C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming.

C++ is regarded as a middle-level language, as it comprises a combination of both high-level and low-level language features.

C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983.

C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

Note: A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.

Object-Oriented Programming

C++ fully supports object-oriented programming, including the four pillars of object-oriented development:

 Encapsulation

 Data hiding

 Inheritance

 Polymorphism

Standard Libraries

Standard C++ consists of three important parts:

 The core language giving all the building blocks including variables, data types and literals, etc.

 The C++ Standard Library giving a rich set of functions manipulating files, strings, etc.

C++ Compiler:

This is actual C++ compiler, which will be used to compile your source code into final executable program.

Most C++ compilers don't care what extension you give your source code, but if you don't specify otherwise, many will use .cpp by default

Most frequently used and free available compiler is GNU C/C++ compiler, otherwise you can have compilers either from HP or Solaris if you have respective Operating Systems.

Installing GNU C/C++ Compiler:

Object - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating. An object is an instance of a class.

 Class - A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.

 Methods - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

 Instant Variables - Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.

* The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header <iostream> is needed.
* The line using namespace std; tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.
* The next line // main() is where program execution begins. is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.
* The line int main() is the main function where program execution begins.
* The next line cout << "This is my first C++ program."; causes the message "This is my first C++ program" to be displayed on the screen.
* The next line return 0; terminates main( )function and causes it to return the value 0 to the calling process.

# Comments in C++

Program comments are explanatory statements that you can include in the C++ code that you write and helps anyone reading it's source code. All programming languages allow for some form of comments.

C++ supports single-line and multi-line comments. All characters available inside any comment are ignored by C++ compiler.

C++ comments start with /\* and end with \*/. For example:

/\* This is a comment \*/

/\* C++ comments can also

\* span multiple lines

\*/

Within a /\* and \*/ comment, // characters have no special meaning. Within a // comment, /\* and \*/ have no special meaning. Thus, you can "nest" one kind of comment within the other kind. For example:

/\* Comment out printing of Hello World:

cout<< "Hello World"; // prints Hello World

\*/

# C++ Basic Input/Output

The C++ standard libraries provide an extensive set of input/output capabilities which we will see in subsequent chapters. This chapter will discuss very basic and most common I/O operations required for C++ programming.

C++ I/O occurs in streams, which are sequences of bytes. If bytes flow from a device like a keyboard, a disk drive, or a network connection etc. to main memory, this is called **input operation** and if bytes flow from main memory to a device like a display screen, a printer, a disk drive, or a network connection, etc, this is called **output operation**.

## I/O Library Header Files:

There are following header files important to C++ programs:

## The standard output stream (cout):

The predefined object **cout** is an instance of **ostream** class. The cout object is said to be "connected to" the standard output device, which usually is the display screen. The **cout** is used in conjunction with the stream insertion operator,

## The standard input stream (cin):

The predefined object **cin** is an instance of **istream** class. The cin object is said to be attached to the standard input device, which usually is the keyboard. The **cin** is used in conjunction with the stream extraction operator

# C++ Data Types

While doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

## Primitive Built-in Types:

C++ offer the programmer a rich assortment of built-in as well as user defined data types. Following table lists down seven basic C++ data types:

|  |  |
| --- | --- |
| **Type** | **Keyword** |
| Boolean | bool |
| Character | char |
| Integer | int |
| Floating point | float |
| Double floating point | double |
| Valueless | void |
| Wide character | wchar\_t |

Several of the basic types can be modified using one or more of these type modifiers:

* signed
* unsigned
* short
* long

The following table shows the variable type, how much memory it takes to store the value in memory, and what is maximum and minimum vaue which can be stored in such type of variables.

|  |  |  |
| --- | --- | --- |
| **Type** | **Typical Bit Width** | **Typical Range** |
| char | 1byte | -127 to 127 or 0 to 255 |
| unsigned char | 1byte | 0 to 255 |
| signed char | 1byte | -127 to 127 |
| int | 4bytes | -2147483648 to 2147483647 |
| unsigned int | 4bytes | 0 to 4294967295 |
| signed int | 4bytes | -2147483648 to 2147483647 |
| short int | 2bytes | -32768 to 32767 |
| unsigned short int | Range | 0 to 65,535 |
| signed short int | Range | -32768 to 32767 |
| long int | 4bytes | -2,147,483,647 to 2,147,483,647 |
| signed long int | 4bytes | same as long int |
| unsigned long int | 4bytes | 0 to 4,294,967,295 |
| float | 4bytes | +/- 3.4e +/- 38 (~7 digits) |
| double | 8bytes | +/- 1.7e +/- 308 (~15 digits) |
| long double | 8bytes | +/- 1.7e +/- 308 (~15 digits) |
| wchar\_t | 2 or 4 bytes | 1 wide character |

The sizes of variables might be different from those shown in the above table, depending on the compiler and the computer you are using.

Following is the example, which will produce correct size of various data types on your computer.

# C++ Variable Types

A variable provides us with named storage that our programs can manipulate. Each variable in C++ has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C++ is case-sensitive:

There are following basic types of variable in C++ as explained in last chapter

|  |  |
| --- | --- |
| **Type** | **Description** |
| bool | Stores either value true or false. |
| char | Typically a single octet(one byte). This is an integer type. |
| int | The most natural size of integer for the machine. |
| float | A single-precision floating point value. |
| double | A double-precision floating point value. |
| void | Represents the absence of type. |
| wchar\_t | A wide character type. |

C++ also allows to define various other types of variables, which we will cover in subsequent chapters like **Enumeration, Pointer, Array, Reference, Data structures,** and **Classes**.

Following section will cover how to define, declare and use various types of variables.

## Variable Definition in C++:

A variable definition means to tell the compiler where and how much to create the storage for the variable. A variable definition specifies a data type, and contains a list of one or more variables of that type as follows:

type variable\_list;

Here, **type** must be a valid C++ data type including char, w\_char, int, float, double, bool or any user-defined object, etc., and **variable\_list** may consist of one or more identifier names separated by commas. Some valid declarations are shown here:

int i, j, k;

char c, ch;

float f, salary;

double d;

The line **int i, j, k;** both declares and defines the variables i, j and k; which instructs the compiler to create variables named i, j and k of type int.

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

type variable\_name = value;

Some examples are:

externint d =3, f =5;// declaration of d and f.

int d =3, f =5;// definition and initializing d and f.

byte z =22;// definition and initializes z.

char x ='x';// the variable x has the value 'x'.

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables is undefined.

## Variable Declaration in C++:

A variable declaration provides assurance to the compiler that there is one variable existing with the given type and name so that compiler proceed for further compilation without needing complete detail about the variable. A variable declaration has its meaning at the time of compilation only, compiler needs actual variable declaration at the time of linking of the program.

A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use **extern** keyword to declare a variable at any place. Though you can declare a variable multiple times in your C++ program, but it can be defined only once in a file, a function or a block of code.

# Operators in C++

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

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Assignment Operators
* Misc Operators
* This chapter will examine the arithmetic, relational, logical, bitwise, assignment and other operators one by one.

## Arithmetic Operators:

* There are following arithmetic operators supported by C++ language:
* Assume variable A holds 10 and variable B holds 20, then:
* [**Show Examples**](http://www.tutorialspoint.com/cplusplus/cpp_arithmatic_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + | Adds two operands | A + B will give 30 |
| - | Subtracts second operand from the first | A - B will give -10 |
| \* | Multiplies both operands | A \* B will give 200 |
| / | Divides numerator by de-numerator | B / A will give 2 |
| % | Modulus Operator and remainder of after an integer division | B % A will give 0 |
| ++ | [**Increment operator**](http://www.tutorialspoint.com/cplusplus/cpp_increment_decrement_operators.htm), increases integer value by one | A++ will give 11 |
| -- | [**Decrement operator**](http://www.tutorialspoint.com/cplusplus/cpp_increment_decrement_operators.htm), decreases integer value by one | A-- will give 9 |

## Relational Operators:

There are following relational operators supported by C++ language

Assume variable A holds 10 and variable B holds 20, then:

[**Show Examples**](http://www.tutorialspoint.com/cplusplus/cpp_relational_operators.htm)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

## Logical Operators:

There are following logical operators supported by C++ language

Assume variable A holds 1 and variable B holds 0, then:

[**Show Examples**](http://www.tutorialspoint.com/cplusplus/cpp_logical_operators.htm)

|  |  |  |
| --- | --- | --- |
| **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. |

## Assignment Operators:

There are following assignment operators supported by C++ language:

[**Show Examples**](http://www.tutorialspoint.com/cplusplus/cpp_assignment_operators.htm)

|  |  |  |
| --- | --- | --- |
| **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 a |

## Misc Operators

There are few other operators supported by C++ Language.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| sizeof | [**sizeof operator**](http://www.tutorialspoint.com/cplusplus/cpp_sizeof_operator.htm) returns the size of a variable. For example, sizeof(a), where a is integer, will return 4. |
| Condition ? X : Y | [**Conditional operator**](http://www.tutorialspoint.com/cplusplus/cpp_conditional_operator.htm). If Condition is true ? then it returns value X : otherwise value Y |
| , | [**Comma operator**](http://www.tutorialspoint.com/cplusplus/cpp_comma_operator.htm) causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list. |
| . (dot) and -> (arrow) | [**Member operators**](http://www.tutorialspoint.com/cplusplus/cpp_member_operators.htm) are used to reference individual members of classes, structures, and unions. |
| Cast | [**Casting operators**](http://www.tutorialspoint.com/cplusplus/cpp_casting_operators.htm) convert one data type to another. For example, int(2.2000) would return 2. |
| & | [**Pointer operator &**](http://www.tutorialspoint.com/cplusplus/cpp_pointer_operators.htm) returns the address of an variable. For example &a; will give actual address of the variable. |
| \* | [**Pointer operator \***](http://www.tutorialspoint.com/cplusplus/cpp_pointer_operators.htm) is pointer to a variable. For example \*var; will pointer to a variable var. |

# C++ Loop Types

There may be a situation, when you need to execute a block of code several number of times. In general statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general from of a loop statement in most of the programming languages:

C++ programming language provides the following types of loop to handle looping requirements. Click the following links to check their detail.

|  |  |
| --- | --- |
| **Loop Type** | **Description** |
| [**while loop**](http://www.tutorialspoint.com/cplusplus/cpp_while_loop.htm) | Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body. |
| [**for loop**](http://www.tutorialspoint.com/cplusplus/cpp_for_loop.htm) | Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| [**do...while loop**](http://www.tutorialspoint.com/cplusplus/cpp_do_while_loop.htm) | Like a while statement, except that it tests the condition at the end of the loop body |
| [**nested loops**](http://www.tutorialspoint.com/cplusplus/cpp_nested_loops.htm) | You can use one or more loop inside any another while, for or do..while loop. |

## Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C++ supports the following control statements. Click the following links to check their detail.

|  |  |
| --- | --- |
| **Control Statement** | **Description** |
| [**break statement**](http://www.tutorialspoint.com/cplusplus/cpp_break_statement.htm) | Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch. |
| [**continue statement**](http://www.tutorialspoint.com/cplusplus/cpp_continue_statement.htm) | Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| [**goto statement**](http://www.tutorialspoint.com/cplusplus/cpp_goto_statement.htm) | Transfers control to the labeled statement. Though it is not advised to use goto statement in your program. |

## 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.

# C++ decision making statements

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general from of a typical decision making structure found in most of the programming languages:



C++ programming language provides following types of decision making statements. Click the following links to check their detail.

|  |  |
| --- | --- |
| **Statement** | **Description** |
| [**if statement**](http://www.tutorialspoint.com/cplusplus/cpp_if_statement.htm) | An if statement consists of a boolean expression followed by one or more statements. |
| [**if...else statement**](http://www.tutorialspoint.com/cplusplus/cpp_if_else_statement.htm) | An if statement can be followed by an optional else statement, which executes when the boolean expression is false. |
| [**switch statement**](http://www.tutorialspoint.com/cplusplus/cpp_switch_statement.htm) | A switch statement allows a variable to be tested for equality against a list of values. |
| [**nested if statements**](http://www.tutorialspoint.com/cplusplus/cpp_nested_if.htm) | You can use one if or else if statement inside another if or else if statement(s). |
| [**nested switch statements**](http://www.tutorialspoint.com/cplusplus/cpp_nested_switch.htm) | You can use one swicth statement inside another switch statement(s). |

# C++ Functions

A function is a group of statements that together perform a task. Every C++ program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division usually is so each function performs a specific task.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

The C++ standard library provides numerous built-in functions that your program can call. For example, function **strcat()** to concatenate two strings, function **memcpy()** to copy one memory location to another location and many more functions.

A function is knows as with various names like a method or a sub-routine or a procedure etc.

## Defining a Function:

The general form of a C++ function definition is as follows:

return\_type function\_name( parameter list )

{

body of the function

}

A C++ function definition consists of a function header and a function body. Here are all the parts of a function:

* **Return Type**: A function may return a value. The **return\_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword **void**.
* **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
* **Parameters:** A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
* **Function Body:** The function body contains a collection of statements that define what the function does.

## Example:

Following is the source code for a function called **max()**. This function takes two parameters num1 and num2 and returns the maximum between the two:

// function returning the max between two numbers

int max(int num1,int num2)

{

// local variable declaration

int result;

if(num1 > num2)

result= num1;

else

result= num2;

return result;

}

## Function Declarations:

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts:

return\_type function\_name( parameter list );

For the above defined function max(), following is the function declaration:

int max(int num1,int num2);

Parameter names are not importan in function declaration only their type is required, so following is also valid declaration:

int max(int,int);

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

## Calling a Function:

While creating a C++ function, you give a definition of what the function has to do. To use a function, you will have to call or invoke that function.

When a program calls a function, program control is transferred to the called function. A called function performs defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.

To call a function, you simply need to pass the required parameters along with function name, and if function returns a value, then you can store returned value. For example:

# C++ Arrays

C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

## Declaring Arrays:

To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array as follows:

type arrayName [ arraySize ];

This is called a single-dimension array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C++ data type. For example, to declare a 10-element array called balance of type double, use this statement:

double balance[10];

## Initializing Arrays:

You can initialize C++ array elements either one by one or using a single statement as follows:

double balance[5]={1000.0,2.0,3.4,17.0,50.0};

The number of values between braces { } can not be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array:

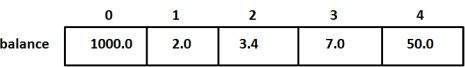
If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write:

double balance[]={1000.0,2.0,3.4,17.0,50.0};

You will create exactly the same array as you did in the previous example.

balance[4]=50.0;

The above statement assigns element number 5th in the array a value of 50.0. Array with 4th index will be 5th, i.e., last element because all arrays have 0 as the index of their first element which is also called base index. Following is the pictorial representaion of the same array we discussed above:



## Accessing Array Elements:

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

# C++ Strings

C++ provides following two types of string representations:

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

The C-Style Character String:

The C-style character string originated within the C language and continues to be supported within C++. This string is actually a one-dimensional array of characters which is terminated by a **null** character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a **null**.

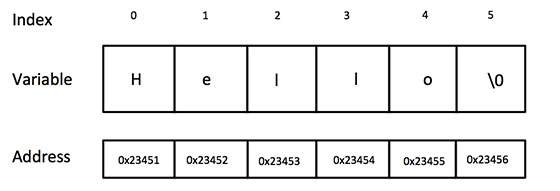
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

char greeting[6]={'H','e','l','l','o','\0'};

If you follow the rule of array initialization, then you can write the above statement as follows:

char greeting[]="Hello";

Following is the memory presentation of above defined string in C/C++:



Actually, you do not 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. Let us try to print above-mentioned string:

|  |  |
| --- | --- |
| **S.N.** | **Function & Purpose** |
| 1 | **strcpy(s1, s2);**  Copies string s2 into string s1. |
| 2 | **strcat(s1, s2);**  Concatenates string s2 onto the end of string s1. |
| 3 | **strlen(s1);**  Returns the length 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 character ch in string s1. |
| 6 | **strstr(s1, s2);**  Returns a pointer to the first occurrence of string s2 in string s1. |

# C++ Pointers

C++ pointers are easy and fun to learn. Some C++ tasks are performed more easily with pointers, and other C++ tasks, such as dynamic memory allocation, cannot be performed without them.

## What Are Pointers?

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

## Using Pointers in C++:

There are few important operations, which we will do with the pointers very frequently. **(a)** we define a pointer variables **(b)** assign the address of a variable to a pointer and **(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. Following example makes use of these operations:

# C++ Classes and Objects

The main purpose of C++ programming is to add object orientation to the C programming language and classes are the central feature of C++ that supports object-oriented programming and are often called user-defined types.

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called members of the class.

## C++ Class Definitions:

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows:

## Classes & Objects in Detail:

So far, you have got very basic idea about C++ Classes and Objects. There are further interesting concepts related to C++ Classes and Objects which we will discuss in various sub-sections listed below:

|  |  |
| --- | --- |
| **Concept** | **Description** |
| [**Class member functions**](http://www.tutorialspoint.com/cplusplus/cpp_class_member_functions.htm) | A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. |
| [**Class access modifiers**](http://www.tutorialspoint.com/cplusplus/cpp_class_access_modifiers.htm) | A class member can be defined as public, private or protected. By default members would be assumed as private. |
| [**Constructor & destructor**](http://www.tutorialspoint.com/cplusplus/cpp_constructor_destructor.htm) | A class constructor is a special function in a class that is called when a new object of the class is created. A destructor is also a special function which is called when created object is deleted. |
| [**C++ copy constructor**](http://www.tutorialspoint.com/cplusplus/cpp_copy_constructor.htm) | The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. |
| [**C++ friend functions**](http://www.tutorialspoint.com/cplusplus/cpp_friend_functions.htm) | A **friend** function is permitted full access to private and protected members of a class. |
| [**C++ inline functions**](http://www.tutorialspoint.com/cplusplus/cpp_inline_functions.htm) | With an inline function, the compiler tries to expand the code in the body of the function in place of a call to the function. |
| [**The this pointer in C++**](http://www.tutorialspoint.com/cplusplus/cpp_this_pointer.htm) | Every object has a special pointer **this** which points to the object itself. |
| [**Pointer to C++ classes**](http://www.tutorialspoint.com/cplusplus/cpp_pointer_to_class.htm) | A pointer to a class is done exactly the same way a pointer to a structure is. In fact a class is really just a structure with functions in it. |
| [**Static members of a class**](http://www.tutorialspoint.com/cplusplus/cpp_static_members.htm) | Both data members and function members of a class can be declared as static. |

# C++ Inheritance

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base**class, and the new class is referred to as the **derived** class.

The idea of inheritance implements the **is a** relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

## Base & Derived Classes:

A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form:

Where access-specifier is one of **public, protected,** or **private**, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

Consider a base class **Shape** and its derived class **Rectangle** as follows:

Access Control and Inheritance:

A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class.

We can summarize the different access types according to who can access them in the following way:

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **public** | **protected** | **private** |
| Same class | yes | yes | yes |
| Derived classes | yes | yes | no |
| Outside classes | yes | no | no |

A derived class inherits all base class methods with the following exceptions:

* Constructors, destructors and copy constructors of the base class.
* Overloaded operators of the base class.
* The friend functions of the base class.

Type of Inheritance:

When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance. The type of inheritance is specified by the access-specifier as explained above.

We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used. While using different type of inheritance, following rules are applied:

* **Public Inheritance:** When deriving a class from a **public** base class, **public** members of the base class become **public** members of the derived class and **protected** members of the base class become**protected** members of the derived class. A base class's **private**members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.
* **Protected Inheritance:** When deriving from a **protected** base class, **public** and **protected**members of the base class become**protected** members of the derived class.
* **Private Inheritance:** When deriving from a **private** base class,**public** and **protected** members of the base class become **private**members of the derived class.

Multiple Inheritances:

A C++ class can inherit members from more than one class and here is the extended syntax:

# Interfaces in C++ (Abstract Classes)

An interface describes the behavior or capabilities of a C++ class without committing to a particular implementation of that class.

The C++ interfaces are implemented using **abstract classes** and these abstract classes should not be confused with data abstraction which is a concept of keeping implementation details separate from associated data.

The purpose of an **abstract class** (often referred to as an ABC) is to provide an appropriate base class from which other classes can inherit. Abstract classes cannot be used to instantiate objects and serves only as an **interface**. Attempting to instantiate an object of an abstract class causes a compilation error.

Thus, if a subclass of an ABC needs to be instantiated, it has to implement each of the virtual functions, which means that it supports the interface declared by the ABC. Failure to override a pure virtual function in a derived class, then attempting to instantiate objects of that class, is a compilation error.

Classes that can be used to instantiate objects are called **concrete classes**

## Designing Strategy:

An object-oriented system might use an abstract base class to provide a common and standardized interface appropriate for all the external applications. Then, through inheritance from that abstract base class, derived classes are formed that all operate similarly.

The capabilities (i.e., the public functions) offered by the external applications are provided as pure virtual functions in the abstract base class. The implementations of these pure virtual functions are provided in the derived classes that correspond to the specific types of the application.

This architecture also allows new applications to be added to a system easily, even after the system has been defined.

# C++ Overloading (Operator and Function)

C++ allows you to specify more than one definition for a **function** name or an**operator** in the same scope, which is called **function overloading** and**operator overloading** respectively.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

When you call an overloaded **function** or **operator**, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called **overload resolution**.

## Function overloading in C++:

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You can not overload function declarations that differ only by return type.

## Operators overloading in C++:

You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

declares the addition operator that can be used to **add** two Box objects and returns final Box object. Most overloaded operators may be defined as ordinary non-member functions or as class member functions. In case we define above function as non-member function of a class then we would have to pass two arguments for each operand as follows:

# Polymorphism in C++

The word **polymorphism** means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

## Virtual Function:

A **virtual** function is a function in a base class that is declared using the keyword **virtual**. Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function.

What we do want is the selection of the function to be called at any given point in the program to be based on the kind of object for which it is called. This sort of operation is referred to as **dynamic linkage**, or **late binding**.

## Pure Virtual Functions:

It's possible that you'd want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class.

**Assignment-1 (Intro to C++ how to create & run)**

#include<iostream.h>//header File Input Output Stream

#include<stdio.h> //Standarad input Output

#include<conio.h> //console input output

void main() //main program

{

clrscr(); //to clear the screen

cout<<"my name is devang.\n"; //to print on display

cout<<"i reside at malad";

cout<<"\n\t\tquary road,";

cout<<"\n\t\tmalad(East),";

cout<<"\n\t\tmumbai 400097";

getch();

}

Explanation :-

#include<stdio.h>🡪This is header file standard input out

#include<conio.h>🡪This is also header file Console Input Out put

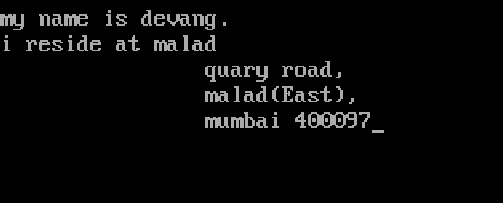
Void main() 🡪 Main program where u have to write code

COUT 🡪 It will show out put on turbo C

Getch() 🡪 it will hold the output.

Clrscr(); 🡪 it will clear the screen

OUTPUT



**Assignment- 2 (Declaring And Intitalizing Variables)**

#include<iostream.h>//header File Input Output Stream

#include <stdio.h>

#include <conio.h>

void main()

{

//Declaring And Intitalizing Variables

char xyz= 'A';

int inum= 21;

float fnum=87.65;

clrscr();

//Displaying the values with Conversion And Escape Characters

cout<<"\n\n";

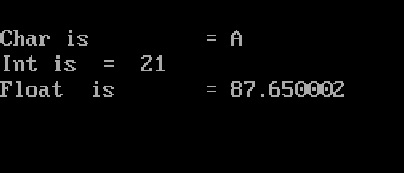
cout<<"Char is \t= "<<xyz;

cout<<"\nInt is \t= "<<inum;

cout<<"\nFloat is \t= "<<fnum;

getch();

}



**Assignment- 3 (Use of cout() and cin() function with output)**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

//Declaring Variables

int rollno;

float height;

char abc[50];

//Use of cout() and cin() function

clrscr();

cout<<"\nEnter Your name ";

cin>>abc;

cout<<"\nEnter Your Roll No ";

cin>>rollno;

cout<<"\nEnter Your height ";

cin>>height;

//Displaying the values entered

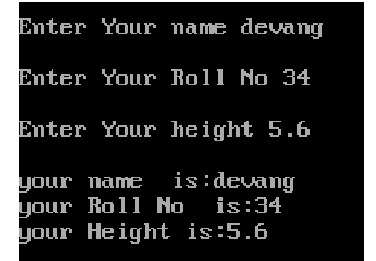
cout<<"\nyour name is:"<<abc;

cout<<"\nyour Roll No is:"<<rollno;

cout<<"\nyour Height is:"<<height;

getch();

}



**Assignment- 4 (Arithmetic Performance)**

#include<iostream.h>

#include <stdio.h>

#include <conio.h>

void main()

{

//Declaration and Intialization of the variable

int a,b,c,d;

int sum, multi, div,remainder, minus, increase, decrease;

c=25;

d=12;

cout<<"\nEnter First Number: ";

cin>>a;

cout<<"\nEnter Second Number: ";

cin>>b;

// Use of Arithmatic operators

sum = a+b;//Addition

minus = a-b;//Subtraction

multi = a\*b;//Multiplication

div = b/a;//Division

remainder = a%b;//Modular Division

increase = ++c;

decrease = --d;

//Displaying the results

clrscr();

cout<<"\nSum is "<<sum;

cout<<"\nsubstraction is "<<minus;

cout<<"\nmultiplication is "<<multi;

cout<<"\nDivision is "<<div;

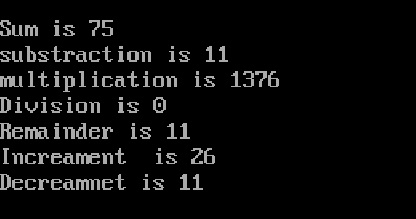
cout<<"\nRemainder is "<<remainder;

cout<<"\nIncreament is "<<increase;

cout<<"\nDecreamnet is "<<decrease;

getch();

}



**Assignment- 5 (Swap Program)**

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

void main()

{

int a,b,c;

clrscr();

cout<<"\nenter the first number:";

cin>>a;

cout<<"\nenter the Second number:";

cin>>b;

// Displayingthe numbers before interchanging

cout<<"\n\n printing the numbers before interchanging";

cout<<"\nthe first number is:"<<a;

cout<<"\nthe second number is:"<<b;

//Interchangingthe numbers

c=a;

a=b;

b=c;

// Displayingthe numbers after interchanging

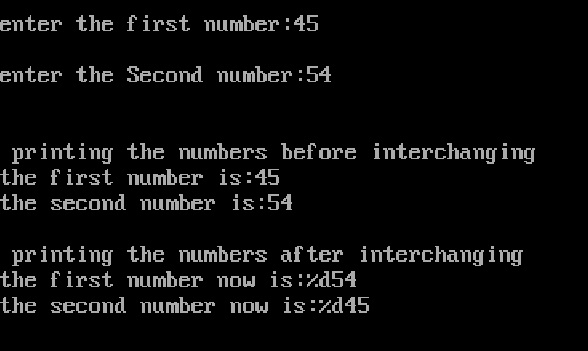
cout<<"\n\n printing the numbers after interchanging";

cout<<"\nthe first number now is:%d"<<a;

cout<<"\nthe second number now is:%d"<<b;

getch();

}



**Assignment- 6 (Salary Calculation With If Else)**

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

#include<string.h>

void main()

{

float basic,da,hra,salary;

char d[50];

clrscr();

cout<<"\nEnter Your Name : ";

cin>>d;

cout<<"\n enter the basic salary :";

cin>>basic;

// Calculate the da,hra and salary

da=basic\*40/100;

hra=basic\*25/100;

salary=basic+da+hra;

//Displaying the details

cout<<"\nyour name is: "<<d;

cout<<"\n\n salary details :";

cout<<"\n Basic salary is: "<<basic;

cout<<"\n Dearness Allowance is:"<<da;

cout<<"\n House Rent Allowance is: "<<hra;

cout<<"\n\n Total salary earned \n"<<salary;

if(salary>=80000)

{

cout<<"\nYou are ceo ";

}

else if(salary>=50000)

{

cout<<"\nYou are purchase manager";

}

else if(salary>=25000)

{

cout<<"\nyou are sales manager";

}

else

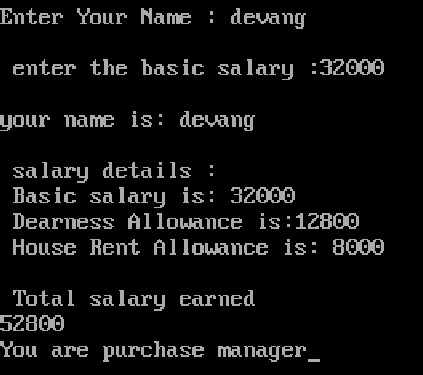
{

cout<<"\nyou are clerk";

}

getch();

}



**Assignment- 7 (Use Of If Else)**

#include<iostream.h>

#include <stdio.h>

#include <conio.h>

void main()

{

int num1,num2, sum;

clrscr();

cout<<"enter two numbers:";

cin>>num1>>num2;

sum= num1+num2;

if(sum>100)

{

cout<<"\n the sum of two numbers is greater than 100\n";

}

else

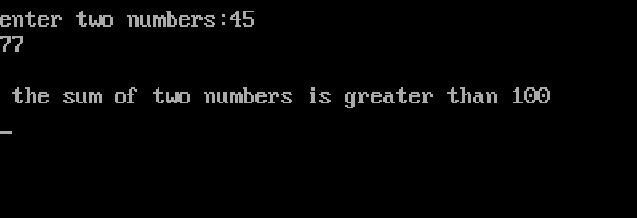
{

cout<<"\n the sum of two numbers is smaller than 100\n";

}

getch();

}



**Assignment- 8 (Voting System Using if Else)**

#include <iostream.h>

#include <string.h>

#include <stdio.h>

#include <conio.h>

void main()

{

int age;

char name[100];

clrscr();

cout<<"\n enter your name:";

cin>>name;

cout<<"\n enter your age:";

cin>>age;

if (age>=19)

{

cout<<"\n you are eligiable for voting\n";

}

else

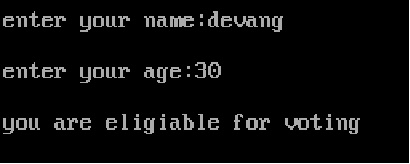
{

cout<<"\n you are not eligible for voting\n";

}

getch();

}



**Assignment- 9 (Use Of Ladder If)**

#include<iostream.h>

#include <stdio.h>

#include <conio.h>

void main()

{

int num1,num2, num3;

clrscr();

cout<<"\n enter 3 numbers:";

cin>>num1>>num2>>num3;

if((num1>num2) && (num1>num3))

{

cout<<"\n the largest of three numbers is \n"<<num1;

}

if((num2>num1) && (num2>num3))

{

cout<<"\n the largest of three numbers is\n"<<num2;

}

if((num3>num1) && (num3>num2))

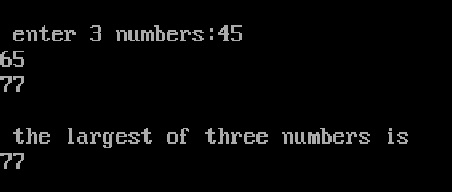
{

cout<<"\n the largest of three numbers is\n"<<num3;

}

getch();

}



**Assignment- 10 (Switch Program)**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

void main()

{

//Declaration and Intialization of the variable

int a;

clrscr();

start : cout<<"\nEnter Number with in one to seven: ";

cin>>a;

cout<<"you have entered number is \n"<<a;

switch(a)

{

case 1 : cout<<"you have selected monday";

break;

case 2 : cout<<"you have selected tuesday";

break;

case 3 : cout<<"you have selected wednesday";

break;

case 4 : cout<<"you have selected thursday";

break;

case 5 : cout<<"you have selected friday";

break;

case 6 : cout<<"you have selected saturday";

break;

case 7 : cout<<"you have selected sunday";

break;

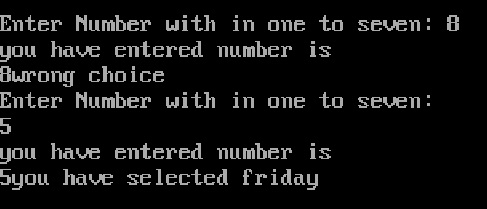
default : cout<<"wrong choice";

goto start;

}

getch();

}



**Assignment- 11 (While Loop with calculation)**

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

void main()

{

int x=1,n,r;

clrscr();

cout<<"enter any number ";

cin>>n;

while(x<=10)

{

r=n\*x;

cout<<"\n"<<n;

cout<<"="<<x;

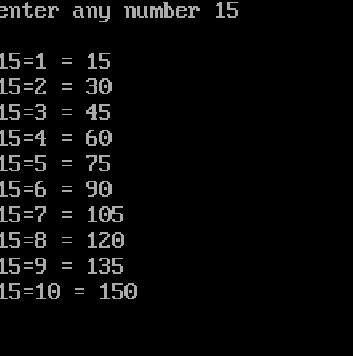
cout<<" "<<r;

x++;

}

getch();

}



**Assignment- 12 (Star Printing using While Loop)**

#include<iostream.h>

#include <stdio.h>

#include <conio.h>

void main()

{

int nstars=1, stars;

clrscr();

while(nstars <=10)

{

stars=1;

while (stars <= nstars)

{

cout<<"\*";

stars++;

}

cout<<"\n";

nstars++;

}

getch();

}



**Assignment- 13 (Do While Loop )**

#include<iostream.h>

#include <stdio.h>

#include <conio.h>

void main ()

{

/\* local variable definition \*/

int a = 10;

clrscr();

/\* do loop execution \*/

do

{

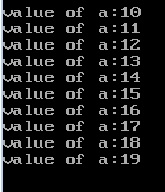
cout<<"\nvalue of a:"<< a;

a = a + 1;

}while( a < 20 );

getch();

}



**Assignment- 14 (table printing using For Loop)**

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

void main()

{

int a,b,c;

clrscr();

cout<<"enter table number : \n";

cin>>b;

for(a=1;a<=10;a++)

{

c=a\*b;

cout<<"\n "<<b;

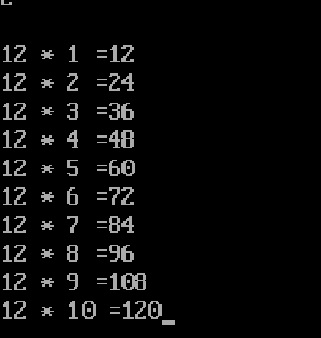
cout<<" \* "<<a;

cout<<" ="<<c;

}

getch();

}



**Assignment- 15 (One Dimension Array)**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

void main()

{

int x[3];

x[0]=10;

x[1]=20;

x[2]=30;

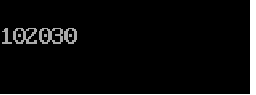
cout<<x[0];

cout<<x[1];

cout<<x[2];

getch();

}



**Assignment- 16 (Two Dimension Array)**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

void main()

{

int x[2][3];

//first row of array

x[0][0]=10;

x[0][1]=20;

x[0][2]=30;

clrscr();

cout<<x[0][0];

cout<<"\n";

cout<<x[0][1];

cout<<"\n";

cout<<x[0][2];

cout<<"\n";

//secon row of array

x[1][0]=40;

x[1][1]=50;

x[1][2]=60;

cout<<x[1][0];

cout<<"\n";

cout<<x[1][1];

cout<<"\n";

cout<<x[1][2];

getch();

}



**Assignment- 17 (String Compare Function)**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char a[50], b[50];

clrscr();

cout<<"Enter the first string\n";

cin>>a;

cout<<"Enter the second string\n";

cin>>b;

if (strcmp(a,b) == 0)

{

cout<<"Entered strings are equal.\n";

}

else

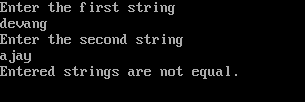
{

cout<<"Entered strings are not equal.\n";

}

getch();

}



**Assignment- 18 (Strcat use for joining two string)**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char a[20],b[20];

clrscr();

cout<<"ENTER 1st THE STRING";

cin>>a;

cout<<"ENTER 2ndTHE STRING";

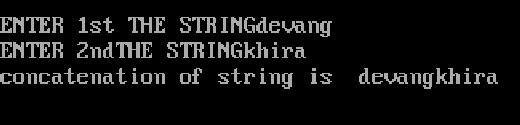
cin>>b;

strcat(a,b);

cout<<"concatenation of string is "<<a;

getch();

}



**Assignment- 19(string copy &str lenth)**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

char a[20],b[20];

int len;

clrscr();

cout<<"ENTER 1st THE STRING";

cin>>a;

len =strlen(a);

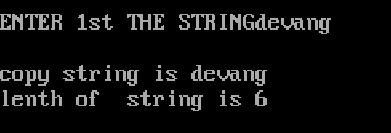
strcpy(b,a);

cout<<" \ncopy string is "<<b;

cout<<"\nlenth of string is "<<len;

getch();

}



**Assignment- 20(Structure Program)**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

struct Books

{

char title[50];

char author[50];

char subject[100];

int book\_id;

};

void main( )

{

struct Books Book1; /\* Declare Book1 of type Book \*/

struct Books Book2; /\* Declare Book2 of type Book \*/

clrscr();

/\* book 1 specification \*/

strcpy( Book1.title, "C Programming");

strcpy( Book1.author, "Nuha Ali");

strcpy( Book1.subject, "C Programming Tutorial");

Book1.book\_id = 6495407;

/\* book 2 specification \*/

strcpy( Book2.title, "Telecom Billing");

strcpy( Book2.author, "Zara Ali");

strcpy( Book2.subject, "Telecom Billing Tutorial");

Book2.book\_id = 6495700;

/\* print Book1 info \*/

cout<<"Book 1 title : "<<Book1.title;

cout<<"\nBook 1 author : "<<Book1.author;

cout<<"\nBook 1 subject : "<<Book1.subject;

cout<<"\nBook 1 book\_id : "<<Book1.book\_id;

/\* print Book2 info \*/

cout<<"\nBook 2 title : "<<Book2.title;

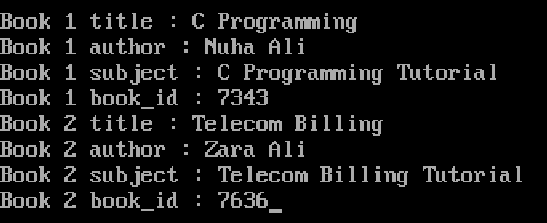
cout<<"\nBook 2 author : "<<Book2.author;

cout<<"\nBook 2 subject : "<<Book2.subject;

cout<<"\nBook 2 book\_id : "<<Book2.book\_id;

getch();

}



**Assignment- 21 Array Using For Loop**

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

int dev [] = {16, 2, 77, 40, 12071};

int n;

void main ()

{

clrscr();

for ( n=0 ; n<5 ; n++ )

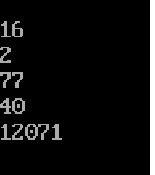
{

cout<<"\n"<<dev[n];

}

getch();

}



**Assignment- 22 Function In C++**

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

int addition (int a, int b)

{

int r;

clrscr();

r=a+b;

return (r);

}

void main ()

{

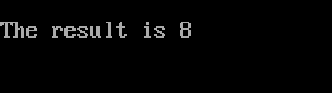
int z;

z = addition (5,3);

cout<< "The result is " << z;

getch();

}



**Assignment- 23 Another Example Of Function**

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

int subtraction (int a, int b)

{

int r;

r=a-b;

return (r);

}

void main ()

{

int x=5, y=3, z;

z = subtraction (7,2);

clrscr();

cout<< "The first result is " << z << '\n';

cout<< "The second result is " << subtraction (7,2) << '\n';

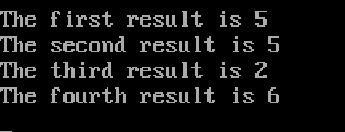
cout<< "The third result is " << subtraction (x,y) << '\n';

z= 4 + subtraction (x,y);

cout<< "The fourth result is " << z << '\n';

getch();

}



**Assignment- 24 Class & Object Simple Example**

C++ Class Definitions:

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

//oop : object oriented progeamming {class & object}

class Demo

{

public:

void disp1()

{

cout<<"\nPUBLIC SECTION";

}

void disp2()

{

cout<<"\nPublic Part 2";

}

};

void main()

{

clrscr();

Demo obj; //object is class variable

obj.disp1(); //accessing member of class

obj.disp2();//acessing another member of class

getch();

}



|  |  |  |  |
| --- | --- | --- | --- |
| Public inheritance | | | |
| Base access specifier | Derived access specifier | Derived class access? | Public access? |
| Public | Public | Yes | Yes |
| Private | Private | No | No |
| Protected | Protected | Yes | No |

|  |  |  |  |
| --- | --- | --- | --- |
| Private inheritance | | | |
| Base access specifier | Derived access specifier | Derived class access? | Public access? |
| Public | Private | Yes | No |
| Private | Private | No | No |
| Protected | Private | Yes | No |

|  |  |  |  |
| --- | --- | --- | --- |
| Protected inheritance | | | |
| Base access specifier | Derived access specifier | Derived class access? | Public access? |
| Public | Protected | Yes | No |
| Private | Private | No | No |
| Protected | Protected | Yes | No |

**Assignment- 25oop : object oriented programming {class & object}**

#include<iostream.h>

#include<conio.h>

//oop : object oriented programming {class & object}

class Sum

{

int x,y,a,m,s,d,r;

public:

void input()

{

cout<<"\nEnter Any Two Number : ";

cin>>x>>y;

}

void process()

{

a=x+y;

m=x\*y;

s=x-y;

d=x/y;

r=x%y;

}

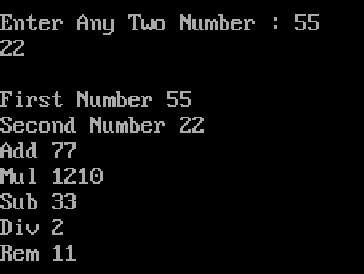
void output()

{

cout<<"\nFirst Number "<<x;

cout<<"\nSecond Number "<<y;

cout<<"\nAdd "<<a;

cout<<"\nMul "<<m;

cout<<"\nSub "<<s;

cout<<"\nDiv "<<d;

cout<<"\nRem "<<r;

}

};

void main()

{

clrscr();

Sum obj; //object is class variable

obj.input();

obj.process();

obj.output();

getch();

}

**Assignment- 26 Salary Sheet Class & Object**

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

class salary

{

int b,h,t,g,pf,tx,n;

public :

void input()

{

cout<<"\nEnter Basic Salary ";

cin>>b;

}

void process()

{

g=b+h+t;

h=b\*0.10;

pf=b\*0.010;

tx=b\*0.05;

t=b\*0.010;

n=pf+t-g;

}

void output()

{

cout<<"\nHRA ="<<h;

cout<<"\nTravelling Allowance = "<<t;

cout<<" \nGross Salary = "<<g;

cout<<"\nPF = "<<pf;

cout<<"\nTax Deduction = "<<tx;

cout<<" \nNet Salary = "<<n;

}

};

void main()

{

clrscr();

salary obj;

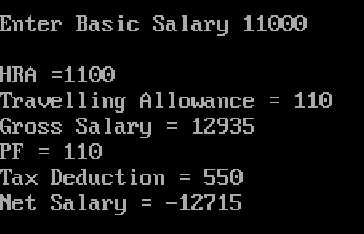
obj.input();

obj.process();

obj.output();

getch();

}



**Assignment- 27 SWITCH CASE WITH CLASS**

#include<iostream.h>

#include<conio.h>

const int MAX=100;

class Details

{

private:

int salary;

float roll;

clrscr();

public:

void getname()

{

cout<<"\n Enter the salary:";

cin>>salary;

cout<<"\n Enter the roll:";

cin>>roll;

}

void putname()

{

cout<<"Employees salary"<<salary<<"and roll is"<<roll<<"\n";

}

};

void main()

{

Details det[MAX];

int n=0;

char ans;

clrscr();

do{

cout<<"Enter the employee number :"<<n+1;

det[n++].getname();

cout<<"Enter another(y/n)?:";

cin>>ans;

}

while(ans!='n');

for(int j=0;j<n;j++)

{

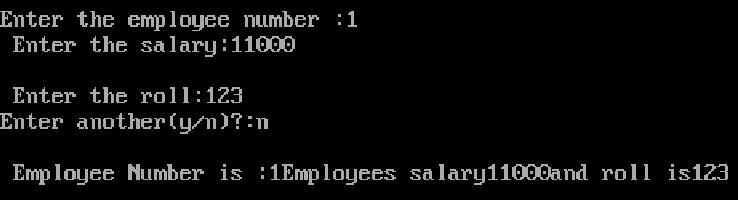
cout<<"\n Employee Number is :"<<j+1;

det[j].putname();

}

getch();

}



**Assignment- 28The public members:**

A public member is accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function as shown in the following example:

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

// Class Declaration

class person

{

//Access - Specifier

public:

//Varibale Declaration

char name[20];

int number;

};

//Main Function

void main()

{

// Object Creation For Class

person obj;

clrscr();

//Get Input Values For Object Varibales

cout<<"Enter the Name :";

cin>>obj.name;

cout<<"Enter the Number :";

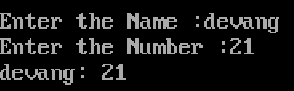
cin>>obj.number;

//Show the Output

cout<< obj.name << ": " << obj.number << endl;

getch();

}



**Assignment- 29The private members:**

A private member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.

By default all the members of a class would be private, for example in the following class width is a private member, which means until you label a member, it will be assumed a private member:

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

class Box

{

public:

double length;

void setWidth( double wid );

double getWidth( void );

private:

double width;

};

// M

ember functions definitions

double Box::getWidth(void)

{

return width ;

}

void Box::setWidth( double wid )

{

width = wid;

}

// Main function for the program

void main( )

{

Box box;

// set box length without member function

box.length = 10.0; // OK: because length is public

clrscr();

cout<< "Length of box : " << box.length <<endl;

// set box width without member function

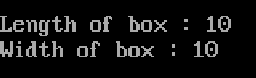
// box.width = 10.0; // Error: because width is private

box.setWidth(10.0); // Use member function to set it.

cout<< "Width of box : " << box.getWidth() <<endl;

getch();

}



**Assignment- 30The protected members:**

A protected member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

You will learn derived classes and inheritance in next chapter. For now you can check following example where I have derived one child class SmallBox from a parent class Box.

Following example is similar to above example and here width member will be accessible by any member function of its derived class SmallBox.

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

class Box

{

protected:

double width;

};

class SmallBox:Box // SmallBox is the derived class.

{

public:

void setSmallWidth( double wid );

double getSmallWidth( void );

};

// Member functions of child class

double SmallBox::getSmallWidth(void)

{

return width ;

}

void SmallBox::setSmallWidth( double wid )

{

width = wid;

}

// Main function for the program

void main( )

{

SmallBox box;

// set box width using member function

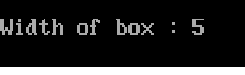
box.setSmallWidth(5.0);

clrscr();

cout<< "Width of box : "<< box.getSmallWidth() << endl;

getch();

}



**Assignment- 31 Call by function & value**

#include<iostream.h>

#include<conio.h>

class Square

{

int x,s;

public:

void input(int n) //function defination n is formal arguments

{

x=n;

}

void process()

{

s=x\*x;

}

void output()

{

cout<<"\nSquare "<<s;

}

};

void main()

{

int a;

clrscr();

Square obj; //object is class variable

cout<<"\nEnter Any Number : ";

cin>>a;

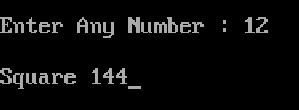
obj.input(a); //call a is actual arguments | call by value

obj.process();

obj.output();

getch();

}



**Assignment- 32 Single Inheritence**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

class base

{

public:

void disp()

{

cout<<"\nBASE CLASS";

}

};

class derived : public base

{

public:

void show()

{

cout<<"\nDERIVED CLASS";

disp();

}

};

void main()

{

clrscr();

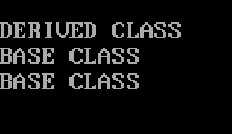
derived dobj;

dobj.show();

dobj.disp();

getch();

}



**Assignment- 33 Multilple Inheritence**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

//multiple

class base1

{

public:

void disp1()

{

cout<<"\nBASE1 CLASS";

}

};

class base2

{

public:

void disp2()

{

cout<<"\nBASE2 CLASS";

}

};

class derived : public base1, public base2

{

public:

void show()

{

cout<<"\nDERIVED CLASS";

}

};

void main()

{

clrscr();

derived dobj;

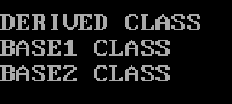
dobj.show();

dobj.disp1();

dobj.disp2();

getch();

}



**Assignment- 34 Hierarchical Inheritence**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

//hierarchical

class base

{

public:

int x;

void disp()

{

cout<<"\nBASE CLASS";

}

};

class d1 : public base

{

public:

void disp2()

{

x=56;

cout<<"\nDerived1 CLASS "<<x;

}

};

class d2 : public base

{

public:

void show()

{

cout<<"\nDERIVED2 CLASS";

}

};

void main()

{

clrscr();

d1 dobj1;

dobj1.disp();

dobj1.disp2();

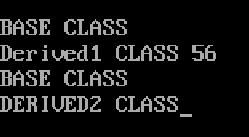
d2 dobj2;

dobj2.disp();

dobj2.show();

getch();

}



**Assignment- 35 Function overloading in C++:**

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You can not overload function declarations that differ only by return type.

Following is the example where same function print() is being used to print different data types:

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

class printData

{

public:

void print(int i) {

cout<< "Printing int: " << i << endl;

}

void print(double f) {

cout<< "Printing float: " << f << endl;

}

void print(char\* c) {

cout<< "Printing character: " << c << endl;

}

};

void main()

{

printData pd;

clrscr();

// Call print to print integer

pd.print(5);

// Call print to print float

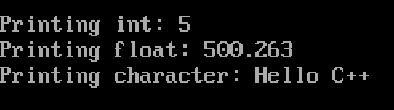
pd.print(500.263);

// Call print to print character

pd.print("Hello C++");

getch();

}



**Assignment- 36 Operators overloading in C++:**

You can redefine or overload most of the built-in operators available in C++. Thus a programmer can use operators with user-defined types as well.

Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. Like any other function, an overloaded operator has a return type and a parameter list.

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

class Box

{

public:

double getVolume(void)

{

return length \* breadth \* height;

}

void setLength( double len )

{

length = len;

}

void setBreadth( double bre )

{

breadth = bre;

}

void setHeight( double hei )

{

height = hei;

}

// Overload + operator to add two Box objects.

Box operator+(const Box& b)

{

Box box;

box.length = this->length + b.length;

box.breadth = this->breadth + b.breadth;

box.height = this->height + b.height;

return box;

}

private:

double length; // Length of a box

double breadth; // Breadth of a box

double height; // Height of a box

};

// Main function for the program

void main()

{

clrscr();

Box Box1; // Declare Box1 of type Box

Box Box2; // Declare Box2 of type Box

Box Box3; // Declare Box3 of type Box

double volume = 0.0; // Store the volume of a box here

// box 1 specification

Box1.setLength(6.0);

Box1.setBreadth(7.0);

Box1.setHeight(5.0);

// box 2 specification

Box2.setLength(12.0);

Box2.setBreadth(13.0);

Box2.setHeight(10.0);

// volume of box 1

volume = Box1.getVolume();

cout<< "Volume of Box1 : " << volume <<endl;

// volume of box 2

volume = Box2.getVolume();

cout<< "Volume of Box2 : " << volume <<endl;

// Add two object as follows:

Box3 = Box1 + Box2;

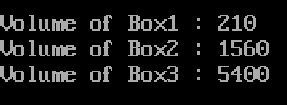
// volume of box 3

volume = Box3.getVolume();

cout<< "Volume of Box3 : " << volume <<endl;

getch();

}



**Assignment- 37The Class Constructor:**

A class constructor is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

Following example explains the concept of constructor:

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

class Example

{

// Variable Declaration

int a,b;

public:

//Constructor without Argument

Example()

{

// Assign Values In Constructor

a=50;

b=100;

cout<<"\nI am Constructor";

}

//Constructor with Argument

Example(int x,int y)

{

// Assign Values In Constructor

a=x;

b=y;

cout<<"\nIm Constructor";

}

void Display() {

cout<<"\nValues :"<<a<<"\t"<<b;

}

};

void main()

{

clrscr();

Example Object(10,20);

Example Object2;

// Constructor invoked.

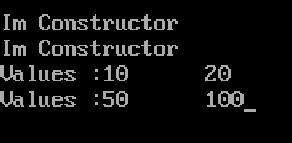
Object.Display();

Object2.Display();

// Wait For Output Screen

getch();

}



**Assignment- 38 Parameterized Constructor**

A constructor with arguments is called a Parameterized Constructor. With the help of such constructor the data elements of various objects can be initialized with different values. This is performed by passing different values to arguments of the constructor function while creating an object.

/\* Program to show use of Parameterized Constructor \*/

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

#include<string.h>

class student

{

public:

student(char name[50],int age,int rol,char clas[50])

{

cout<<"\n R.No.\tAge\tName\tClass";

cout<<"\n "<<rol<<"\t"<<age<<"\t"<<name<<"\t"<<clas;

}

};

void main()

{

clrscr();

char name[50],clas[50];

int age,rol;

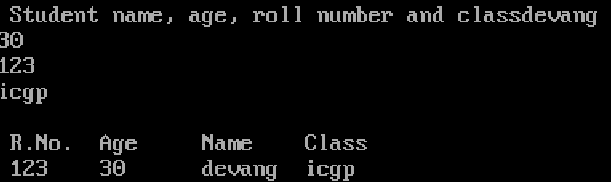
cout<<"\n Student name, age, roll number and class";

cin>>name>>age>>rol>>clas;

class student s(name,age,rol,clas);

getch();

}



**Assignment- 39 Constructor Overloading**

/\* Program to overload arithmetic operators using constructors\*/

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

class operation

{

public:

int a,b,c,e;

float d;

operation()

{

cout<<"\n Enter any interger value of a and b";

cin>>a>>b;

c=a+b;

cout<<"\n Addition of "<<a<<" and "<<b<<" is "<<c;

}

operation(int a,float b)

{

d=a-b;

cout<<"\n Subtraction of "<<b<<" from "<<a<<" is "<<d;

}

operation(operation &o)

{

e=(o.c);

int f=e\*10;

cout<<"\n\n Multiplication of "<<e<<" and 10 is "<<f;

}

};

void main()”

{

clrscr();

int a;

float b;

class operation o1;

operation o3(o1);

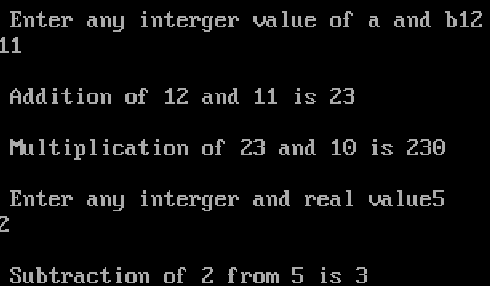
cout<<"\n\n Enter any interger and real value";

cin>>a>>b;

operation o2(a,b);

getch();

}



**Assignment- 40 DESTRUCTORS**

These are the functions that are complimentary to constructors. These are used to re-initialize objects when they are destroyed. A Destructor is called when an OBJECT of the CLASS goes out of scope, or when the memory space used by it is de-located with the help of delete operator.

Basic thing about Destructors

1. Destructor re-initializes the value of constructors.

2. Destructor can’t be overloaded.

PROPERTIES OF Destructors

1. Destructor functions are evoked automatically when the objects are destroyed.

2. Destructor may not be static.

3. Destructor can’t be inherited.

4. Destructor does not take any value and also does not return any value.

5. An object of a CLASS with a destructor can’t become member of UNION.

6. Member function may be called or accessed within the destructor.

7. It is not possible to take address of destructor.

8. No argument is provided to destructor.

9. Destructor level also access 3 access levels private, protected and public similar to constructors.

10. If a CLASS has a destructor, each object of that class will be re-initialized before the object goes out of scope.

/\* Program to show use of destructor\*/

#include<iostream.h>

#include<conio.h>

class stud

{

public:

char name[10],clas[10];

int age;

stud()

{

cout<<"\n Enter student name, age and class";

cin>>name>>age>>clas;

cout<<"\n Age\tName\tClass";

cout<<"\n "<<age<<"\t"<<name<<"\t"<<clas;

}

~stud()

{

cout<<"\n\n Now this is the application of Destructor";

cout<<"\n\n Enter student name, age and class";

cin>>name>>age>>clas;

cout<<"\n Age\tName\tClass";

cout<<"\n "<<age<<"\t"<<name<<"\t"<<clas;

}

};

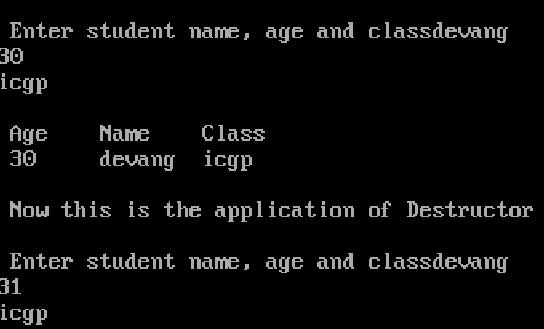
main()

{

class stud s;

getch();

}



**Assignment- 41 Polymorphism**

Before getting into this section, it is recommended that you have a proper understanding of pointers and class inheritance. If any of the following statements seem strange to you, you should review the indicated sections:

Statement: Explained in:

int a::b(c) {}; Classes

a->b Data Structures

class a: public b; Friendship and inheritance

Pointers to base class

One of the key features of derived classes is that a pointer to a derived class is type-compatible with a pointer to its base class. Polymorphism is the art of taking advantage of this simple but powerful and versatile feature, that brings Object Oriented Methodologies to its full potential.

We are going to start by rewriting our program about the rectangle and the triangle of the previous section taking into consideration this pointer compatibility property:

// pointers to base class

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

class CPolygon

{

protected:

int width, height;

public:

void set\_values (int a, int b)

{

width=a; height=b; }

};

class CRectangle: public CPolygon

{

public:

int area ()

{

return (width \* height); }

};

+

class CTriangle: public CPolygon

{

public:

int area ()

{

return (width \* height / 2); }

};

int main () {

CRectangle rect;

CTriangle trgl;

CPolygon \* ppoly1 = &rect;

CPolygon \* ppoly2 = &trgl;

ppoly1->set\_values (4,5);

ppoly2->set\_values (4,5);

cout<< rect.area() << endl;

cout<< trgl.area() << endl;

getch();

}



**Assignment- 42 Virtual members**

A member of a class that can be redefined in its derived classes is known as a virtual member. In order to declare a member of a class as virtual, we must precede its declaration with the keyword virtual:

#include<iostream.h>

#include<conio.h>

class base

{

public:

virtual void disp()

{

cout<<"\nBase class function disp";

}

virtual void show()

{

cout<<"\nBase class function show";

}

};

class sub : public base

{

public:

void disp()

{

cout<<"\nSub class function disp";

}

void show()

{

cout<<"\nSub class function show";

}

};

void main()

{

clrscr();

base bobj;

sub sobj;

base \*bptr;

cout<<"\n--POINT TO BASE--";

bptr=&bobj;

bptr->disp();

bptr->show();

cout<<"\n--POINT TO SUB--";

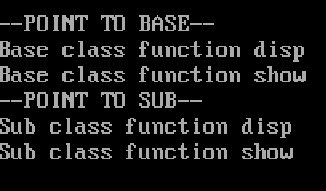
bptr=&sobj;

bptr->disp();

bptr->show();

getch();

}



//writing to file (working with file)

|  |  |
| --- | --- |
| Data Type | Description |
| ofstream | This data type represents the output file stream and is used to create files and to write information to files. |
| ifstream | This data type represents the input file stream and is used to read information from files. |
| fstream | This data type represents the file stream generally, and has the capabilities of both ofstream and ifstream which means it can create files, write information to files, and read information from files. |

**Assignment- 43 Writing to File**

#include<fstream.h>

#include<conio.h>

void main()

{

ofstream out("my.txt");///

char name[20];

int marks;

clrscr();

cout<<"Enter Student Name : ";

cin>>name;//

out<<"Name : "<<name<<"\n";

cout<<"Enter Marks :";

cin>>marks;

out<<"Marks : "<<marks<<"\n";

out.close();

cout<<"\nFile Write Done";

getch();

}

**Assignment- 44 read data from file(working with file)**

#include<fstream.h>

#include<conio.h>

void main()

{

ifstream inf("my.txt");

char name[20];

int marks;

clrscr();

inf>>name;

inf>>marks;

cout<<"\n";

cout<<"\nStudent Name : "<<name<<"\n";

cout<<"Marks : "<<marks<<"\n";

inf.close();

cout<<"\nFile read Done";

getch();

}

**Assignment- 45 Simple Program for Exception Handling Divide by zero Using C++ Programming**

#include<iostream.h>

#include<conio.h>

void main()

{

int a,b,c;

float d;

clrscr();

cout<<"Enter the value of a:";

cin>>a;

cout<<"Enter the value of b:";

cin>>b;

cout<<"Enter the value of c:";

cin>>c;

try

{

if((a-b)!=0)

{

d=c/(a-b);

cout<<"Result is:"<<d;

}

else

{

throw(a-b);

}

}

catch(int i)

{

cout<<"Answer is infinite because a-b is:"<<i;

}

getch();

}

**Assignment- 46 Simple Program for Exception Handling with Multiple Catch Using C++ Programming**

#include<iostream.h>

#include<conio.h>

void test(int x)

{

try

{

if(x>0)

throw x;

else

throw 'x';

}

catch(int x)

{

cout<<"Catch a integer and that integer is:"<<x;

}

catch(char x)

{

cout<<"Catch a character and that character is:"<<x;

}

}

void main()

{

clrscr();

cout<<"Testing multiple catches\n:";

test(10);

test(0);

getch();

}

**Assignment- 47 Data Encapsulation in C++**

Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding.

Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

C++ supports the properties of encapsulation and data hiding through the creation of user-defined types, called classes. We already have studied that a class can contain private, protected and public members. By default, all items defined in a class are private. For example:

Any C++ program where you implement a class with public and private members is an example of data encapsulation and data abstraction. Consider the following example:

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

class Adder{

public:

// constructor

Adder(int i = 0)

{

total = i;

}

// interface to outside world

void addNum(int number)

{

total += number;

}

// interface to outside world

int getTotal()

{

return total;

};

private:

// hidden data from outside world

int total;

};

void main( )

{

clrscr();

Adder a;

a.addNum(10);

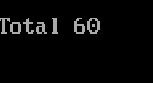
a.addNum(20);

a.addNum(30);

cout<< "Total " << a.getTotal() <<endl;

getch();

}



**Assignment- 48 CLASS SHOPPING**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

class shoping

{

int c,p;

static int s;

char name[150];

public :

void input();

void process();

void output();

};

int shoping::s;

void shoping :: input()

{

cout<<"\nEnter Item Name = ";

cin>>name;

cout<<"\nEnter Item Code = ";

cin>>c;

}

void shoping :: process()

{

if(c<=25)

{

p=100;

s=s+p;

cout<<"\nPrice Is "<<p;

}

else if (c<=50)

{

p=200;

s=s+p;

cout<<"\nPrice Is "<<p;

}

else if (c<=75)

{

p=300;

s=s+p;

cout<<"\nPrice Is "<<p;

}

else if (c<=100)

{

p=400;

s=s+p;

cout<<"\nPrice Is "<<p;

}

else if (c>=100)

{

cout<<"\nWrong Code ";

}

}

void shoping :: output()

{

cout<<"\nYour Item Name is : "<<name;

cout<<"\nYour Code is : "<<c;

cout<<"\nPlz Pay : "<<p;

cout<<"\n Total Amount Is = "<<s;

}

void main ()

{

clrscr();

int i;

shoping sobj[4];

for (i=0;i<=3;i++)

{

cout<<"\n--------------\*-----------";

cout<<"\nDetail Of Purchased Material "<<i+1;

sobj[i].input();

}

cout<<"\n Detail Of Purchased ";

for (i=0;i<=3;i++)

{

cout<<"\n--------------\*-----------";

cout<<"\nPurechased "<<i+1;

sobj[i].process();

}

for (i=0;i<=3;i++)

{

cout<<"\n--------------\*-----------";

cout<<"\nPurechased "<<i+1;

sobj[i].output();

}

getch();

}

**Assignment- 49Abstarct Class using Virtual**

#include<iostream.h>

#include<conio.h>

class base

{

public:

virtual void disp()

{

cout<<"\nBase class function disp";

}

virtual void show()

{

cout<<"\nBase class function show";

}

};

class sub : public base

{

public:

void disp()

{

cout<<"\nSub class function disp";

}

void show()

{

cout<<"\nSub class function show";

}

};

void main()

{

clrscr();

base bobj;

sub sobj;

base \*bptr;

cout<<"\n--POINT TO BASE--";

bptr=&bobj;

bptr->disp();

bptr->show();

cout<<"\n--POINT TO SUB--";

bptr=&sobj;

bptr->disp();

bptr->show();

getch();

}

**Assignment- 50Abstarct Class using Virtual**

#include<iostream.h>

#include<conio.h>

class Base

{

public:

void display() {cout<<"\n Display base";}

virtual void show(){cout<<"\n show base";}

};

class Derived : public Base

{

public:

void display(){cout<<"\n Display derived";}

void show(){cout<<"\n show derived";}

};

int main()

{

Base B;

Derived D;

Base \*bptr;

cout<<"\n bptr points to Base \n";

bptr=&B;

bptr-> display();

bptr-> show();

cout<<"\n\n bptr points to Derived\n";

bptr=&D;

bptr-> display();

bptr-> show();

getch();

}

**Assignment- 51 *(Pointer will read the address value of Variable)***

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

void main()

{

int a=10;

int \*p; /\*pointer variable\*/

p=&a; /\*assign memory address of variable \*/

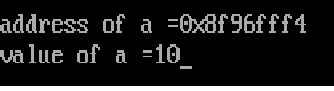
clrscr();

cout<<"address of a ="<<p;

cout<<"\nvalue of a ="<<\*p;

getch();

}



**Assignment- 52 *(Pointer of Pointer)***

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

void main()

{

int x,\*p1,\*\*p2;

x=5;

p1=&x;

p2=&p1;

clrscr();

cout<<"\nx="<<x;

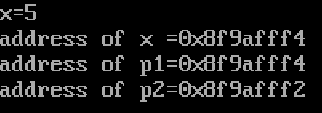
cout<<"\naddress of x ="<<&x;

cout<<"\naddress of p1="<<p1;

cout<<"\naddress of p2="<<p2;

getch();

}



int\*ip;// pointer to an integer

double\*dp;// pointer to a double

float\*fp;// pointer to a float

char\*ch // pointer to character

**Assignment- 53 *(Pointer of Pointer)***

#include<iostream.h>

#include<conio.h>

#include<stdio.h>

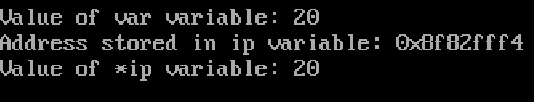
void main ()

{

int var = 20; // actual variable declaration.

int \*ip; // pointer variable

ip = &var; // store address of var in pointer variable



cout<< "Value of var variable: ";

cout<< var << endl;

// print the address stored in ip pointer variable

cout<< "Address stored in ip variable: ";

cout<< ip << endl;

// access the value at the address available in pointer

cout<< "Value of \*ip variable: ";

cout<< \*ip << endl;

getch();

}