2076 Ashwin

**1. What are benefits of object oriented programming over procedural language? Compare C++ with C. List out the features of C++.**

Ans: Benefits of object oriented programming

A:Redundant code is eliminated by various techniques like inheritance

and templates.

B:Through data hiding programmer can biuld secure program.

C:Existing classes can serve as library class for further

enhancements.

D:Division of program into objects makes software development easy.

E:code reusability is much easier.

F:upgrading and maintenance of software is easily manageable.

|  |  |
| --- | --- |
| C programming | C++ programming |
| Languages such as c++,perl,php are influenced by C | Languages such as java,c# are influenced by C++ |
| It follows procedural approach of program development | It follows object oriented approach of program development |
| C application are faster to compile than c++ application | C++ application are compararively slower to compile than c |
| C has comparatively weaker type checking than c++ | C++ provides stronger type checking than c |
| C does not support extension in programming | C++ directly supports extension in programming like operator overloading |
| C has comparatively lesser implementation than c++ | C++ has larger implementation in area like GUI ad graphics as c++ support robust and powerful GIU |
| C has fewer libraries than c++ | C++ has improved and extended form of libraries than that of C developed by many vendors and opne source community |
| C has fewer keywords than c++ | C++ has more keywords than c |

Features of C++

.Namespace

.classes

.Derived class

.Access controllers

.Default arguments

.function overloading

.operator overloading

**2. What is a constant function? What is its reation with constant object? Write a meaningful function that shows the use of constant object and constant function along with use of const cast operator.**

Ans: Function members that guarantees not to change the data members value or object value and prevents accidental alteration these types of function that guarantee not to change object value or data member value are called constant function.

Constant function can be invoked by constant as well as non constant objects. But a non constant function cannot be called by constant object because it can change the object value or data member value.

#include<iostream>

**using** **namespace** std;

**class** **test**

{

**private:**

**int** data;

**public:**

test(**int** n=**0**)

{

data=n;

}

**void** setdata(**int** n)

{

data=n;

}

**int** getdata()**const**

{

**return** data;

}

};

**int** **main**()

{

test t1(**1**);

**const** test t2(**2**);

t1.setdata(**5**);

cout<<t1.getdata()<<endl;

test \*p=**const\_cast**<test \*>(&t2);

p->setdata(**7**);

cout<<t2.getdata();

**return** **0**;

}

**3. What do you understand by default arguments? How can you relate default argument function with function overloading? Write a program to find volume of different shapes using function overloading.**

Ans: A default arguments is a value provided in a function declaration that is automatically assigned by the compiler if the calling function doesn’t provide a value for the argument.

#include<iostream>

**using** **namespace** std;

**void** **add**(**int**,**int** b=**5**);

**void** **add**(**double**,**float** b=**2.5**);

**int** **main**()

{

add(**5**);

add(**2.5**);

**return** **0**;

}

**void** **add**(**int** a,**int** b)

{

cout<<"add is: "<<(a+b)<<endl;

}

**void** **add**(**double** a,**float** b)

{

cout<<"add is: "<<(a+b)<<endl;

}

#include<iostream>

#include<math.h>

**using** **namespace** std;

**void** **volume**(**int** l,**int** b,**int** h)

{

cout<<"Volume of rectangle is: "<<(l\*b\*h)<<endl;

}

**void** **volume**(**int** s)

{

cout<<"volume of cube is: "<<pow(s,**3**)<<endl;

}

**void** **volume**(**float** p,**int** r,**int** h)

{

cout<<"Volume of cylinder is: "<<(p\*r\*h);

}

**int** **main**()

{

**int** l,b,h,s,r,h1;

**float** p=**3.14**;

cout<<"Enter length,breadth,hight of reactangle: "<<endl;

cin>>l>>b>>h;

cout<<"Enter side of cube: "<<endl;

cin>>s;

cout<<"Enter radius and height of cylinder "<<endl;

cin>>r>>h1;

volume(l,b,h);

volume(s);

volume(p,r,h1);

**return** **0**;

}

**4. What do you mean by operator overloading? Write down its syntax. Write a class that represent the distance class and overload ++ and – operator to increment and decrement distance.**

Ans: Operator overloading is a features of c++ in which we can extend the functionalities of the existing operator for user defined data types.

Syntax:

Return\_type operator operator\_symbol(argument\_list)

{

//body

}

#include<iostream>

**using** **namespace** std;

**class** **distancess**

{

**private:**

**int** a;

**public:**

distancess()

{

a=**0**;

}

distancess(**int** c)

{

a=c;

}

**void** **operator** ++();

**void** **operator** --();

**void** **display**()

{

cout<<a<<endl;;

}

};

**void** distancess::**operator** ++()

{

a=++a;

}

**void** distancess::**operator** --()

{

a=--a;

}

**int** main()

{

distancess a(**2**);

++a;

a.display();

--a;

a.display();

**return** **0**;

}

**5. Explain the need of inheritance in programming? Explain various forms of inheritance. Write a program to create a derived class by inheriting two base classes with same function names. Your program should be complete and meaningful.**

Ans: Inheritance allows programmers to create classes that are built upon existing classes, to specify a new implementation while maintaining the same behaviors, to reuse code and helps in reusability of code.

Various forms of inheritance:

A:single

When a class is derived from only one base class then such derivation is called single inheritance.

Syntax:

Class derived:[access\_to\_base] base

{

//……

}







B:Multiple

When a class is derived from two or more base class, then such type of inheritance is called multiple inheritance.

Syntax:

Class derived:[access\_to\_base]base1,[access\_to\_base]base2

{

//………

}



C:Multilevel

The derivation of a class from another derived class is called multilevel inheritance.





D: Hierarchical

When different classes are derived from single base class such type of inheritance is called hierarchical inheritance.

E: Hybrid

When we mix more than one form of inheritance such type of inheritance is called hybrid inheritance.

F:Multipath

Derivation of a class from other derived classes, which are derived from same base class, is called multipath inheritance.



#include<iostream>

**using** **namespace** std;

**class** **base**

{

**protected:**

**int** a;

**public:**

**void** **getdata**(**int** x)

{

a=x;

}

};

**class** **base1**

{

**protected:**

**int** b;

**public:**

**void** **getdata**(**int** x)

{

b=x;

}

};

**class** **derived**:**public** base,**public** base1

{

**public:**

**void** display()

{

cout<<a<<endl<<b;

}

};

**int** **main**()

{

derived d;

d.base::getdata(**5**);

d.base1::getdata(**6**);

d.display();

**return** **0**;

}

**6. What is the purpose of stream manipulation? Explain different file modes that are used in opening the file. Write a program that will copy the content from one file, change the case of letters to upper case if they are in lower case and store in next file.**

Ans: File modes that are used in opening the file are:

1. ios::in

This mode opens a file for reading.

1. ios::out

This mode opens a file for writing.

1. ios::ate

when a file is opened in this mode, file access pointer is set at the end of file.

1. ios::app

When a file is opened in this mode, file is opened in write mode with the file access pointer at the end of file.

1. ios::trunc

when a file is opened in this mode, the file is truncated if a file with specified name already exists.

1. ios::binary

when a file is opened in this mode, the file is opened as a binary file and not as an ASCII text file.

#include<iostream>

#include<fstream>

#include<cstring>

**using** **namespace** std;

**int** **main**()

{

**char** st[**100**],s[**100**],c;

**int** i;

fstream file;

file.open("file.txt",ios::in);

file.read(**reinterpret\_cast**<**char** \*>(st),**sizeof**(st));

**for**(i=**0**;i<=strlen(st);i++)

{

c=(**char**)toupper(st[i]);

s[i]=c;

}

file.close();

file.open("updatefile.txt",ios::out);

file.write(**reinterpret\_cast**<**char** \*>(s),**sizeof**(s));

file.close();

cout<<"Complete";

**return** **0**;

}

**7. What do you mean by polymorphic class? What are different RTTI mechanisms in C++? Write a program that shows the use of pure virtual function.**

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

RTTI is a mechanism that exposes information about an object’s data type during runtime. This feature can be available only when the class has at least one virtual function.

#include<iostream>

**using** **namespace** std;

**class** **shape**

{

**protected:**

**int** volume;

**public:**

**virtual** **void** find\_vol()=**0**;

};

**class** **cube**:**public** shape

{

**protected:**

**int** side;

**public:**

cube()

{

cout<<"Enter side of the cube: "<<endl;

cin>>side;

}

**void** find\_vol()

{

volume=side\*side\*side;

cout<<"Volume of cube is: "<<volume;

}

};

**class** **sphere**:**public** shape

{

**protected:**

**int** radius;

**public:**

sphere()

{

cout<<"Enter radius of the sphere: "<<endl;

cin>>radius;

}

**void** find\_vol()

{

volume=(**4**/**3**)\***3.14**\*radius\*radius\*radius;

cout<<"Volume of sphere is: "<<volume;

}

};

**int** **main**()

{

shape \*ptr;

cube c;

sphere s;

ptr=&c;

ptr->find\_vol();

ptr=&s;

ptr->find\_vol();

**return** **0**;

}

**8. Why do we need class templates? Write a program to create class to represent stack data structure and use exception handling to control empty and full cases.**

Ans: A template class in C++ is a class that allows the programmer to operate with generic data types. This allows the class to be used on many different data types as per the requirements without the need of being re-written for each type.

Syntax:

template<class template\_type,……>

#include<iostream>

#define max 2

**using** **namespace** std;

**class** **stack**

{

**protected:**

**int** s[max];

**int** top;

**public:**

**class** **full**

{

};

**class** **empty**

{

};

stack()

{

top=-**1**;

}

**void** push(**int** x)

{

**if**(top==max-**1**)

{

**throw** full();

}

**else**

{

s[++top]=x;

}

}

**int** pop()

{

**if**(top==-**1**)

{

**throw** empty();

}

**else**

{

**return** s[top--];

}

}

};

**int** **main**()

{

stack s;

try

{

s.push(**11**);

s.push(**22**);

s.push(**33**);

cout<<"Number popped: "<<s.pop()<<endl;

cout<<"Number popped: "<<s.pop()<<endl;

cout<<"Number popped: "<<s.pop()<<endl;

}

**catch**(stack::full)

{

cout<<"stack is full"<<endl;

}

**catch**(stack::empty)

{

cout<<"stack is empty"<<endl;

}

**return** **0**;

}