:
2
-1
4

Delta = -31.000000
Delta is negative, there is no solution
```

Exercise 15: Cola Machine

Write a program that presents the user a choice of 4 beverages (Water, Cola, Sprite, and Fanta). Then ask the user to choose a beverage by entering a number from 1-4; and displays the name of the beverage that he chooses. If the user enters a choice other than 1-4 then output: "Invalid choice, here is your money back."

Exercise 16: If you have two fractions, a/b and c/d, their sum can be obtained from the formula:

$$\frac{a}{b} + \frac{c}{d} = \frac{a * d + b * c}{b * d}$$

for example, 1/4 plus 2/3 is:

$$\frac{1}{4} + \frac{2}{3} = \frac{1 * 3 + 4 * 2}{4 * 3} = \frac{3+8}{12} = \frac{11}{12}$$

Write a program that encourages the user to enter two fractions and the display their sum in fractional form. The interaction with the user might look like this:

```
Enter first fraction: 1 / 2
Enter second fraction: 2 / 5
sum = 9 / 10
```

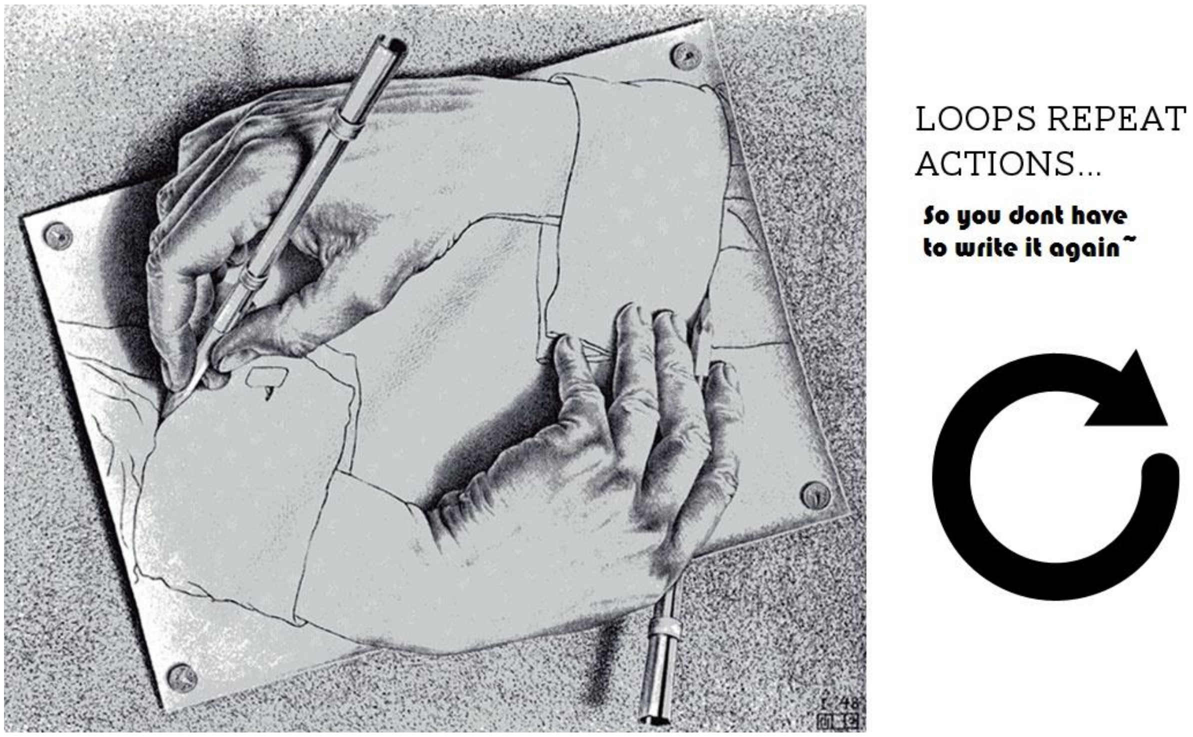
You can take the advantage if the fact that the extraction operator (>>) can be chained to read in more than one quantity at once: cin >> a>> ch >>b

Exercise 17: Write a C++ program that read X and calculates Y as follow:

$$Y = \left| \begin{array}{lll} X + 1 & \textbf{if} & X > 0 \\ X^3 + 4X+10 & \textbf{if} & X = 0 \\ X^2 + 3 & \textbf{if} & X < 0 \end{array} \right.$$

Chapter 4

Loops in C++



Objectives:

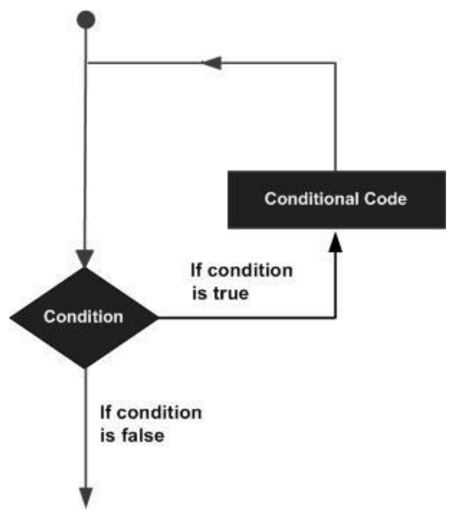
There may be a situation, when you need to execute a block of code for a several number of times. A loop statement allows you to execute a statement or a group of statements multiple times without having to write the same code again. In this chapter you will learn the following forms of programming loops:

- The for statement
- The while statement
- The do while statement
- Nested loops
- Practical exercises

Loops in C++

4.1- Looping:

Loops repeat a statement a certain number of times, or while a condition is fulfilled. For example, if you want to print the same word 50 times. You could type 50 cout<< statements, but it is easier to use a loop to execute the same cout<< statement 50 times.

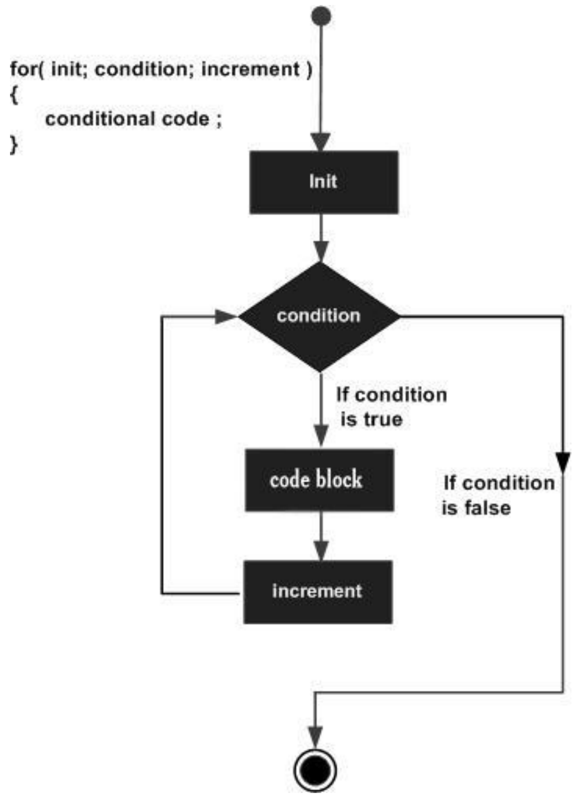


C++ programming language provides the following types of loops:

Loop Type	Description
for loop	Execute a sequence of statements multiple times. It is mostly used when the number of repetition is defined and known.
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body. It is mostly used when the number of repetition is not defined exactly.
do...while loop	Like a while statement, except that it tests the condition at the end of the loop body. It is mostly used for more control on input.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

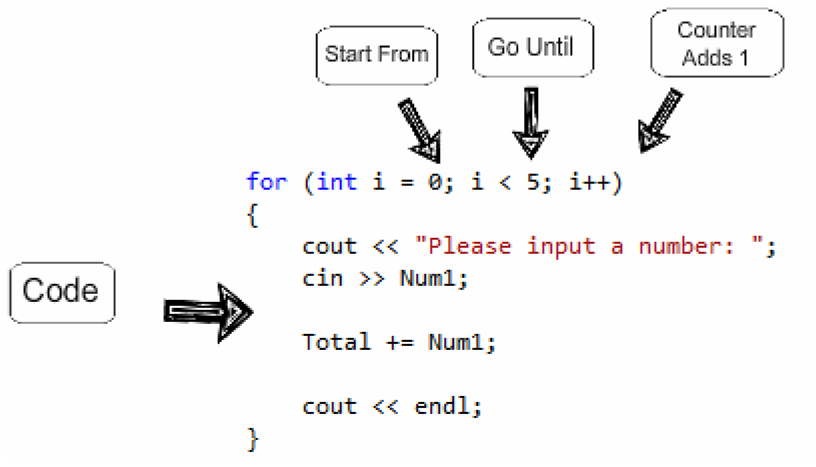
4.1.1- The for loop:

The ***for*** loop executes a section of code a fixed number of times. It's usually used when you know, before entering the loop, how many times you want to execute the code. The **syntax** of a for loop in C++ is:



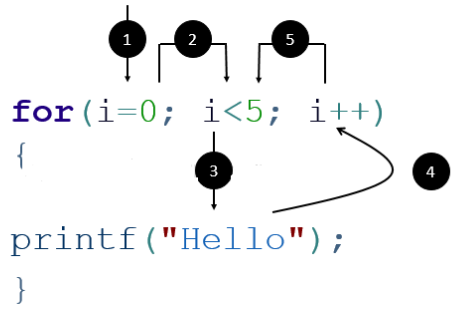
- The **init** step is executed first, and only once. This step allows you to declare and initialize the loop counter. This statement can be left blank, as long as a semicolon appears after the condition.
- Next, the **condition** is evaluated. If it is true, the code block inside the loop is executed. If it is false, the flow of control jumps to the next statement just after the for loop (exit the loop).
- After the code block executes, the flow of control jumps back up to the **increment** statement. This statement allows you to update the counter variable. This statement can be left blank, as long as a semicolon appears after the condition.
- The condition is now evaluated again. If it is true, the loop executes and the process **repeats** itself (body of loop, then increment step, and then again condition). After the condition becomes false, the for loop **terminates**.

Example: the following code takes in 5 numbers and calculate the total.



```
for (int i = 0; i < 5; i++)
{
    cout << "Please input a number: ";
    cin >> Num1;

    Total += Num1;

    cout << endl;
}
```

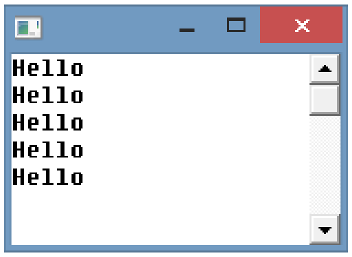
Example: Displaying the word “Hello” five times on the screen



```
for(i=0; i<5; i++)
{
printf("Hello");
}
```

```
#include <iostream.h>
void main( )
{  for ( int I = 1; I <=5; I++ )
        cout<< “Hello” <<endl; }
```

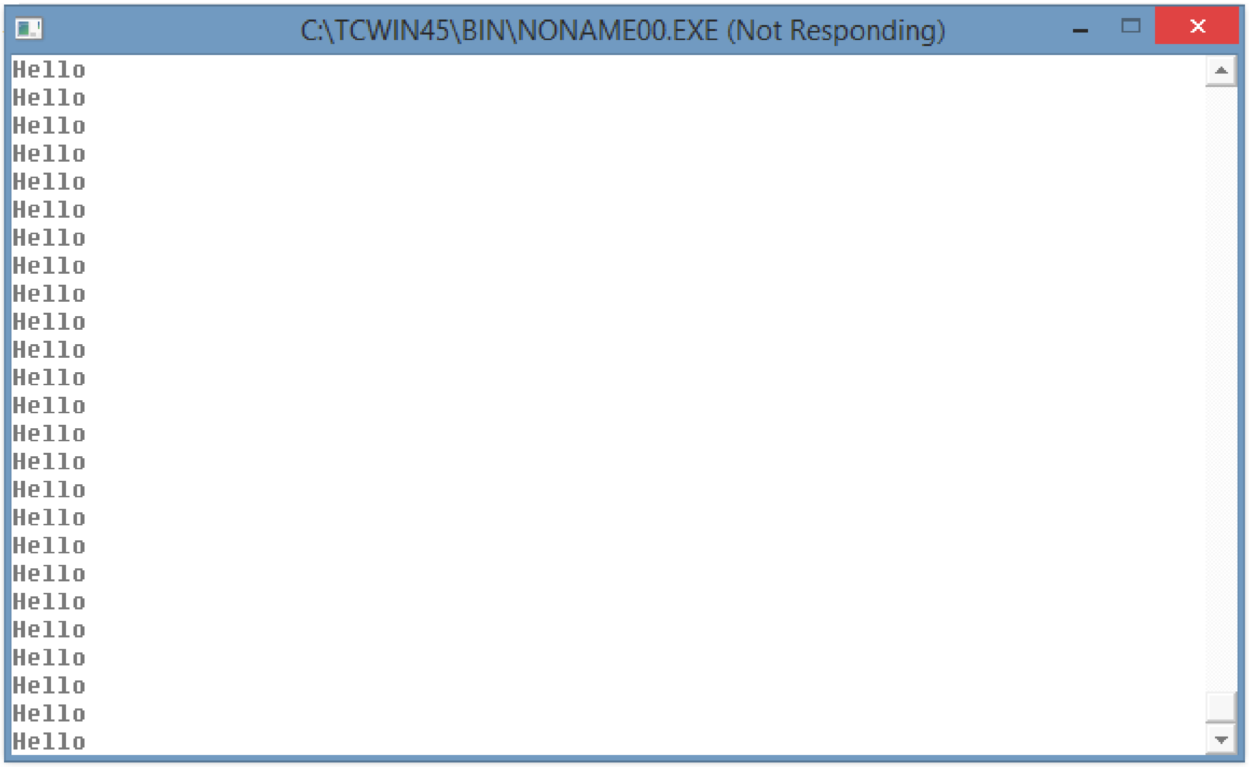
- When the above code is executed, it produces the following results:



```
Hello
Hello
Hello
Hello
Hello
```

☝ **Remember to increment the counter inside the loop (I++). If you forget this, the loop becomes infinite!**

```
for ( int I= 1; I <= 5;    )
{
        cout<< “Hello” <<endl;  // this will be displayed endlessly !
 }
```



```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

Exercise 18: Write a C++ program that displays the numbers from 1 to 20.

Exercise 19: Write a C++ program that displays the numbers from 100 to 1.

Exercise 20: Write a program that displays the squares of the numbers from 0 to 14. Here’s the output: 0 1 4 9 16 25 36 49 64 81 100 121 144 169 196

Exercise 21: Write a program that prints the cubes of the numbers from 1 to 10, using a two-column format. Here's the output:

```
1   1
2   8
3   27
4   64
5   125
6   216
7   343
8   512
9   729
10  1000
```

Exercise 22: Write a program that accepts 10 integers and calculate the Sum.

Exercise 23: Write a C++ program that accepts 5 grades and determine the average and the result (succeeded or failed).

Exercise 24: Write a C++ program that accepts N integers and count the number of even integers within N.

Exercise 25: Write a C++ program that accepts 10 numbers and determine the SUM of positive numbers and the SUM of negative numbers.

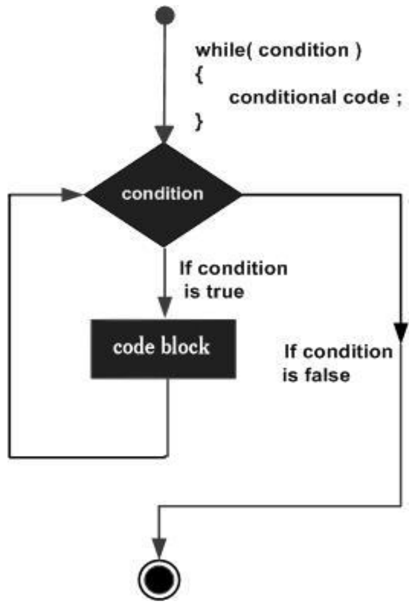
Exercise 26: Write a C++ program that accepts 10 numbers and determine the average of positive numbers and the average of negative numbers.

Exercise 27: Write a program that prints the values:

```
10
100
1000
10000
100000
1000000
10000000
100000000
1000000000
10000000000
```

4.1.2- The while loop:

The ***while*** loop simply repeats statement while expression is true. If, after any execution of statement, expression is no longer true, the loop ends, and the program continues right after the loop. The **syntax** of a while loop in C++ is shown in the figure:

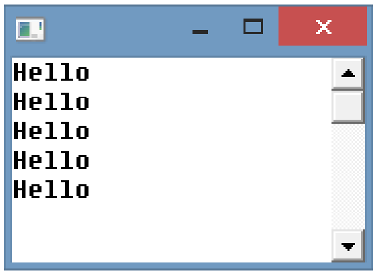


The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

- **Example: Displaying the word "Hello" five times on the screen using while statement.**

```
#include <iostream.h>
void main( )
{  int I=1;
  while ( I <= 5)
      {
         cout<< "Hello\n";
         I=I+1;  // I++
      }
  }
```

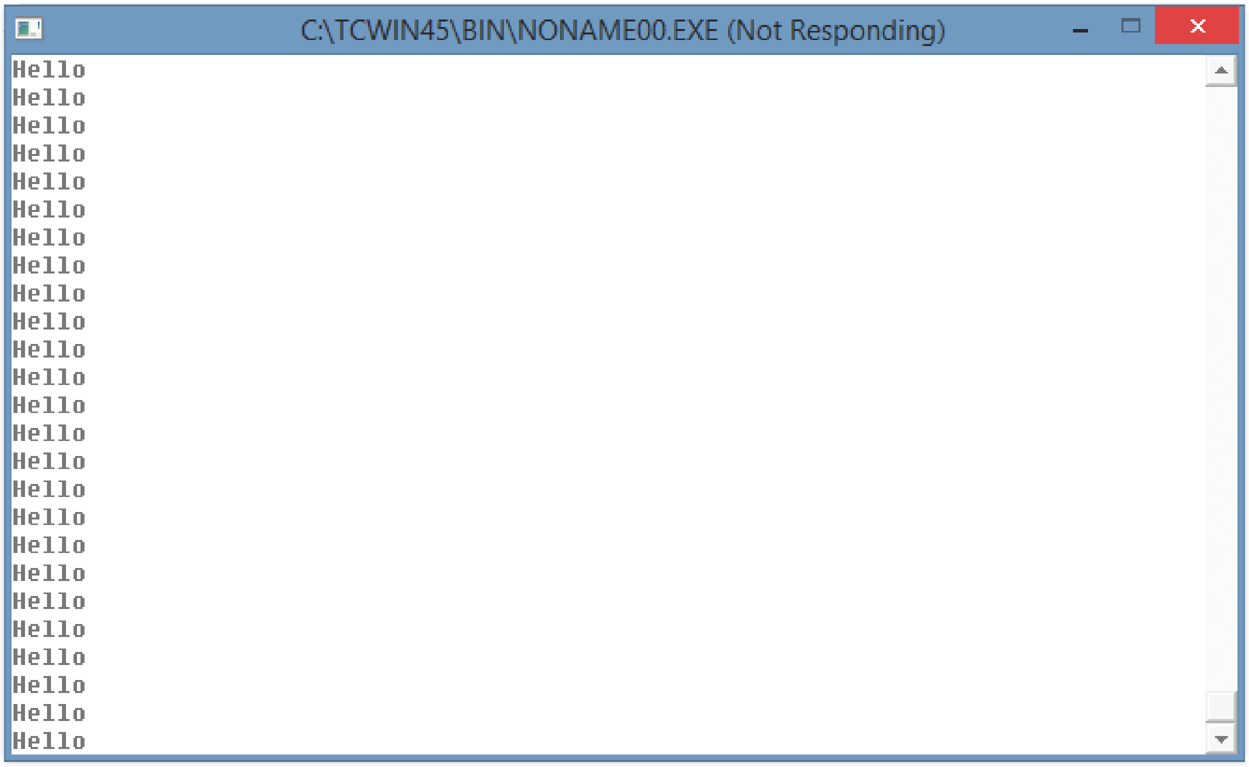
- When the above code is executed, it produces the following results:



```
Hello
Hello
Hello
Hello
Hello
```

☝ **Remember to increment the counter inside the loop (I++). If you forget this, the loop becomes infinite!**

```
while ( I <= 5)
  {
     cout<< "Hello\n";
  }
```



C:\TCWIN45\BIN\NONAME00.EXE (Not Responding)

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

Example: Displaying the integers from 10 to 19 using a while loop

```
#include <iostream>
using namespace std;
 int main ()
{
   int a = 10; // Local variable declaration:
   while ( a < 20 )
   {
       cout << "value of a: " << a << endl;
       a++;
   }
    return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

- **Exercise 28: Write a C++ program that displays the numbers from 1 to 10 using a while loop.**

- **Exercise 29: Write a C++ program that displays the numbers from 50 to 1 using a while loop.**

- **Exercise 30: Write a program that displays all even numbers between 2 and 50 using a while loop.**

- **Exercise 31: Write a C++ program that accepts 10 Letters and count the occurrence of letter ‘A’.**

- **Exercise 32: Using a while loop, write a program that reads N grades and calculate the average of negative grades.**

- **Exercise 33: Write a program that accepts 10 integers and calculate the Sum using a while loop.**

- **Exercise 34: Write a C++ program that accepts N integers and count the number of even integers within N (use a while loop).**

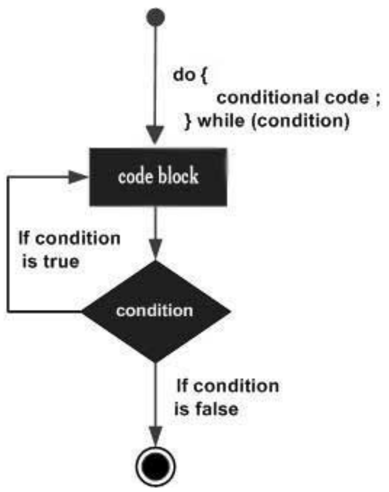
- **Exercise 35: Write a C++ program that accepts 10 numbers and determine the average of positive numbers and the average of negative numbers (use a while loop).**

- **Exercise 36: Write a C++ program that accepts 10 Letters and count the occurrence of letter ‘A’ and the occurrence of letter ‘C’ (use a while loop).**

- **Exercise 37: Write a C++ program that displays the numbers *a – z* using while loop. The variables *a* and *z* are determined by the user.**

4.1.3- The do .. while loop:

Unlike ***for*** and ***while*** loops, which test the loop condition at the top of the loop, the ***do...while*** loop checks its condition at the bottom of the loop. A ***do...while*** loop is guaranteed to execute at least one time, that's why it is mostly used to make control on the inputs. The syntax of a ***do...while*** loop in C++ is:

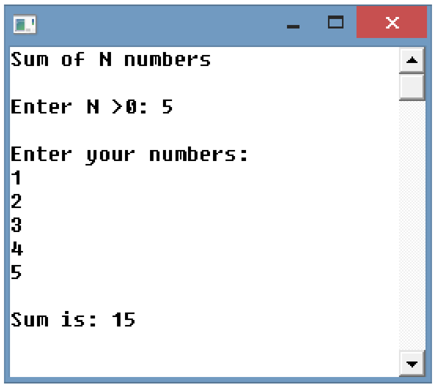


If the ***condition*** is true, the flow of control jumps back up to ***do***, and the statement(s) in the loop execute again. This process repeats until the given ***condition*** becomes false.

Exercise 38: Write a C++ program that accepts N integers and calculates the sum. N is a positive number.

```
#include <iostream.h>
void main ()
{  int N, x, I =1, s=0;
   cout<<"Sum of N numbers\n\n";
   cout<<"Enter N >0: ";
   do
  {
     cin>>N;
 }while( N <=0 );
 cout<<"\nEnter your numbers:\n";
 for (I=1; I<=N; I++)
 {
   cin>>x;
   s=s+x;
  }
cout<<"\nSum is: " <<s;
}
```

When the above code is compiled and executed, it produces the following result:



```
Sum of N numbers

Enter N >0: 5

Enter your numbers:
1
2
3
4
5

Sum is: 15
```

Exercise 39: **Write a C++ program that displays the numbers from 10 to 19 using a *do while* loop.**

```
#include <iostream.h>
void main ()
{
  int a = 10;
  do
  {
     cout << "value of a: " << a << endl;
     a = a + 1;
  }while( a < 20 );
 }
```

When the above code is compiled and executed, it produces the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Exercise 40: **Write a C++ program that takes in N integers and calculates the occurrence of digit 8. N>0 is given by the user.**

Exercise 41: **Write a C++ program that takes in a set of positive integers and calculates the average. The program stops when the user enter a negative number.**

Exercise 42: **Write a C++ program that takes in a positive number and calculates its factorial.**

N! = N * N-1 * N-2 … * 1 ; for example, 5!= 5 * 4 *3 *2 *1 = 120

The Nested loop:

A loop can be nested inside of another loop. C++ allows at least 256 levels of nesting.

The **syntax** for a **nested for loop** statement in C++ is as follows:

```
for ( init; condition; increment )
{
  for ( init; condition; increment )
  {
    statement(s);
  }
  statement(s);
}
```

The **syntax** for a **nested while loop** statement in C++ is as follows:

```
while (condition)
{
  while(condition)
  {
    statement(s);
  }
  statement(s); // you can put more statements.
}
```

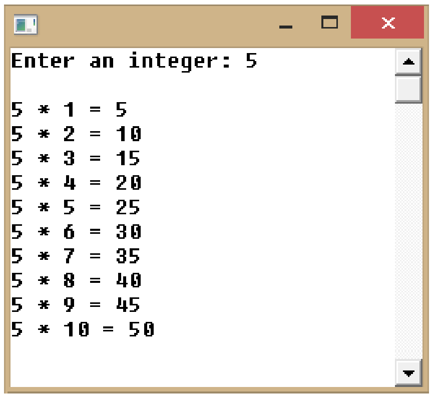
➾ The syntax for a **nested do...while loop** statement in C++ is as follows:

```
do
{
  statement(s); // you can put more statements.
  do
  {
    statement(s);
  }while( condition );

}while( condition );
```

Exercise 43: Write a C++ program that display the multiplication table (up to 10) of a given number:

```
#include <iostream.h>
int main() {
        int n;
        cout << "Enter an integer: ";
        cin >> n;
        cout<<endl;
        for (int i = 1; i <= 10; ++i) {
                cout << n << " * " << i << " = " << n*i << endl;
        }

        return 0;
}
```

Output:



```
Enter an integer: 5

5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

The Infinite Loop:

A loop becomes **infinite loop** if a condition never becomes false. The for loop is traditionally used for this purpose. Since none of the three expressions that form the for loop are required, you can make an endless loop by leaving the conditional expression empty.

```
#include <iostream.h>
void main ()
{
  for(   ;   ;   )
  {
    printf ("This loop will run forever.\n");
  }
}
```

```
#include <iostream.h>
void main ()
{
  for( int I=1   ;  I<10 ;    )
  {
    printf ("This loop will run forever.\n");
  }
}
```

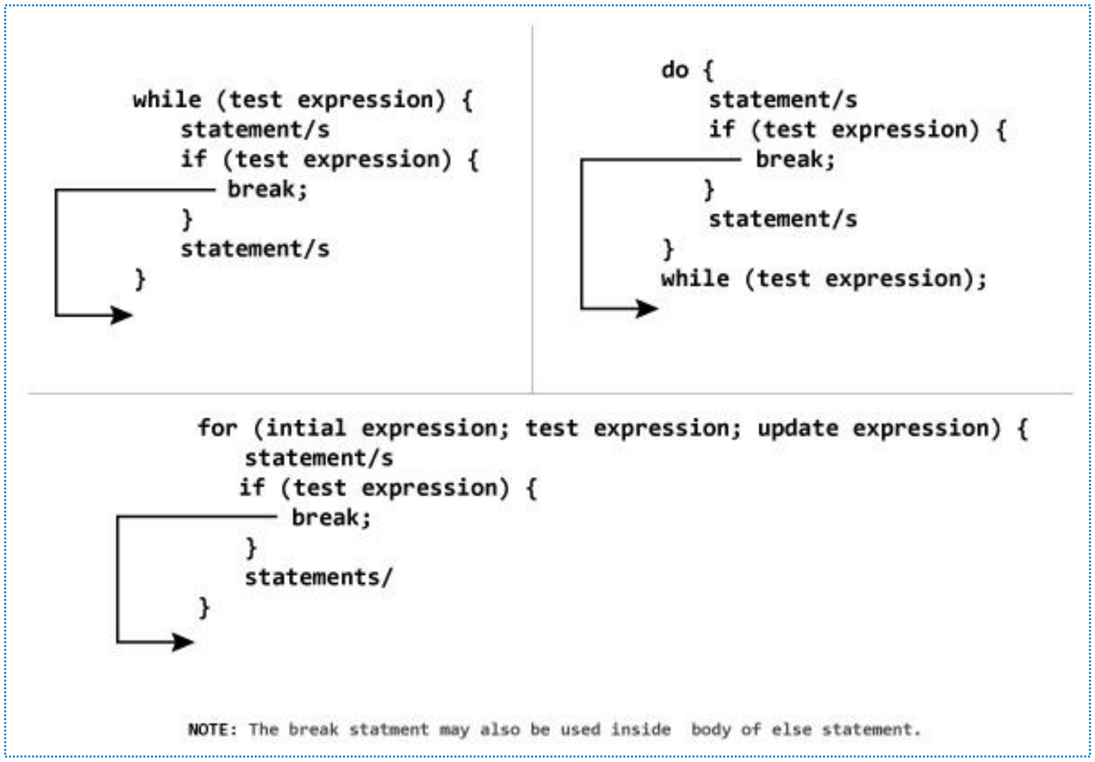
☝ **NOTE:** You can terminate an infinite loop by pressing Ctrl + C keys.

4.1.4- The Loop Control Statements:

Loop control statements are used to alter the normal flow of program such as **exiting** a loop or **bypassing** some iterations.

Control Statement	Description
break	Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or the switch.
continue	Causes the loop to skip the current iteration and move to the next directly.
goto	Transfers control to the labeled statement. Though it is not advised to use goto statement in programming anymore.

Example: How the 'break' Statement works?



```
while (test expression) {
    statement/s
    if (test expression) {
        break;
    }
    statement/s
}
```

```
do {
    statement/s
    if (test expression) {
        break;
    }
    statement/s
}
while (test expression);
```

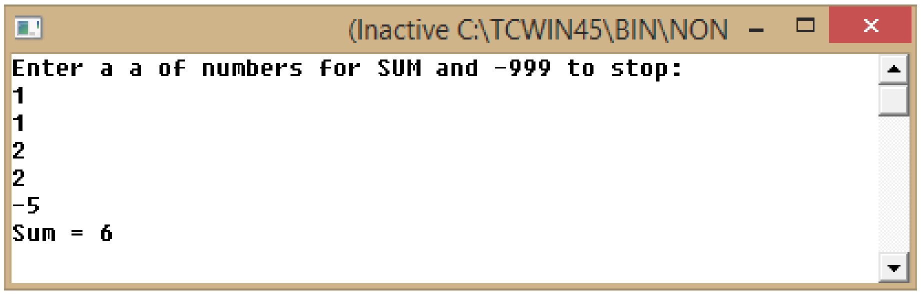
```
for (intial expression; test expression; update expression) {
    statement/s
    if (test expression) {
        break;
    }
    statements/
}
```

NOTE: The break statment may also be used inside body of else statement.

Exercise 44: Write a C++ program that takes in up to 10 numbers and calculate the SUM. Note that the program stops if the user enter any negative number.

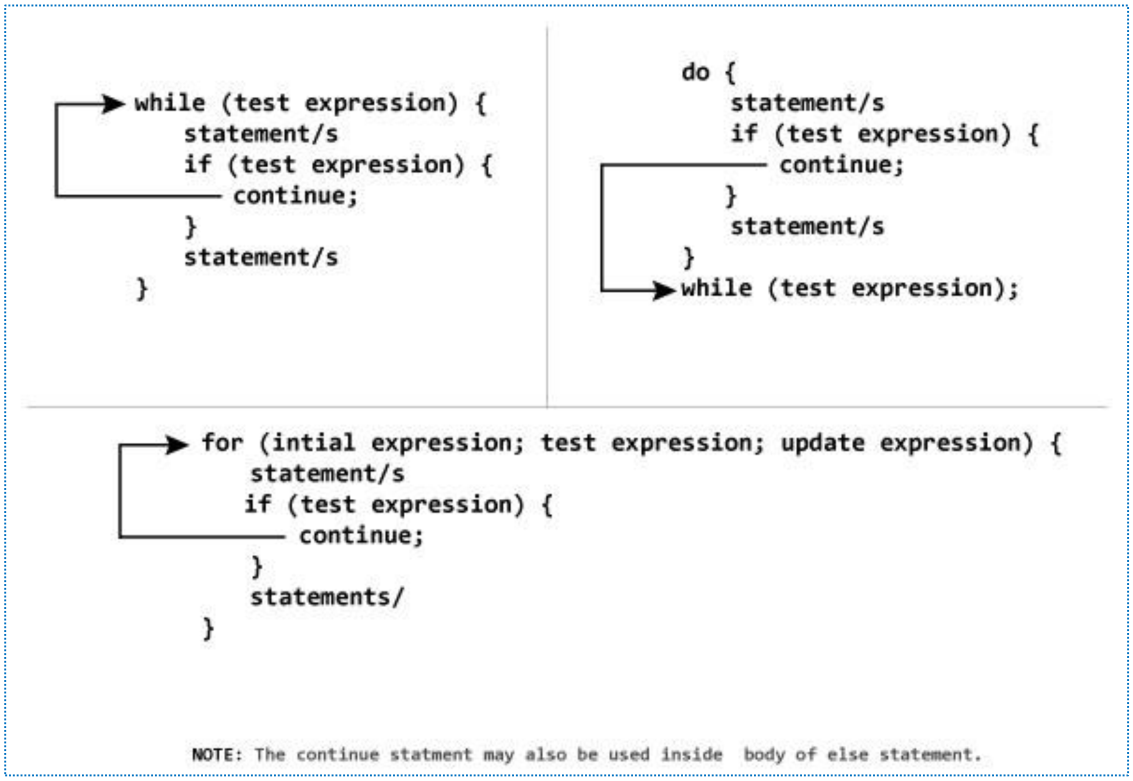
```
// C++ Program to demonstrate working of break statement

#include <iostream.h>
void main() {
        int n, s= 0,I;

        cout<<"Enter a set of 10 numbers for SUM and -999 to stop:\n";
        for (I=1; I<=10 ; I++)
                {
                cin>>n;
                if (n <0)
                        break;
                else
                        s= s +n;
                }

        cout<<"Sum = "<<s;
}
```

Output:



```
Enter a a of numbers for SUM and -999 to stop:
1
1
2
2
-5
Sum = 6
```

Example: How the 'continue' Statement works?



```
while (test expression) {
     statement/s
     if (test expression) {
          continue;
     }
     statement/s
}
```

```
do {
     statement/s
     if (test expression) {
          continue;
     }
     statement/s
}
while (test expression);
```

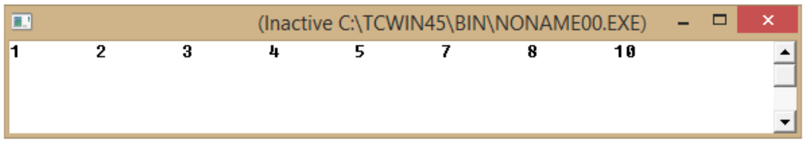
```
for (intial expression; test expression; update expression) {
     statement/s
    if (test expression) {
          continue;
     }
     statements/
}
```

NOTE: The continue statment may also be used inside body of else statement.

Exercise 45: Write a C++ program to display the numbers from 1 to 10 except 5 and 9.

```
// C++ Program to demonstrate working of continue statement
#include <iostream.h>
void main()
{
   for (int i = 1; i <= 10; ++i) {
      if ( i == 6 || i == 9) {
         continue;
      }
      cout<<i<<"\t";
   }
 }
```

Output:



(Inactive C:\TCWIN45\BIN\NONAME00.EXE)

```
1       2       3       4       5       7       8       10
```

Chapter 5

Practical Exercises

Objectives:

At the end of your first four chapters, it is time to solve more advanced programs. In this chapter you will practice C++ programming using all statements and concepts that you have learned in the previous chapters. Note that completing all exercises from each chapter, help solidify and expand your programming experience.

You should complete all the problems that you can, because every problem solved and every line of code makes you a better programmer. Don't worry about whether or not it took you 30 minutes or 6 hours, what's important is that you solved the problem. One last advice: even if you read a program code and it is 100% that you have understood that piece of code: Writing and compiling code using a computer is a completely different story!

46. Write a program that asks the user to type an integer and writes "YOU WIN" if the value is between 56 and 78 (both included). In the other case it writes "YOU LOSE".

47. Write a program that asks the user to type all the integers between 8 and 23 (both included) using a for loop.

48. Same exercise as 47 but you must use a while loop.

49. A program that determine the maximum of N numbers.

50. A program that determines the minimum of N numbers.

51. A program that will ask the user to input a set of positive numbers and calculates the SUM. The program will terminate when a negative nb is entered.

52. A program that will ask the user to input n positive numbers and counts the EVEN numbers only. The program will terminate when a negative number is entered.

53. Request the user to type numbers, each time printing its triple, until the user enters -999.

54. Write a C++ code segment using the while loop to find the sum of the even integers 2,4,6,8,...,500. Display the resulting sum. Be sure to give code to declare and initialize variables used.

55. Write a program that writes out the decimal equivalent of the reciprocals
1/2, 1/3, 1/4, ... , 1/19, 1/20.

56. Write a program that reads in 3 non-negative integers from the user (one at a time) and assigns the maximum value to a variable maxVal.

57. A program that determines a student's grade (A , B , C or D). The program will read three types of scores (quiz, mid-term, and final scores (score: between 0 and 100)) and determine the grade based on the following rules:

- if the average score >=90 ➔grade=A
- if the average score >= 70 and <90 ➔ grade=B
- if the average score >=50 and <70 ➔ grade=C
- if the average score <50 ➔ grade=F

58. Write a program to count the numbers which are divisible by 3 in a given Data Series.

59. A program that will ask the user to input three integer values from the keyboard. Then it will print the smallest and largest of those numbers.

60. A program that generate the results as shown below:

```
Results: =======Quizzes==========

Enter the score of the first quiz: 90
Enter the score of the second quiz: 75
Enter the score of the third quiz: 91

 =======Mid-term=============
Enter the score of the mid-term: 80

=======Final=================
Enter the score of the final: 89
Quiz Total :      256
Mid-term   :       80
Final      :       89
……………………
Total:            425
```

61. Write a C++ program that request the user to enter numbers, for each number printing its triple, until the user enters -999 to stop. At the end, display the sum of all input numbers.

62. A program that determine if a number is prime or not. A prime number can be divided only by 1 and itself. For example, 7 is prime because only 7%1➔0 and 7%7➔ 0

63. A program that determine if a number is perfect or not. A perfect number is equal to the sum of its divisors. For example, 6 is a perfect number because: 6= 1+2+3

64. A program that determine if a number is Armstrong or not. For example, 153 is Armstrong because: $153 = 1^3 + 5^3 + 3^3$

65. Write a C++ program that takes in a set of numbers, identifies and calculates the sum of prime numbers within the set.

66. A C++ program that will print the pattern as shown below (the number of lines L is defined by the user):

```
*****
*****
*****
*****
```

67. A C++ program that will print the pattern as shown below:

```
******
*****
****
***
**
*
```

68. A program that will output the multiplication table as show below:

```
1*1=1      2*1=2          3*1=3      ……   9*1=9
1*2=2      2*2=4          3*2=6      ……   9*2=18
…….         …….            …….       ……   ………
1*9=9      2*8=18         3*9=27     ……   9*9=81
```

69. A program that displays the following shape of stars (the number of lines L is defined by the user).

```
Triangle of Stars

Enter the number of Lines: 5

* * * * *
* * * *
* * *
* *
*
```

70. A C++ program that will print the pattern as shown below:

```
1******
12*****
123****
1234***
12345**
123456*
1234567
```

71. A program that displays a pyramid of stars. The number of lines of the pyramid is given by the user in a variable N.

```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

72. A program that accepts a 4-digit number x and count the number of 1s in x. Example: If x is 1122 ➔ nb of 1's is 2.

73. Write a C++ program that displays the following pyramid:

```
    1
   121
  12321
 1234321
123454321
```

The number of Lines of the pyramid (here L= 5) is given by the user.

74. A C++ program that will print the pattern as shown below (nb of lines: L must be even):

```
*
***
*****
*******
*********
*********
*******
*****
***
*
```

75. Write a program, using the loop 'while', that displays the N first even numbers of the series 2, 4, 6, 8, 10, 12,….N, at the rate of five numbers per line.

76. Write a program that accepts 5 grades of N students (0< N < 20). Determines and displays the average of each student, and the grade letter based on the following rules:

- Average score >=90➔ Grade Letter is: A
- Average score >= 70 and <90 ➔ Grade Letter is: B
- Average score>=50 and <70 ➔ Grade Letter is: C
- Average score<50➔ Grade Letter is: F

At the end, display the student who got the maximum average.

77. The country A has 50M inhabitants, and its population grows 3% per year. The country B, 70M and grows 2% per year. Write a C++ program that tell in how many years A will surpass B.

78. Currency conversion. Write a program Currency Converter that asks the user to enter today's exchange rate between U.S. dollars and the euro. Then the program reads U.S. dollar values and converts each to euro values. Stop when the user enters q.

    Here is a sample session:

```
How many euros is one dollar? 0.79447

Dollar value (q to quit): 100
100.00 dollar = 79.45 euro

Dollar value (q to quit): 20
20.00 dollar = 15.89 euro

Dollar value (q to quit): q
```

79. Print the pattern like shown below with loops, where the number of stars in first row rectangle is input by user. Example if number of stars are 11 in first rectangle:

```
*                          -                          ***********
***                         -                           *********
*****                        -                            *******
*******                       -                             *****
*********                      -                              ***
***********                     -                               *
```

80. Write a program that asks the user to type a positive integer. When the user types a negative value the program writes ERROR and asks for another value. When the user types 0, that means that the last value has been typed and the program must write the average of the positive integers. If the number of typed values is zero the program writes 'NO AVERAGE'.

81. Write a program in C++ language that displays the N first numbers of the series 2, 5, 8, 11, 14, 17,…. ,on consecutives lines, at the rate of five numbers per line.

82. Write a C++ program that will display the calculator menu. The program will prompt the user to choose the operation choice (from 1 to 5). Then it asks the user to input two integer vales for the calculation. See the sample below.

```
MENU
1. Add
2. Subtract
3. Multiply
4. Divide
5. Modulus

Enter your choice: 1
Enter your two numbers: 12   15
Result: 27

Continue?  y
```

The program also asks the user to decide whether he/she wants to continue the operation. If he/she input ‘y’, the program will prompt the user to choose the operation gain. Otherwise, the program will terminate.

83. Write a C++ program that asks the user to type an integer N and computes the sum of the cubes from 5^3 to N^3.

84. Write a C++ program that asks the user to type an integer N and that writes the number of prime numbers lesser or equal to N.

85. Write a C++ program to calculate the SUM of the following series:

$$SUM = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \ldots + \frac{1}{99} - \frac{1}{100}$$

86. Write a C++ program to calculate the SUM of the following series:

$$SUM = 1 - \frac{1}{2!} + \frac{1}{3!} - \frac{1}{4!} + \ldots + \frac{1}{99!} - \frac{1}{100!}$$

87. Write a C++ program to calculate the SUM of the following series:

$$1-1/2+1/3+ \; . \; . \; . \; +1/999$$

88. Write a C++ program to perform date arithmetic, such as how many days there are between 6/1/90 and 8/3/92.

89. Write a C++ program to determine the age of a person, in number of days.

90. Write a C++ program that read natural number n, determine the prime numbers p1 and p2 such that n= p1 + p2.

91. Write a C++ program to determine the twin prime numbers p1 and p2 immediately larger than the given non-null natural number n. Two prime numbers p and q are called twin if q-p = 2.

92. For a given natural number n find the minimal natural number m formed with the same digits. E.g. n=3658, m=3568.

93. For a given natural number n find the largest natural number written with the same digits. E.g. n=3658, m=8653.

94. A program that displays all Perfect numbers between 1 and 1000. A perfect number is a whole number, an integer greater than zero; and when you add up all of the factors less than that number, you get that number. Examples: The factors of 6 are 1, 2, 3 and 6 ➔ 1 + 2 + 3 = 6.

95. A program that displays all Prime numbers between 1 and 100. A Prime Number can be divided evenly only by 1 or itself.

96. A program that displays all Armstrong numbers. A number is Armstrong if the sum of cubes of individual digits of a number is equal to the number itself. For example 371 is an Armstrong number as 33 + 73 + 13 = 371. Some other Armstrong numbers are: 0, 1, 153, 370, 407.

97. A program that displays N Fibonacci numbers. In mathematics, the Fibonacci numbers or Fibonacci sequence are the numbers in the following integer sequence: 1 1 2 3 5 8 13 21 34 55 89 144 …

98. A program that checks whether a given number is palindrome or not. A palindromic number or numeral palindrome is a number that remains the same when its digits are reversed. Like 16461, for example, it is "symmetrical". Another example is 121.

99. The palindrome of a number is the number obtained by reversing the order of digits. E.g. palindrome (237) = 732). For a given natural number n, determine its palindrome.

100. Write a C++ program that displays a Floyd's triangle. The number of rows is entered by the user. First four rows of Floyd's triangle are as follows:

```
1
2 3
4 5 6
7 8 9 10
```

101. Write a C++ program that displays the following rectangle, where the number of lines and the number of columns should be given by two variables NL and NC. Example: The number of lines (here 5) and the number of columns (here 4) give the following shape:

```
+--+
|  |
|  |
|  |
+--+
```

102. Generate the smallest perfect number larger than a given integer number n. A number is perfect if it is equal to the sum of its divisors, except itself. E.g. 6 is a perfect number (6=1+2+3). Maximum integer value is 32767. If such a number does not exist, a message should be displayed.
PS: 6, 28, 496, and 8128 are perfect numbers.

Consider the following output:

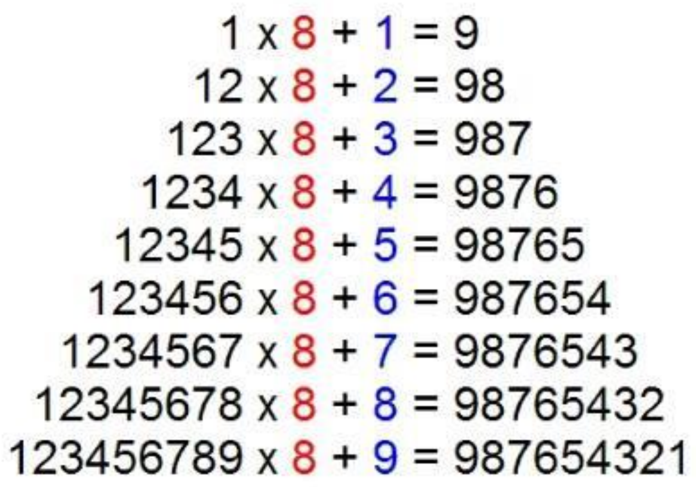
```
Enter n >0:   4
➔First perfect nb after 4 is: 6

Enter n>0: 1000
➔First perfect nb after 1000 is: 8128

Enter n>0: 9000
➔No perfect numbers after 9000
```

103. Generate the largest prime number **smaller** than a given integer number n. If such a number does not exist, a message should be displayed.

104. Write a C++ program to generate the first prime number **larger** than a given integer number n.

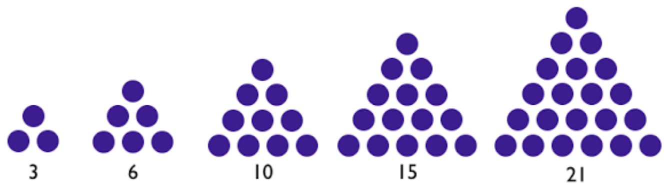
105. A serial transmission line can transmit 960 characters a second. Write a program that will calculate how long it will take to send a file, given the file's size. Try it on a 400MB (419,430,400 byte) file. Use appropriate units. (A 400MB file takes days.).

106. Write a C++ program that displays the following sequence. The number of lines L is given by the user.



```
        1 x 8 + 1 = 9
       12 x 8 + 2 = 98
      123 x 8 + 3 = 987
     1234 x 8 + 4 = 9876
    12345 x 8 + 5 = 98765
   123456 x 8 + 6 = 987654
  1234567 x 8 + 7 = 9876543
 12345678 x 8 + 8 = 98765432
123456789 x 8 + 9 = 987654321
```

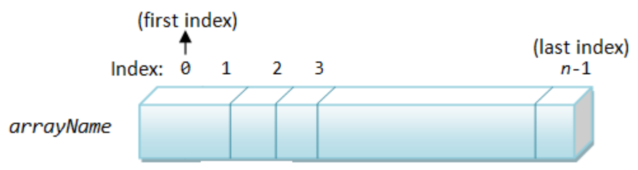
107. Write a C++ program that displays the following hollow triangles:

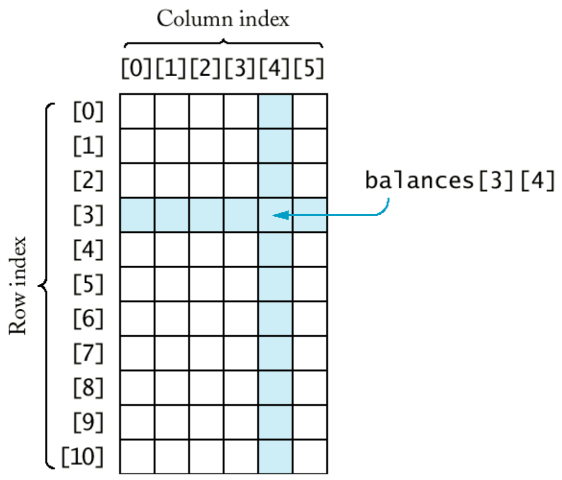


The sequence is 3, 6, 10, 15, 21, ... The first one, 3, is 1 + 2, the second one, 6, is 1 + 2 + 3, the third one, 10, is 1 + 2 + 3 + 4 and so on. For a triangle with a base of n blobs the number of blobs in the triangle is n∗(n+1)/2

Chapter 6

Arrays in C++





Objectives:

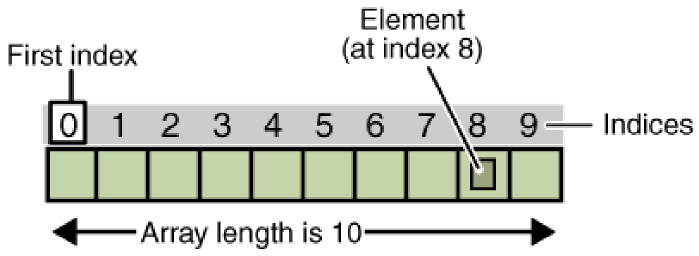
- Define and explain one dimensional and two dimensional arrays.
- Arrays declaration.
- Arrays Initialization.
- Use arrays to store, sort, and search values.
- Describe and implement basic sorting techniques.

Arrays in C++

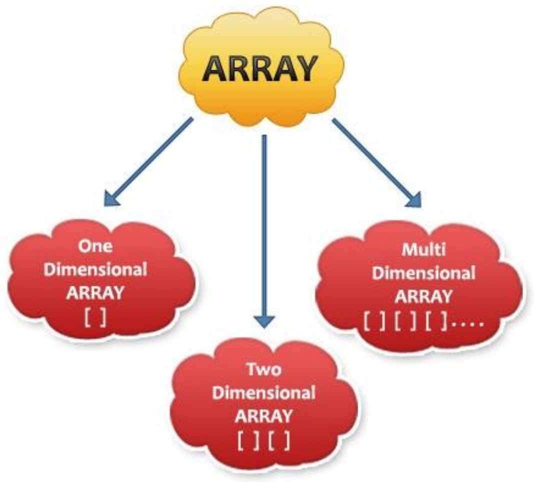
6.1- What Is an Array?

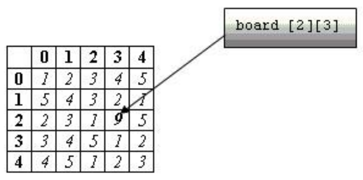
An **array** is a data structure that lets you store a fixed-size sequential collection of elements of the same type. You can think of an array as a collection of variables of the same type. Instead of declaring individual variables, such as A0, A1... and A9, you declare one array variable such as A[10] and use A[0], A[1], and ..., A[9] to represent individual variables.

A specific element in an array is accessed by an **index**. All arrays consist of connecting memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Arrays come in two types: **one dimensional** and **multi-dimensional** arrays. A multidimensional array has more than one subscript. A two-dimensional array has two subscripts; a three-dimensional array has three subscripts, and so on.


ARRAY
One Dimensional ARRAY [ ]
Two Dimensional ARRAY [ ] [ ]
Multi Dimensional ARRAY [ ] [ ] [ ]....



	0	1	2	3	4
0	1	2	3	4	5
1	5	4	3	2	1
2	2	3	1	9	5
3	3	4	5	1	2
4	4	5	1	2	3

6.2- Single or One-Dimensional Array:

A **single-dimensional array** has only a single subscript (index). This index identifies the number of elements in the array or the array size.

✍ Naming and Declaring Arrays:

data_type array_name [array_size];

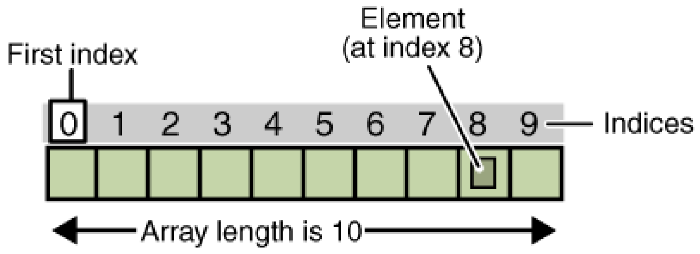
The number of elements in the array must be enclosed in square brackets immediately following the array name. Let's take some examples:

int a[10]; /* array of integers */

char s[100]; /* array of characters (strings) */

float f[20]; /* array of reals */

Array elements are always numbered starting at 0. Example: **int array[10];** The array is named array, and it contains 10 integer elements. The 10 elements are numbered 0 through 9.



The rules for assigning names to arrays are the same as for variable names; an array name must be unique. It can't be used for another array or for any other variable.

✍ Initializing Single-Dimensional Arrays:

You can initialize all or part of an array when you first declare it. Follow the array declaration with an equal sign and a list of values enclosed in braces and separated by commas. For example, the following code assigns the value 100 to array[0], 200 to array[1], 300 to array[2], and 400 to array[3] : **int array[4] = { 100, 200, 300, 400 };**

If you omit the array size, the compiler creates an array just large enough to hold the initialization values. Thus, the following statement would have exactly the same effect as the previous array declaration statement: **int array[ ] = { 100, 200, 300, 400 };**

Let's take other examples:

- **int a[100];**
- **a[2] = 9;**
- **a[2 + 3] = 100;** /* equivalent to: a[5]= 100; */
- **a[a[2]] = 100;** /* equivalent to: a[9]= 100; */

The easiest way to initialize an array can be done using a loop:

```
int a[5];
for(i=0;i<5;i++)
      cin >> a[i];
```

Processing the Elements of an Array:

Each **element** of the array, also called a **member** of the array, has a specific and constant **position**. The position of an item is also called its **index**.

The first member of the array, the most left, has an index of **0**. The second member of the array has an index of **1.** Since each array has a number of items which can be specified as **n**, the last member of the array has an index of **n-1**.

Example:

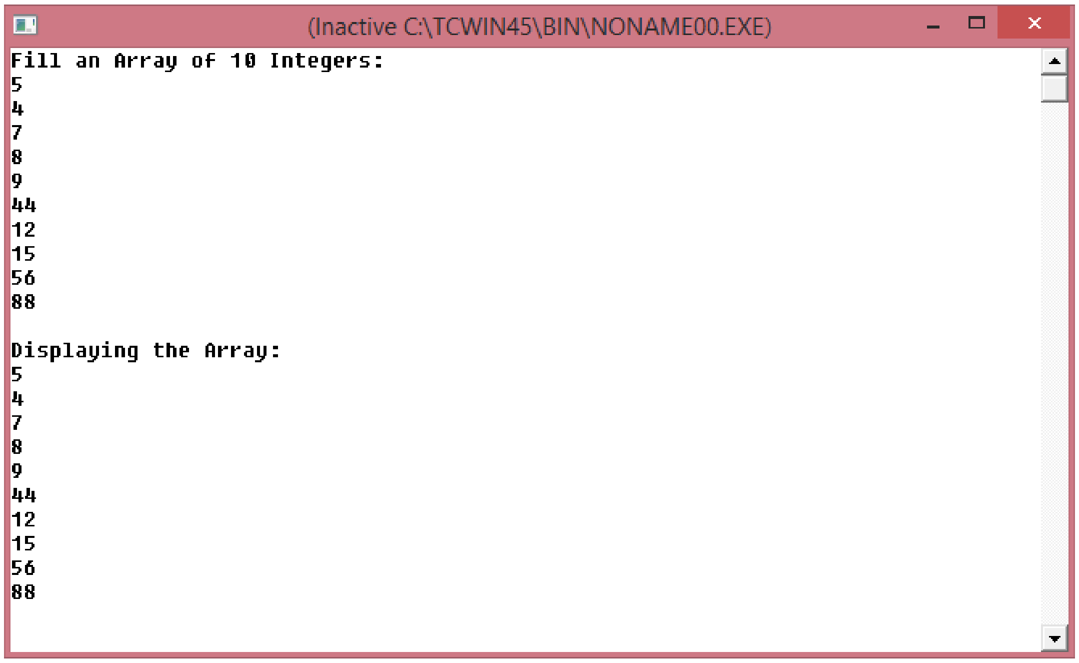
int D[ ] = {50, 40, 20, 3, 6};

To locate the value of the **3rd member** of the array, you would type **D[2]**. In the same way, the **1st member** of the array can be located with **D[0].**

Exercise 108: **Write a C++ program that creates, fills and displays an array of 10 integers.**

```
//Creates, Fills and Displays an array of 10 integers
#include<iostream.h>
int main()
{
  int a[10], i;
  cout<<"Fill an Array of 10 Integers:\n";
  for(i=0; i<=9; i++)
       cin>>a[i];

  cout<<"\nDisplaying the Array:\n";
  for(i=0; i<=9; i++)
      cout<< a[i] <<endl;
  return 0;
}
```

This would produce:

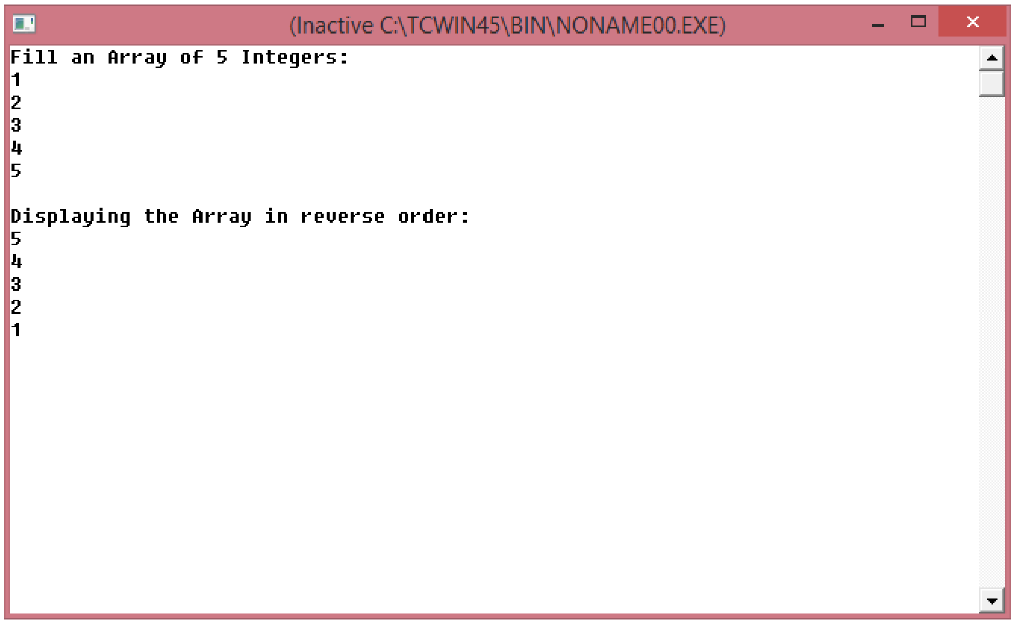


```
Fill an Array of 10 Integers:
5
4
7
8
9
44
12
15
56
88

Displaying the Array:
5
4
7
8
9
44
12
15
56
88
```

Exercise 109: Write a C++ program that creates, fills an array of 5 integers, and displays the array in reverse order:

```
//Creates, Fills and Displays an array of 5 integers in reverse order

#include<iostream.h>
int main()
{
  int a[5], i;
  cout<<"Fill an Array of 5 Integers:\n";
  for(i=0; i<=4; i++)
      cin>>a[i];

  cout<<"\nDisplaying the Array in reverse order:\n";
  for(i=4; i>=0; i--)
     cout<< a[i] <<endl;
  return 0;
}
```

This would produce:



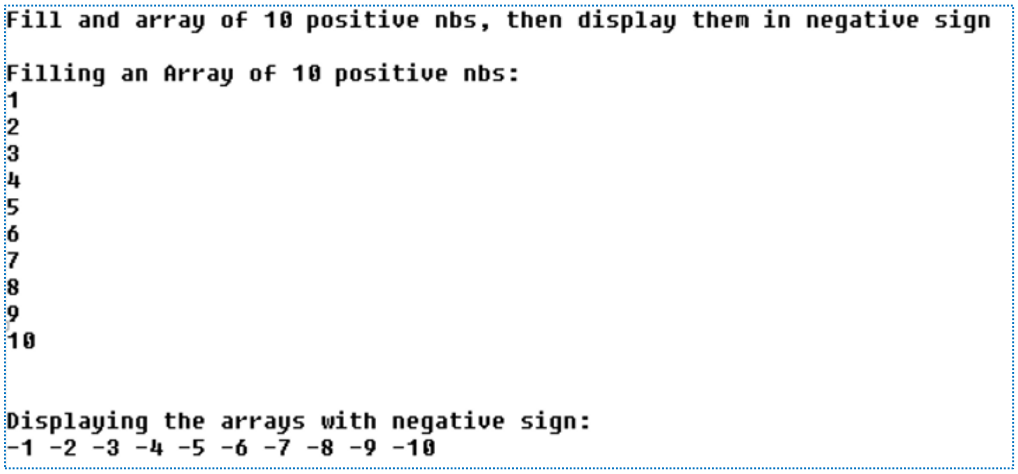
```
Fill an Array of 5 Integers:
1
2
3
4
5

Displaying the Array in reverse order:
5
4
3
2
1
```

Exercise 110: **Write a C++ program to store 5 numbers entered by user in an array and display first and last number only.**

Exercise 111: **Write a C++ program to create an Array A[ ], of N integers (N max 20) and calculate the average of its numbers.**

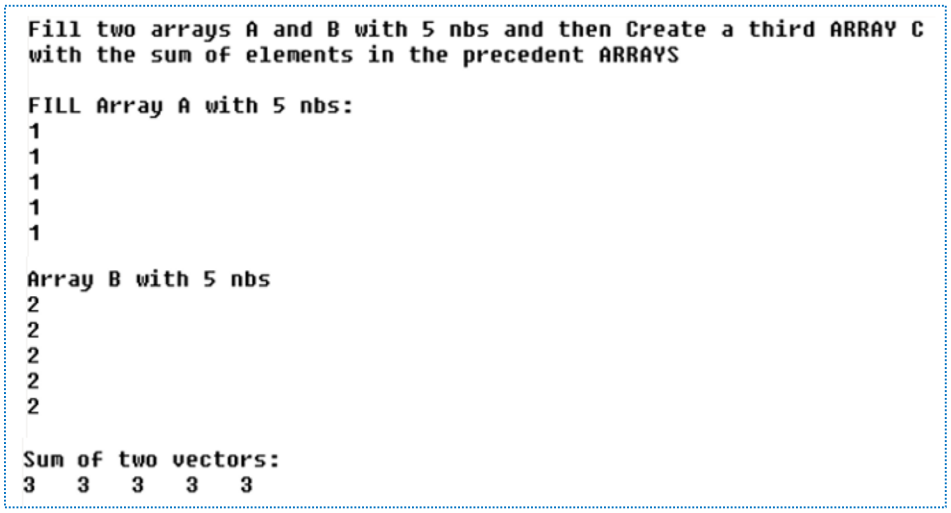
Exercise 112: **Write a C++ program which reads 5 numbers into an array and prints out the value of each number which has a value less than 10.**

Exercise 113: **Write a program to produce the following:**



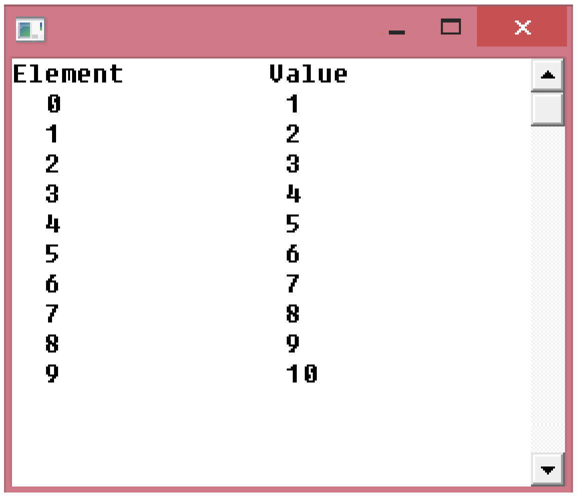
```
Fill and array of 10 positive nbs, then display them in negative sign

Filling an Array of 10 positive nbs:
1
2
3
4
5
6
7
8
9
10


Displaying the arrays with negative sign:
-1 -2 -3 -4 -5 -6 -7 -8 -9 -10
```

Exercise 114: **Write a program to produce the following:**



```
Fill two arrays A and B with 5 nbs and then Create a third ARRAY C
with the sum of elements in the precedent ARRAYS

FILL Array A with 5 nbs:
1
1
1
1
1

Array B with 5 nbs
2
2
2
2
2

Sum of two vectors:
3    3    3    3    3
```

Exercise 115: Write a C++ program that creates, fills an array of 10 integers, and displays the array in the following format:



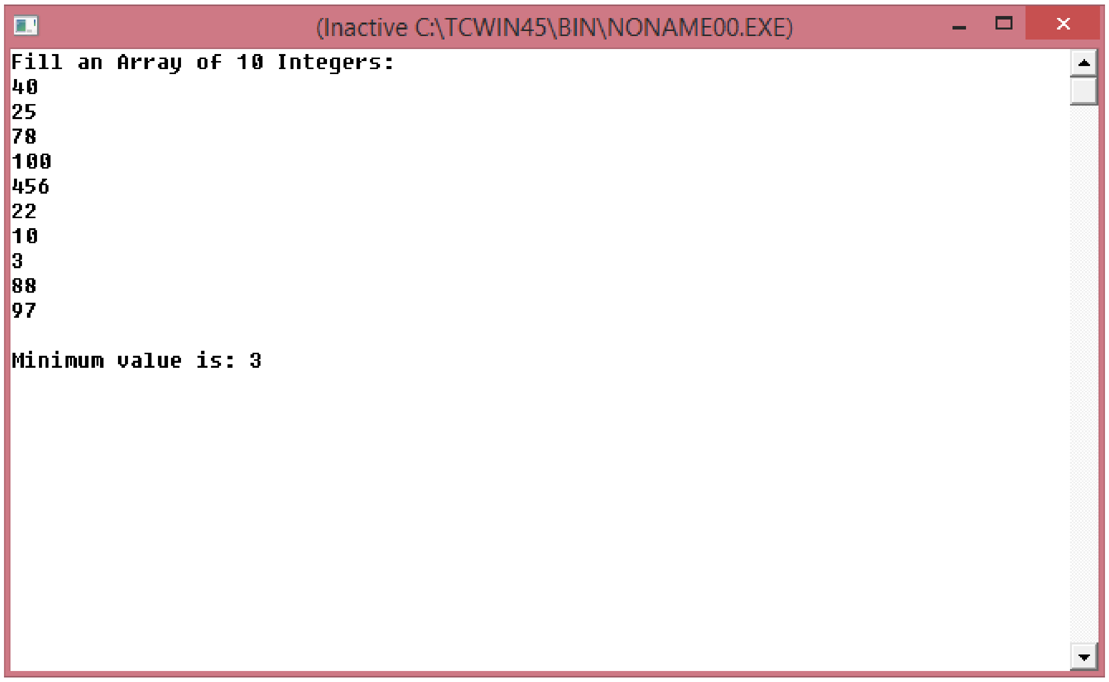
```
Element          Value
  0                1
  1                2
  2                3
  3                4
  4                5
  5                6
  6                7
  7                8
  8                9
  9                10
```

```
#include <iostream.h>
int main ()
{
        // A is an array of 10 integers
        int A[10];

        // initialize elements of array
        for (int i = 0; i < 10; i++) {
                A[i] = i + 1;
        }

        cout << "Element \tValue" << endl;

        for (i = 0; i < 10; i++) {
                cout << "  "<<i << "\t \t " << A[i] << endl;
        }

        return 0;
}
```

Exercise 116: Write a C++ program that creates, fills an array of 10 integers, and find the minimum:

```
#include<iostream.h>
int main()
{
	int a[10], i, min;
	cout<<"Fill an Array of 10 Integers:\n";
	for(i=0; i<=9; i++)
		cin>>a[i];

	//Finding th minimum
	min=a[0]; // initial value is equal to the first element in the array
	for(i=1; i<=9; i++)
	  if (a[i]< min)
		min = a[i];

	cout<<"\nMinimum value is: " <<min;
	return 0;
}
```

This would produce:



```
Fill an Array of 10 Integers:
40
25
78
100
456
22
10
3
88
97

Minimum value is: 3
```

- **Exercise 117:** Write a C++ program that creates, fills an array of 10 integers, and find the maximum.

- **Exercise 118:** Write a C++ program that creates, fills an array of 10 integers, and check if the digit 5 exists or not in the array.

- **Exercise 119:** Write a C++ program that creates, fills an array of 10 integers, and calculates the occurrence of digit 7.

- **Exercise 120:** Write a program to create an Array A[ ], of N integers (N max 50) and search if a given number x is found in A[ ] with its position.

- **Exercise 121:** Write a program that prompts the user to input 5 grades in an array grades [ ]. Then, prints the average score. (Note that the maximum grade is 100). Use the #define directive.

- **Exercise 122:** Write a C++ program that will prompt the user to input ten integer values. The program will display the smallest and greatest of those values.

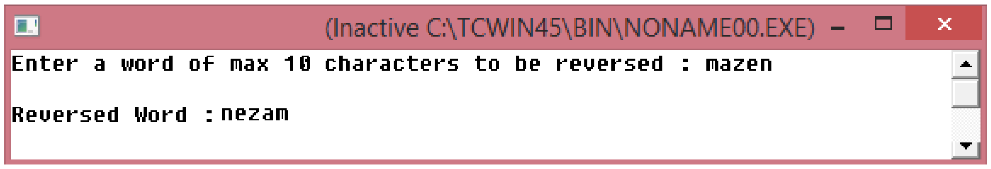
- **Exercise 123:** Write a C++ program which takes 2 arrays of 10 integers each, A[10] and B[10]. C[20] is an array with 20 integers. The program should put into C[20] the elements of A[10] and B[10], the first 10 integers of C from array A, the latter 10 from B. Then the program should display C[20].

- **Exercise 124:** Write a program to create an Array A[10] containing integers (positive and negative). Then, split this array into two arrays P[ ] containing the positive numbers, and N[ ]containing the negative numbers.

- **Exercise 125:** Write a program to create an Array V[10] containing integers. Then, split this array into two arrays O[ ] containing the Odd numbers, and E[]containing the Even numbers.

- **Exercise 126:** Write a program which accepts a set of N numbers and save only the prime numbers into an array and then displays out the array values. (N is max 100).

Exercise 127: Write a C++ program that reads a word (called in programming: a string of characters) and reverses that word and prints it on the screen as the following:



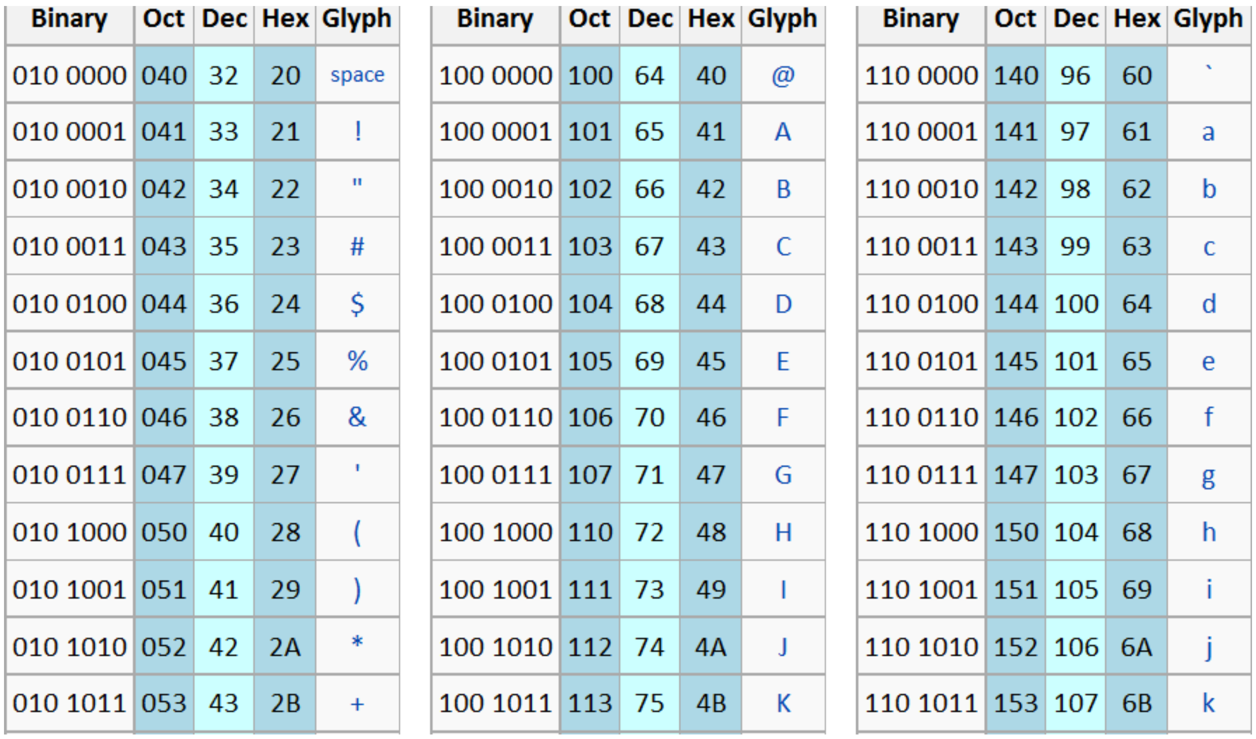
```
Enter a word of max 10 characters to be reversed : mazen

Reversed Word :nezam
```

Exercise 128: Write a C++ program that asks the user to type 10 integers of an array and an integer value V. The program must search if the value V exists in the array and must remove the first occurrence of V, shifting each following element left and adding a zero at the end of the array. The program must then write the final array.

Exercise 129: Write a C++ program that asks the user to type 10 integers of an array and an integer value V and an index value i between 0 and 9. The program must put the value V at the place i in the array, shifting each element right and dropping off the last element. The program must then write the final array.

Sorting Arrays:

Array’s sorting is the process of putting data in order either numerically or alphabetically. It is often necessary to arrange the elements in an array in numerical order from highest to lowest values (descending order) or vice versa (ascending order). If the array contains string values, alphabetical order may be needed (which is actually ascending order using ASCII values).



Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph	Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	space	100 0000	100	64	40	@	110 0000	140	96	60	`
010 0001	041	33	21	!	100 0001	101	65	41	A	110 0001	141	97	61	a
010 0010	042	34	22	"	100 0010	102	66	42	B	110 0010	142	98	62	b
010 0011	043	35	23	#	100 0011	103	67	43	C	110 0011	143	99	63	c
010 0100	044	36	24	$	100 0100	104	68	44	D	110 0100	144	100	64	d
010 0101	045	37	25	%	100 0101	105	69	45	E	110 0101	145	101	65	e
010 0110	046	38	26	&	100 0110	106	70	46	F	110 0110	146	102	66	f
010 0111	047	39	27	'	100 0111	107	71	47	G	110 0111	147	103	67	g
010 1000	050	40	28	(	100 1000	110	72	48	H	110 1000	150	104	68	h
010 1001	051	41	29	)	100 1001	111	73	49	I	110 1001	151	105	69	i
010 1010	052	42	2A	*	100 1010	112	74	4A	J	110 1010	152	106	6A	j
010 1011	053	43	2B	+	100 1011	113	75	4B	K	110 1011	153	107	6B	k

The process of sorting an array requires exchanging of values. In order to swap two values, you must use a third variable, (temp), to temporarily hold the value you do not want to lose:

```
//swapping a[1] and a [2]
temp = a[1];     // holding variable
a[1] = a[2];
a[2] = temp;
```

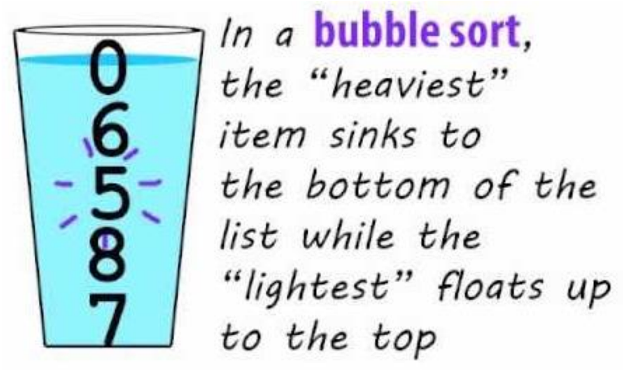
Ways to sort arrays:

There are different ways or methods to sort arrays. The basic goal of each of these methods is the same: to compare each array element to another array element and swap them if they are in the wrong position.

1. **Bubble Sort // simplest and most used**
2. Exchange Sort
3. Selection Sort
4. Insertion Sort
5. Shell Sort
6. Quick Sort
7. Merge Sort
8. …

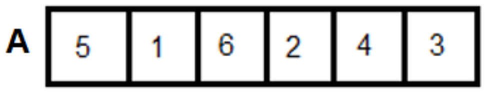
Bubble sort:

In the bubble sort, as elements are sorted they gradually "bubble" or "rise up" to their proper location in the array. The bubble sort repeatedly compares adjacent elements of an array. The first and second elements are compared and swapped if out of order. Then the second and third elements are compared and swapped if out of order and so on until the last two elements of the array are compared and swapped if out of order.


0
6
5
8
7
In a bubble sort, the "heaviest" item sinks to the bottom of the list while the "lightest" floats up to the top

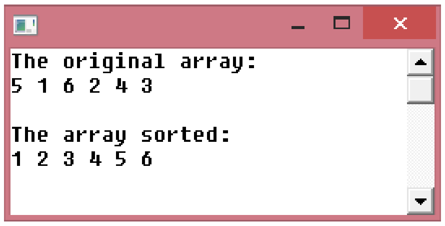
When the first pass through the array is complete, the bubble sort returns to elements one and two and starts the process all over again until the entire array is examined and no "swaps" are needed.

Exercise 130: **Write a C++ program to sort the following array in ascending order (using the Bubble sort) and display it before and after sorting.**



```
//Sorting an array in ascending order using the Bubble sort
#include<iostream.h>
void main()
{
  int A[6]={5,1,6,2,4,3}, i;
  cout<<"The original array:\n";
  for(i=0; i<6; i++)
     cout<<A[i]<<" ";

 //Sorting the array in ascending order
  int  I, J, temp;

  for (I=0; I <6; I++)
     for (J=I+1; J<6; J++)
                          if (A[I] > A[J])
                          {
                                                   temp =A[I];
                                                   A[I] = A[J];
                                                   A[J] = temp;
                          }

  //Displaying the array sorted
  cout<<"\n\nThe array sorted in ascending order:\n";
  for(i=0; i<6; i++)
     cout<<A[i]<<" ";

}
```

This would produce:



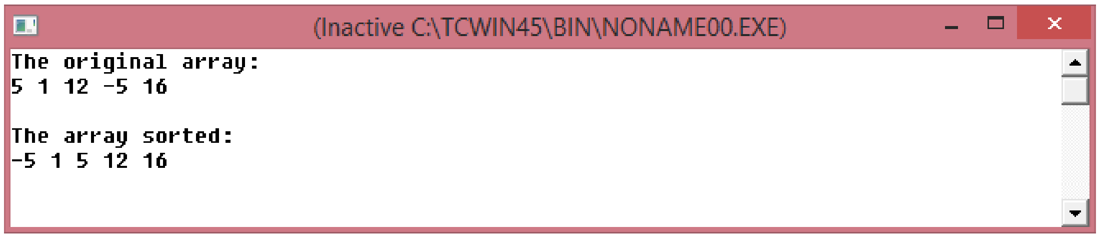
```
The original array:
5 1 6 2 4 3

The array sorted:
1 2 3 4 5 6
```

Exercise 131: **Write a C++ program to sort the following array in ascending order (and display it before and after sorting.**

$\boxed{5}\ \boxed{1}\ \boxed{12}\ \boxed{-5}\ \boxed{16}$ unsorted

```
//Sorting an array in ascending order using the Bubble sort
#include<iostream.h>
#define n 5
int main(){
  int A[n]={5,1,12,-5,16}, i, J, temp;
  cout<<"The original array:\n";
  for(i=0; i<n; i++)
     cout<<A[i]<<" ";

 //Sorting the array in ascending order
     for (i=0; i <n; i++)
     for (J=i+1; J<n ; J++)
                 if (A[i] > A[ J]){
                   temp =A[i];
                  A[i] = A[J];
                 A[J] = temp;}

  //Displaying the array sorted
  cout<<"\n\nThe array sorted:\n";
  for(i=0; i<n; i++)
     cout<<A[i]<<" ";

return 0;}
```

This would produce:



```
The original array:
5 1 12 -5 16

The array sorted:
-5 1 5 12 16
```

Exercise 132: Write a C++ program that create and fill an array of n numbers_(n<=30)_ and sort the array in descending order (using the Bubble sort).

Sorting an array in Descending order using the Bubble sort

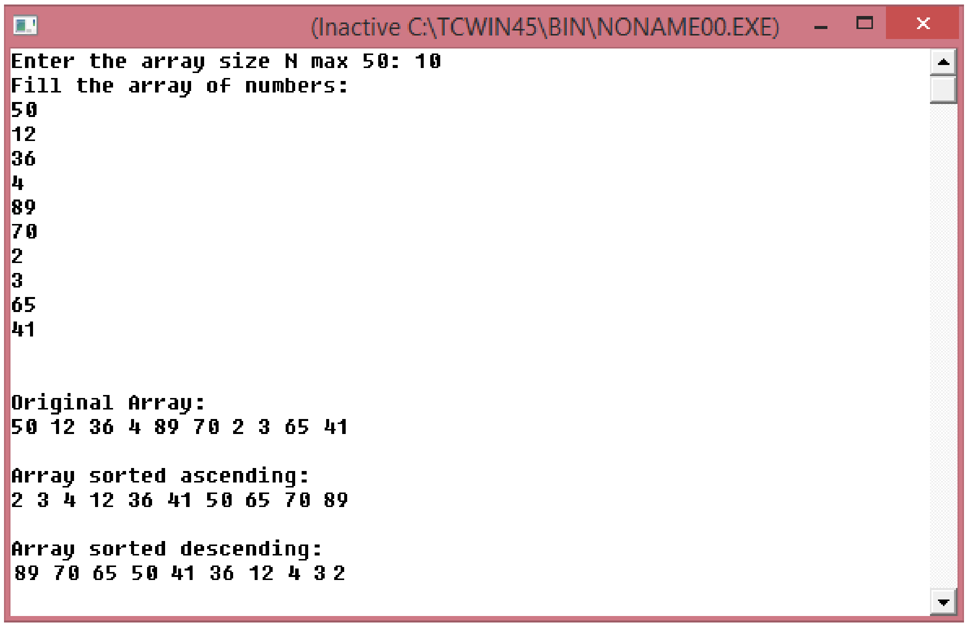
```
#include <iostream.h>
int main ()
{
int A[30];
int n, i, j, temp;

cout<<"Enter the value of  n<=30: \n";
cin>>n;
cout<<"\nEnter the numbers:\n";
for (i = 0; i < n; ++i)
 cin>>A[i];

//sorting
for (i = 0; i < n; ++i)
 {
  for (j = i + 1; j < n; ++j)
          {
              if (A[i] < A[j])
                     {
                            temp = A[i];
                            A[i] = A[j];
                             A[j] = temp;
                       }
            }
  }

        for (i = 0; i < n; ++i)
                  cout<< A[i] <<" ";

return 0;}
```

Exercise 133: Write a C++ program to solve the following:

- **Create and fill an array A[ ] of N integers. N (positive) is maximum 50.**
- **Sort the array A[ ] in ascending order and display it.**
- **Sort the array A[ ] in descending order and display it.**

```
 //Sorting Method 2– Exercise 133
#include <iostream.h>
main( ){
int a[50], N, i, pass, J , temp;
cout<< "Enter the array size N max 50: ";
do{
cin>>N;
}while (N <=0 || N>50);

cout<<"Fill the array of numbers:\n";
for(i = 0; i <N; i++)
  cin>>a[i];

 cout<<"\n\nOriginal Array:\n";
  for(i = 0; i <N; i++)
            cout<<a[i]<<" ";

//sorting in ascending order
for (pass=0; pass <N; pass++)
for (J=0; J<N - pass; J++)    //method 2
  if (a[J] > a[ J+1]){
    temp =a[J];
    a[J] = a[J+1];
   a[J+1] = temp;}


 cout<<"\n\nArray sorted ascending:\n";
  for(i = 0; i <N; i++)
    cout<<a[i]<<" ";

//sorting in descending order
for (pass=0; pass <N; pass++)
 for (J=0; J<N - 1- pass; J++)    //method 2
  if (a[J] < a[ J+1]){
     temp =a[J];
     a[J] = a[J+1];
     a[J+1] = temp;}

  cout<<"\n\nArray sorted descending:\n";
  for(i = 0; i <N; i++)
  cout<<a[i]<<" ";
 }
```

This would produce:



```
Enter the array size N max 50: 10
Fill the array of numbers:
50
12
36
4
89
70
2
3
65
41


Original Array:
50 12 36 4 89 70 2 3 65 41

Array sorted ascending:
2 3 4 12 36 41 50 65 70 89

Array sorted descending:
 89 70 65 50 41 36 12 4 3 2
```

Exercise 134: Write a program that asks the user to enter 10 integers between 0 and 20 that will be stored in an array V[10]. Then, display the occurrence of 0, 1 , 2 …20.

Exercise 135: Write a C++ program that:

1) **Creates and fills an Array of integers V[20].**
2) **Display the array V[ ] in reverse order.**
3) **Calculate the occurrence of a given number x in the array V[ ] ➔ (How many times x is found in the array V).**
4) **Sort the array V[ ] in ascending order.**
5) **Split the array V [ ] into two arrays O[ ] containing the Odd numbers, and E[]containing the Even numbers.**
6) **Display the arrays O[] and E[].**

Exercise 136: **Write a C++ program that:**

1. **Creates the array VECTA of N elements containing positive integers or null numbers.**

2. **Fills the array VECTA by numbers between 0 and 255 (include 0 and 255).**

3. **Creates from the values of array VECTA the array VECTB containing in each of its cases the sum of two successive cases of VECTA. Only the first value of VECTB corresponds to the first value of VECTA.**

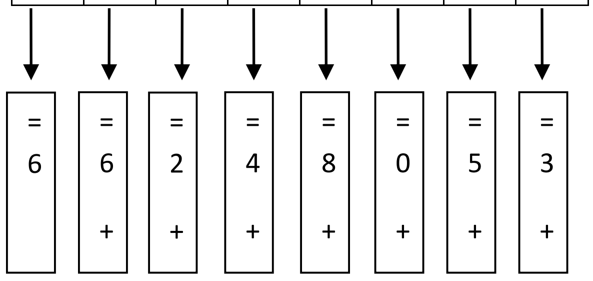
   Example:

   Array VECTA

6	2	4	8	0	5	3	9

   Array VECTB

6	8	6	12	8	5	8	12



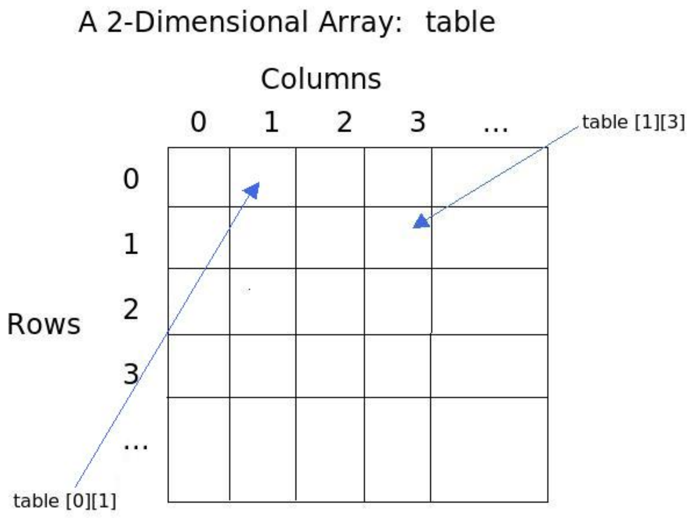
4. **Displays the contents of array VECTB.**

5. **Sorts the array VECTB in ascending order.**

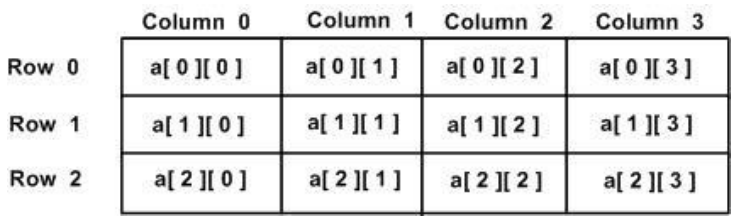
6. **Displays the contents of array VECTB sorted.**

6.3- Two-Dimensional Arrays:

A two-dimensional array is a simple form of multidimensional arrays that is used to organize data in a table (matrix) which will have "**I**" number of **rows** and "**J**" number of **columns**: e.g. **int table[I][J];**



As with one-dimensional arrays, two-dimensional arrays can contain data of any type, although each entry in a table must have the same type. Also, the size of a two-dimensional array must be declared when it is first defined. For example: **int A[3][4]** is a 2-dimensional array, that contains three rows and four columns as shown below:



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Initializing Two-Dimensional Arrays:

A **two dimensional array** may be initialized as follow:

- $\mathbf{int\ a[3][4] = \{}\underbrace{\mathbf{0,1,2,3}}_{\text{row 0}},\ \underbrace{\mathbf{4,5,6,7}}_{\text{row 1}},\ \underbrace{\mathbf{8,9,10,11}}_{\text{row 2}}\mathbf{\}}$

Also, **two dimensional arrays** may be initialized by specifying bracketed values for each row. The following is an array with 3 rows and each row has 4 columns:

```
int A [3][4] = {
 {0, 1, 2, 3},        // row 0
 {4, 5, 6, 7},       // row 1
 {8, 9, 10, 11}    // row 2
};
```

Filling and Processing a Two-Dimensional Array:

An element in a two-dimensional array is accessed by using the subscripts (row index and column index) of the array. For example:

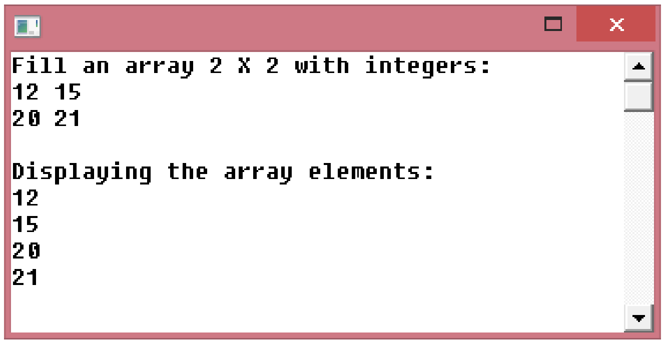
x = a[2][3];

- You cannot skip the index size in 2-D arrays:
  int A[ ][ ] ;   int A[3][ ] ; int A[ ][3] :   are invalid declarations.

Example: Filling and displaying out a 2D-Array:

```
#include <iostream.h>
void main ()
{
  int A[2][2] ; /* declaring an array with 2 rows and 2 columns*/
  int i, j;
  cout<<"Fill an array 2 X 2 with integers:\n";
   for ( i = 0; i < 2; i++ )
        for ( j = 0; j < 2; j++ )
              cin>>A[i][j];

  /* output each array element's value */
  cout<<"\nDisplaying the array elements:\n";
  for ( i = 0; i < 2; i++ )
  {
    for ( j = 0; j < 2; j++ )
              cout <<A[i][j] <<endl;
  }
}
```

- When the above code is executed, it produces the following results:



```
Fill an array 2 X 2 with integers:
12 15
20 21

Displaying the array elements:
12
15
20
21
```

Exercise 137: **Write a C++ program that create a two-dimensional array A[4][4]. Fill the array by 16 numbers and displays it line per line.**

Exercise 138: **Write a program that fills a 2-Dimensional array of size 3 by 3, and displays the sum of all the elements.**

Exercise 139: **Write a program that fills a 2-Dimensional array of size 3 by 3. Then displays it line by line with the sum of elements at the end of each line.**

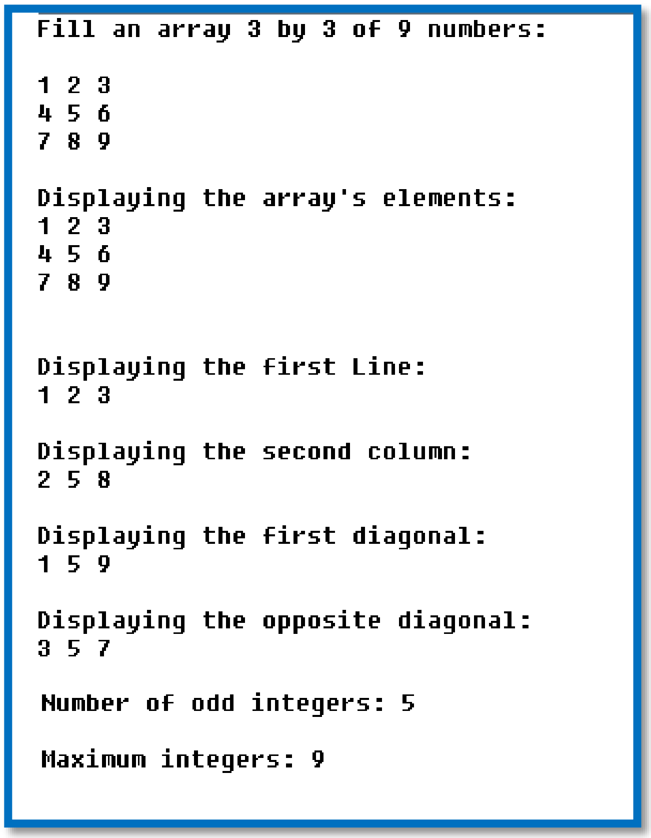
```
Fill an Array 3 by 3 of 9 numbers:
1 1 1
2 2 2
3 3 3


Array with lines Sum:
1 1 1 3
2 2 2 6
3 3 3 9
```

Exercise 140: **Write a C++ program to fill and display a table of numbers as shown below:**

1	**2**	**3**	**4**	**5**
6	**7**	**8**	**9**	**10**
11	**12**	**13**	**14**	**15**
16	**17**	**18**	**19**	**20**
21	**22**	**23**	**24**	**25**

Exercise 141: Write a program that performs the following:

1. Create a two dim array of size 3 by 3.
2. Fill the array.
3. Display the array's elements.
4. Display the elements of the first row.
5. Display the elements of the second column.
6. Display the first diagonal elements.
7. Display the opposite diagonal elements.
8. Calculate the SUM of array's elements.
9. Determine the number of ODD values in the array.
10. Determine the maximum value.
11. Determine the minimum value.



```
Fill an array 3 by 3 of 9 numbers:

1 2 3
4 5 6
7 8 9

Displaying the array's elements:
1 2 3
4 5 6
7 8 9


Displaying the first Line:
1 2 3

Displaying the second column:
2 5 8

Displaying the first diagonal:
1 5 9

Displaying the opposite diagonal:
3 5 7

 Number of odd integers: 5

 Maximum integers: 9
```

Exercise 142: Write a program to display a matrix as shown below. The diagonal of the matrix fills with 0. The lower side fills will -1s and the upper side fills with 1s.

```
0     1     1     1     1

-1    0     1     1     1

-1    -1    0     1     1

-1    -1    -1    0     1

-1    -1    -1    -1    0
```

Exercise 143: Write a program for addition of two matrices. Suppose two matrices A and B having same order 3×3.

```
Matrix A:            Matrix B:
4   6   5            9   2   3
1   3   1            8   0   6
2   3   7            4   1   2

Resultant Matrix is addition of matrices A and B:
13   8   8
 9   3   7
 6   4   9
```

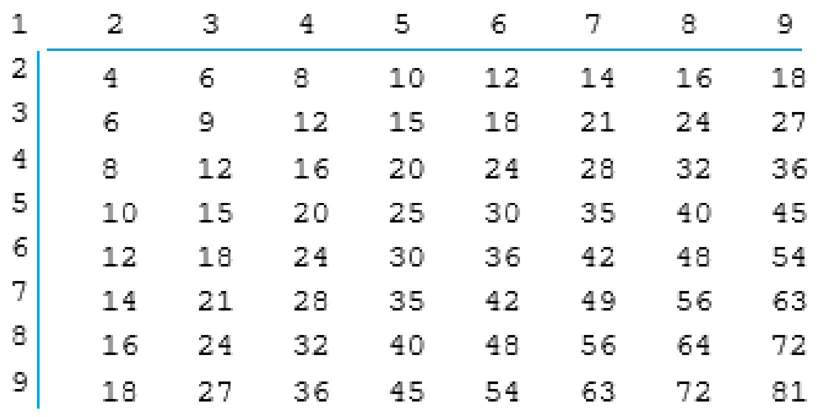
Exercise 144: Transpose of matrix is that interchanging rows and columns of matrix. In transpose, order of square matrix remains same. But order of a matrix in m x n order changes after transpose to order n x m. Suppose matrices have order 3×3:

```
                 Matrix A:
                 1   2   3
                 4   5   6
                 7   8   9

      Then transpose of Matrix A will be:
                 1   4   7
                 2   5   8
                 3   6   9
```

Exercise 145: **Write a program to solve the following:**

- Demands to enter NL the number of lines and NC the number of columns of a two-dimensional array **COURSE [20][20]** containing integer numbers.
- Fills the array **COURSE [ ][ ]** by numbers between 0 and 20 (including 0 and 20) only.
- At the end, determines and display the array with the maximum grade of each line .

Exercise 146: **Write a C++ program to calculate and prints a multiplication table for all values between 1 and 9 (inclusive). Note that when printing the table, the for loops start from 1 instead of 0. Here is the output:**



```
1    2    3    4    5    6    7    8    9
   --------------------------------------------
2 |  4    6    8    10   12   14   16   18
3 |  6    9    12   15   18   21   24   27
4 |  8    12   16   20   24   28   32   36
5 |  10   15   20   25   30   35   40   45
6 |  12   18   24   30   36   42   48   54
7 |  14   21   28   35   42   49   56   63
8 |  16   24   32   40   48   56   64   72
9 |  18   27   36   45   54   63   72   81
```

Exercise 147: **Write a program that solves the following problem:**

1. **Takes in the number of students** N. **(**N is positive and maximum 20**).**
2. **Read 5 grades of each student in a two dimensional array named** Grades [student][grades]**. The grades are over 20.**
3. **For each student, display the student number, its grades, average, and its average_letter.**

Average	Average Letter
18 - 20	A
15- 17	B
12- 14	C
10 - 11	D
Less than 10	F

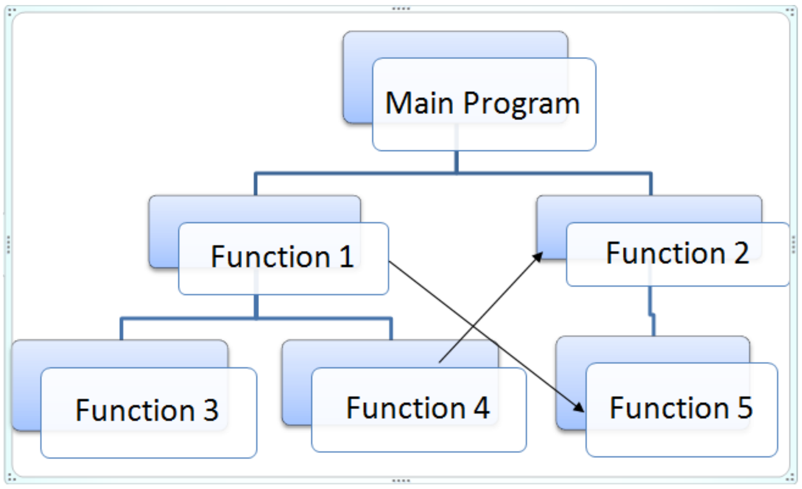
4. **At the end, display the Student of maximum average with its average.**

Exercise 148: **Write a C++ program to display a table that represents a Pascal triangle of any size. In Pascal triangle, the first and the second rows are set to 1. Each element of the triangle (from the third row downward) is the sum of the element directly above it and the element to the left of the element directly above it. See the example Pascal triangle (size=5) below:**

```
1
1          1
1          2          1
1          3          3          1
1          4          6          4          1
```

Chapter 7

Functions in C++



Objectives:

- Purpose of using functions.
- Predefined functions.
- User defined functions: Function definition, function prototype, function header, function name, function type, parameter list/arguments, function body, local variables, function calling, return..
- References and reference parameters: pass-by-reference versus pass-by-value.
- Passing Arrays to functions.
- Random number generation: rand( ) & srand( ).

Functions in C++

7.1- What is a Function?

Functions allow to structure programs in segments of code to perform individual tasks which simplify the programming process. Technically, a **Function** is an independent section of a Program that performs a specific task such as displaying a text message, sorting an array in a specific order, or calculating the factorial of a number… and optionally **returns a value** to the calling program.

- **Two types of functions:**

- void functions: do not have a data type
- value-returning functions: have a data type

Some Functions are **predefined** such as **pow( ) and sqrt( )** that belong to the **<math.h> library.** Other functions are **user defined functions** that you create while writing your programs.

7.2- Using a Predefined Function:

Some of the predefined mathematical functions are:

- sqrt(x)
- pow(x,y)

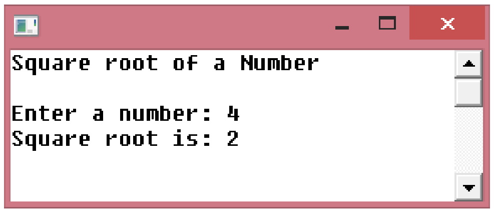
Predefined functions are organized into separate libraries:

- Math functions are in <cmath> header
- I/O functions are in <iostream> header
- File functions are <fstream> header
- string functions are in <string> header
- …

Exercise 149: **Write a program to determine the square root of a given number using the sqrt( ) function.**

```
#include <iostream.h>
#include <math.h>
void main()
{
  int x, s ;
  cout<<"Square root of a Number\n\n";
  cout<<"Enter a number: ";
  cin>> x;
  s= sqrt(x);
  cout<<"Square root is: " <<s;   // cout<<sqrt (x);
  }
```

The output is:



```
Square root of a Number

Enter a number: 4
Square root is: 2
```

Exercise 150: **Write a program to calculate X Power Y using the pow(X , Y) function.**

```
#include <iostream.h>
#include <math.h>
void main()
{
  int x, y, p ;
  cout<<"Enter two numbers of the form: X power Y:\n";
  cin>> x >> y;
  p= pow(x, y);
  cout<< p;
  }
```

7.3- Writing Functions:

The common syntax to define a function is:

```
Type   Name (Parameter1, Parameter2...)
{
  Statement(s);
}
```

Where:

- **Type:** is the type of the **value returned** by the function. It can be: **int, float, double , char** … or **void** which means that the function has no return value.

- **Name:** is the **function name** by which the function can be called. A function name should be unique.

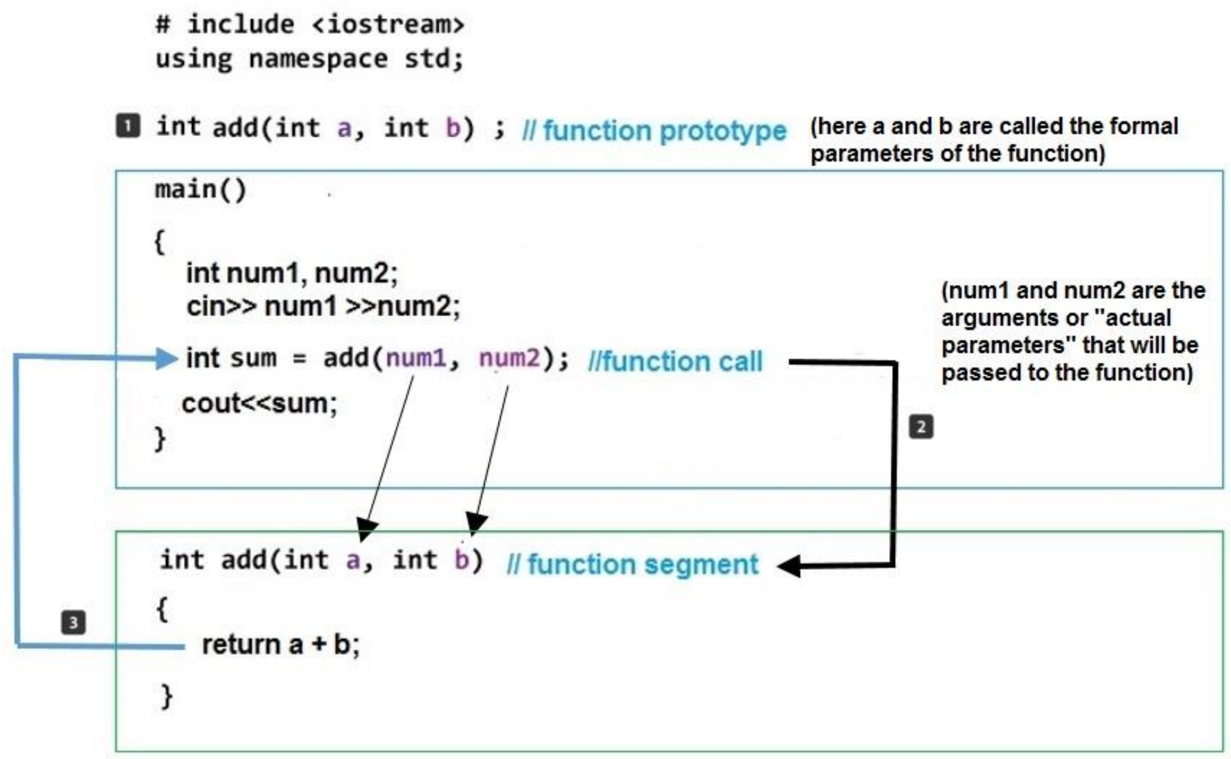
- **Parameters:** The purpose of **parameters** is to allow passing **arguments** to the function from the location where it is called from.

  Each parameter consists of a type followed by an identifier, with each parameter being separated from the next by a comma. Each parameter looks very much like a regular variable declaration (for example: **int x**), and acts within the function as a local variable of the function.

- **Statements:** It is a block of statements surrounded by braces **{ }** that specify what the function actually does.

- When the function's statements have finished, execution passes back to the same location in the program that called the function.

Example:



```
# include <iostream>
using namespace std;

int add(int a, int b) ; // function prototype

main()
{
   int num1, num2;
   cin>> num1 >>num2;
   int sum = add(num1, num2); //function call
   cout<<sum;
}

int add(int a, int b) // function segment
{
   return a + b;
}
```

(here a and b are called the formal parameters of the function)

(num1 and num2 are the arguments or "actual parameters" that will be passed to the function)

In the example above:

- The program is divided in two functions: **add( )** and **main( )**. Remember that a C++ program always starts by calling the **main function**.

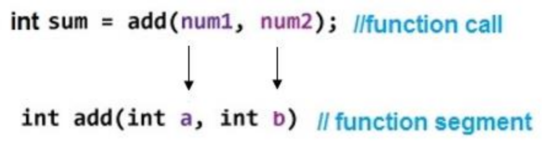
1. **Function Prototype:** A function prototype provides the compiler with a description about a function that will be defined at a later point in the program. The prototype includes:

- A **return type** indicating the type of variable that the function will return.
- The **function name**.
- The **types of the arguments** that will be passed to the function.
- Optionally, it can contain the names of the variables that will be passed.
- A function prototype should always end with a semicolon (**;**).

Examples:

✓ int max (int a , int b , int c);
✓ float average (int x , int y , int z);
✓ double cube (double number);
✓ void print_report (int report_number);

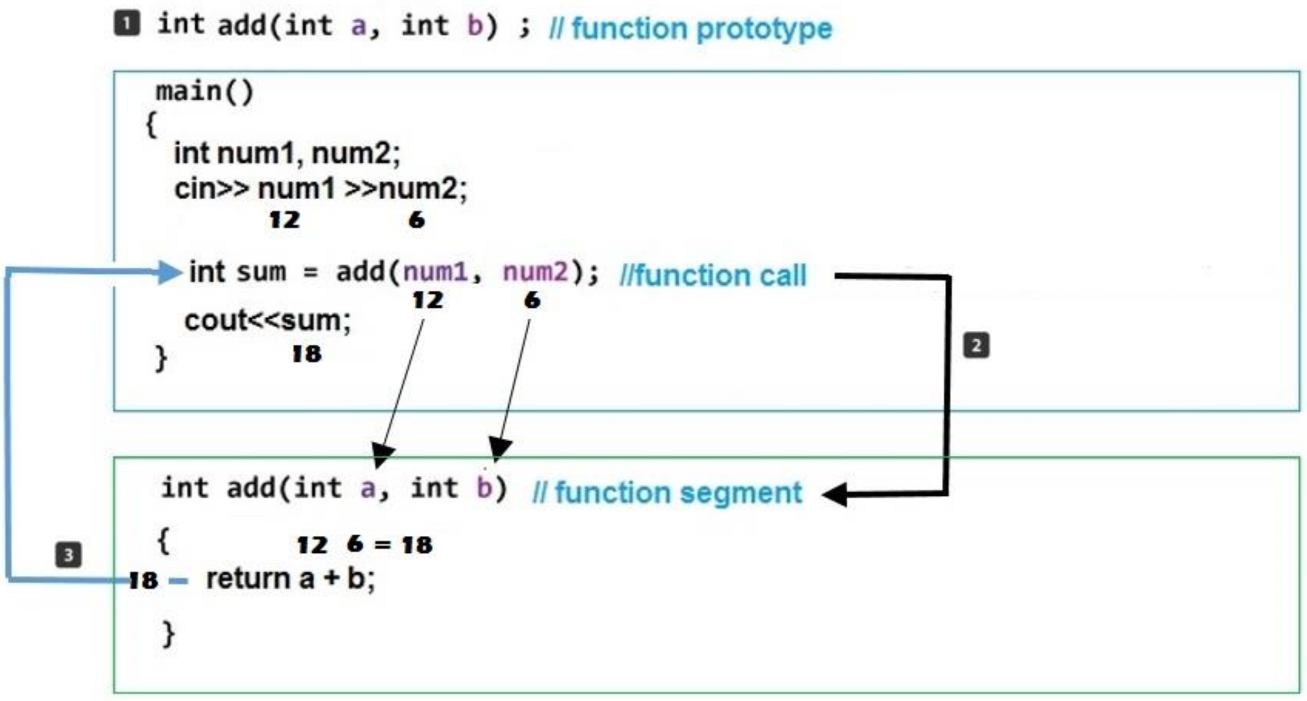
2. **Function call:** in main the **arguments** (variables called **actual parameters**) **num1** and **num2** of type int are declared, and right after that, the first function **add( )** is called with passing the arguments to the function **parameters (**called **formal parameters a and b)**. The call to a function follows a structure very similar to its declaration:



int sum = add(num1, num2); //function call
int add(int a, int b) // function segment

- At the point at which the function is called from within main, the control is passed to function **add( ).** So the execution of **main**( ) is stopped, and will only resume once the **add( )** function ends.

- At the moment of the function call, the value of both **arguments num1** and **num2** are copied to the local variables (**parameters**) **int a** and **int b** within the function. Inside the function **add( )**, the result of **(a + b)** is returned to the variable **sum** that called the function **add( )**. Look how it works actually:

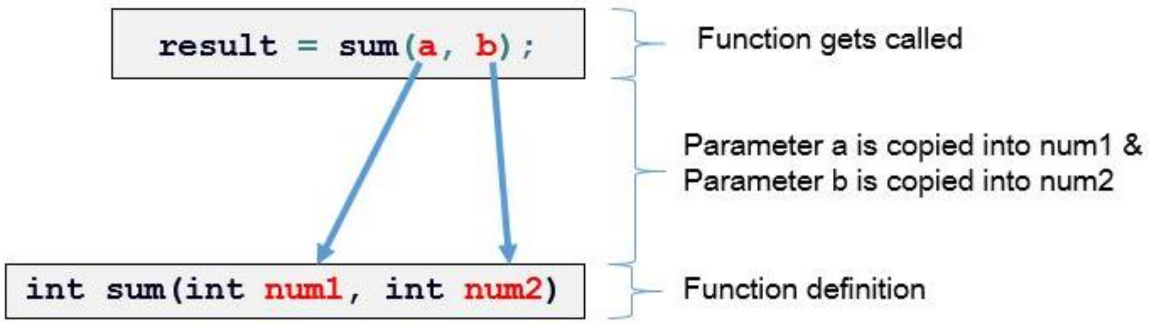


```
int add(int a, int b) ; // function prototype

main()
{
  int num1, num2;
  cin>> num1 >>num2;
  int sum = add(num1, num2); //function call
  cout<<sum;
}

int add(int a, int b) // function segment
{
  return a + b;
}
```

7.4- Passing arguments to a function:

To pass arguments to a function, you list them in parentheses following the function name. The number of arguments and the type of each argument must match the parameters in the function header and prototype.

For **example**, if a function is defined to take two type int arguments, you must pass for it exactly two int arguments--no more, no less--and no other type. Remember to pass the multiple arguments in order.


Function gets called
Parameter a is copied into num1 &
Parameter b is copied into num2
Function definition

```
result = sum(a, b);
```

```
int sum(int num1, int num2)
```

❖ "Pass by Value" vs. "Pass by Reference":

When you call a function, the ***arguments*** can be passed to the function in two different ways: ***Pass by value*** or ***Pass by reference.***

1. ***Pass by value*** means that a copy of the contents of the **arguments** is passed in to the function parameters. ➔Typically if you aren't going to change the actual variable permanently, you use pass by value.

2. In ***pass by reference*** (also called pass by address), a copy of the address of the **arguments** is passed into the function parameters. The called function does not create its own copy, rather it refers to original value only by different name <reference or alias to the actual parameter in calling function>. This may change the actual variable permanently.

✍ **The following figures illustrate how passing values to a function works:**

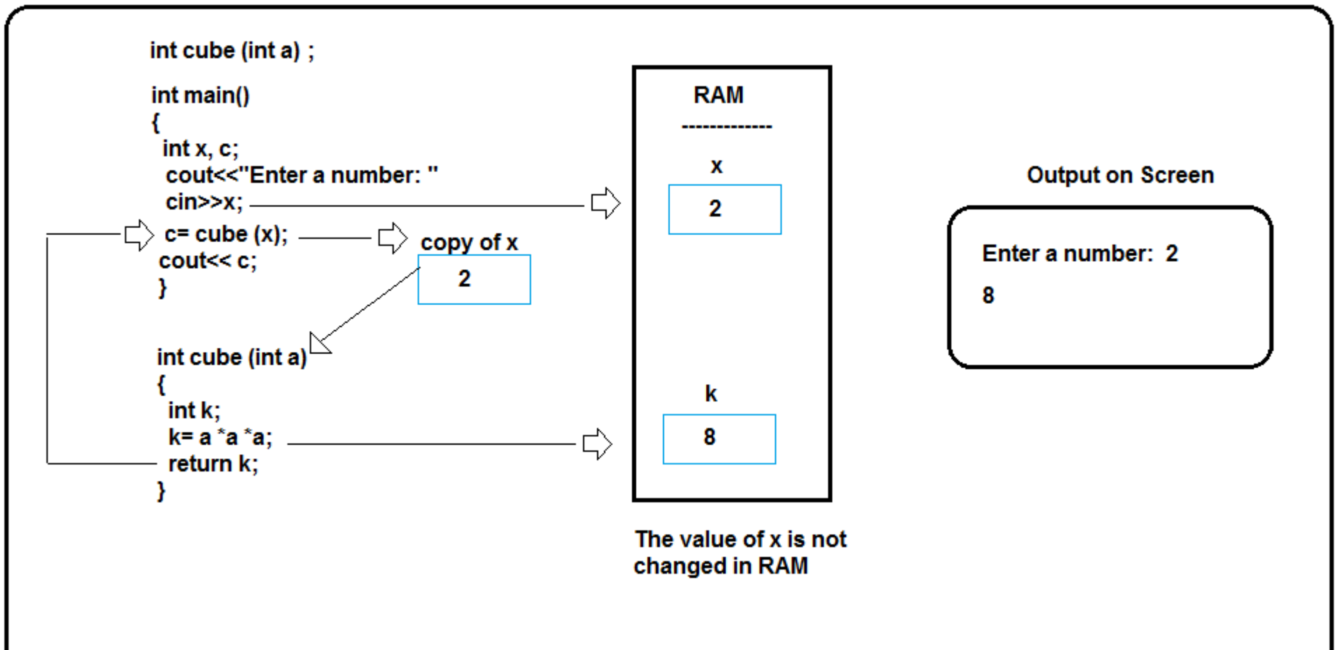


Figure.Pass by Value

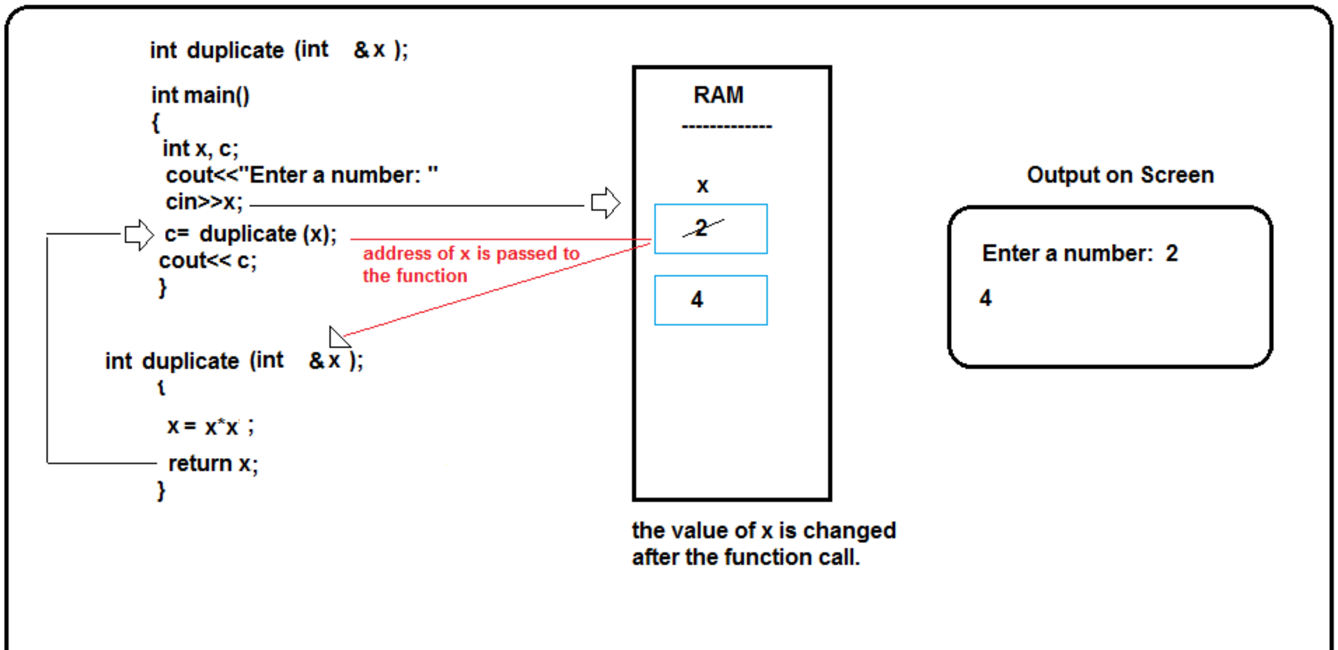


Figure.Pass by Reference

Exercise 151: Consider a **swapping function** to demonstrate ***pass by value*** vs. ***pass by reference***.

```
#include<iostream.h>

void swapByVal (int num1, int num2);

void swapByRef (int &num1, int &num2) ;
```

```
void main()
{
  int i = 10,  j = 20;

  swapByVal (i, j);
  cout<< i << "  " << j <<endl;     // this will display:  10  20

  swapByRef(i, j);
  cout<< i << "  " << j <<endl;     // this will display:  20  10
}
```

```
void swapByVal (int num1, int num2)
{
      int temp = num1;
      num1 = num2;
      num2 = temp;

     //here, the change occurs only inside the function. No changes in the variable of main
   }
```

```
void swapByRef (int &num1, int &num2)
{
  int temp = num1;
  num1 = num2;
  num2 = temp;

    //here, the real variable in main are changed
}
```

7.5- Passing arrays to a function:

An **array** can be passed to a function as argument, and also it can be returned by a function. There are **three popular ways to pass an array to a function**. And all three declaration methods produce similar results.

- **Suppose you want to pass the array A[10] to a function:**

❖ **Way-1: Send an unsized array as follows:**

```
void average (int A [ ] , int size)
{
...
}

// the function call will be something like: v= average (A, S);
```

❖ **Way-2: Send a sized array as follows:**

```
void average (int A [10])
{
...
}

// the function call will be something like: v= average (A);
```

❖ **Way-3: Send a pointer parameter on the array as follows:**

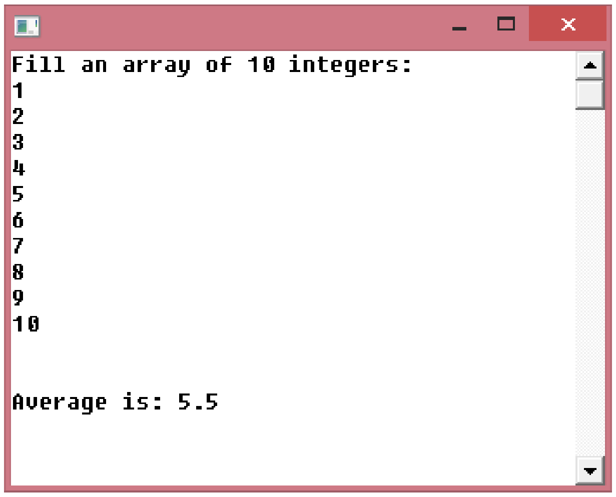
```
void average (int  *A)
{
...
}

// the function call will be something like: v= average (A);
```

Exercise 152 (way 1): A function average ( ) **that takes in an unsized array** A[ ] **with its** size **(10 elements maximum) as arguments and that will return the average of its numbers:**

```
#include <iostream.h>
double average (int A[ ], int size);     //prototype
void main( )
{
 int arr[10], s =10 ;
 float v;
 cout<<"Fill an array of 10 integers:\n";
 for(int i=0; i<10;i++)
   cin>>arr[i];

 v= average (arr , s );   // function call

 cout<<"\n\nAverage is: " <<v;
}
```

```
double average (int A[ ], int size)
{ int i;
 double avg;
 double sum;
 for (i = 0; i < size; i++)
           sum = sum + A[i];

  avg = sum / size;
  return avg; }
```

The output is:



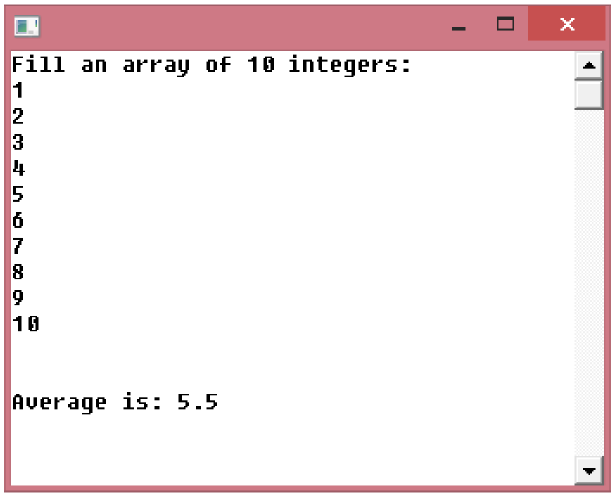
```
Fill an array of 10 integers:
1
2
3
4
5
6
7
8
9
10


Average is: 5.5
```

Exercise 153 (way 2): A function average( ) **that will take a sized array** A[10] **as argument and that will return the average of its numbers:**

```
#include <iostream.h>
double average (int A[10]);     //prototype
void main( )
{
 int arr[10];
 float v;
 cout<<"Fill an array of 10 integers:\n";
 for(int i=0; i<10;i++)
   cin>>arr[i];

 v= average (arr);   // function call

 cout<<"\n\nAverage is: " <<v;
}
```

```
double average (int A[10])
{ int i;
 double avg;
 double sum;
 for (i = 0; i < 10; i++)
            sum = sum + A[i];

  avg = sum / 10;
  return avg; }
```

The output is:



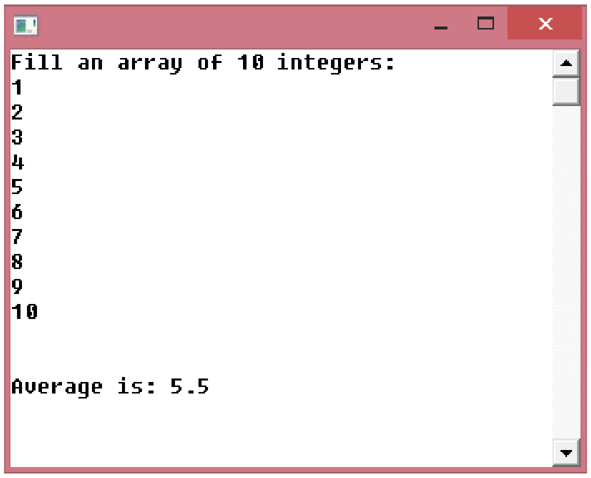
```
Fill an array of 10 integers:
1
2
3
4
5
6
7
8
9
10


Average is: 5.5
```

Exercise 154 (way 3): **A function** average( ) **that will take a pointer on an array** *A **as argument and that will return the average of its numbers:**

```
#include <iostream.h>
double average (int *A);     //prototype
void main( )
{
 int arr[10];
 float v;
 cout<<"Fill an array of 10 integers:\n";
 for(int i=0; i<10;i++)
   cin>>arr[i];

 v= average (arr);    // function call

 cout<<"\n\nAverage is: " <<v;
}
```

```
double average (int *A)
{ int i;
 double avg;
 double sum;
 for (i = 0; i < 10; i++)
            sum = sum + A[i];

  avg = sum / 10;
  return avg; }
```

The output is:



```
Fill an array of 10 integers:
1
2
3
4
5
6
7
8
9
10


Average is: 5.5
```

Exercise 155: **Write a program that accepts 5 integers and calculates the sum using a function SUM( ).**

Exercise 156: **Write a function that accepts an Array A[] and its size S; return the number of negative values in A[].**

Exercise 157: **Write a program that accepts a number and calculates and return its cubic value using a function. Example: 2 ➔ 2 * 2 *2 = 8**

Exercise 158: **Write a program that uses a function for sum_of_square_up to n numbers. In this program first you enter a number then it gives sum of square up to that number. Suppose the number entered is 4, then output will be $1^2+2^2+3^2+4^2 = 30$**

Exercise 159: **Write a program that uses a function to check whether a given number is Prime or not.**

Exercise 160: **Write a program that uses a function to check whether a given number is Perfect or not.**

Exercise 161: **Write a program that uses a function to check whether a given number is Armstrong or not.**

Exercise 162: **Write a function that accepts an array A [10] as argument and returns the average of EVEN numbers.**

Exercise 163: **Write a program that uses a function that accepts an array A[5] and returns its maximum value.**

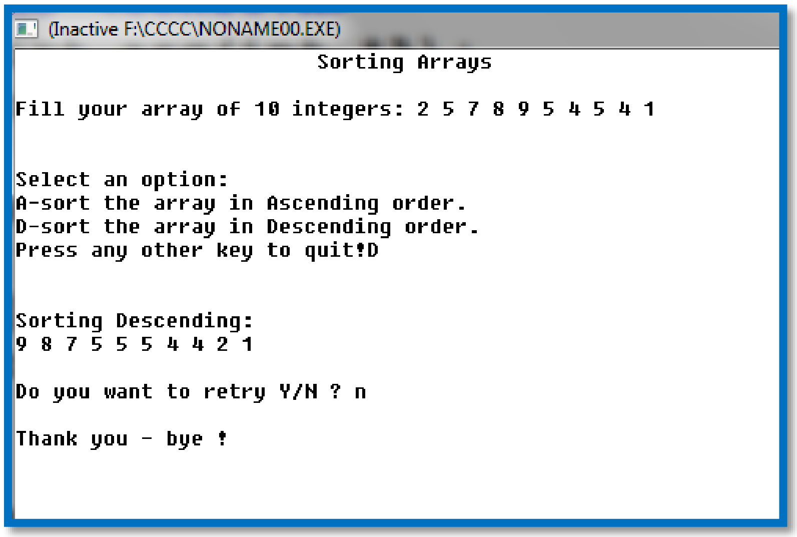
Exercise 164: **Write a function that accepts N as number of integers, and displays the factorial of the N integers. Note: N!= N * (N-1) * (N-2) * ….2 *1**

Exercise 165: **Write a C++ function that accepts a double dimension Array A[3][3] and count how many EVEN numbers are in the array.**

Exercise 166: **Write a program that uses a function for check whether number is palindrome or not.**

Exercise 167: **Write a program to reverse a number. For example if user enter 1234 as input then 4321 is printed as output. To reverse a number you can use the modulus (%) operator to obtain the digits of a number (from right to left).**

Exercise 168: **Write a C++ program that includes two functions** sort_asc **and** sort_desc. **The user can choose how to display his array sorted. First, the user must fills the array A[10]. The prog should allow the user to repeat the process.**


(Inactive F:\CCCC\NONAME00.EXE)

```
                         Sorting Arrays

Fill your array of 10 integers: 2 5 7 8 9 5 4 5 4 1


Select an option:
A-sort the array in Ascending order.
D-sort the array in Descending order.
Press any other key to quit!D


Sorting Descending:
9 8 7 5 5 5 4 4 2 1

Do you want to retry Y/N ? n

Thank you - bye !
```

7.6- Functions with no type: void

If the function does not need to return a value, the type to be used is **void**. For example, a function that simply displays a message may not need to return any value:

```
#include <iostream.h>
void display ( )
{
  cout << "Thank You!";
}

void main ( )
{
display ( );
}
```

void can also be used in the function's parameter list to specify that the function takes no actual parameters when it is called:

```
void printmessage (void)
{
 cout << "Thank You!";
}
```

The return value of main( ):

You may have noticed that the return type of **main** is **int**. If the execution of **main** ends normally without encountering a return statement, the compiler assumes the function main ( ) ends with an implicit return statement: **return 0 ;**

When **main** returns **zero** (either implicitly or explicitly), it is interpreted by the environment as that the program ended successfully. Other values may be returned by main which indicates an error.

Value	Description
0	The program was successful
EXIT_SUCCESS	The program was successful. This value is defined in header <cstdlib>.
EXIT_FAILURE	The program failed. This value is defined in header <cstdlib>.

7.7- Random Values in C++ with Rand:

To generate a random number we can use the **rand()** and **srand()** functions of **<stdlib.h>**. This will produce a result in the range 0 to RAND_MAX, where RAND_MAX is a constant defined by the implementation.

- **rand ( )** does not actually generate a random number. What **rand( )** gives you is a pseudo random number. In fact the sequence will eventually repeat and is predictable. We can **seed** the generator with the **srand( )** function.
- **srand ( )** is used to select where to start reading from the list of pseudo random numbers. Think **seed == start index**. By using **srand ( ),** you can make the numbers generated by **rand( )** seem random.
  - ✓ The typical thing to do, is to put **srand(time(0));** in main**( )**. You really should only call srand ( ) once, prior to the first call to rand( ).
  - ✓ The value of **RAND_MAX** varies according to your compiler, and can be as low as **32767**, which would give a range from **0 to 32767** for rand( ).
  - ✓ **`RAND_MAX`** is a **`const int`** defined in the include file **`stdlib.h`**. To find out the value of RAND_MAX for your compiler run the following small piece of code:

```
//Determine the Rand Max value
#include <iostream.h>
#include <stdlib.h> // srand(), rand()
void main ( ){
        cout << "The value of RAND_MAX is " <<  RAND_MAX; }
```

Exercise 169: A program to generate a single random number.

```
#include <iostream.h>
#include <stdlib.h> // srand(), rand()
void main ( ){
        int n =  rand();
        cout << n ;}
```

Exercise 170: A program to seed the generator with the system time then output a single random number, which should be different each time we run the program.

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

main ( )
{
  srand(time(0));
  int n = rand();
  cout << n << endl;
}
```

- The use of `time(0)` as an argument to `srand` ensures that a new starting value is used each time the program is run.

- Note that the inclusion of the files `stdlib.h` and `time.h` is required before these functions can be used.

Exercise 171: A program to display ten different random integers:

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

main ( )
{
  int n;
  srand(time(0));
  for(int i=1; i<=10; i++)
  {
          n = rand();
          cout << n << endl;
  }
}
```

Generating a number in a specific range:

If you want to produce numbers in a specific range, rather than between 0 and RAND_MAX, you can use the **modulo operator**: **rand( )%n** that generates a number from **0 to n-1**.

By adding an offset to the result we can produce a range that is not zero based.

Exercise 172: A program that produce 20 random numbers from 1 to 10:

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
void main ( )
{
srand(time(0));
int random_integer;
for(int index=0; index<20; index++)
 {
  random_integer = (rand( )%10)+1;
  cout << random_integer << endl;
  }
 }
```

Exercise 173: Write in C++ a Random Data Analyzer program that generates 100 random numbers between 0 and 1000. Print out the average and the maximum.

Exercise 174: Write in C++ a Random Data Analyzer program that generates 10 random numbers between 0 and 10. Print out the occurrence of each number.

Chapter 8

Recursion

“There are two types of people in this world:
those who understand recursion and those who don't understand it”.

Objectives:

- Purpose of using Recursive functions.
- Understanding the differences between Iteration and Recursion.
- Advantages of using recursions.
- Writing Recursive functions.
- Exercises.

Recursions

8.1- What is a Recursive Function?

Recursive function is a function that calls itself either directly or indirectly through another function. Some problems are more easily solved by recursion, like solving tasks that can be defined in terms of similar subtasks. For instance, sort, search, and traversal problems often have simple recursive solutions. For example, **recursion** can be used to calculate the factorial of a number. The factorial of a number **x** is written **x!** and is calculated as follows:

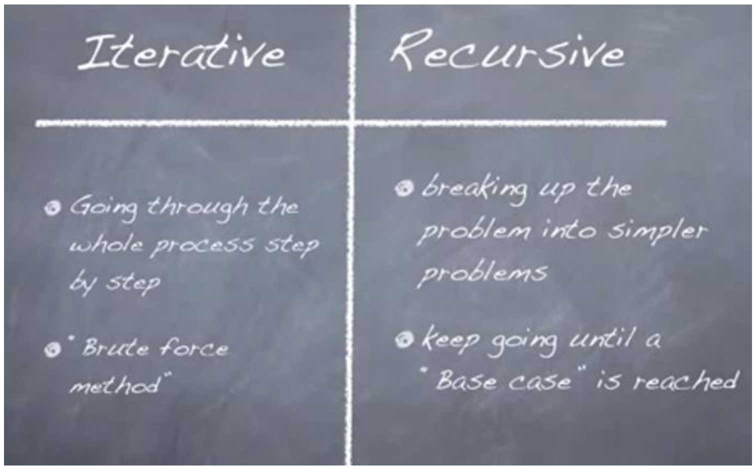
x! = x * (x-1) * (x-2) * (x-3) * ... * (2) * 1

However, you can also calculate x! like this:
x! = x * (x-1)!

Going one step further, you can calculate (x-1)! using the same procedure:
(x-1)! = (x-1) * (x-2)!

You can continue calculating recursively until you're down to a value of 1...

Iteration vs Recursion:



Iterative

- Going through the whole process step by step
- "Brute force method"

Recursive

- breaking up the problem into simpler problems
- keep going until a "Base case" is reached

Iteration solutions are often better than recursion with respect to performance, because recursion produces a series of function calls and copies of variables, requiring more overhead. On the other side, using recursion makes it possible to write complicated programs in very simple and elegant way.

8.2- Writing Recursive Functions:

A recursive function has the following general form:

```
ReturnType  Function ( Pass appropriate arguments )
{

  if (base case) // stopping condition
           return the simplest value without recursion call;
  else
          call function with a simpler version of problem (recursivity) ;
}
```

This is how it works:

1. **Write "if":** There must be at least 2 cases: a **recursive case** where the method calls itself and a **base case** where the method does not (stopping condition).
2. **First, handle the base case:** no recursive call needed (no looping).
3. **Second, write the recursive call:** for the next input/state

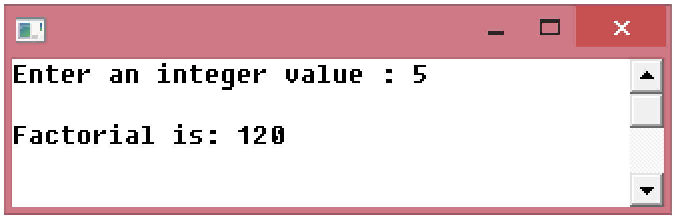
☝ When a function calls itself, a new copy of that function is run. The local variables in the second version are independent of the local variables in the first, and they cannot affect one another directly.

☝ The local variables in the main( ) function can affect the local variables in any function it calls.

Exercise 175: **Write a recursive function to calculate factorials.**

```
// Using a recursive function to calculate factorials

#include <iostream.h>
int f, n;

long fact( int n);   //function prototype

void main( )
{
  cout<< "Enter an integer value : ";
  cin>> n;
  f = fact(n);
  cout<<"\nFactorial is: " << f;
}

long fact( int n)
{

 if ((n == 0) || (n == 1))   // base cases

     return 1;
  else
     n = n* fact (n-1);

 return n;
}
```

Output:



```
Enter an integer value : 5

Factorial is: 120
```

☝ **This is how it works:**

If 5 is entered:

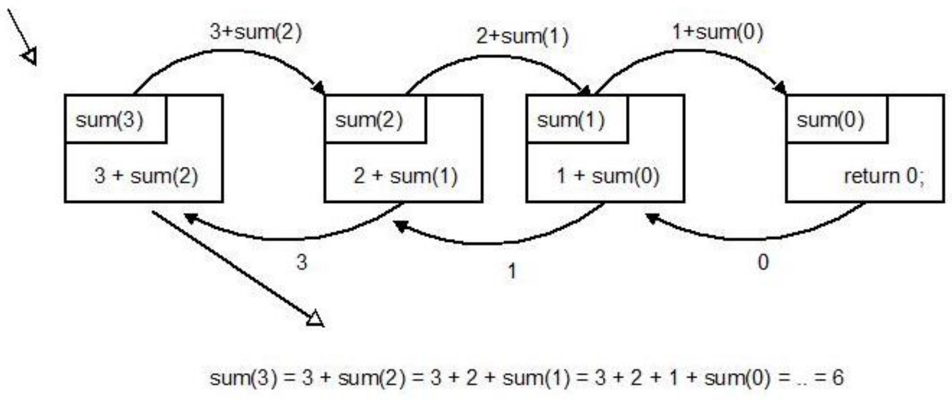
1) **call fact( ) with 5, result is stored as 5 * fact(4)**
2) **call fact( ) with 4, result is stored as 5 * 4 * fact(3)**
3) **call fact( ) with 3, result is stored as 5 * 4 * 3 * fact(2)**
4) **call fact( ) with 2, result is stored as 5 * 4 * 3 * 2 * fact(1)**
5) **call fact( ) with 1, result is stored as 5 * 4 * 3 * 2 * 1**

Since it all happened under the same instance, it kept the value of the previous. In other words, when it got to one it still had all the previous calculations in a chain so it got calculated when it hit the base case. That's the power of recursion!

Exercise 176: **Write a program to calculate the sum of 1 + 2 + .. + 9 + 10. Use a recursive function.**

```
#include <iostream.h>
int sum(int n);
void main()
{
 int n ;
 cout <<"Enter the nb of terms n: ";
 cin>>n;

int s= sum (n);
cout <<"\nSum is: " <<s;
}

int sum(int n) {

             if ( n == 0 ) return 0;   //base case
               else
                 n= n+ sum(n-1);   //recursion case
return n;
}
```

Now, how would we go on by computing summation using recursion? Let's come up with a formula by calculating the summation of 1, 2, and 3:



Exercise 177: **Write a program that decomposes a number n into smaller dolls until 1 using a recursive function. Example: n = 5 ➔ 5 4 3 2 1**

Exercise 178: **Write a recursive function pow (a , n) for Raising a Number to a Power.**

$$\begin{aligned} A^4 &= A * A^3 \\ &= A * A * A^2 \\ &= A * A * A * A^1 \\ &= A * A * A * A * A^0 \\ &= A * A * A * A * 1 \end{aligned}$$

Exercise 179: **Write a C++ program to display N number of the following sequence (using recursion): 4 7 10 13 16 …**

Exercise 180: **Write a C++ program to display N number of the following sequence (using recursion): 8 2 -4 -10 -16 …**

Exercise 181: **Use recursion to write a function that displays out the numbers:**
1 2 3 4 5 6 7 8 9 9 8 7 6 5 4 3 2 1.

Exercise 182: **Write the Euclid's algorithm (Iterative method) for finding the greatest common divisor of two integers (GCD).** The greatest common divisor of a and b is usually written as gcd (a, b). For example, gcd (12, 18) = 6, gcd (-4, 14) = 2 and gcd (5, 0) = 5. Two numbers are called coprime or relatively prime if their greatest common divisor equals 1. For example, 9 and 28 are relatively prime.

❖ **Euclid's Algorithm:** Given two natural numbers a and b, not both equal to zero:

Check if b is zero ➔if yes, a is the gcd; if not, repeat the process using, respectively, b, and the remainder after dividing a by b. The remainder after dividing a by b is usually written as a mod b. In C++ we implement a mod b using the % operator.

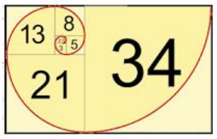
Exercise 183: **Write the Euclid's algorithm (Recursion method) for finding the greatest common divisor of two integers (GCD).**

Exercise 184: **Write a C++ program for two Matrix Multiplication using Iterative method.**

Exercise 185: **Write a C++ program for two Matrix Multiplication using Recursive method.**

Exercise 186: **Write a recursive function to display the first 13 Fibonacci numbers. The mathematical definition of the Fibonacci number is:**

- **fib(0) = 1**
- **fib(1) = 1**
- **fib(n) = fib(n-1) + fib(n-2)**

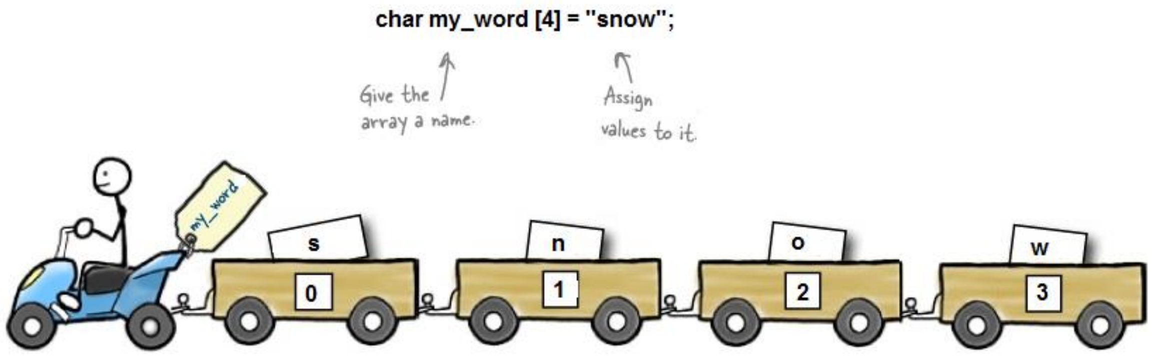


Hint: *The Fibonacci sequence is a set of numbers that starts with a one or a zero, followed by a one, and proceeds based on the rule that each number (called a Fibonacci number) is equal to the sum of the preceding two numbers. If the Fibonacci sequence is denoted F (n), where n is the first term in the sequence, the following equation obtains for n = 0, where the first two terms are defined as 0 and 1 by convention. F (1) = 1, 1, 2, 3, 5, 8, 13, 21...*

Fibonacci numbers are of interest to biologists and physicists because they are frequently observed in various natural objects and phenomena. The branching patterns in trees and leaves, for example, and the distribution of seeds in a raspberry are based on Fibonacci numbers.

Chapter 9

C++ Strings and Pointers



String of characters

Objectives:

- Fundamentals of Characters and Strings
- Understanding the C-Style Character String
- Manipulating Strings.
- Pointers basics.
- Exercises.

C++ Strings

9.1- Fundamentals of Characters and Strings:

In programming, Strings are in fact, sequences of characters that can be represented as arrays of type char. We can distinguish between the following:

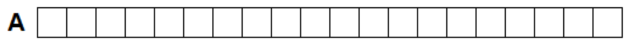
- **Character constant:** Integer value represented as character in single quotes, example: **'z'**
- **String:** Series of characters treated as single unit. It can include letters (A_Z, a-z) , digits (0-9), or special characters +, -, * ...
  - A string is enclosed in double quotes, for example: **"I like Programming"**
  - Array of characters, ends with null character **'\0'**
  - There is a pointer to string's first character like in arrays

C++ provides **two types** of string representations:

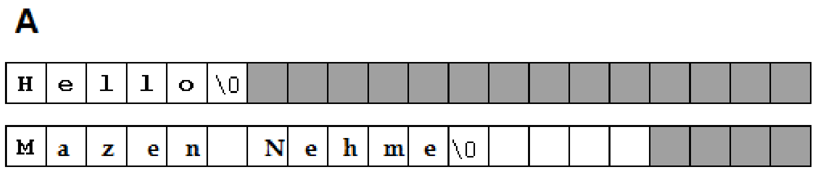
1. **The *C-style character* string.**
2. **The *string class* type introduced with Standard C++.**

9.2- The C-Style Character String:

This string is actually a **one-dimensional array of characters** which is terminated by a **null** character **'\0'** (backslash zero). Example: **char A[20] ;** this is an array that can store up to 20 elements of type char. It can be represented as:



Therefore, this array has a capacity to store sequences of up to 20 characters. But this capacity does not need to be fully exhausted; the array can also accommodate shorter sequences. For example, at some point in a program, either the sequence **"Hello"** or the sequence "**Mazen Nehme**" can be stored in **A[ ]**, since both would fit with a capacity of 20 characters.



String assignment:

- **char A[6] = {'H', 'e', 'l', 'l', 'o', '\0'};**
- **char A[ ] = "Hello"; // Creates 5 element char array with last element is '\0'**
- **A[0] = 'H';**
  A[1] = 'e';
  A[2] = 'l';
  A[3] = 'l';
  A[4] = 'o';
  A[5] = '\0';

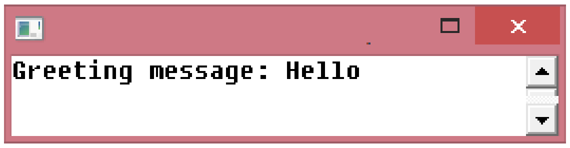
☝ Actually, you do not need to place the null character at the end of a string constant. The C++ compiler automatically places the '\0' at the end of the string when it initializes the array.

☝ To hold the null character at the end of the array, the size of the character array must be one more than the number of characters of the value.

Example:

```
#include <iostream.h>
main ()
{
        char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

        cout << "Greeting message: ";
        cout << greeting << endl;
 }
```

When the above code is compiled and executed, it produces the following result:



```
Greeting message: Hello
```

Reading strings:

Assign input to character array **word [20]:**

- **cin** >> **word;**
    - Reads characters until whitespace or EOF
- **cin** >> **setw(20)** >> **word;**
    - Reads 19 characters (space reserved for '\0')
- **cin.getline**
    - Read line of text
- **cin.getline (array, size, delimiter);**
    - Copies input into specified **array** until either One less than **size** is reached or **delimiter** character is input
    - Example: **char A[80];**
      cin.getline (A, 50); //read a string of max 50 character
- **cin.sync( );**
    - If you're using **cin.getline** after **cin** >> something, you need to flush the newline out of the buffer in between. You can use **cin.sync( );** to discard unread characters and avoid this problem.

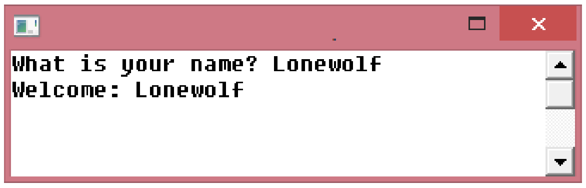
Reading and Printing out strings and characters:

Strings can be read or printed out using: **cin** and **cout**

Example:

```
#include <iostream.h>
#include <string.h>
 main ()
{
  char question1[] = "What is your name? ";
  char answer1 [80];
  cout << question1;
  cin >> answer1;
  cout << "Welcome: " << answer1;
}
```

Output:



```
What is your name? Lonewolf
Welcome: Lonewolf
```

9.3- String Manipulation Functions:

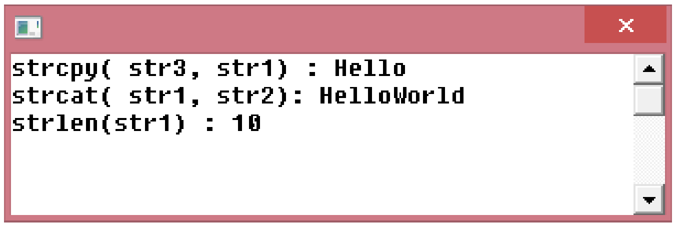
C++ supports a wide range of predefined functions that manipulate null-terminated strings:

	Functions and Purpose
1	**strlen(s1);** Returns the length of string s1.
2	**strcpy(s1, s2);** Copies string s2 into string s1.
3	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of the character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.
---	**More functions are included in: <string.h> and <cstring.h>**

Exercise 187: Practicing Strings predefined functions:

```
#include <iostream.h>
#include <cstring.h>     // you can use  <string.h>
int main ( )
{
   char str1[10] = "Hello";
   char str2[10] = "World";
   char str3[10];
   int  len ;

   // copy str1 into str3
   strcpy( str3, str1);
   cout << "strcpy( str3, str1) : " << str3 << endl;

   // concatenates str1 and str2
   strcat( str1, str2);
   cout << "strcat( str1, str2): " << str1 << endl;

   // total lenghth of str1 after concatenation
   len = strlen(str1);
   cout << "strlen(str1) : " << len << endl;

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:



```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

Exercise 188: **Write a function that accepts two strings S1 and S2 and display which String is larger.**

Exercise 189: **Write a function that accepts two strings S1 and S2 and display if they are identical.**

Exercise 190: **Write a function that accepts an Array of characters A[50] and return the number of spaces values in A[].**

Exercise 191: **Write a C++ function that accepts an Array of characters A[50] and return the number of spaces values in A[ ].**

Exercise 192: **Write a C++ program that includes a function int count(char car1, char car2, char *word) that returns the number of occurrences of two successive characters car1 and car2 in the string word.**

Exercise 193: **Write a function for printing a String Reversed.**

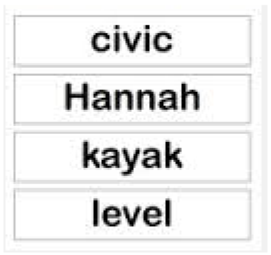
Exercise 194: **Write a function display (char *ch) that returns the number of characters (f, u, l, a, s) present in the string ch.**

Exercise 195: **Write a function NBRWORD (char *ch) that returns the number of words present in the string ch ending by "ers". Note: the words are separated by a space (" ") in the string.**

Exercise 196: **Write a function that accepts one character X and a Sentence S; return the occurrence of X in S.**

Exercise 197: **Write a function that accepts a Sentence S of max 50 char, and determine how many words ends with "r" in S.**

Exercise 198: **Write a program that uses a function for check whether a given string is palindrome or not.**



civic
Hannah
kayak
level

C++ Pointers:

What is a Pointer?

A **pointer** is a variable whose value is the **address** of **another variable**. Like any variable, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is:

type   *var-name;

Here, ***type*** is the pointer's type; it must be a valid C++ type and ***var-name*** is the name of the pointer variable. The ***asterisk**** is used to designate a variable as a pointer. Following are the valid pointer declaration:

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to character
```

The **actual data type** of the value of all pointers, whether integer, float, character, or otherwise, is the same, **a long hexadecimal number** that represents a **memory address**.

Using Pointers in C++:

Some C++ tasks are performed more easily with pointers, such as dynamic memory allocation, cannot be performed without them.

(a) We define a pointer variables

(b) Assign the address of a variable to a pointer

(c) Finally access the value at the address available in the pointer variable. This is done by using unary **operator *** that returns the value of the variable located at the address specified by its operand.

The memory addresses_of operator & :

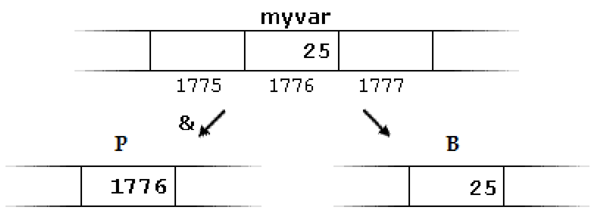
For a C++ program, the memory of a computer is like a succession of memory cells, each one byte in size, and each with a unique address.

When a variable is declared, the memory needed to store its value is assigned (by the OS) a specific location in memory (an address). However, the address of a variable can be obtained by preceding the name of a variable with an ampersand sign (**&**), known as **address_of** operator. For example: **P = &myvar;**

This would assign the address of variable **myvar** to **P**; by preceding the name of the variable **myvar** with the address-of operator (**&**), we are no longer assigning the content of the variable itself to **P**, but its address.

Take a look at this example:

- **myvar = 25;**
- **P = &myvar;**
- **B = myvar;**



Memory at Run time

The variable that stores the address of another variable (like **P** in the previous example) is what in C++ is called a pointer.

Example: Consider the following example which will display the **address** of the variables defined:

```
#include <iostream.h>
int main (){
        int  var1;
        char var2[10];
        cout << "Address of var1 variable: ";
        cout << &var1 << endl;
        cout << "Address of var2 variable: ";
        cout << &var2 << endl;
        return 0;  }
```

When the above code is compiled and executed, it produces result something as follows:

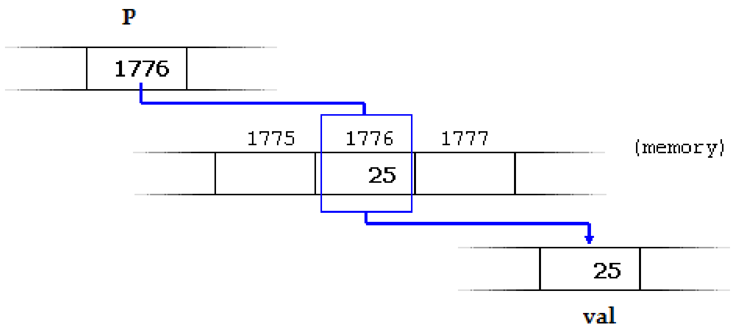
```
Address of var1 variable: 0xbfebd5c0
Address of var2 variable: 0xbfebd5b6
```

The dereference operator (*):

An interesting property of pointers is that they can be used to access the variable they point to directly. This is done by preceding the pointer name with the **dereference operator** (*). The operator itself can be read as "**value pointed to by**".

Therefore, following with the values of the previous example, the following statement:
val = *P;

This could be read as: **"val equal to the value pointed to by P",** and the statement would actually assign the value **25** to **val**, since **P** is **1776**, and the value pointed to by **1776** is **25**.



It is important to clearly differentiate that **P** refers to the value **1776**, while ***P** (with an **asterisk *** preceding the identifier) refers to the **value** stored at address **1776**, which in this case is **25**.

```
val = P;   // val equal to P (1776)

val = *P;  // val equal to value pointed to by P(25
```

Exercise 199: Write a program that find the maximum within data values entered by the user. The program uses a function *findMax( ) **that accepts the array of data and its size. The return from the function is the pointer that points to the max value.**

Exercise 200: Write a C++ program to accept five integer values from keyword. The five values will be stored in an array using a pointer. Then print the elements of the array on the screen.

Final Exercises

- **Exercise 201:** **Write a program that read a sentence and writes the second word in an array of characters. Words are separated by spaces.**

- **Exercise 202:** **Write a program that read a sentence and then display it without the vowels: e o u a i**

- **Exercise 203:** **Write a function CountDigit which receives two arguments: a char array and the size of the array (of type int). This function counts the number of digit letters in the char array, and returns the count (of type int). Use the C++ predefined function** isdigit **(defined in** <cctype>**).**

- **Exercise 204:** **Write a piece of code which calls CountDigit to find out how many digit characters are in arr.**

- **Exercise 205:** **Write a program to solve the following:**

  1. **Creates a 2-D array with 3 rows and 4 columns.**

  2. **Populate it 12 random integers in the range 1 to 9, including the end points.**

  3. **Print the contents of the array, appropriately labeled, with 3 rows and 4 columns. For readability, include a space between each number.**

     Your printout should look like the following, with the x's representing numbers in the array:

```
x x x x
x x x x
```

x x x x

Exercise 206: Write a program to solve the following:

1. In the main method, create a 10 x 8 two dimensional array consisting of random integers in the range 50 to 99, including the end points.
2. Call a method named printArray that will print the array, appropriately labeled, 8 numbers per line, with a space between each.
3. Call a method named findSmallest that will find and print, appropriately labeled, the smallest number in the array.
4. Call a method named findAverage that will find and print, appropriately labeled, the average of the numbers in the array.
5. Call a method named findMode that will find and print, appropriately labeled, the number that occurs most often.
6. Call a method named removeEvens, that will modify the array such that all even numbers are replaced by the number 11.
7. Call the method named printArray discussed in part 2 above and print the array as modified in part 6 above.

Exercise 207: Write a program to solve the following:

1. In the main method create a random array of 9 integers.
2. Call a method to print the array, all on one line with a space between.
3. Call a method to reverse the order of the digits.
4. Call the print method above to print the new contents.

Exercise 208: **Write a program to solve the following:**

1. **In the main method:**

   a. **Declare and initialize a two-D array of integers containing 10 rows and 10 columns. The diagonal cells that run from upper left to lower right contain the integer 1. The other cells contain 0.**

   b. **Prompt the user to enter a row number and a column number - these will be used below.**

   c. **Populate the array as described above using either a for loop or a while loop.**

2. **Call a method named printArray that will print the contents of the array, 10 numbers per line.**

3. **Call a method named printCol that will print the contents of the col number referenced above, one item per line. Do this using a for or a while loop- and any other structures needed.**

4. **Call a method named printRow that will print the contents of the row number referenced above, all on one line with a space between each. Do this using a for or a while loop - and any other structures needed.**

5. **Call a method nemed printDiagonal that will print the numbers on the diagonal, all on the same line, with a space between. Do this using a for or a while loop - and any other structures needed.**

Exercise 209: **Write a program that asks the user to input 5 sequences of characters. Then it will ask the user for a character to search for and will output the maximum number of times that it occurred between the 5 sequences.**

Example:
Sequence 1: aabc
Sequence 2: aaaa

If the user chooses to search for character 'a', the program will output "Character a occurred a maximum of 4 times" because it occurred 4 times in the second sequence, and only twice in the first sequence.

Exercise 210: **Write a program that asks the user to type a random number. Write in output if this number is a prime number or not. Write, by which numbers can your number be divided.**

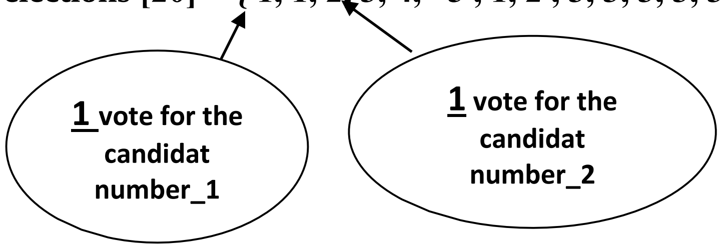
Exercise 211: **By using C++ pointer, write a C++ program to find the max of an integral data set. The program will ask the user to input the number of data values in the set and each value. Then your program will show the max of the data set. See example below. Your C++ program will use a function that accepts the array of data values and its size. The return from the function is the pointer that points to the max value.**

```
Enter number of data values: 3
Enter value 1: 21
Enter value 2: 12
Enter value 3: 4
The max is 21.
Result: 2/9
```

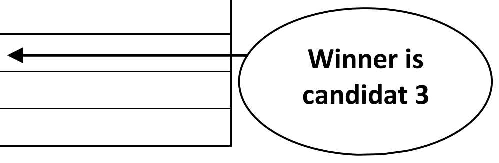
Exercise 212: **Write a C++ function to sort an array of ten integer values in ascending order. The function will accept two arguments-- a pointer that points to the array and the array size. The function returns a pointer that points to the sorted array.**

Exercise 213: **You have 20 people and their votes for 5 candidates. Write a C++ program to read the votes and count and display the number of votes that each candidate got.**

Given: int elections [20] = { 1, 1, 2, 3, 4,   5 , 1, 2 , 3, 3, 3, 3, 3, 3, …};


1 vote for the candidat number_1
1 vote for the candidat number_2

Candidats	Votes
1	2
2	2
3	14
4	1
5	1



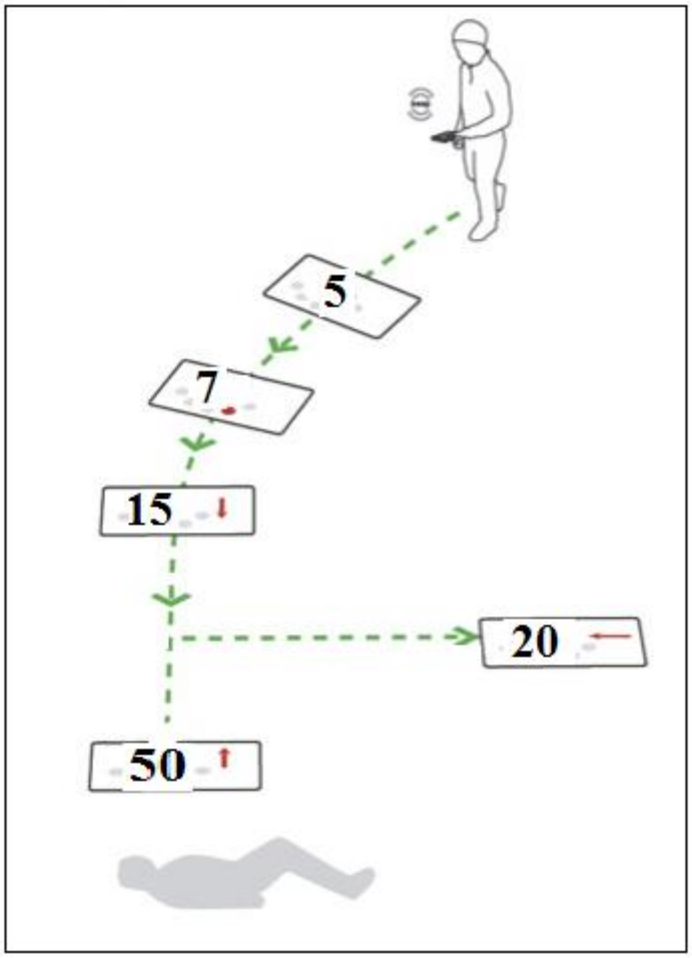
Exercise 214: Bracketing Search

Write a C++ program that calculates a random number (1 through 100). The program then asks the user to guess this number.

If the user guesses too high or too low then the program should output "too high" or "too low" accordingly.

The program must let the user continue to guess until the user correctly guesses the number.

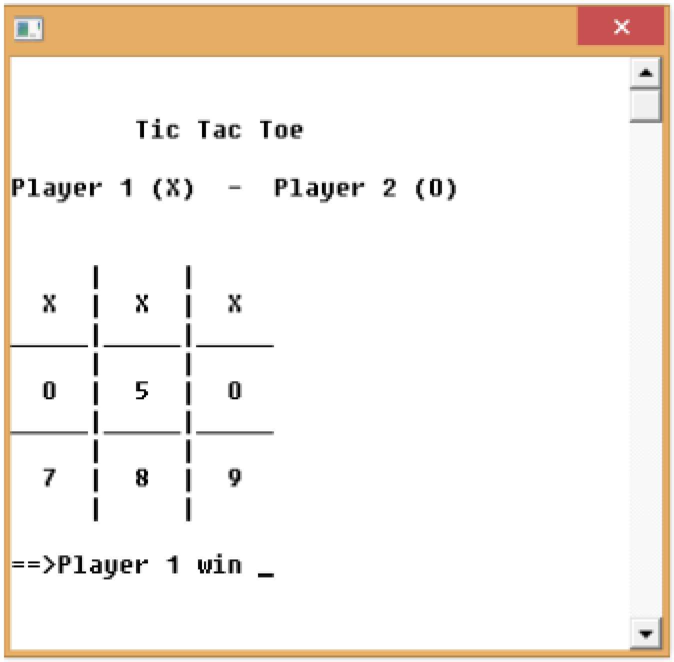
The program must also output how many guesses it took the user to correctly guess the right number.



Exercise 215: **Bracketing Search (2)**

Modify the program of exercise 214 so that instead of the user guessing a number the computer came up with, the computer guesses the number that the user has secretly decided. The user must tell the computer whether it guessed too high or too low.

Exercise 216: **Write a C++ program for the Tic Tac Toe game**



```
          Tic Tac Toe

Player 1 (X)  -  Player 2 (O)


       |     |
   X   |  X  |  X
_______|_____|_______
       |     |
   O   |  5  |  O
_______|_____|_______
       |     |
   7   |  8  |  9
       |     |

==>Player 1 win _
```

Finally, we got here, Good work!
Hope to see you in the next levels where we will discuss advanced programming topics…

C++ Programming

mazennehme@hotmail.com
03/728019