1. Write a C++ program to create a templatized Class MAT of size m\*n. Define all possible matrix operations for MAT type objects (operator overloading program).

**Ans:**

#include<iostream>

#define MAX 20

using namespace std;

int i,j;

template <class T>

class MAT

{

T arr[MAX][MAX];

int row;

int col;

public:

void getdata()

{

cout << "Enter the number of rows and columns in the matrix:";

cin >> row >> col;

cout << "Enter the elements in the matrix:" << endl;

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

{

for(j=0;j<col;j++)

cin >> arr[i][j];

}

}

void display()

{

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

{

for(j=0;j<col;j++)

cout << arr[i][j] << " ";

cout << endl;

}

cout << "----------------------------------------------------------------------------" << endl;

}

MAT operator \*(int k)

{

MAT m;

m.row = row;

m.col = col;

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

{

for(j=0;j<col;j++)

m.arr[i][j] = arr[i][j]\*k;

}

return m;

}

friend int check(MAT m1, MAT m2,char sign)

{

if (sign == '+' || sign == '-')

{

if ((m1.row == m2.row) && (m1.col == m2.col))

return 1;

return -1;

}

if (sign == '\*')

{

if (m1.col == m2.row)

return 1;

return -1;

}

}

MAT operator + (MAT x)

{

MAT m;

m.row = row;

m.col = col;

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

{

for(j=0;j<col;j++)

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

}

return m;

}

MAT operator - (MAT x)

{

MAT m;

m.row = row;

m.col = col;

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

{

for(j=0;j<col;j++)

m.arr[i][j] = arr[i][j] - x.arr[i][j];

}

return m;

}

MAT operator \*(MAT x)

{

MAT m;

m.row = row;

m.col = x.col;

int sum,k;

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

{

for(j=0;j<x.col;j++)

{

sum = 0;

for(k=0;k<col;k++)

{

sum = sum + arr[i][k]\*x.arr[k][j];

}

m.arr[i][j] = sum;

}

}

return m;

}

};

int main()

{

MAT <int>m1;

MAT <int>m2;

MAT <int>m3;

int k,addsub\_check,mul\_check;

m1.getdata();

m2.getdata();

cout << "The elements in matrix 1 are:" << endl;

m1.display();

cout << "The elements in matrix 2 are:" << endl;

m2.display();

cout << " MULTIPLYING A MATRIX BY A SCALAR " << endl;

cout << "Enter a scalar to be multiplied to matrix 1:";

cin >> k;

m3 = m1\*k;

m3.display();

cout << " ADDING TWO MATRICES " << endl;

addsub\_check = check(m1,m2,'+');

if (addsub\_check == 1)

{

m3 = m1 + m2;

cout << "The result of m1 + m2 is:" << endl;

m3.display();

}

else

{

cout << "Matrix 1 and 2 can't be added" << endl;

cout << "----------------------------------------------------------------------------" << endl;

}

cout << " SUBTRACTING TWO MATRICES " << endl;

addsub\_check = check(m1,m2,'-');

if (addsub\_check == 1)

{

m3 = m1 - m2;

cout << "The result of m1 - m2 is:" << endl;

m3.display();

}

else

{

cout << "Matrix 1 and 2 can't be subtracted" << endl;

cout << "----------------------------------------------------------------------------" << endl;

}

cout << " MULTIPLYING TWO MATRICES " << endl;

mul\_check = check(m1,m2,'\*');

if (mul\_check == 1)

{

m3 = m1 \* m2;

cout << "The result of m1 \* m2 is:" << endl;

m3.display();

}

else

cout << "Matrix 1 and 2 can't be multiplied" << endl;

return 0;

}

2. Write a C++ program to implement templatized functions to swap two elements.

**Ans:**

#include<iostream>

#include<iomanip>

using namespace std;

template<class T>

void swap(T \*x,T \*y)

{

T tmp = \*x;

\*x = \*y;

\*y = tmp;

}

int main()

{

int a = 10,b = 20;

float c = 1.1,d = 1.5;

char s1 = 'H',s2 = 'W';

cout << " SWAPPING TWO INTEGER VALUES " << endl;

cout << "----------------------------------------\n";

cout << "The values before swapping:" << endl;

cout << "a = " << a << setw(10) << " b = " << b << endl;

swap<int>(&a,&b);

cout << "The values before swapping:" << endl;

cout << "a = " << a << setw(10) << " b = " << b << endl;

cout << "----------------------------------------\n";

cout << " SWAPPING TWO FLOAT VALUES " << endl;

cout << "----------------------------------------\n";

cout << "The values before swapping:" << endl;

cout << "c = " << c << setw(10) << " d = " << d << endl;

swap<float>(&c,&d);

cout << "The values after swapping:" << endl;

cout << "c = " << c << setw(10) << " d = " << d << endl;

cout << "----------------------------------------\n";

cout << " SWAPPING TWO CHAR VALUES " << endl;

cout << "----------------------------------------\n";

cout << "The values before swapping:" << endl;

cout << "s1 = " << s1 << setw(10) << " s2 = " << s2 << endl;

swap<char>(s1,s2);

cout << "The values before swapping:" << endl;

cout << "s1 = " << s1 << setw(10) << " s2 = " << s2 << endl;

return 0;

}

3. Distinguish between the two following statements: Explain Time T2(T1) Time T2=T1;

**Ans:** Time T2(T1) is an example of copy constructor while Time T2 = T1 is a case of overloading of **=** operator

4. Define a string class “**String**” that could work as a user defined string type. Include

constructors that will enable us to create an uninitialized string

**String s1;** // ie with length 0 and also to initialize an object with a string constant at

the time of creation like **string s2(“ new delhi”).** Include a function that adds two

strings to make a third string . Write a complete C++ program to test your class to

see that it does the following tasks:

**a.** Create uninitialized string objects

**b.** Create objects with string constants

**c.** Concatenates two strings properly

**d.** Display a desired string object

**Ans:**

#include<iostream>

#include<cstring>

using namespace std;

class String

{

int len;

char arr[30];

public:

String()

{

len = 0;

strcpy(arr,"\0");

}

String(char a[30])

{

len = strlen(a);

strcpy(arr,a);

}

friend void operator << (ostream &x, String &y)

{

x << y.arr;

}

friend void operator >> (istream &x, String &y)

{

x >> y.arr;

}

String operator + (String &x)

{

String s;

s.len = len + x.len;

strcpy(s.arr,arr);

strcat(s.arr," ");

strcat(s.arr,x.arr);

return s;

}

};

int main()

{

//CREATE UNINITIALIZED STRING OBJECTS

String s1,s2,s3;

//CREATE OBJECTS WITH STRING CONSTANTS

String s4("New Delhi");

cout << "String 4 is initialized to ";

cout << s4;

cout << "\nEnter two strings:";

cin >> s1;

cin >> s2;

//CONCATENATES TWO STRINGS PROPERLY

cout << "The concatenated string s3 is:";

s3 = s1 + s2;

//DISPLAY A DESIRED STRING OBJECT

cout << s3;

cout << endl;

}

5. What is a friend function? what are the merits and demerits of using friend function

**Ans:** A friend function is a function that is declared outside a class but is capable of accessing private and protected members of a class.

Syntax:

class class-name

{

…

…

public:

…

…

friend return-type function-name (arguments);

}

Merits of friend function:

1. While defining a friend function there is no need to use scope resolution operator.

2. It can be used to increase versatility of overloading operator

3. It can be invoked like a normal function without help of object

4. It allows sharing of private class information by a non-member function

Demerits of friend function:

1. It violates the purpose of data hiding in C++ by allowing access to private members of class from outside the class.

2. Breach of data integrity

3. A derived class does not inherit friend function

6. Write a C++ program to define two classes’ centimeters’ and ‘meters’. Define a

conversion routine to convert centimeters to meters (class to class type conversion)

**Ans:**

#include<iostream>

#include<iomanip>

using namespace std;

class centimeters

{

float cm;

public:

centimeters(float c)

{

cm = c;

}

void display()

{

cout << "The distance in cm is:";

cout.setf(ios::fixed,ios::floatfield);

cout.precision(3);

cout << cm << endl;

}

float returnc()

{

return cm;

}

};

class meters

{

float m;

public:

meters()

{

m = 0.0;

}

meters(centimeters c)

{

m = c.returnc()/100.0;

}

void display()

{

cout << "The distance in m is:";

cout.setf(ios::fixed,ios::floatfield);

cout.precision(3);

cout << m << endl;

}

};

int main()

{

float c;

cout << "Enter distance in cm:";

cin >> c;

centimeters cm1 = centimeters(c);

cm1.display();

meters m1;

m1 = cm1;

m1.display();

return 0;

}

7. Write all the file error handling functions and diff file pointers.

**Ans:** Following are all the file error handling functions:

1. eof(): Returns true if end of file is encountered while reading. Otherwise return false/zero

2. fail(): Returns true when an input/output operation has failed

3. bad(): Returns true if an invalid operation is attempted or any unrecoverable error has occurred. However, if it is false, it may be possible to recover from any other error reported, and continue operation.

4. good(): Returns true if no error has occurred. This means, all the above functions are false. For instance, if file.good() is true, all is well with the stream file and we can proceed to perform I/O operations. When it returns false, no further operations can be carried out.

Each file has two associated pointers known as file pointers. One of them is called input pointer/ get pointer and other is called output pointer/ put pointer. We can use these pointers to move through files while reading/writing. The input pointer is used for reading the contents of a given file location and the output pointer is used for writing to a given file location. Each time an input/output operation takes place, the appropriate pointer is automatically advanced.

8. Write a C++ program to show how to use read() and write() functions for file

operations.

**Ans:**

#include<iostream>

#include<iomanip>

#include<fstream>

using namespace std;

int main()

{

float arr[4];

int n=4,i=0;

ofstream fout;

ifstream fin;

cout << "Enter the elements:" << endl;

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

cin >> arr[i];

fout.open("File.txt");

fout.write((char \*)&arr,sizeof(arr)); //Writing arr[] to File.txt using write() function

fout.close();

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

arr[i] = 0;

fin.open("File.txt");

fin.read((char \*)&arr,sizeof(arr)); //Reading from File.txt using read() function

fin.close();

cout << "The elements in the array arr are:" << endl;

cout << "[ ";

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

{

cout << setprecision(3) << arr[i] << " ";

}

cout << "]" << endl;

return 0;

}

9. Explain the different STL components with examples of each.

**Ans:** The three key STL components are containers, algorithms and iterators.

**Container:** is an object that actually stores data, It is a way data is organized in memory. The STL containers are implemented by template classes and therefore can be easily customized to hold different types of data. Example: Sequence containers (vector, deque, list), Associative containers (set, multiset, map, multimap) and Derived containers (stack, queue, priority queue)

**Algorithm:** is a procedure that is used to process the data contained in containers. The STL includes many different kinds of algorithms to provide support to tasks such as initializing, searching, copying, sorting, and merging. Algorithms are implemented by template functions.

**Iterator:** is an object (like a pointer) that points to an element in a container. We can use iterators to move through the contents of containers. Iterators are handled just like pointers. We can increment or decrement them. Iterators connect algorithms with containers and play a key role in the manipulation of data stored in the containers. Example: random access, bidirectional

10.What are STL iterators, explain all the different categories **Ans:** STL iterators behave like pointers and used to access container elements. They are often used to traverse from one element to another, a process known as iterating through the container. The different types of iterators are as follows: **1. Input iterators:** are considered to be the weakest as well as simplest of all iterators available based on their functionality and what can be achieved using them. They are the iterators that can be used in sequential input operations where each value pointed by iterator is read-only once and then iterator is incremented. Direction of movement is forward only and access method is linear.

**2. Output iterators:** are considered to be exactly opposite of input iterators as they perform the opposite function of input iterators. They can be assigned values in a sequence but can’t be used to access values unlike input iterators which do the reverse of accessing values and cannot be assigned values. Direction of movement is forward only and access method is linear. I/O capability is write only.

**3. Forward iterators:** are considered to be combination of input as well as output iterators. It provides support to the functionality of both of them. It permits values to be both accessed and modified. Direction of movement is forward only and access method is linear. I/O capability is read/write.

**4. Bidirectional iterators:** are iterators that can be used to access sequence of elements in a range in both directions. They are similar to forward iterators except that they can move in backward direction unlike forward iterators which can move in forward direction only. Access method is linear. I/O capability is read/write. Example: list, map, multimap, set and multiset.

**5. Random iterators:** are iterators that can be used to access elements at an arbitrary offset position relative to the element they point to, offering the same functionality as pointers. Random- access iterators are the most complete iterators in terms of functionality. All pointer types are also valid random-access iterators. Direction of movement is forward as well as backward and access method is random. I/O capability is read/write. Example: vector, deque