Object Oriented Programming

November 7, 2022

Object Oriented Programming

Unit 1

Introduction

Programmers write instructions in various programming languages to perform their computation tasks such as:

- Machine Level Language.
- Assembly Level Language.
- High Level Language.

Introduction

- Machine Level Language: Machine code or machine language is a set of instructions executed directly by a computer's central processing unit (CPU). Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory. Every program directly executed by a CPU is made up of a series of such instructions..

- Assembly Level Language: An assembly language (or assembler language) is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

Introduction

- High Level Language: High level language is any programming language that enables development of a program in much simpler programming context and is generally independent of the computer's hardware architecture. High-level language has a higher level of abstraction from the computer, and focuses more on the programming logic rather than the underlying hardware components such as memory addressing and register utilization.

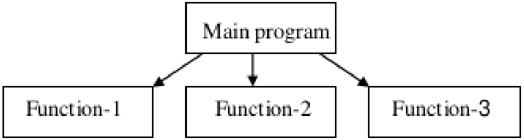
Introduction

The high-level programming languages are broadly categorized in to two categories:

- Procedure Oriented Programming (POP) language.
- Object Oriented Programming (OOP) language.

Procedure Oriented Programming Language

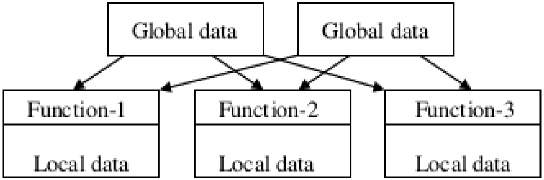
- In the procedure oriented approach, the problem is viewed as sequence of things to be done such as reading, calculation and printing.
- Procedure oriented programming basically consist of writing a list of instruction or actions for the computer to follow and organizing these instruction into groups known as functions



Procedure Oriented Programming Language

Characteristics of procedure oriented programming:

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



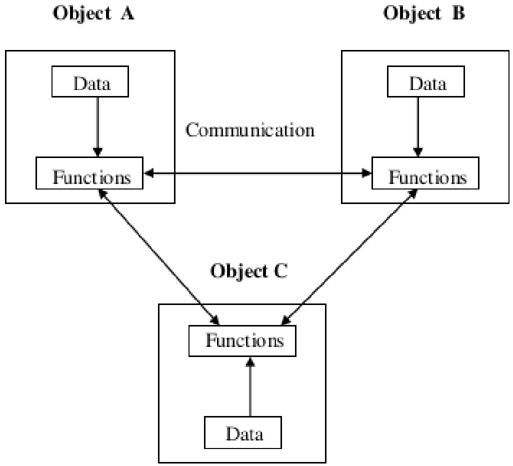
Procedure Oriented Programming Language

The disadvantage of the procedure oriented programming languages is:

- Global data access.
- It does not model real world problem very well.
- No data hiding.

Object Oriented Programming

In object oriented programming, program is divided into small parts called objects



Difference between Procedure Oriented Programming and Object Oriented Programming

Procedure Oriented Programming	**Object Oriented Programming**
In Procedure Oriented Programming, program is divided into small parts called functions.	In object oriented programming, program is divided into small parts called objects.
Procedure Oriented Programming follows top down approach.	Object oriented programming follows bottom up approach.
There is no access specifier in Procedure Oriented Programming.	Object oriented programming have access specifiers like private, public, protected etc.
Adding new data and function is not easy.	Adding new data and function is easy.

Difference between POP and OOP

Procedure Oriented Programming does not have any proper way for hiding data so it is less secure.	Object oriented programming provides data hiding so it is more secure.
In Procedure Oriented Programming, overloading is not possible.	Overloading is possible in object oriented programming.
In Procedure Oriented Programming, function is more important than data.	In object oriented programming, data is more important than function.
Procedure Oriented Programming is based on unreal world.	Object oriented programming is based on real world.
Examples: C, FORTRAN, Pascal, Basic etc.	Examples: C++, Java, Python, C# etc.

Difference between Top-Down Approach and Bottom-Up Approach in Programming

Top-Down Approach	**Bottom-Up Approach**
In this approach We focus on breaking up the problem into smaller parts.	In bottom up approach, we solve smaller problems and integrate it as whole and complete the solution.
Mainly used by structured programming language such as COBOL, Fortan, C etc.	Mainly used by object oriented programming language such as C++, C#, Python.
Each part is programmed separately therefore contain redundancy.	Redundancy is minimized by using data encapsulation and data hiding.
In this the communications is less among modules.	In this module must have communication.

Basics Concepts of Object Oriented Programming (OOP)

- Objects
- Classes
- Data Abstraction
- Data Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

Basics Concepts of Object Oriented Programming (OOP)

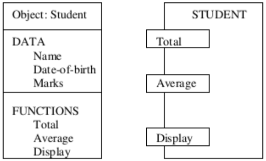
Objects

- Objects are the basic run-time entities in an object-oriented system.
- For instance: object may represent a person, a place, a bank account, a table of data or any item that the program must handle.
- The fundamental idea behind object oriented approach is to combine both data and function into a single unit and these units are called objects.

Basics Concepts of Object Oriented Programming (OOP)

Objects

- The term objects means a combination of data and program that represent some real word entity. For example: consider an example named Amit; Amit is 25 years old and his salary is 2500. The Amit may be represented in a computer program as an object. The data part of the object would be (name: Amit, age: 25, salary: 2500)
- The program part of the object may be collection of programs (retrive of data, change age, change of salary). In general even any user –defined type-such as employee may be used. In the Amit object the name, age and salary are called attributes of the object.



Basics Concepts of Object Oriented Programming (OOP)

Class

- The building block of C++ that leads to Object-Oriented programming is a Class.
- It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.
- For Example, Class of Cars - There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.
- The data member will be speed limit, mileage etc and member functions can apply brakes, increase speed etc.

Basics Concepts of Object Oriented Programming (OOP)

Data Abstraction

- Abstraction refers to the act of representing essential features without including the back ground details or explanations. Classes use the concept of abstraction and are defined as size, width and cost and functions to operate on the attributes.

Basics Concepts of Object Oriented Programming (OOP)

Data Encapsulation

- The wrapping up of data and function into a single unit (called class) is known as encapsulation. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. These functions provide the interface between the objects data and the program.

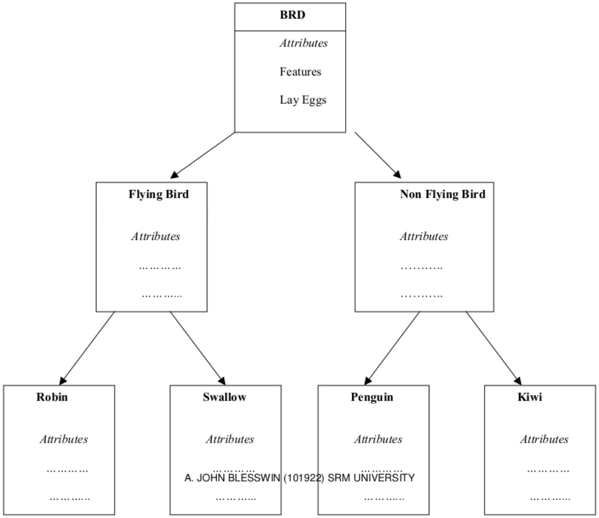
Basics Concepts of Object Oriented Programming (OOP)

Inheritance

- Inheritance is the process by which objects of one class acquire the properties of another class. In the concept of inheritance provides the idea of reusablity. This mean that we can add additional features to an existing class with out modifying it. This is possible by designing a new class will have the combined features of both the classes.

Basics Concepts of Object Oriented Programming (OOP)

Inheritance



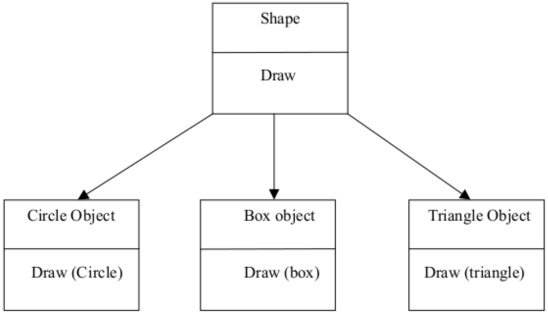
Basics Concepts of Object Oriented Programming (OOP)

Polymorphism

- Polymorphism means the ability to take more than one form. An operation may exhibit different instance. The behaviour depends upon the type of data used in the operation.
- A language feature that allows a function or operator to be given more than one definition. The types of the arguments with which the function or operator is called determines which definition will be used.
- For example: The expression x + y denote the sum of x and y, for many different types of x and y; integers , float and complex no. You can even define the + operation for two strings to get the concatenation of the strings.

Basics Concepts of Object Oriented Programming (OOP)

Polymorphism



Basics Concepts of Object Oriented Programming (OOP)

Dynamic Binding

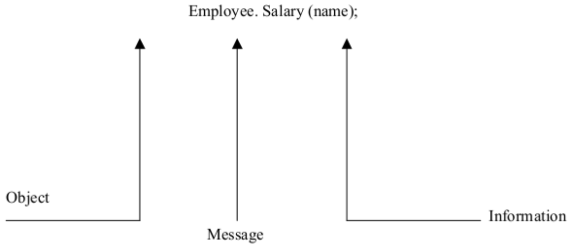
- In dynamic binding, the code to be executed in response to function call is decided at runtime.
- Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time.

Basics Concepts of Object Oriented Programming (OOP)

Message Passing

- An object-oriented program consists of a set of objects that communicate with each other.
- Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. The concept of message passing makes it easier to talk about building systems that directly model or simulate their real-world counterparts.
- A Message for an object is a request for execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired results. Message passing involves specifying the name of object, the name of the function (message) and the information to be sent.

Message Passing



Benefits of OOP

OOP offers several benefits to both the program designer and the user. Object- Orientation contributes to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The principal advantages are:

- Through inheritance, we can eliminate redundant code extend the use of existing Classes.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure program that can not be invaded by code in other parts of a programs.

Benefits of OOP

- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map object in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more detail of a model can implemental form.
- Object-oriented system can be easily upgraded from small to large system.
- Message passing techniques for communication between objects makes to interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

Application of OOP

OOP has become one of the programming buzzwords today. There appears to be a great deal of excitement and interest among software engineers in using OOP. Applications of OOP are beginning to gain importance in many areas.

- Real-time system
- Simulation and modeling
- Object-oriented data bases
- Hypertext, Hypermedia, and experts
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

Basics of C++

Brief History

- C ++ is an object oriented programming language
- C ++ was developed by Jarney Stroustrup at AT & T Bell lab, USA in early eighties.
- C ++ was developed from c and simula 67 language.
- C ++ was early called ‘C with classes’.

Output Operator

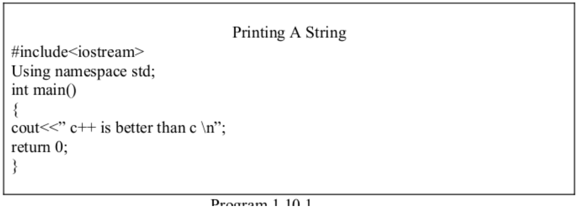
- The statement cout «"Hello, world" displayed the string with in quotes on the screen. The identifier cout can be used to display individual characters, strings and even numbers.
- It is a predefined object that corresponds to the standard output stream.
- The cout object, whose properties are defined in "iostream.h" represents that stream.

Applications of C++

C++ is a versatile language for handling very large programs; it is suitable for virtually any programming task including development of editors, compilers, databases, communication systems and any complex real life applications systems.

- Since C++ allow us to create hierarchy related objects, we can build special object-oriented libraries which can be used later by many programmers.
- While C++ is able to map the real-world problem properly, the C part of C++ gives the language the ability to get closed to the machine-level details.
- C++ programs are easily maintainable and expandable. When a new feature needs to be implemented, it is very easy to add to the existing structure of an object.
- It is expected that C++ will replace C as a general-purpose language in the near future.

Structure of a Program


Printing A String

```
#include<iostream>
Using namespace std;
int main()
{
cout<<" c++ is better than c \n";
return 0;
}
```

Program 1.10.1

Figure 1. This simple program demonstrates several C++ features.

Program feature: Like C, the C++ program is a collection of function. The above example contain only one function main(). As usual execution begins at main(). Every C++ program must have main(). C++ is a free form language. With a few exception, the compiler ignore carriage return and white spaces. Like C, the C++ statements terminate with semicolons.

Structure of a Program

Comments: Like C, the C++ program is a collection of function. The above example contain only one function main(). As usual execution begins at main(). Every C++ program must have main(). C++ is a free form language. With a few exception, the compiler ignore carriage return and white spaces. Like C, the C++ statements terminate with semicolons.

- C++ introduces a new comment symbol // (double slash). Comment start with a double slash symbol and terminate at the end of the line. A comment may start anywhere in the line, and whatever follows till the end of the line is ignored. Note that there is no closing symbol.
- The C comment symbols /*,*/ are still valid and are more suitable for multi-line comments. The following comment is allowed:

Input Operator: The statement "cin» number 1;" 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 number1.

The identifier **cin** is a predefined object in C++ that corresponds to the standard input stream. Here this stream represents the key board.

The operator » is known as get from operator. It extracts value from the keyboard and assigns it to the variable on its right.

Return Statement: In C++ main ( ) returns an integer type value to the operating system. Therefore every main() in C++ should end with a return (0) statement, otherwise a warning or an error might occur.

Structure of a Program

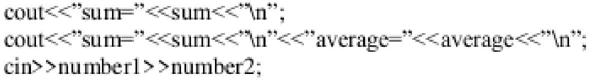
The iostream File: We have used the following include directive in the program: $include < iostream >$. The include directive instructs the compiler to include the contents of the file enclosed within angular brackets into the source file.

The header file $iostream.h$ should be included at the beginning of all programs that use input/output statements.

Namespace: Namespace is a new concept introduced by the ANSI C++ standards committee. This defines a scope for the identifiers that are used in a program. For using the identifier defined in the namespace scope we must include the using directive, like, $Usingnamespacestd$;

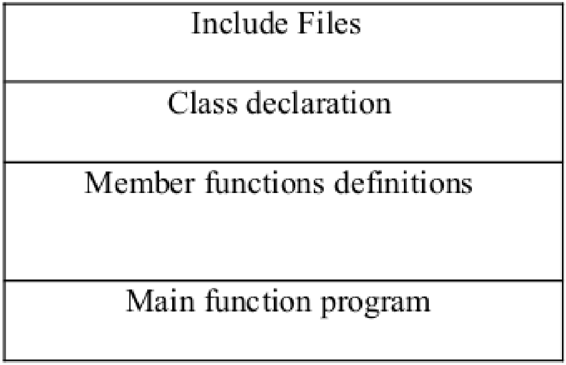
Here, std is the namespace where ANSI C++ standard class libraries are defined. All ANSI C++ programs must include this directive. This will bring all the identifiers defined in std to the current global scope. Using and namespace are the new keyword of C++.

Cascading of I/O Operator



```
cout<<"sum="<<sum<<"\n";
cout<<"sum="<<sum<<"\n"<<"average="<<average<<"\n";
cin>>number1>>number2;
```

Structure of C++ Program



Structure of a C++ Program

As it can be seen from program, a typical C++ program would contain four sections as shown in figure. This section may be placed in separate code files and then compiled independently or jointly.

It is a common practice to organize a program into three separate files. The class declarations are placed in a header file and the definitions of member functions go into another file.

This approach enables the programmer to separate the abstract specification of the interface from the implementation details (member function definition).

Finally, the main program that uses the class is places in a third file which “includes: the previous two files as well as any other file required.

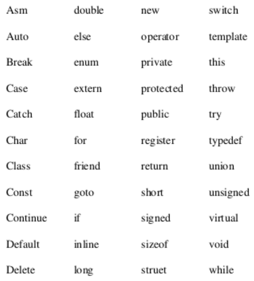
Tokens

The smallest individual units in program are known as tokens. C++ has the following tokens.

- Keywords
- Identifiers
- Constants
- Strings
- Operators

Tokens

Keyboards The keywords implement specific C++ language feature. They are explicitly reserved identifiers and can't be used as names for the program variables or other user defined program elements. The keywords not found in ANSI C are shown in red letter.

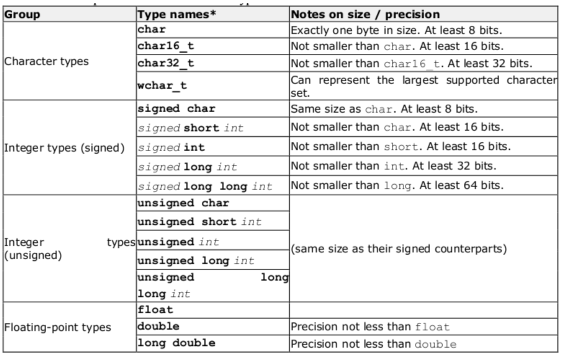


Asm	double	new	switch
Auto	else	operator	template
Break	enum	private	this
Case	extern	protected	throw
Catch	float	public	try
Char	for	register	typedef
Class	friend	return	union
Const	goto	short	unsigned
Continue	if	signed	virtual
Default	inline	sizeof	void
Delete	long	struet	while

Identifiers Identifiers refers to the name of variable , functions, array, class etc. created by programmer. Each language has its own rule for naming the identifiers.

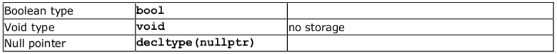
Basic Datatypes in C++

Both C and C++ compilers support all the built in types. With the exception of void the basic datatypes may have several modifiers preceding them to serve the needs of various situations. The modifiers signed, unsigned, long and short may applied to character and integer basic data types. However the modifier long may also be applied to double.



Group	Type names*	Notes on size / precision
Character types	**char**	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	**signed char**	Same size as char. At least 8 bits.
	signed **short** *int*	Not smaller than char. At least 16 bits.
	signed **int**	Not smaller than short. At least 16 bits.
	signed **long** *int*	Not smaller than int. At least 32 bits.
	signed **long long** *int*	Not smaller than long. At least 64 bits.
Integer types (unsigned)	**unsigned char**	(same size as their signed counterparts)
	unsigned short *int*	
	unsigned *int*	
	unsigned long *int*	
	unsigned long long *int*	
Floating-point types	**float**	
	double	Precision not less than float
	long double	Precision not less than double

User Defined Datatypes



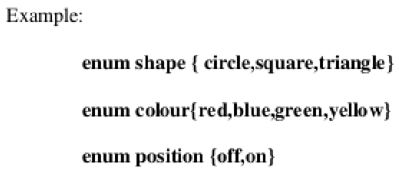
Boolean type	`bool`	
Void type	`void`	no storage
Null pointer	`decltype(nullptr)`	

Structures and Classes We have used user defined data types such as STRUCT and UNION in C. While these more features have been added to make them suitable for object oriented programming. C++ also permits us to define another user defined data type known as class which can be used just like any other basic data type to declare a variable. The class variables are known as objects, which are the central focus of oops.

Enumerated Data An enumerated data type is another user defined type which provides a way for attaching names to number, these by increasing comprehensibility of the code. The enum keyword automatically enumerates a list of words by assigning them values 0,1,2 and soon. This facility provides an alternative means for creating symbolic.

User Defined Datatypes

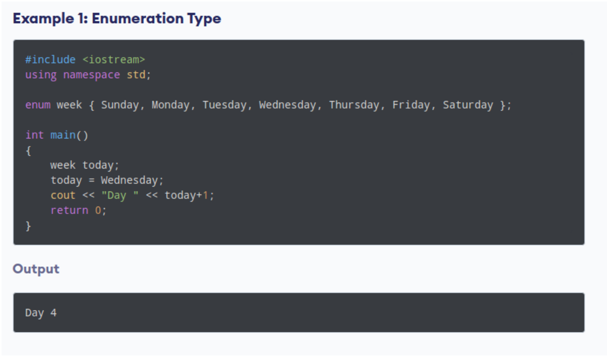
Enumerated Data


Example:

enum shape { circle,square,triangle}

enum colour{red,blue,green,yellow}

enum position {off,on}


Example 1: Enumeration Type

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

enum week { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

int main()
{
    week today;
    today = Wednesday;
    cout << "Day " << today+1;
    return 0;
}
```

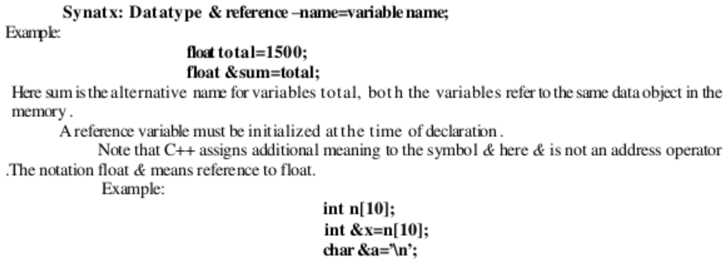
Output

```
Day 4
```

Reference Variables

C++ interfaces a new kind of variable known as the reference variable. A references variable provides an alias.(alternative name) for a previously defined variable. For example, if we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to represent the variable.

A reference variable is created as follows:



Synatx: Datatype & reference –name=variable name;

Example:

```
float total=1500;
float &sum=total;
```

Here sum is the alternative name for variables total, both the variables refer to the same data object in the memory .

A reference variable must be initialized at the time of declaration .

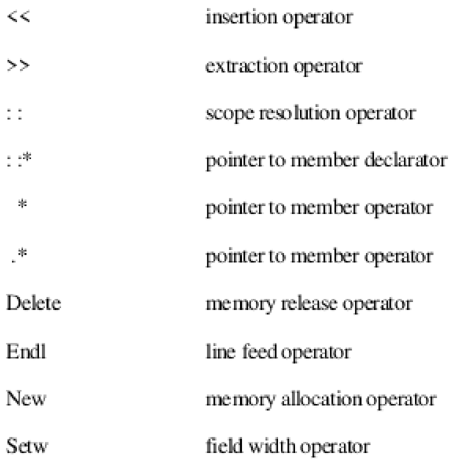
Note that C++ assigns additional meaning to the symbol & here & is not an address operator .The notation float & means reference to float.

Example:

```
int n[10];
int &x=n[10];
char &a='\n';
```

Operators

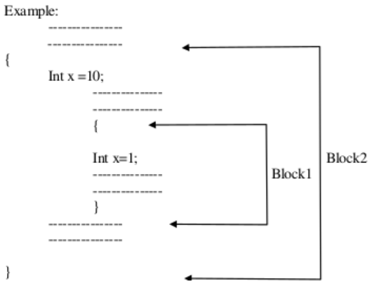
C++ has a rich set of operators. All C operators are valid in C++ also. In addition. C++ introduces some new operators.



<<	insertion operator
>>	extraction operator
: :	scope resolution operator
: :*	pointer to member declarator
*	pointer to member operator
.*	pointer to member operator
Delete	memory release operator
Endl	line feed operator
New	memory allocation operator
Setw	field width operator

Scope Resolution Operator

Like C,C++ is also a block-structured language. Block -structured language. Blocks and scopes can be used in constructing programs. We know same variables can be declared in different blocks because the variables declared in blocks are local to that function.

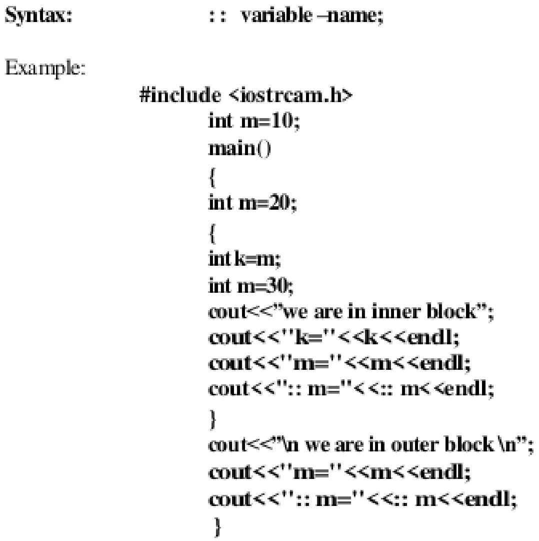


Scope Resolution Operator

Block2 contained in block1. Note that declaration in an inner block hides a declaration of the same variable in an outer block and therefore each declaration of x causes it to refer to a different data object. With in the inner block the variable x will refer to the data object declared there in.

In C, the global version of a variable can't be accessed from with in the inner block. C++ resolves this problem by introducing a new operator :: called the scope resolution operator. This can be used to uncover a hidden variable.

Scope Resolution Operator



Syntax: :: variable –name;

Example:

```
#include <iostrcam.h>
        int m=10;
        main()
        {
        int m=20;
        {
        int k=m;
        int m=30;
        cout<<"we are in inner block";
        cout<<"k="<<k<<endl;
        cout<<"m="<<m<<endl;
        cout<<":: m="<<:: m<<endl;
        }
        cout<<"\n we are in outer block \n";
        cout<<"m="<<m<<endl;
        cout<<":: m="<<:: m<<endl;
         }
```

Memory Management Operator

C uses malloc and calloc functions to allocate memory dynamically at run time.

Similarly it uses the functions Free( ) to free dynamically allocated memory.

We use dynamic allocation techniques when it is not known in advance how much of memory space as needed.

C++ also support those functions it also defines two unary operators new and delete that perform the task of allocating and freeing the memory in a better and easier way.
Syntax: $pointerVariable = newdatatype;$
Example: $p = new\ int$**;** $q = new\ int;$
Example: $*p = 25;\ *q = 7.5;$

Memory Management Operator

new can be used to create a memory space for any data type including user defined such as arrays,structures,and classes .The general form for a one-dimensional array is:
Syntax: $pointerVariable = new\ datatypes[size]$**;**

If a data object is no longer needed, it is destroyed to release the memory space for reuse.
Syntax: $delete\ pointerVariable$**;**
Example: $delete\ p; delete\ q;$

If we want to free a dynamically allocated array ,we must use the following form of delete.
$delete\ [size]\ pointerVariable$**;**
$delete\ []\ pointerVariable$**;**

Difference between C structures and C++ structures

In C++, struct and class are exactly the same things, except for that struct defaults to public visibility and class defaults to private visibility.

- Member functions inside structure: Structures in C cannot have member functions inside structure but Structures in C++ can have member functions along with data members.
- Direct Initialization: We cannot directly initialize structure data members in C but we can do it in C++.

Difference between C structures and C++ structures

- Using struct keyword: In C, we need to use struct to declare a struct variable. In C++, struct is not necessary. For example, let there be a structure for Record. In C, we must use "struct Record" for Record variables. In C++, we need not use struct and using 'Record' only would work.

- Static Members: C structures cannot have static members but is allowed in C++.

- Constructor creation in structure: Structures in C cannot have constructor inside structure but Structures in C++ can have Constructor creation.

Difference between C structures and C++ structures

- Data Hiding: C structures do not allow concept of Data hiding but is permitted in C++ as C++ is an object oriented language whereas C is not.
- Access Modifiers: C structures do not have access modifiers as these modifiers are not supported by the language. C++ structures can have this concept as it is inbuilt in the language.

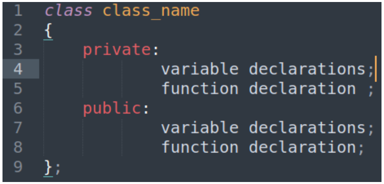
Class

- Class is a group of objects that share common properties and relationships.
- In C++, a class is a new data type that contains member variables and member functions that operates on the variables.
- A class is defined with the keyword class. It allows the data to be hidden, if necessary from external use.
- When we defining a class, we are creating a new abstract data type that can be treated like any other built in data type.

Generally a class specification has two parts:

- Class declaration
- Class function definition

Class

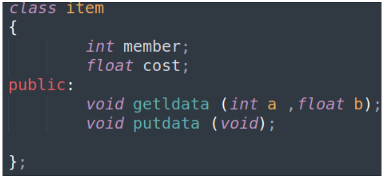


```
class class_name
{
    private:
            variable declarations;
            function declaration ;
    public:
            variable declarations;
            function declaration;
};
```

1. The members that have been declared as private can be accessed only from with in the class. On the other hand, public members can be accessed from outside the class also. The data hiding is the key feature of oops. The use of keywords private is optional by default, the members of a class are private.

2. The variables declared inside the class are known as data members and the functions are known as members mid the functions. Only the member functions can have access to the private data members and private functions. However, the public members can be accessed from the outside the class.

Class


```
class item
{
        int member;
        float cost;
public:
        void getldata (int a ,float b);
        void putdata (void);

};
```

The class item contains two data members and two function members, the data members are private by default while both the functions are public by declaration.

The function $getdata()$ can be used to assign values to the member variables member and cost, and $putdata()$ for displaying their values.

These functions provide the only access to the data members from outside the class.

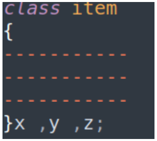
Creating Objects:

Once a class has been declared we can create variables of that type by using the class name.

Example: $item\ x$

Creates a variable x of type item. In C++, the class variables are known as objects. Therefore x is called an object of type item.

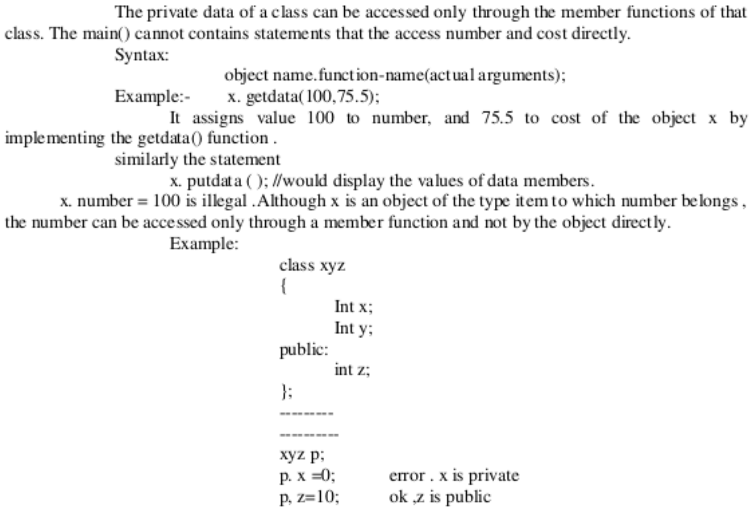
Item x, y, z also possible.



```
class item
{
-----------
-----------
-----------
}x ,y ,z;
```

It would create the objects x, y, z of type item.

Accessing a Member Function



The private data of a class can be accessed only through the member functions of that class. The main() cannot contains statements that the access number and cost directly.

Syntax:

object name.function-name(actual arguments);

Example:- x. getdata(100,75.5);

It assigns value 100 to number, and 75.5 to cost of the object x by implementing the getdata() function .

similarly the statement

x. putdata ( ); //would display the values of data members.

x. number = 100 is illegal .Although x is an object of the type item to which number belongs , the number can be accessed only through a member function and not by the object directly.

Example:

```
class xyz
{
        Int x;
        Int y;
public:
        int z;
};
--------
---------
xyz p;
p. x =0;        error . x is private
p, z=10;        ok ,z is public
```