**2070 Ashad**

1. **Write down the limitations of procedural programming. Compare procedural and object-oriented programming. Write a program to find prime number in procedural and object-oriented ways.**

Ans:

Limitations of procedural programming are:

* The program code is harder to write when Procedural Programming is employed.
* The Procedural code is often not reusable, which may pose the need to recreate the code if it is needed to use in another application
* Difficult to relate with real-world objects
* The importance is given to the operation rather than the data, which might pose issues in some data-sensitive cases
* The data is exposed to the whole program, making it not so much security friendly

|  |  |  |
| --- | --- | --- |
| S. N | Procedural Oriented Programming | Object-Oriented Programming |
| 1. | In procedural programming, the program is divided into small parts called functions. | In object-oriented programming, the program is divided into small parts called objects. |
| 2. | There is no access specifier in procedural programming. | Object-oriented programming has access specifiers like private, public, protected, etc. |
| 3. | Procedural programming does not have any proper way of hiding data so it is less secure. | Object-oriented programming provides data hiding so it is more secure. |
| 4. | In procedural programming, there is no concept of data hiding and inheritance. | In object-oriented programming, the concept of data hiding and inheritance is used. |
| 5. | Procedural programming is based on the unreal world. | Object-oriented programming is based on the real world. |

|  |  |
| --- | --- |
| 6. WAP to find prime number in: | |
| Procedural oriented way | Object oriented way |
| #include<stdio.h>  #include<conio.h>  void main()  {      int num,count=0,i;      printf("Enter a number: ");      scanf("%d",&num);      for(i=1;i<=num;i++)      {          if(num%i==0)          {          count++;          }      }      if(count==2)      {          printf("%d is a prime number",num);      } else {          printf("%d is not a prime number",num);      }  } | #include<iostream>  using namespace std;  int main()  {      int num,count=0;      cout<<"Enter a number: ";      cin>>num;      for(int i=1;i<=num;i++)      {          if(num%i==0)          {          count++;          }      }      if(count==2)      {          cout<<num<<" is a prime number";      } else {          cout<<num<<" is not a prime number";      }      return 0;  } |

**2) What do you understand by friend functions and classes? Explain with example. Write a program to add members of objects of two different classes.**

Ans:

The outside functions can’t access the private data of a class. But there could be a situation where we would like two classes to share a particular. C++ allows the common function to have access to the private data of the class. Such a function may or may not be a member of the classes. To make an outside function “friendly” to a class, we have to simply declare this function as a friend of the class as:

class ABC

{

 - - - - - -

 - - - - - -

 public:

 - - - - - -

 - - - - - -

 friend void xyz(arg\_list); //declaration

} ;

 void xyz(arg\_list) //function definition

{

 //function body

}

#include<iostream>

using namespace std;

class Second;

class First{

    int num1;

    public:

    First()

    {

        cout<<"Enter a number: ";

        cin>>num1;

    }

    friend add (First, Second);

};

class Second {

    int num2;

    public:

    Second ()

    {

        cout<<"Enter another number: ";

        cin>>num2;

    }

    friend add (First, Second);

};

int add(First a, Second b)

{

    return(a.num1+b.num2);

}

int main()

{

    First f;

    Second s;

    cout<<"Sum = "<<add(f,s);

    return 0;

}

**3) What do you mean by namespace? Explain how namespace can be used. Write a program that uses pass by reference to change metre to centimetre using pass by reference along with the namespace.**

Ans:

A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc.) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.

The general syntax is:

 namespace namespace\_name

 {

 // Declaration of variable, function, classes, etc

 }

#include<iostream>

using namespace std;

namespace MtoCm{

    float x;

 void convert(float &x)

 {

    x=x\*100;

 }

}

using namespace MtoCm;

int main()

{

    float temp;

    cout<<"Enter meter: ";

    cin>>x;

    temp=x;

    convert(x);

    cout<<temp<<"m"<<" = "<<x<<"cm";

    return 0;

}

**4) Explain the binary and unary operator overloading along with their syntax and example. Write a program to add two matrices by overloading the + operator.**

Ans:

Binary operator overloading: An operator which contains two operands to perform a mathematical operation is called the binary operator overloading. It is a polymorphic compile technique where a single operator can perform various functionalities by taking two operands from the programmer or user. There are multiple binary operators like +, -, \*, /, etc., that can directly manipulate or overload the object of a class.

 Syntax:

return\_type operator binary\_operator\_symbol (arg)

{

 //function definition

}

Unary operator overloading: Unary operators are those which operate on a single variable. Overloading unary operator means extending the operator’s original functionality to operate upon object of the class.

 Syntax:

void operator unary\_operator\_symbol ()

{

 //function definition

}

#include<iostream>

using namespace std;

class Matrix{

    int m[3][3];

    public:

    void getData()

    {

        cout<<"Enter elements of 3x3 matrix: ";

        for(int i=0;i<=2;i++)

        {

            for(int j=0;j<=2;j++)

            {

                cin>>m[i][j];

            }

        }

    }

    Matrix operator +(Matrix x)

    {

        Matrix temp;

        for(int i=0;i<=2;i++)

        {

            for(int j=0;j<=2;j++)

            {

                temp.m[i][j]=m[i][j]+x.m[i][j];

            }

        }

        return temp;

    }

    void displaySum()

    {

        cout<<"Sum of entered matrix = "<<endl;

        for(int i=0;i<=2;i++)

        {

            for(int j=0;j<=2;j++)

            {

                cout<<m[i][j]<<"\t";

            }

            cout<<endl;

        }

    }

};

int main()

{

    Matrix m1,m2,m3;

    m1.getData();

    m2.getData();

    m3=m1+m2; // m3=m1.operator +(m2);

    m3.displaySum();

    return 0;

}

**5) Explain the constructor and destructor invocation order in single and multiple inheritance. Also show how a parameterized base class constructor is called when derived class object are created. Write a program to create classes to represent student, teaching staffs and non-teaching staff from the base class person. Use proper members in the classes to make your program meaningful.**

Ans:

In single and multiple inheritance, the constructor of the base class is automatically executed first and then the constructor of the derived class is executed. Destructors of are executed in reverse order, i.e., from derived class to base class.

2

To call the parameterized constructor of base class when derived class’s parameterized constructor is called, we have to explicitly specify the base class’s parameterized constructor in derived class as shown in below program:

#include <iostream>

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

**class** **Parent** {

**int** x;

**public:**

Parent(**int** i)

{

x = i;

cout << "Inside base class's parameterized constructor"<<endl;

}

};

**class** **Child** : **public** Parent {

**public:**

Child(**int** x): Parent(x)

{

cout << "Inside sub class's parameterized constructor"<<endl;

}

};

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

{

Child obj1(**10**);

**return** **0**;

}

Output:

Inside base class's parameterized constructor

Inside sub class's parameterized constructor

#include<iostream>

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

**class** **Person**{

**protected:**

**char** name[**50**];

**int** age;

**public:**

**void** **getData**()

{

cout<<"Enter name: ";

cin>>name;

cout<<"Enter age: ";

cin>>age;

}

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

{

cout<<endl<<"Name: "<<name;

cout<<endl<<"Age: "<<age;

}

};

**class** **Student**:**public** Person{

**int** roll;

**public:**

Student()

{

cout<<"Student Info"<<endl;

}

**void** getRoll()

{

cout<<"Enter roll number: ";

cin>>roll;

display();

cout<<endl<<"Roll: "<<roll<<endl;

}

};

**class** **TeachingStaff**:**public** Person{

**int** nPeriods;

**float** salary;

**public:**

TeachingStaff()

{

cout<<endl<<"Teaching staff info: "<<endl;

}

**void** getInfo()

{

cout<<"Enter number of periods: ";

cin>>nPeriods;

cout<<"Enter salary: ";

cin>>salary;

display();

cout<<endl<<"Number of periods: "<<nPeriods;

cout<<endl<<"Salary: "<<salary<<endl;

}

};

**class** **NonTeachingStaff**:**public** Person{

**float** salary;

**public:**

NonTeachingStaff()

{

cout<<endl<<"Non-teaching staff info: "<<endl;

}

**void** getInfo()

{

cout<<"Enter salary: ";

cin>>salary;

display();

cout<<endl<<"Salary: "<<salary<<endl;

}

};

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

{

Student s;

s.getData();

s.getRoll();

TeachingStaff t;

t.getData();

t.getInfo();

NonTeachingStaff n;

n.getData();

n.getInfo();

**return** **0**;

}

**6) What do you mean by manipulators? Explain different manipulators available in C++. Write a program that stores information of a student in a file and display the file’s content in descending order according to their marks obtained.**

Ans:

The manipulators in C++ are stream functions that change the properties of an input or output stream. It's used to format the input and output streams by modifying the stream's format flags and values.

Here is the list of standard input/output Manipulators and their Functions in C++

* setw (int n) – To set field width to n.
* Setbase – To set the base of the number system.
* setprecision (int p) – The precision is fixed to p.
* setfill (Char f) – To set the character to be filled.
* endl – Gives a new line.
* skipws – Omits white space in input.
* noskipws – Does not omit white space in the input.
* ends – Adds null character to close an output string.
* flush – Flushes the buffer stream.
* hex, oct, dec – Displays the number in hexadecimal or octal or in decimal format

#include<iostream>

#include<fstream>

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

**class** **Student**{

**private:**

**char** name[**50**];

**int** marks;

**public:**

**void** **read\_data**()

{

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

cin>>name;

cout<<"Marks obtained: ";

cin>>marks;

}

**void** **show\_data**()

{

cout<<"Name: "<<name<<endl;

cout<<"Marks obtained: "<<marks<<endl;

}

**int** **getMarks**()

{

**return** marks;

}

};

**void** **write2file**()

{

Student stu;

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

stu.read\_data();

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

}

**void** **readfromfile**()

{

Student stu,temp[**10**];

**int** count=**0**;

cout<<"**\n**\*\*\*\*\*\*\*\*\*\* Data From File \*\*\*\*\*\*\*\*\*\*"<<endl;

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

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

{

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

{

temp[count] = stu;

count++;

}

}

**for**(**int** j=**0**;j<count-**1**;j++)

{

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

{

**if**(temp[i].getMarks() < temp[i+**1**].getMarks())

{

Student temp2;

temp2=temp[i];

temp[i]=temp[i+**1**];

temp[i+**1**]=temp2;

}

}

}

**for**(**int** i=**0**;i<count;i++)

{

temp[i].show\_data();

}

}

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

{

**int** choice;

cout<<"\*\*\*\*\*\*\*\*\*\* Student Record System \*\*\*\*\*\*\*\*\*";

cout<<endl<<"Select any option below: "<<endl;

cout<<"**\t**1. Write records to file";

cout<<"**\t**2. Read records from file";

cout<<"**\t**3. Exit form Program";

**while**(true)

{

cout<<"**\n**Enter your choice: ";

cin>>choice;

**switch**(choice)

{

**case** **1**:

write2file();

**break**;

**case** **2**:

readfromfile();

**break**;

**case** **3**:

exit(**0**);

**break**;

**default:**

cout<<"**\n**Choice not available";

exit(**0**);

}

}

**return** **0**;

}

**7) What are virtual functions and pure virtual functions? Explain abstract class and its uses. Write a program having students as an abstract class and create derived class such as Engineering, Science and Medical. Show the use of virtual functions in this program.**

Ans:

A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class’s version of the function.

Syntax:

virtual return\_type function\_name()

{

//function definition

}

A pure virtual function is a virtual function with null body (=0). The base class which has at least one pure virtual function is called abstract class. Abstract class impose compulsion to the derived class to override the pure virtual function.

For example:

class Shape{

public:

Virtual void draw()=0;

}

#include<iostream>

using namespace std;

class Student{

protected:

char name[50];

int roll;

public:

virtual void getData()=0;

virtual void display()=0;

};

class Engineering:public Student{

public:

void getData()

{

cout<<"Enter name and roll number of engineering student: ";

fflush(stdin);

gets(name);

cin>>roll;

}

void display()

{

cout<<"Name: "<<name<<endl<<"Roll: "<<roll<<endl;

}

};

class Science:public Student{

public:

void getData()

{

cout<<"Enter name and roll number of science student: ";

fflush(stdin);

gets(name);

cin>>roll;

}

void display()

{

cout<<"Name: "<<name<<endl<<"Roll: "<<roll<<endl;

}

};

class Medical:public Student{

public:

void getData()

{

cout<<"Enter name and roll number of medical student: ";

fflush(stdin);

gets(name);

cin>>roll;

}

void display()

{

cout<<"Name: "<<name<<endl<<"Roll: "<<roll<<endl;

}

};

int main()

{

Student \*ptr;

Engineering e;

Medical m;

Science s;

ptr = &e;

ptr->getData();

ptr->display();

ptr=&m;

ptr->getData();

ptr->display();

ptr=&s;

ptr->getData();

ptr->display();

return 0;

}

**8) What do you understand by function template? Write down the syntax and use of function template. Write a program that will find the sum and average of elements in an array using function templates.**

Ans:

Function templates are special functions that can operate with generic types. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

Syntax:

template <class identifier> function\_declaration;

#include<iostream>

using namespace std;

template <class X,class Y=int>

void Find(X \*p,Y size)

{

    X sum=0,average;

    int i;

    for(i=0;i<size;i++)

    {

        sum=sum+p[i];

    }

    cout<<"Sum = "<<sum<<endl<<"Average = "<<sum/(i\*1.0);

}

int main()

{

    int size,\*arr;

    float \*floatArray;

    cout<<"How many integer numbers do you want to enter: ";

    cin>>size;

    arr = new int[size];

    for(int i=0;i<size;i++)

    {

        cin>>arr[i];

    }

    Find(arr,size);

     cout<<endl<<"How many float type numbers do you want to enter: ";

    cin>>size;

    floatArray = new float[size];

    for(int i=0;i<size;i++)

    {

        cin>>floatArray[i];

    }

    Find(floatArray,size);

    return 0;

}

#include<iostream>

using namespace std;

template <class X,class Y=int>

void Find(X \*p,Y size)

{

X sum=0,average;

int i;

for(i=0;i<size;i++)

{

sum=sum+p[i];

}

cout<<"Sum = "<<sum<<endl<<"Average = "<<sum/(i\*1.0);

}

int main()

{

int size,\*arr;

float \*floatArray;

cout<<"How many integer numbers do you want to enter: ";

cin>>size;

arr = new int[size];

for(int i=0;i<size;i++)

{

cin>>arr[i];

}

Find(arr,size);

cout<<endl<<"How many float type numbers do you want to enter: ";

cin>>size;

floatArray = new float[size];

for(int i=0;i<size;i++)

{

cin>>floatArray[i];

}

Find(floatArray,size);

return 0;

}