1. **Why object oriented programming necessary in Programming? With suitable example explain the importance of object as function argument and returning object.**

**Answer:**

**Speaking in general, Programming is done in order to solve the difficult problems easily and efficiently. Yes, there are a lot of programming approach that we can use which also includes object oriented programming. Object oriented programming is necessary because it helps the programmer to break down the problem into many object so that the solution of that problem can be acquired efficiently.**

**Considering a program to add two complex number;**

**Source Code:**

**Analyzing the program above, we were able to create the object of class Complex where in object C3 we passed the object C1 and C2 in order to add them. Therefore using “passing object as an argument”, addition is done efficiently.**

1. **What do you mean by constructor? Explain different types of constructors. Create a class called “time” with data member hour, minute, second, and day. Initialize all the data members using constructor. Write a program to add two time object using necessary member functions and display the result.**

**Answer:**

**A constructor is a special type of member function of a class which initializes the members of its object. There are three type of constructors. They are:**

* **Default constructor**
* **Parameterized constructor**
* **Copy constructor**

**The type of constructor which has no arguments is known as default constructor.  
 The type of constructor which takes some argument is called parameterized constructor.  
 The type of constructor which initializes the data member of the object by copying the values of   
 another object initialized by either default or parameterized constructor is called copy   
 constructor.**

**Source code:**

**#include<iostream>**

**#include<conio.h>**

**using namespace std;**

**class time**

**{**

**private:**

**int day, hour, minute, second;**

**public:**

**time ()**

**{**

**day=0;**

**hour=0;**

**minute=0;**

**second=0;**

**}**

**time (int d, int h , int m , int s)**

**{**

**day=d;**

**hour=h;**

**minute=m;**

**second=s;**

**}**

**time addtime(time t1 , time t2)**

**{**

**time temp;**

**temp.day=t1.day+t2.day;**

**temp.hour=t1.hour+t2.hour;**

**temp.minute=t1.minute+t2.minute;**

**temp.second=t1.second+t2.second;**

**if(temp.second>=60)**

**{**

**temp.second=temp.second-60;**

**temp.minute=temp.minute+1;**

**}**

**if(temp.minute>=60)**

**{**

**temp.minute=temp.minute-60;**

**temp.hour=temp.hour+1;**

**}**

**if(temp.hour>=24)**

**{**

**temp.hour=temp.hour-24;**

**temp.day=temp.day+1;**

**}**

**return (temp);**

**}**

**void showtime ()**

**{**

**cout<<"Day = "<<day<<endl;**

**cout<<"Hour = "<<hour<<endl;**

**cout<<"Minute = "<<minute<<endl;**

**cout<<"Second = "<<second<<endl;**

**}**

**};**

**int main()**

**{**

**time t1(1,2,3,4),t2(5,6,7,8),t3;**

**t3=t3.addtime (t1, t2);**

**t3.showtime ();**

**getch ();**

**return (0);**

**}**

1. **Compare C and C++. Why do we need dynamic memory management? Explain the operators in C++ that enables dynamic memory management with example.**

**Answer:**

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **C** | **SN** | **C++** |
| **1** | **This language was introduced in 1972 by Dennis Ritchie at Bell Lab.** | **1** | **This language was introduced in 1985 by Bjarne Stroustup at Bell Lab.** |
| **2** | **This language is influenced by B, BCPL, ALGOL68, etc.** | **2** | **This language is influenced by C, Simula68, Ada83, etc.** |
| **3** | **Languages such as C++,Javascript,PHP,etc are influenced by C** | **3** | **Languages such as Java, C#, etc are influenced by C++ language.** |
| **4** | **It follows POP (procedural oriented programming) approach of program development.** | **4** | **It follows OOP (Object oriented programming) approach of program development.** |
| **5** | **The applications built using C are comparatively faster to compile.** | **5** | **The applications built using C++ are comparatively slower to compile** |
| **6** | **C language has comparatively weaker type checking than C++.** | **6** | **C++ language has comparatively weaker type checking than C.** |

**Dynamic memory allocation is very useful for managing the memory during coding a program. Some circumstances where we can use dynamic memory allocation are:**

* **When we do not know how much amount of memory would be needed for the program beforehand.**
* **When we want data structures without any upper limit of memory space.**
* **When you want to use your memory space more efficiently**

**The operators in C++ that enables dynamic memory management are:**

* + - **“new” (obtains memory at run time)**
    - **“delete” (frees the allocated memory)**

**Source code: (Dynamic Memory Allocation Example)**

**#include<iostream>**

**#include<conio.h>**

**using namespace std;**

**int main()**

**{**

**int \*p, i, n,sum=0;**

**float temp,avg;**

**cout<<"How many elements do you want to enter \n";**

**cin>>n;**

**p=new int [n];**

**cout<<"Enter "<<n<<" elements \n";**

**for (i=0; i<n; i=i+1)**

**{**

**cout<<"Enter element no "<< (i+1) <<"\n";**

**cin>>p[i];**

**sum=sum+p[i];**

**}**

**temp=static\_cast <float>(sum);**

**avg=temp/n;**

**cout<<"Sum is "<<sum<<"\n";**

**cout<<"Average is "<<avg<<"\n";**

**delete []p;**

**getch ();**

**return (0);**

**}**

1. **What is token, write its detail? With example explain function overloading in object oriented programming.**

**Answer:**

**Tokens are the valid sets (collection) of different characters, symbols, operators, punctuators. The compiler collects valid character sets tom make a sensible token. They can also be defined as the smallest meaningful individual units in a program.**

**The types of tokens are:**

* **Keywords**
* **Identifiers**
* **Constants**
* **Operators**
* **Punctuators**

**Keywords are the reserved words that have fix meaning.  
The name given to the variables, functions, classes, etc by the programmer is called identifiers.  
Constants are those which have fixed values that cannot be changed.  
Operators are special symbol used for arithmetic and logical operation.**  
**Punctuators in C++ have syntactic and semantic meaning to the compiler but do not, of themselves, specify an operation that yields a value.**

**Source code: (Function overloading by using different number of arguments)**

**#include<iostream>**

**#include<conio.h>**

**#define pi 3.14**

**using namespace std;**

**float volume(float r,float s);**

**float volume (float l, float b, float h);**

**int main ()**

**{**

**float r, s, l, b, h, cyl, rec;**

**r=1, s=10;**

**l=5, b=5, h=5;**

**cyl=volume(r,s);**

**rec=volume (l, b, h);**

**cout<<"Volume of cylinder is "<<cyl<<endl;**

**cout<<"volume of rectangle is "<<rec<<endl;**

**getch ();**

**return(0);**

**}**

**float volume(float r,float s)**

**{**

**return(pi\*r\*r\*s);**

**}**

**float volume(float l , float b, float h)**

**{**

**return(l\*b\*h);**

**}**

**Source code: (Function overloading by using different types of argument)**

**#include<iostream>**

**#include<conio.h>**

**using namespace std;**

**void display(char);//char function**

**void display(int);//integer function**

**void display(float);//float function**

**int main()  
{**

**char ch='a';**

**int num=1;**

**float pi=3.1415;**

**cout<<"Character Function"<<endl;**

**display(ch);**

**cout<<"Integer Function"<<endl;**

**display(num);**

**cout<<"Float Function"<<endl;**

**display(pi);**

**getch();**

**return(0);**

**}**

**void display(char ch)  
{**

**cout<<"character = "<<ch<<endl;**

**}**

**void display(int num)  
{**

**cout<<"num = "<<num<<endl;**

**}**

**void display (float pi)  
{**

**cout<<"pi = "<<pi<<endl;**

**}**

1. **Explain which operators cannot be overloaded in c++? Explain how a class type (user defined type) of data can be converted to a basic data (in built data) type? Write a program to concatenate two user given string by overloading binary plus operator.**

**Answer:**

**There are some rules that a programmer has to follow while operator overloading and in C++ there are some operators that cannot be overloaded. They are:**

* **:: (Scope resolution operator)**
* **. (Dot operator / Member access operator)**
* **.\* (Pointer to member access operator)**
* **?: (Conditional operator)**
* **sizeof() (Size of operator)**
* **typeid() (Type id operator)**

**To convert a class type to basic type, we can define an overloaded casting operator but the operator function must have following characteristics:**

* **It must be a member of class.**
* **It must not specify a return type.**
* **It must not have any argument.**

**Syntax : (Casting operator)**

**Operator type\_name( )  
{  
 //body  
}**

**Source code:**

**#include<iostream>**

**#include<conio.h>**

**#include<string.h>**

**using namespace std;**

**class mystring**

**{**

**private:**

**char str[20];**

**public:**

**void getvalue()**

**{**

**cout<<"Enter string"<<endl;**

**cin>>str;**

**}**

**friend mystring operator +(mystring s1,mystring s2);**

**void display()**

**{**

**cout<<str<<endl;**

**}**

**};**

**mystring operator +(mystring s1,mystring s2)**

**{**

**mystring s3;**

**strcpy(s3.str,s1.str);**

**strcat(s3.str,s2.str);**

**return(s3);**

**}**

**int main()**

**{**

**mystring s1,s2,s3;**

**cout<<"Enter string 1"<<endl;**

**s1.getvalue();**

**cout<<"Enter string 2"<<endl;**

**s2.getvalue();**

**cout<<"BEFORE CONCATENATION"<<endl;**

**s1.display();**

**s2.display();**

**s3=s1+s2;**

**cout<<"AFTER CONCATENATION"<<endl;**

**s3.display();**

**getch();**

**return(0);**

**}**

1. **Explain why inheritance is important in object oriented programming? With suitable example write details on member function overriding?**

**Answer:**

**Inheritance is the procedure in which one class inherits the attributes and methods of another class. The class whose properties and methods are inherited is known as the Parent class. And the class that inherits the properties from the parent class is the Child class.** **The interesting thing is, along with the inherited properties and methods, a child class can have its own properties and methods.  
Syntax:  
class base\_class\_name  
{  
 - - - - - - - - -   
 - - - - - - - - -  
}  
class derived\_class\_name : visibility\_mode base\_class\_name  
{  
 - - - - - - - - -  
 - - - - - - - - -   
}  
Inheritance is one of the most important aspects of Object Oriented Programming (OOP). The key to understanding Inheritance is that it provides code re-usability. In place of writing the same code, again and again, we can simply inherit the properties of one class into the other.** **This, as you can imagine, saves a ton of time. And time is money in data science!** **OOP is all about real-world objects and inheritance is a way of representing real-world relationships. Here’s an example – car, bus, bike – all of these come under a broader category called Vehicle. That means they’ve inherited the properties of class vehicles i.e. all are used for transportation. We can represent this relationship in code with the help of inheritance.  
Function overriding in C++ is a feature that allows us to use a function in the child class that is already present in its parent class. The child class inherits all the data members, and the member functions present in the parent class. If you wish to override any functionality in the child class, then you can implement function overriding. Function overriding means creating a newer version of the parent class function in the child class.**

**Source code:**

**#include<iostream>**

**#include<conio.h>**

**using namespace std;**

**class Base**

**{**

**protected:**

**int num;**

**public:**

**void getdata()**

**{**

**cout<<"Enter the number in base"<<endl;**

**cin>>num;**

**}**

**void showdata()**

**{**

**cout<<"Number in Base class = "<<num<<endl;**

**}**

**};**

**class derived : public Base**

**{**

**private:**

**int num;**

**public:**

**void getdata()**

**{**

**cout<<"Enter numbers in derived : "<<endl;**

**cin>>num;**

**}**

**void showdata()**

**{**

**cout<<"Number in Derived class = "<<num<<endl;**

**}**

**};**

**int main()**

**{**

**derived d1;**

**d1.getdata();**

**d1.showdata();**

**getch();**

**return(0);**

**}**

1. **Explain compile time binding and run time binding. Differentiate between abstrace base class and concrete class. Write an abstract class of your own choice and use it in a program. Your program should be meaningful.**

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **Compile Time Binding** | **SN** | **Run Time Binding** |
| **1** | **Compiler interacts with operating system memory manager to perform it.** | **1** | **It is done by processor at the time of program execution.** |
| **2** | **It is static address binding.** | **2** | **It is dynamic address binding.** |
| **3** | **Compiler is responsible for the compile time address binding.** | **3** | **Execution time address binding is done by processor.** |
| **4** | **Code is compiled here.** | **4** | **From memory instructions are executed by CPU.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **Abstract Class** | **SN** | **Concrete Class** |
| **1** | **A class declared with an abstract keyword which is a collection of abstract and non-abstract method is called abstract class.** | **1** | **A class that allows to create an instance or an object using the new keyword is called concrete class.** |
| **2** | **Programmers cannot create objects using an abstract class.** | **2** | **Programmers can create objects using a concrete class.** |
| **3** | **An abstract class can have unimplemented methods.** | **3** | **All methods in a concrete class are implemented.** |

**Source code: (General Program for the concept of abstract class)**

**#include<iostream>**

**#include<conio.h>**

**using namespace std;**

**class Point**

**{**

**public:**

**virtual void Draw()**

**{**

**cout<<"Point"<<endl;**

**}**

**};**

**class Line : public Point**

**{**

**public:**

**void Draw()**

**{**

**cout<<"Line"<<endl;**

**}**

**};**

**class Triangle : public Point**

**{**

**public:**

**void Draw()**

**{**

**cout<<"Triangle"<<endl;**

**}**

**};**

**class Polygon : public Point**

**{**

**public:**

**void Draw()**

**{**

**cout<<"Polygon"<<endl;**

**}**

**};**

**class Circle : public Point**

**{**

**public:**

**void Draw()**

**{**

**cout<<"Circle"<<endl;**

**}**

**};**

**int main()**

**{**

**Point Pt;**

**Line Ln;**

**Triangle Tr;**

**Polygon Py;**

**Circle Cr;**

**Point \*basepointer[]={&Pt,&Ln,&Tr,&Py,&Cr};**

**for (int i=0;i<5;i=i+1)**

**{**

**basepointer[i]->Draw();**

**}**

**getch();**

**return(0);**

**}**

**Source code: (General Program for the concept of abstract class(pure abstract class))  
#include<iostream>**

**#include<conio.h>**

**using namespace std;**

**class Polygon**

**{**

**public:**

**virtual void Draw()=0;**

**};**

**class Rectangle : public Polygon**

**{**

**public:**

**void Draw()**

**{**

**cout<<"Drawing Rectangle"<<endl;**

**}**

**};**

**class Triangle : public Polygon**

**{**

**public:**

**void Draw()**

**{**

**cout<<"Drawing Triangle"<<endl;**

**}**

**};**

**class Pentagon : public Polygon**

**{**

**public:**

**void Draw()**

**{**

**cout<<"Drawing Pentagon"<<endl;**

**}**

**};**

**int main ()**

**{**

**Polygon \*P[3];**

**Rectangle Rect;**

**Triangle Tri;**

**Pentagon Pent;**

**P[0]=&Rect;**

**P[1]=&Tri;**

**P[2]=&Pent;**

**int i;**

**for(i=0;i<3;i=i+1)**

**{**

**P[i]->Draw();**

**}**

**getch();**

**return(0);**

**}**

1. **Sequential and random access are two methods to access a data file. Which one do you prefer and why? Write a program to show opening, reading objects from file, checking end of file and closing the file.  
   Answer:**

**When we are talking about sequential or random access to data files we refer to the way data is written or read from a file on a computer system.  
Sequential Access to a data file means that the computer system reads or writes information to the file sequentially, starting from the beginning of the file and proceeding step by step.  
On the other hand, Random Access to a file means that the computer system can read or write information anywhere in the data file. This type of operation is also called “Direct Access” because the computer system knows where the data is stored (using Indexing) and hence goes “directly” and reads the data.  
I prefer random file access method because if the amount of data stored in a file is fairly large, the use of random access will allow you to search through it quicker. If it had been a sequential access file, you would have to go through one record at a time until you reach the target data. A random access file lets you jump directly to the target address where data is located.  
Source code:   
#include<iostream>**

**#include<conio.h>**

**#include<fstream> //headerfile for filestream**

**using namespace std;**

**class Student //class declaration**

**{**

**private: //private access specifier**

**char Name[30]; //private class member**

**int Roll;**

**public: //public access specifier**

**void getdata() //public class member function**

**{**

**cout<<"Enter Name"<<endl;**

**cin>>Name;**

**cout<<"Enter Roll Number"<<endl;**

**cin>>Roll;**

**}**

**void showdata()**

**{**

**cout<<"Name : "<<Name<<endl;**

**cout<<"Roll Number : "<<Roll<<endl;**

**}**

**};**

**void writetofile() //writing the datas in a file**

**{**

**Student S;**

**ofstream outfile("record.dat",ios::binary|ios::app);**

**S.getdata();**

**outfile.write(reinterpret\_cast <char \* >(&S),sizeof(S));**

**outfile.close();**

**}**

**void readfromfile()//reading the data form the file**

**{**

**Student S;**

**cout<<"DATA FROM FILE"<<endl;**

**ifstream infile("record.dat",ios::binary|ios::app);**

**while(!infile.eof())**

**{**

**if(infile.read(reinterpret\_cast<char \* >(&S),sizeof(S))>0)**

**{**

**S.showdata();**

**}**

**}**

**infile.close();**

**}**

**void readonerec() //reading specific data from the file**

**{**

**Student S;**

**int n; //n is declarec to specifically choose the file number**

**ifstream infile("record.dat",ios::binary);**

**cout<<"Enter record Number : ";**

**cin>>n;**

**infile.seekg((n-1)\*sizeof(S)); //searching the data in the file**

**infile.read(reinterpret\_cast<char\*>(&S),sizeof(S)); //reading the data from file**

**S.showdata(); //displaying the readed data from file**

**infile.close();**

**}**

**int main()**

**{**

**int Choice;**

**cout<<"STUDENT RECORD SYSTEM "<<endl;**

**cout<<"SELECT ONE OPTION BELOW "<<endl;**

**cout<<"1 to write record to file"<<endl;**

**cout<<"2 to read from file"<<endl;**

**cout<<"3 to read one record"<<endl;**

**cout<<"Any other Key to Exit"<<endl;**

**while(true)**

**{**

**cout<<"Enter Your Choice : "<<endl;**

**cin>>Choice;**

**switch(Choice)**

**{**

**case 1:**

**writetofile();**

**break;**

**case 2:**

**readfromfile();**

**break;**

**case 3:**

**readonerec();**

**break;**

**default:**

**cout<<"Choice Not Available";**

**exit(0);**

**}**

**}**

**getch();**

**return(0);**

**}**

1. **Why template is important in C++ programming? Write a program using template to add two numbers. Use the function template to pass the integer, float, and double. Display the returned result.**

**Answer:  
Templates in C++ is defined as a blueprint or formula for creating a generic class or a function. To simply put, you can create a single function or single class to work with different data types using templates. C++ template is also known as generic functions or classes which is a very powerful feature in C++.   
Syntax:  
template <class type >  
return\_type function\_name( parameter\_list )  
{  
 //body of function  
}  
Source code:   
#include<iostream>**

**#include<conio.h>**

**using namespace std;**

**template <class T1 ,class T2>**

**void add(T1 a, T2 b)**

**{**

**cout<<"Sum = "<<(a+b)<<endl;**

**}**

**int main()**

**{**

**int a,b;**

**cout<<"Enter two integer numbers"<<endl;**

**cin>>a>>b;**

**cout<<"The sum of two integer numbers is "<<endl;**

**add(a,b);**

**float c,d;**

**cout<<"Enter two float numbers"<<endl;**

**cin>>c>>d;**

**cout<<"The sum of two float numbers is "<<endl;**

**add(c,d);**

**double e,f;**

**cout<<"Enter two double numbers"<<endl;**

**cin>>e>>f;**

**cout<<"The sum of two double numbers is "<<endl;**

**add(e,f);**

**getch();**

**return(0);**

**}**

1. **How is exception handling mechanism better than traditional error handling? Explain how the exception is rethrown with a suitable program.  
   Answer:  
   Exception handling is the process of responding to unwanted or unexpected events when a computer program runs. Exception handling deals with these events to avoid the program or system crashing, and without this process, exceptions would disrupt the normal operation of a program.  
   Exception handling mechanism have different advantages over traditional error handling which are mentioned below:  
   Advantage 1: Separating Error Handling Code from "Regular" Code**

**Advantage 2: Propagating Errors up the Call Stack**

**Advantage 3: Grouping Error Types and Error Differentiation  
Source code:   
#include<iostream>**

**#include<conio.h>**

**#include<cmath>**

**using namespace std;**

**class Number**

**{**

**private:**

**double data;**

**public:**

**Number()**

**{**

**data=0;**

**}**

**Number (double d)**

**{**

**data=d;**

**}**

**class Neg**

**{**

**};**

**void readdata()**

**{**

**cout<<"\n Enter Number : "<<endl;**

**cin>>data;**

**}**

**double sqroot()**

**{**

**if(data<0)**

**{**

**throw Neg();**

**}**

**return((sqrt(data)));**

**}**

**double gmean(Number nm)**

**{**

**Number an(data\*nm.data);**

**double res;**

**try**

**{**

**res=an.sqroot();**

**}**

**catch(Neg)**

**{**

**cout<<"Square root of negative number is not possible "<<endl;**

**throw;**

**}**

**return res;**

**}**

**};**

**int main()**

**{**

**Number nm1,nm2(53.53);**

**nm1.readdata();**

**try**

**{**

**double gm=nm1.gmean(nm2);**

**cout<<"The geometric mean = "<<gm<<endl;**

**}**

**catch (Number :: Neg)**

**{**

**cout<<"The geometric mean of negative number cannot be calculated "<<endl;**

**}**

**getch();**

**return(0);**

**}**