13-06-2016

Principles of Object Oriented Programming

Characteristics exhibited by procedure-oriented-programming (POP)

1. Emphasis is on doing the task, i.e to follow the algorithm
2. Large programs are divided into smaller programs known as functions
3. Most of these functions share the global data.
4. Data moves openly around the system from function to function.
5. It employs top-down approach in program design.

Example: C

Characteristics of Object-oriented-Programming (OOP)

1. Emphasis is on the data rather than on the procedure.
2. Programs are divided into objects.
3. Data structures are designed such that they characterize the objects
4. Data is hidden and cannot be accessed by external functions.
5. Objects may communicate with each other through functions.
6. It follows bottom-up approach in program design

Example: C++

OOP – Data and instructions for processing the data are combined into a self contained module known as an object which can be used in other programs.

Object - The basic runtime-entities in an OOP system.

Data Encapsulation - The wrapping up of data and functions into a single unit called class

Class - The entire set of data and code of an object can be made by a user defined data-type with the help of class. A class is thus a collection of objects of similar type.

Data Abstraction- Refers to the act of representing essential features without including the background details or explanations

Binding- Refers to the procedural call to the code to be executed in response to that call

Dynamic Binding (Late Binding)- means that the code associated with a given procedural call is not known until the time of the call at runtime.

Inheritance - process by which objects of one class acquire some or all properties of another class

Polymorphism- Ability to take more than one form

Types of Polymorphism

1. Function overloading
Using a single function name to perform different types of tasks

2. Operator overloading
Making an operator to exhibit different behaviours in different instances

Message Passing- An OOP program consists of a set of objects that communicate with each other.

Steps for message passing

1. Creating classes that define objects and their behaviour.
2. Creating objects from class definitions
3. Establishing communication among objects in the form of

procedural call

Advantages of OOP

1. Through inheritance one can eliminate redundant code and extend the use of existing classes.
2. It is possible to have multiple instances of an object to co-exist without any interference.
3. These systems can easily be updated from basic to complex designs.
4. It is easy to partition diffent tasks in a project based on objects.
5. Software complexity can easily be managed.

Applications of OOP

1. Real Time Systems
2. Simulation And Modelling
3. Object oriented databases
4. Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM).

Beginning with C++

C++ Character set

1. Source characters
   (i) Small case alphabets → a-z
   (ii) Capital " " → A-Z
   (iii) Digits → 0-9
   (iv) Special symbols → +, -, \, *, [,], {, }, etc.

2. Escape sequence/white space characters.
   (i) \n end of line, takes the control to next line
   (ii) \0 end of string

(iii)	\t	horizontal tab
(iv)	\r	Takes the control to next paragraph
(v)	\f (form feed)	Takes the control to next page
(vi)	\v	vertical tab
(vii)	\b	backspace
(viii)	\\	backslash
(ix)	\a	audible alert
(x)	\'	single quote
(xi)	\"	double quote
(xii)	\?	Question mark
(xiii)	\ooo	display an octal number
(xiv)	\hhh	display hexadecimal number

Variable - is a way of refering to a memory location in a program.

Comments - statements in a program which gives details about specific instructions. They are not compiled by the compiler

Different ways of writing a comment

1. //_____
2. /*_____*/
3. /*_____
_____
_____*/

Input - Output Operators

1. cout - a predefined object that represents the standard output string. The operator << (insertion or put to operator) inserts the contents of the variable to the object displaying it on the screen.
2. cin - a predefined object that represents the standard input

string. The operator >> (extraction or get from operator) inserts the contents of the variable from the keyboard to the object.

3. The multiple use of << (output/insertion/put to) operator or >> (input/extraction/get from) operator in one statement is called cascading of input-output operators.

Structure of a procedure oriented C++ programme

1. Comments (optional).
2. header files.
3. user defined function (optional).
4. main( ).
5. function definition (if 3.).

Different header files and their meaning

1. <iostream.h>/<stdio.h>	contains function prototypes for standard input-output functions
2. <math.h>	contains function prototypes for math library functions.
3. <string.h>	contains function prototypes for C-style string processing functions.
4. <iomanip.h>	contains function prototypes for string manipulators that enable formatting of streams of data
5. <fstream.h>	contains function prototypes for functions that perform input from files on the disc and output to files on the disc.
6. <ctype.h>	contains function prototypes for function that test characters for certain properties.

7. <stdlib.h> contains function prototypes for conversion of numbers to text, text to numbers, responsible for random memory allocation and other utility programs.

8. <float.h> contains the floating point size limits of the system

9. <assert.h> contains macros and information that helps in program debugging.

Tokens

The smallest individual units in a program - C++ has following tokens

1. Keywords - explicitly reserved identifiers which cannot be used as variables or other user defined program elements

Table 3.1 C++ keywords

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	**int**	**static**	**volatile**
do	**long**	**struct**	**while**
Added by ANSI C++			
bool	export	reinterpret_cast	typename
const_cast	false	static_cast	using
dynamic_cast	mutable	true	wchar_t
explicit	namespace	typeid	

Note: The ANSI C keywords are shown in bold face

2. Identifiers - refers to the names of the variables, functions, arrays, classes, etc. created by the programmer

3. constants - refer to the fixed values that do not change during execution of a program.

Eg: 123 → integer constant

12.5 → floating point constant

"C++" → string constant

'C' → character constant

037 → octal constant.

C++ datatypes

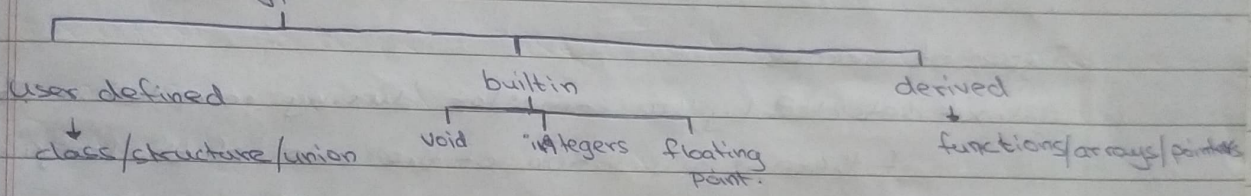


Table 3.2 Size and Range of C++ Basic Data Types

Type	Bytes	Range
char	1	−128 to 127
unsigned char	1	0 to 255
signed char	1	−128 to 127
int	2	−32768 to 32767
unsigned int	2	0 to 65535
signed int	2	−31768 to 32767
short int	2	−31768 to 32767
unsigned short int	2	0 to 65535
signed short int	2	−32768 to 32767
long int	4	−2147483648 to 2147483647
signed long int	4	−2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	4	3.4E−38 to 3.4E+38
double	8	1.7E−308 to 1.7E+308
long double	10	3.4E−4932 to 1.1E+4932

Operators in C++

1. Arithmetic operators → +, −, ×, /, %
2. Relational operators → >, <, <=, >=, ==
3. Logical operators → && , || , !
4. Some other operators → :: (scope resolution operator),
   endl (line feed operator) (same as \n),
   new (memory allocation operator),
   delete (memory release operator),

Errors and types of errors

1. Errors - Errors may be made during program creation which

will make a software to work unexpectedly. Certain types of errors are detected by the compiler

Debugging- process of finding and reducing number of errors in a program or software thus making it behave as expected.

Types of errors

1. Syntax- detected by the compiler during compilation. They occur if there is a violation of the rules of the language (grammatical errors). It also displays the error message on the screen
2. Logical- Occur when the logic of the program is wrong. They are most difficult to debug as no error messages are displayed on the screen.
3. Execution time- successfully compiling and linking need not always give the desired output. This may happen due to errors in logic like divide by zero condition in the code.
4. Linking - if the user does not include the necessary header files then the compiler compiles the program successfully but fails during linking. These errors are detected by the compiler and the error message is displayed.

Expressions and their types

Expression- A combination of operators, constants and variables arranged as per the rules of the language. It may also include function calls which return values.

Types of expressions

1. Constant expression- Consists of only constant values

2. Integral expression - produce integer results.
eg. If m and n are two integers then m+n, m*n, etc are integral expressions.

3. Floating point expressions - produce floating point results
eg. If m and n are floating point variables then m+n, m*n, are floating point expressions.

4. Relational expressions - produce results of type binary which takes the values true or false or zeros or ones.
eg: x >= y

5. Logical expressions - combine two or more relational expressions and produce binary type of result. eg. a+b >= c+d, a>b || c==10

6. Bit wise expressions - used to manipulate data at bit level.
eg. x << 3 → It shifts 3 bit positions to left
x >> 1 → shifts 1 bit position to right

7. Pointer expressions - produces expressions with address values

8. special Assignment expressions -

① chained assignment - eg. x=(y=10) or x=y=10 → both are valid.
Note: A chained statement cannot be used to initialize variables at the time of declaration
eg: int x=y=10 is invalid.

② embedded assignment - eg. x=(y=10)+50 or y=10; x=y+50 → valid

③ compound assignment — a compound assignment operator is a combination of the assignment operator (=) with binary arithmetic operators.
eg. x+=10 ⇒ x=x+10

Sytax of.

1. for loop → for (initialize counter ; condition to be tested ; increment or decrement)

(2) while loop → initialize counter;

```
while (condition to be tested)
{ ==========
  increment or decrement; }
```

(3) do while loop → initialize counter;

```
do { ==========
  increment or decrement }
while (condition to be tested);
```

Types of functions

1. Library functions (Built-in functions)
2. User-defined functions

Syntax for writing functions in C++ is similar to that of C

1. Function Declaration

Syntax:- type of function name of function (arguments);

2. Function calling

Syntax:- name of function (arguments);

3. Function Definition

Syntax:- type of function name of function (arguments)

{ statements to be executed

}

When a user defined function is called the control is transferred to the first statement in the function body. The other statements in the function body are then executed and control returns to the main program when } is encountered.

Importance of function prototyping

The prototype describes the function interface to the compiler by giving details such as the number of arguments, type of the arguments and type of value to be returned. When a function is called, the compiler matches the prototype with the given values and ensures that proper function is executed and the correct value is returned.

Advantages of using functions

1. A complex program can be divided into small subtasks and independent functions could be written to implement each
2. It makes the program user-friendly if many tasks are to be executed
3. There are many inbuilt functions present in the C++ library which could be used in the program directly.
4. There is also a facility wherein one function which calls itself to be included in the program also known as recursive functions. and the process is known as recursion.
5. If a function is created and the same has to be used later in the program, the same function could be reused again and again without redefining, hence it helps in reuse of a code.

1. Write a function factorial( ) in C++ to return factorial of individual digits of the given positive integer. Display the original number and the factorials of the digits.

Soln
```
#include <iostream.h>
#include <conio.h>
int factorial (int);
void main()
{int n,d,m;
clrscr();
cout<<"Enter the number";
```

```
cin >> n;
cout<<"the number is "<<n;
while (n>0)
{d= n%10; cout<<"Factorial of "<<d<<" is "
<<fact (d);
   n=n/10;}
   getch();}
int fact (int x)
   { int f=1,i;
  for(i=1;i<=x;i++)
    f=f*i;
    return(f);}
```

sum = sum + pow(x,i)/p; p=1;

return sum;}

Passing arrays to functions.

Arrays are always passed to functions by address. The name of an array is actually the address of the first element it stores. When a function has an array as a parameter, the corresponding parameters in both, the calling and the called function must be compatible.

1. Write a function ① get_array() having two arguments, integer array and size of the array which accepts the elements of the array ② Create a function calc_sum() with two arguments, integer array and size of the array which perfo the addition of all elements of the array ③ Function display to display the elements of the array before finding the s

and the total sum on the screen

Solo

```
#include <iostream.h>
#include <conio.h>
void get_array(int [],int);
void calc_sum(int [],int);
void display(int [],int);
void main()
{int a[100],n ;
cout<<"Enter the size of the array
cin>>n;
get_array(a,n);

display(a,n);
getch();}
void get_array(int a[100],int n)
{int i;
 cout<<"Enter the elements of the array";
    for(i=0;i< n ;i++)
    cin>>a[i];
 }
void calc_sum(int a[100],int n)
{int i,sum=0;
  for(i=0;i<n;i++)
   sum=sum+a[i];
   cout<<sum;}
void display(int a[100],int n)
{int i;
cout<<"The array is";
for(i=0;i<n;i++)
   cout<<a[i];
calc_sum(a,n);}
```

Functions with default arguments

Default arguments are useful in situations where some arguments have the same value. Eg: A bank interest may be same for all customers. In such cases such arguments can be initialized with a default value and only the necessary input may be taken from the user.

Declarations for default arguments.

An argument with a default value should be initialized last in the argument list → If 'a', 'b' and 'c' are three variables where b=5 and 'a' and 'c' are inputs passed as arguments to function product(), then the declaration of the function is

void product(int a, int c, int b=5);

→ If a=1, b=1 and c=1 the the declaration is

void product (int a=1, int b=1, int c=1)

→ If a=1, b=2 and c is the input the the declaration is

void product (int c, int a=1, int b=2)

void product (int a=1, int b=2, int c) is invalid.

11. Write a function to find the total sum amount earned at a rate of interest of 0.15. The principal amount and number of years are inputs by the user and the function calculate() should accept these as parameters and interest rate by default and returns the result back to the calling function. The total sum is calculated as sum = sum(1+r) + p for n years.

Soln →

```
#include <iostream.h>
#include <conio.h>
float calculate (int, int, float r=0.15);
void main()
{ int n, p;
  float r;
  cout << "Enter the principal amount and number of ye
  cin >> p >> n;
cout << "The total sum is " << calculate(p,n,r);
  getch();}
float calculate (int p, int n, float r= 0.15)
 { int i;
   float sum=p;
   for(i=1;i<=n;i++)
     sum=sum*(1+r);
     return sum;}
```

Function overloading.

Method by which functions performing similar tasks having same function names are created. This means that we can

use the same function name to create functions that perform different tasks. (polymorphism).

1. When a function call is made when functions are overloaded the compiler first matches the prototype of the function, that is the number of arguments, type of the arguments, type of the function and then calls the correct function for execution the best match must be unique in order to execute successfully.
2. If an exact match is not found the compiler uses internal conversion to find the exact match. When either of them fails, the compiler tries to use built-in conversions to the actual arguments and finds the unique match.
3. If the conversion is possible to have multiple matches then the compiler may generate an error or terminate abruptly because functions with same name and arguments cannot be called simultaneously.

Limitations

1. One should not try to overload unrelated functions and should reserve function overloading only for those functions which perform similar tasks.

Eg: Write an overloaded function area() to find

(1) area of circle where radius is float
(2) area of square where side is integer
(3) area of triangle where base and height are integers.

All the functions should return the result back to the calling function.

```
#include <iostream.h>
#include <conio.h>
float area(float);
int area(int);
float area(int,int);
```

```
void main()
{int s,b,h;
 float r;
 clrscr();
cout<<"Enter the radius of the circle";
cin>>r;
cout<<"The area of the circle is"<<area(r);
cout<<"Enter the side of the square";
cin>>s;
cout<<"The area of the square is"<<area(s);
cout<<"Enter the base and height of the triangle";
cin>>b,>>h;
cout<<"The area of the triangle is"<<area(b,h);
getch();}
float area (float r)
  {return (3.14*r*r);}
 int area (int s)
 {return (s*s);}
float area (int b,int h)
  {return (0.5*b*h);}
```

Write an overloaded function volume, to calculate the volume of the cone, volume of the sphere and volume of the cylinder, where all the radii values are of the type integer except for cylinder which is integer. and heights are float. All the functions return the value to the calling function.

Soln
```
#include<iostream.h>
#include<conio.h>
   float volume (int ,float);
   float volume (int);
int volume (int ,int);
void main()
```

To eliminate the cost of calls to small functions, c++ introduces a new feature called 'inline function.

An inline function is a function that is expanded inline when it is invoked that is the compiler replaces the function call with the corresesponding function code.

To make a function inline, just a keyword inline must be prefixed at the time of function definition. This function must be defined before they are called

```
inline type name (argument list)
{
==========
}
```

The keyword inline notifies the compiler that a simple function is being executed and the compiler may treat it as inline or not depending upon the complexity of the function definition.

Situations where inline functions do not work.

1. For functions returning values if a loop or switch exists.
2. For functions not returning values if a return statement exists.
3. If functions are recursive.

Eg: Write a function to add two numbers, to divide two numbers and to multiply two numbers using functions add(), divide() and multiply() respectively. All the three functions return values.

Soln
```
#include <iostream.h>
#include <conio.h>
inline int add(int x, int y)
{return (x+y);}
inline float divide(int x, int y)
{return (1.0*x/y);}
inline int multiply(int x, int y)
{return (x*y);}
void main()
{int x,y;
clrscr();
cout<<"Enter the numbers";
cin>>x>>y;
cout<<"the sum is"<<add(x,y)<<"the quotient is"
<<divide(x,y)<<"the product is"<<multiply(x,y);
getch();}
```

Conclusion

Functions with arguments can be invoked by

1. Call by value - the values of the actual parameters are copied

into the formal parameters which appears in function definition, hence function creates its own copy of arguments and operates on them

2. call by reference - It provides an alternate name for the variables and hence the same variables value can be used by two different names, i.e. original name and alternate name. A reference to the actual arguments is passed so the called function does not create its own copy of the values but works with original values of different names. Any change in the original data of the called function gets reflected back to the calling function.

Classes And Objects

Class

Combines the related data and functions together. The data is also known as data members and functions are known as member functions.

Definition:- wrapping up of data members and member functions into a single unit. It typically consists of the following

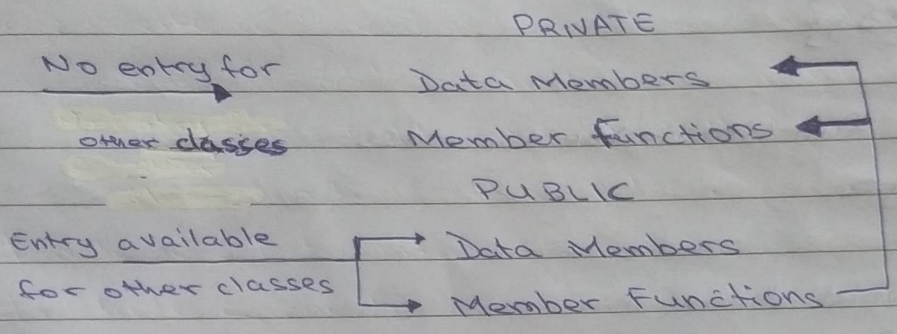
```
class classname
{private: data members;
          member functions;
 public: data members;
          member functions;};
```

Visibility modes.

1. private - class members that have been declared as private can be accessed only from within the class. By default, the members of the class are private. Hence a keyword

private is optional. If both the labels are missing, then by default all the members are private. Such a class is completely hidden from the other classes and does not serve any purpose.

2. public - public members can be accessed by the functions of the other class, hence, the data is available for other classes to use and update.



Syntax for creating objects
Name of class object name;

Calling Member functions
Syntax: object name . function name (arguments if any);

Defining Member functions
There are two ways to define member functions
1. Outside the class
2. Inside the class.

Syntax for defining member functions outside the class
type of function name of the class :: function (arguments if any)
{
}

Examples of defining member functions outside the class.

```
class xyz
{int a;
 public: void show();};
void xyz::show()
{
}
void main()
{xyz a;
 a.show();}
```

Defining the function inside the class

```
class xyz
{int a;
 public: void show()
         {
         }
};
void xyz::show()
{
}
void main()
{xyz a;
 a.show();}
```

Note:

1. several different classes can use the same function name.
2. Member functions can access the private data of its own class
3. Member functions can call each other in its own function definition.

```
class xyz
{int a;
public: void x(); void y(); };
```

```
void xyz::x()
{ ____
  ____ y();}
void xyz::y()
{ ____
  ____}
void main()
{ xyz a;
  a.x();}
```

This is called nesting of functions.

4. If any member function is under private visibility mode, then the object of its class cannot directly call this function, instead, this function has to be called through one of its public member function.

```
class xyz
{ void y();
  public: void x();};
void xyz::x()
{ ____
  ____ y();}
 void xyz::y()
  { ____
    ____}

void main()
{ xyz a; [a.y(); is invalid because y() is in private]
  a.x();};
```

① class student

private: registration no.- type integer
name- character array of size 30
age- type integer

public: a function input() - to accept the values for all data

members.

function display() - to display all values on the screen

soln
```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#include <string.h>
class student
{int reg_no, age;
  char name[30];
 public: void input();
         void display(); };
void student :: input()
{ cout << "Enter the registration number, age and name";
  cin >> reg_no >> age;
      gets(name); display(); }
            void student :: display()
cout << reg_no << age;
puts(name); }
void main
{student obj; clrscr();
 obj. input(); getch(); }
```

Example of array of objects.

(3) Class name – book

private – book number : type integer
price : float
title – character array size 30
calculate() – which returns the total price as total price = price x n where n is the number of copies and is passed as argument to this function.

public – input() to accept book number, price, title.
purchase() – to enter the number of copies of each book. the function also displays the book number, price of the book, title and total price calculated in function calculate().

The system should take the number of books from the user, recieve the details for all the books and display the data values on the screen.

Soln
```
#include <stdio.h>
#include <iostream.h>
#include <conio.h>
#include <string.h>
  class book
 {int book no;
  float price;
  char title[30];
  float calculate(int);
  public: void input();
          void purchase();};
void book::input()
{cout<<"Enter the book number, price and title of the book";
 cin>>bookno>>price;
 gets(title);}
```

```
void book::purchase()
{int n;
cout<<"Enter the number of copies of the book";
cin>>n;
cout<<"The book number is"<<bookno<<", the price
is"<<price<<"the title is";
puts(title);
cout<<"The total price for"<<n<<"books are"
    <<calculate(n);}
float
    book::calculate(int n)
{return(price*n);}
void main
{int n; book a[30];
cout<<"Enter the number of books";
cin>>n;
for(int i=0;i<n;i++)
{a[i].input();
 a[i].purchase();}}
```

Passing objects as function arguments

(5) class name - distance

private - inches and feet — integer

public - read data() — to read the values for feet and inches.

display( ) - to display the values of feet and inches on the screen

sum( ) - to accept two objects of class distance as parameters which finds the sum of corresponding data members and displays the result.

⑥ Write a function main to create three objects d_1, d_2, d_3 and values for d_1 and d_2 are read and calculate d_3 as a sum of d_1 and d_2. (12 inches is one foot)

Soln

```
#include<iostream.h>
#include<stdio.h>
#include<string.h>
#include<conio.h>
class distance
{int inches,feet;
 public: void readdata();
         void display();
         void sum(distance,distance);
 };
void distance::readdata()
{cout<<"Enter the values for feet and inches";
 cin>>feet>>inches;}
void distance::display()
{cout<<"the value of feet and inches are"<<feet<<inches;
}

void distance::sum(distance d1, distance d2)
{int i,f;
 i=d1.inches+d2.inches
 f=d1.feet+d2.feet+i/12;
 i=i%12;
cout<<"the total feet and inches are"<<f<<i;}
void main()
```