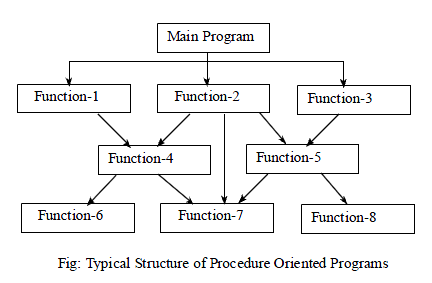
# Overview of Programming Approaches

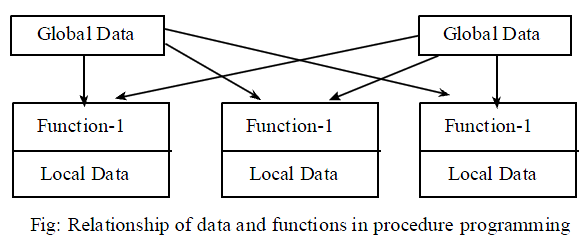
## 1.1 Overview of Procedure-Oriented Programming (POP)

Conventional Programming, using high level languages such as COBOL (Common Business Oriented Language), FORTAN (Formula Translation) and C, is commonly called as procedural oriented programming (POP).

POP basically consists of writing a list of instructions (or actions) for the computer to follow, and organizing these instructions into groups known as function. A typical program structure for procedural programming is shown in the figure below.



In a multi-function program, many important data items are placed as globally. So that they may be accessed by all the functions. Each function may have its own local data. The figure shown below shows the relationship of data and function in a procedure-oriented program.



### Some features of procedure-oriented programming are:

* Emphasis is on doing things (algorithms).
* Large programs are divided into smaller programs called as functions.
* Most of the functions share global data.
* Data move openly around the system from function to function.
* Functions transform data from one form to another.
* Employs top-down approach in program design.

### Some issues with procedural programming are:

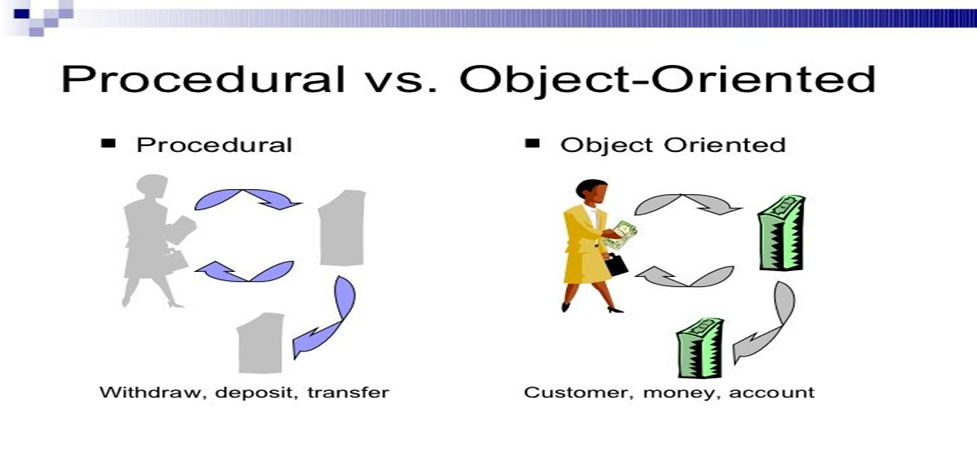
* **Data Undervalued:** Data is given second-class status in the organization of procedural languages.
* **Insecure Data**: A global data can be corrupted by functions. Since many functions access the same global data, the way the data is stored becomes critical.
* **Relationship to the Real World:** Procedural programs are often difficult to design because their chief components – functions and data structures – don’t model the real world very well.
* **New Data Types:** It is difficult to create new data types with procedural languages.
* **Complex and time consuming:** Does not provide code reusability, hence requires lots of coding work.
* **Does not support generic programming:** One can’t write programs independent of particular data types.

### Structured Programming

Structured programming (sometimes known as modular programming) is a subset of procedural programming that enforces a top-down design model, in which developers map out the overall program structure into separate subsections to make programs more efficient and easier to understand and modify. A defined function or set of similar functions is coded in a separate module or sub-module, which means that code, can be loaded into memory more efficiently and that modules can be reused in other programs. In this technique, program flow follows a simple hierarchical model that employs three types of control flows: sequential, selection, and iteration.

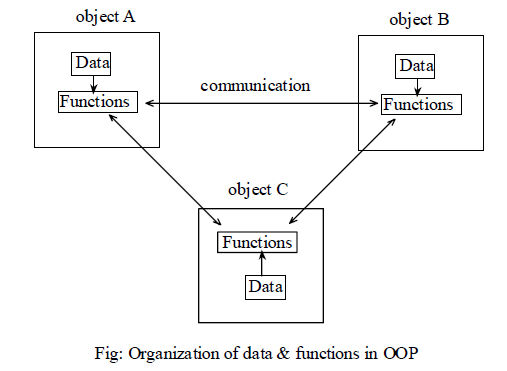
#### Problems with Structured Programming:

As programs grow larger, structured programming approach begins to show signs of strain. No matter how well the structured programming approach is implemented, the project becomes too complex, the schedule slips, more programmers are needed, and costs skyrocket.



## 1.2 Object-oriented Programming

* The object-oriented programming is an approach that combines or encapsulates both data (or instance variables) and functions (or methods) that operate on that data into a single unit. This unit is called an object.
* The data is hidden, so it is safe from accidental alteration. An object’s functions typically provide the only way to access its data.
* In order to access the data in an object, we should know exactly what functions interact with it. No other functions can access the data.
* An object-oriented program typically consists of a number of objects, which communicate with each other by calling one another’s functions. This is called message passing.
* Examples of object-oriented languages include C++, Python, C#, Java etc.

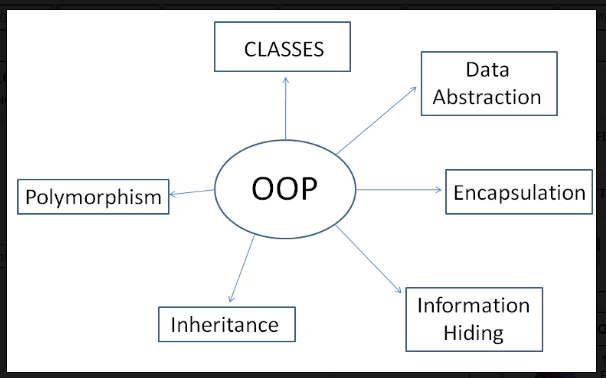


**In object oriented programming:**

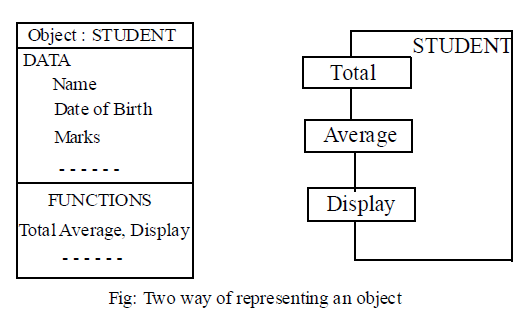
* Emphasis is on data rather than procedure.
* Programs are divided into objects.
* Functions that operate on the data of an object are tied together in the data structure.
* Data is hidden and cannot be accessed by external functions.
* Objects may communicate with each other through functions.
* New data and functions can be easily added whenever necessary.
* Follows bottom-up approach in program design.

### 1.3.1 Characteristics/features of Object-Oriented Language:

* Objects
* Classes
* Data Abstraction
* Encapsulation
* Inheritance
* Polymorphism



**Objects**: This is the basic unit of object oriented programming. That is both data and function that operate on data are bundled as a unit called as object. Objects are the basic run-time entities in an object-oriented language. Objects are instances of classes. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. They may also represent user-defined data such as vectors, time and lists. Program objects should be chosen such that they match closely with the real-world objects. Objects take up space in the memory.



**Classes:** Class is a user-defined data type. It is a collection of objects of similar type. Objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. E.g.: grapes, bananas and orange are the member of class fruit. Classes are user-defined data types and behave like the built-in types of a programming language.

**Data Encapsulation and Data Abstraction:** Combining data and functions into a single unit (class) is known as Encapsulation. ***Data encapsulation*** is important feature of a class. Class contains both data and functions. Data is not accessible from the outside world and only those functions which are present in the class can access the data. The insulation of the data from direct access by the program is called ***data hiding*** or information hiding.

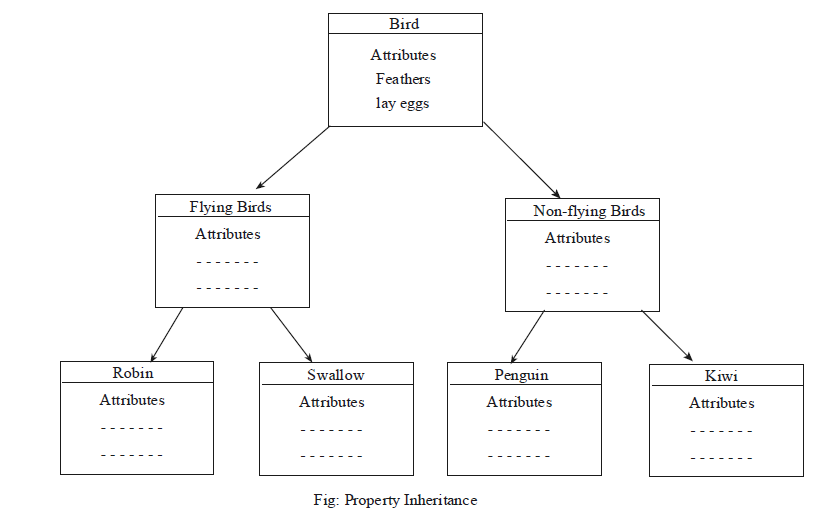
Hiding the complexity of program (*size, cost*) is called ***Abstraction*** and only essential features are represented. In short we can say that internal working is hidden. In C++, classes provides great level of data abstraction. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data, i.e., state without actually knowing how class has been implemented internally.

For example, your program can make a call to the sort() function without knowing what algorithm the function actually uses to sort the given values.

1. Abstraction using Classes: We can implement Abstraction in C++ using classes. Class helps us to group data members and member functions using available access specifiers. A Class can decide which data member will be visible to outside world and which is not.
2. **Abstraction in Header files**: One more type of abstraction in C++ can be header files. For example, consider the pow() method present in math.h header file. Whenever we need to calculate power of a number, we simply call the function pow() present in the math.h header file and pass the numbers as arguments without knowing the underlying algorithm according to which the function is actually calculating power of numbers.
3. **Abstraction using access specifiers:** Access specifiers are the main pillar of implementing abstraction in C++. We can use access specifiers to enforce restrictions on class members. For example: Members declared as public in a class, can be accessed from anywhere in the program. Members declared as private in a class, can be accessed only from within the class. They are not allowed to be accessed from any part of code outside the class.

We can easily implement abstraction using the above two features provided by access specifiers. Say, the members that defines the internal implementation can be marked as private in a class. And the important information needed to be given to the outside world can be marked as public. And these public members can access the private members as they are inside the class.

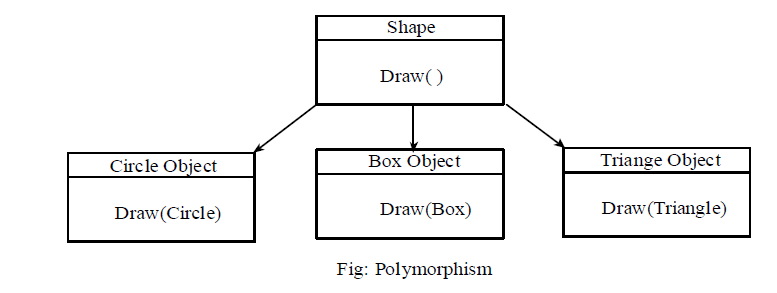
**Inheritance:** It is the process by which object of one class acquire the properties or features of objects of another class. The concept of inheritance provides the idea of reusability, means we can add additional features to an existing class without modifying it. This is possible by driving a new class from the existing one. The new class will have the combined features of both the classes. Object-oriented programming allows classes to inherit commonly used data and functions from other classes. If we derive a class (called derived class) from another class (called base class), some of the data and functions can be inherited so that we can reuse the already written and tested code in our program, simplifying our program. For examples, cars, trucks, buses, and motorcycles inherit all characteristics of vehicles.



**Polymorphism:** Polymorphism means ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation.

**Example:**

* *Operator Overloading:* An operator represents different behaviors in different instances is known as operator overloading. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation. Existing set of operators can be used to operate with user defined types. One can provide additional meaning to the existing set of operators.
* *Function Overloading:* Using a single function name we can deal with different type/number of arguments, it is called function overloading. A single function name can be used to handle different number and different types of arguments as in figure.



**Dynamic Binding:** Binding refers to linking of function call with function definition. And when binding takes place at run-time, it is called dynamic binding. OOP allows dynamic binding. The code associated to the function call is not known during compile time and it is only known during the execution. Dynamic binding is associated with polymorphism and inheritance.

### 1.3.2 Advantages of Object-oriented Programming

* Programs are easy to develop.
* Data hiding is achieved through encapsulation.
* Data centered approach rather than process centered approach.
* Elimination of redundant code due to inheritance, that is, we can re-use the code by deriving a new class from existing one.

## 1.3 C versus C++

|  |  |
| --- | --- |
| **C** | **C++** |
| C Follows the procedural programming approach, importance is given to the procedure of the program. | C++ is an OOP language, importance is given to the data rather than process. |
| Does not provide data hiding facility .i.e., data is insecure. | Provides data hiding facility through encapsulation i.e., data is secure. |
| Functions are the building blocks of a C program. | Objects are building blocks of a C++ program. |
| C doesn’t support function overloading. | C++ enables function overloading. |
| Structures cannot contain functions in C. | In C++, functions can be used inside a structure. |
| Provides little code reusability, thus hard to write complex programs. | Extreme code reusability through inheritance, so easier to write complex programs. |
| Complex dynamic memory allocation (malloc(), calloc(), free()). | Efficient and easy dynamic memory allocation through new and delete operators. |
| C uses top-down approach. | C++ follows bottom-up approach. |

# C++ Basics

## 2.1 A Simple C++ Program

#include <iostream> //allows program to accept data from keyboard, and output data to the screen

using namespace std;

int main() // main() is where program execution begins.

{

cout << " My first C++ program "; // displays the content on the screen

return 0; //indicates that the program is terminated successfully

}

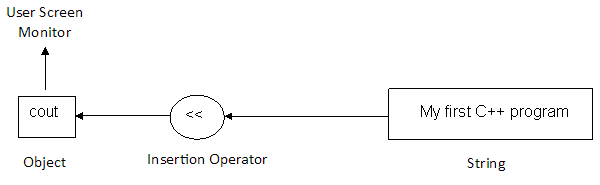
This program demonstrates following C++ features:

1. **#include <iostream>**: *The C++ language defines several headers, which contain information that is either necessary or useful to program. For this program, the header <iostream> is included to perform input/output operations. It tells the preprocessor to include the contents of iostream header file in the program before compilation. This file is required for input output statements.*
2. *The line* **using namespace std***; tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++. In C++, namespace defines the scope of the identifiers. A C ++ namespace, contains classes, variables, constants, functions, etc. For using the identifier defined in the namespace scope we must include the using directive.*

*Here, std is the namespace where ANSI C++ standard class libraries are defined. All ANSI C++ programs must include this directive. This will bring all the identifiers defined in std to the current global scope. Using and namespace are the new keyword of C++.*

1. *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. For multiline comments we can still use /\*…. \*/.*
2. *The line* **int main()** *is the main function where program execution begins.*
3. *The statement* **cout << "My first C++ program";** causes the string in quotation marks (“ ”) to be displayed on the screen. *This statement introduces two new C++ features,* ***cout*** *and* ***<<****. The identifier* **cout** *is a predefined object that represents the standard output stream in C++(i.e., monitor).*

*The << operator is called* **insertion or put to** *operator and directs (inserts or sends) the contents of the variable/operand on its right to the object on its left. The << operator directs the string constant* "My first C++ program" *to* **cout***, which sends it for the display to the monitor.*

**

1. *The next line* **return 0***; terminates main( )function. In C++ main() returns an integer type value to the operating system.*

## 2.2 Input using cin

#include <iostream>

using namespace std;

int main()

{

float a, b, sum, avg;

cout << "Enter two numbers: ";

cin >> a; //Read numbers

cin >> b; //from keyboard

sum = a + b;

avg = sum/2;

cout << “Sum: “ << sum <<”\n” ;

cout<< “Average: “ << avg ;

return 0;

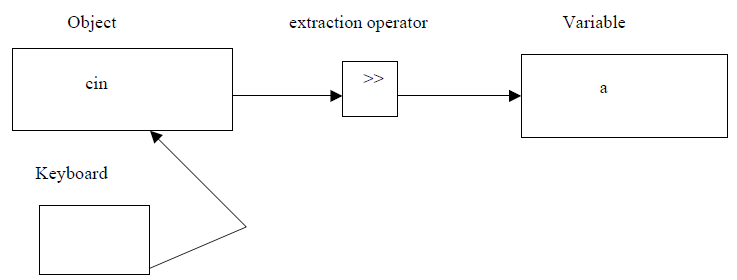
}

The keyword cin (pronounced ‘C in’) is also a stream object, predefined in C++ to correspond to the standard input stream (i.e keyword). This stream represents data coming from the keyboard.

cin >> a;

Is an input statement and causes the program to wait for the user to type in a number. The number keyed in is placed in the variable a. The identifier cin is a predefined object in C++ that corresponds to the standard input stream. Here, this stream represents the keyboard.

The operator >> is known as **extraction or get from** operator and extracts (takes) the value from the stream object **cin** and places it in the variable on its right. For example, in the statement **cin>>a;** the >> operator extracts the value from **cin** object that is entered from the keyboard and assigns it to the variable **a**.



## 2.3 Cascading of I/O operators

The multiple use of **insertion (<<),** or **extraction (>>)** operator in one statement is called cascading of I/O operators.

**For example:**

cout<< “sum: ” << sum <<endl ;

cin>> a >> b;

## 2.4 Manipulators

Manipulators are the operators/C++ programming language construct used with the insertion operator (<<) to modify or format the data display. The most commonly used manipulators are ***endl, setw, and setprecision***.

### 2.4.1 The endl Manipulator

This manipulator causes a linefeed to be inserted into the output stream. It has the same effect as using the newline character “\n”.

**For example:**

cout<< “Sum: ” << sum <<endl ;

cout<< “Average: ”<< avg ;

This is equivalent to cout<< “Sum: ” << sum <<endl << “Average: ”<< avg ;

### 2.4.2 The setw Manipulator

The manipulator s**etw(n)** specifies a field width n for printing the value. The output is right justified within the field. It is defined inside the header file *<iomanip>*.

**For example:**

cout<<setw(11)<<"Kantipur"<<endl<<setw(11)<<"Engineering"<<endl<<setw(11)<<"College";

**Output:**

Kantipur

Engineering

College

### 2.4.3 The setprecision Manipulator

To control the precision of floating point numbers appearing in the output, setprecision(n) and fixed manipulators are used.

It sets the total number of digits to be displayed witth necessary round-offs, when floating point numbers are used.

setprecision(n), means we are setting n significant digits with necessary round-offs for output. **setprecision** is defined inside the header file *<iomanip>*.

**For example**:

float f =3.14669;

cout << setprecision(2)<< f << endl;

cout << setprecision(3) << f << endl;

cout<<fixed << setprecision(3) << f;

**Output:**

3.1

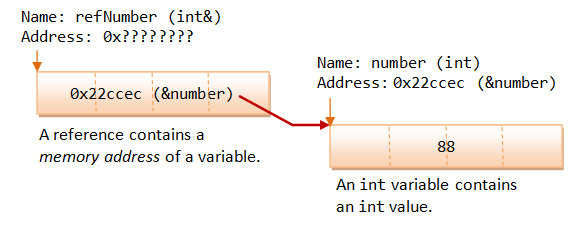
3.15

3.147

## 2.5 Reference Variable

C++ introduces a new kind of variable known as *reference variable*. *Reference variable* is an alias (another name) given to the already existing variable. When we declare a *reference variable*, it refers to the memory of another variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

C++ references allow you to create a second name for the variable that you can use to read or modify the original data stored in that variable.



A reference must be initialized when it is created. You need to initialize the reference variable during declaration. Once a reference is established to a variable, you cannot change the reference to reference another variable. You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.

Once a reference is initialized to an object/variable, it cannot be changed to refer to another object/variable. Pointers can be pointed to another object at any time. Pointers can be initialized at any time.

#### Syntax:

data-type & reference\_variable\_name = variable\_name;

#### Example:

int p = 100;

int & b = p; // b is an integer reference to the variable p.

Here **p** is an ***integer*** type variable that has already been declared, **b** is the alternative name given to represent the variable **p**.

***Consider the case of normal variables;***

10

10

int a = 10;

int b = a;

a

b

++a;

cout<< a<<b; // *output will be 11 10.*

***Consider the case of reference variables;***

10

int a=10;

int &b=a;

a

b

++a;

cout<<a<<b; // *output will be 11 11.*

#### // what will be the output of the following program? Why?

#include<iostream>

using namespace std;

void f( int &); //function prototype, one reference variable as argument.

int main()

{

int s = 4;

cout<<"s: "<< s<<endl;

f(s); //function call, initialization, int & a = s;

cout<<"s: "<< s<<endl;

return 0;

}

void f( int & a) //function definition.

{

a = 10\*a;

}

**Exercise:**

Write a function using reference variable as arguments to swap the values of a pair of integers.

## 2.6 Scope Resolution Operator (::)

Scope resolution operator (::) is used to access the global version of a particular variable. It is also used to declare global variable at local place.

#### Syntax:

:: variable-name;

#### // Sample program code.

char c = 'a'; // global variable

int main()

{

char c = 'b'; // *local variable with the same name,, local to main.*

cout << "Local c: " << c << endl; //Displays a character b.

cout << "Global c: " << ::c << "\n"; // *Accesses the global version. Displays a character a.*

::c ='u' ; // re-declaration of a global variable. Change persists within this scope

cout << "Global c: " << ::c << endl; //Now, displays a character u.

return 0;

}

Suppose in a C program there are two variables with the same name a. Assume that one has become declared outside all functions (global) and another is declared locally inside a function. Now, if we attempt to access the variable a in the function we always access the local variable. This is because the rule says that whenever there is a conflict between a local and a global variable, local variable gets the priority. C++ allows you the flexibility of accessing both the variables. It achieves through a scope resolution operator (: :).

Namespace

## 2.7 Specifying Symbolic Constants in C++

A **constant** is an [identifier](http://en.wikipedia.org/wiki/Identifier_(computer_programming)) with an associated [value](http://en.wikipedia.org/wiki/Value_(computer_science)) which cannot be altered by the [program](http://en.wikipedia.org/wiki/Computer_program) during execution.

### 1. Using const keyword

const int size = 30;

const char n = ‘b’;

const float PI = 3.14;

### 2. Using #define preprocessor directive

#define size 30

#define PI 3.14

## 2.8 String in C++

An ordered sequence of characters is called string. C++ provides following two types of string representations:

### 2.8.1 The C-style character string

A one-dimensional array of characters which is terminated by a null character '\0'.

**Example:**

* char address[14];
* char name[] = “ Harry”;
* char msg[5] = “Hello”; // error in C++.
* char msg[6] = “Hello”; // o.k.

In C++, the size should be one larger than the number of characters in the string. Built-in functions *strlen(), strcpy(), strcmp()* etc. are used to manipulate C-style string.

#### Reading Embedded Blanks

##### 1. Reading a Line of Text

Input stream object “cin” is used to read a word from keyboard. It cannot read a line of text with embedded blanks. It stops reading when first whitespace is encountered. To read text containing blanks, ***get()*** function is used.

**Syntax:**

cin.get(variable\_name, max characters);

**Example:**

#include<iostream>

#include<cstring>

using namespace std;

int main()

{

char name[70];

cout<< "Enter your name: " <<endl;

cin.get(name,70); //can read one line of text with whitespaces, maximum 70.

cout<<"Your name is: "<<name;

return 0;

}

##### 2. Reading Multiple Line of Text

To read multiple line of text, another form of ***get( )*** function is used.

**Syntax:**

cin.get (variable\_name, max, ‘terminator symbol’);

**Example:**

#include<iostream>

#include<cstring>

using namespace std;

int main()

{ const int max = 80;

char poem[max];

cout<< "enter multiple lines of text: " <<endl;

cin.get(poem, max, '$'); //can read multiple lines of text, maximum characters 80.

cout<<"You have entered following lines of text: "<<poem;

return 0;

}

In this example, ***get()*** function continue to accept characters until terminating character ‘$’ or until we exceed the size of max.

### 2.8.2 The string class type introduced with Standard C++

The standard C++ library provides a string class, which reduces several problems related to size, operations etc. with C-style strings. For this we need to include <string> header file.

* + **Example:**

string str; // Declaring a C++ string object:

str = “ Hello”; // Initializing a C++ string object:

string str1 = "Send money!"; // Another way.

* + **Sample program code:**

#include <iostream>

#include <string>

using namespace std;

int main ()

{

string str1 = "Hello";

string str2 = "World";

string str3, str4;

str3 = str1; // copy str1 into str3.

cout << "str3 : " << str3 << endl;

str4 = str1 + str2;

cout << "str4: " << str4 << endl;

cout<<"Size of string 3: "<<str3.size()<<endl; **//ReturnS length of string str3 in bytes.**

return 0;

}

#### Another Program: This program shows the use of getline () function.

#include<iostream>

#include<string>

**TRY this program, and ANALYZE the output.**

using namespace std;

int main()

{

string str1, str2;

cout<<" Enter a line of text: "<<endl;

getline(cin, str2); //to read a line of text with blanks.

cout<<"Your line of text is: "<<endl<<str2<<endl;

cout<<" Enter multiple lines of text: "<<endl;

getline(cin, str1, '$'); //to read multiple lines of text with embedded blanks, $ is the terminator character.

cout<<"You entered following line of text"<<endl<<str1<<" Length is "<<str1.size()<<endl;

string text="jhskfshfs ihfihsifhwsihfi wsh"; **//string constant**

cout<<endl<<"Given string constannt: "<<text<<endl;

return 0;

}

**Structure in C++**

A structure is acollection of simple variables. The variables in a structure can be of different types: some can be int, some can be float and so on. The data items in a structure are called the members of the structure. For example

#include<iostream>

#include<iomanip>

using namespace std;

struct part

{int model;

int partno;

float price;

};

int main()

{part parts;

parts.model=555;

parts.partno=44;

parts.cost=456.60;

cout<<”Model=”<<parts.model;

cout<<”Part=”<<parts.partno;

cout<<”Costs=”<<parts.cost;

return (0);

}

**Enumeration**

An enumeration is a user-defined type consisting of a set of named constants called enumerators. So an enumeration is a list of all possible values. We must give a specific name to every possible values. For example

#include<iostream>

#include<iomanip>

using namespace std;

enum days{sun,mon,tues,wed,thu,fri,sate};

int main()

{days day1,day2;

day1=mon;

day2=thu;

int diff=day1-day2;

cout<<”Days between=”<<diff<<endl;

if(day1<day2)

cout<<”Day1 comes before day2\n”;

return (0);

}

By default, members of an enum lists are given the values of 0,1,2,..etc. We can also initialized the enum members explicitly and unitialized members of the list have values that are one more than the previous value on the list.

enum days{sun=1, mon,tues,wed,thu,fri,sat};

Here, mon=2, tues=3 and so on.

## 2.9 Type Conversion

Type conversion refers to changing an entity of one data type, expression, function argument, or return value into another. There are two types of type conversion: automatic conversion and explicit type conversion.

### 2.9.1 Automatic Conversion (Implicit Type Conversion)

Implicit type conversion, also known as *coercion*, is an automatic type conversion done by the [compiler](http://en.wikipedia.org/wiki/Compiler).

**Example:** int a = 5.6;

**OUTPUT:**

a: 5

cout<< "a: " <<a;

Whenever the compiler expects data of a particular type, but the data is given as a different type, it will try to automatically covert. For e.g.

int a=5.6 ;

float b=7 ;

In the example above, in the first case an expression of type float is given and automatically interpreted as an integer. In the second case, an integer is given and automatically interpreted as a float. There are two types of standard conversion of numeric type: promotion and demotion. Demotion is not normally used in C++ community.

#### Promotion:

Promotion occurs whenever a variable or expression of a smaller type is converted to a larger type.

float a=4 ; //4 is a int constant, gets promoted to float

long b=7 ; // 7 is an int constant, gets promoted to long

double c=a ; //a is a float, gets promoted to double

When two or more operands of different types are encountered in the same expression, the lower type variable is converted to the type of the higher type variable by the compiler automatically. This is also called type promotion.

**Example:**

int a = 5;

float b = 7.6;

cout<< "sum: " << a+b; **//sum will be 12.6**, **WHY?**

**Analyze the output of the following program:**

float c=13.2;

char p='Z';

cout<<c+p; **//What is the output? WHY?**

There is generally no problem with automatic promotion. Programmers should just be aware that it happens.

#### Demotion:

Demotion occurs whenever a variable or expression of a larger type gets converted to smaller type. By default, a floating point number is considered as a double number in C++.

int a=7.5 // double gets down – converted to int ;

int b=7.45; // float gets down – converted to int

char c=b ; // int gets down – converted to char

Standard Automatic demotion can result in the loss of information. In the first example the variable ‘a’ will contain the value 7, since int variable cannot handle floating point values.

The order of types is given below:

**Data Type Order**

long double (highest)

double

float

long

int

char (lowest)

**2.9.2 Explicit Type Conversion (Type Casting)**

Explicit type conversion is a type conversion which is explicitly defined within a program by the programmer. Compile is not responsible for this type of conversion.

### Syntax:

(type-name) expression **//C notation**

type-name (expression) **//C++ notation**

**Example:**

double da = 3.3;

double db = 3.3;

double dc = 3.4;

int result = int (da )+ int (db) + int (dc); //result = 9

cout<< "Result: " <<result;

*Another Example:*

Explicit type conversion from *int* type to *char* type.

**//output will be d.**

int p =100;

cout<< char( p)<<"\n";

#### //WAP that inputs a character from keyboard and displays its corresponding ASCII value on the screen.

#include<iostream>

using namespace std;

int main()

{

char d;

cout<< "Enter any character: "<<endl;

cin>>d;

cout<<"ASCII value of "<<d<<" is "<< int(d); **//type conversion,** *char* type to *int* type.

return 0;

}

#### Analyze the output of the following program:

#include<iostream>

using namespace std;

int main()

{

float a=3.4, b=4.9, c=13.2;

int d=a+b+c;

cout<<d<<endl; **//output is 21, WHY?**

int e= int(a)+int (b)+ int (c);

cout<<e<<endl; **//output is 20, WHY?**

return 0;

}