National University of Computer and Emerging Sciences

Lab Manual 12
Object Oriented Programming

Course Instructor	Miss Hafsa Tariq
Lab Instructor (s)	Ms. Sonia Anum Ms. Yusra Arshad
Section	BCS-2J
Semester	Spring 2022

Department of Computer Science
FAST-NU, Lahore, Pakistan

Objectives:

- Polymorphism and Down casting
- Template classes, functions and specialization
- Error Handling

Task 1:

1. You have to design a C++ **template** function **range**, which takes a dynamic two-dimensional square matrix, its dimensions (rows, columns) size. It returns the range of values in matrix.

- Range = ((max – min)/4) + min.
- **Note:** No specialization is required for this function. **Do not** take **input** from **user**. **Initialize** a 2D array in **main** and call the function.
- **Template <typename T> T range (T **array, int rows, int columns)**

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Range = ((15-0)/4) + 0

Range = 3

A	B	C
H	I	D
G	F	E

Range = ((I-A)/4) + A

Range = C

2. Write another template function that will Shift the columns of matrix by 1 and print the shifted matrix. What would be the prototype of this function?

C	A	B
D	H	I
E	G	F

Task 2:

If we want to define a different implementation for a template when a specific type is passed as template parameter, we can declare a specialization of that template.

```
template <>
char* maximum <char*>(char*x,char*y) {
        if(strcmp(x,y) ==1)
              return x;
        else
              return y;
};
```

Consider a template function **increment**, that receives a variable (it can be int, double, float etc) and increase the value of that variable by 1.

Now Write a template specialization for char * variables (character arrays) that convert all letters

of character arrays to upper case.
Hint:
Lowercase characters ASCII range from 97 to 122. If the character is found to be in this range then the program converts that character into an uppercase character. ASCII of ‘A’ and ‘a’ differs by 32.

Task 3:

Consider the following class template:

```
template <class T, int N>
class Sequence {
   T memblock [N];
  public:
   void setmember (int x, T value);
   T getmember (int x);
};
```

Sequence is a class that stores a sequence of elements. N is an integer. The member function setmember sets the member at position x in the memblock with value and getmember returns the value at index x.

a. Implement the Sequence class w.r.t the following **main**

```
int main ()
{
  Sequence <int,5> myints;
  Sequence <double,5> myfloats;
  myints.setmember (0,100);
  myfloats.setmember (3,3.1416);
  cout << myints.getmember(0) << '\n';
  cout << myfloats.getmember(3) << '\n';
  return 0;
}
```

Task 4:

Define an exception class called tornadoException. The class should have two constructors including the default constructor. If the exception is thrown with the default constructor, the method what should return "Tornado: Take cover immediately!" The other constructor has a single parameter, say m, of the int type. If the exception is thrown with this constructor, the method what should return "Tornado: m miles away; and approaching!" Write a C++ driver program to test the class tornadoException.

Task 5:

For the first part of this lab, we are going to see if base class pointers can point to an object of derived class

and vice versa. Perform the following steps

- Create a class called `Animal`.
- An animal can speak so create a virtual public method named speak in the which returns a `char *`
- Modify the definition of the speak method so that it returns the string "`speak() called.`".
- A Dog is an Animal. Create a class named `Dog`. Use public inheritance.
- The `Dog` class will inherit the `speak` method from `Animal`. Override this method in the `Dog` class so that it returns "woof!" when called.
- Add the following lines in the main function of your program, note the output and paste it in the space provided below.

```
Animal objAnimal;
Dog objDog;
Animal *ptrAnimal = &objAnimal;
Dog *ptrDog = &objDog;

cout << objAnimal.speak() << endl;
cout << objDog.speak() << endl;
cout << ptrAnimal->speak() << endl;
cout << ptrDog->speak() << endl;
```

You can see that we have created two objects, one for each class and two pointers in the same manner. The pointer to `Animal` is pointing to an object of the class `Animal` and the pointer to `Dog` is pointing to the object of class `Dog`. In this example, `ptrAnimal` called the `speak` method of the class `Animal` where as `ptrDog` called the `speak` method of the class Dog. If we want to use the definition of the base class method that we have overloaded in a derived class from a derived class pointer, we have to follow this syntax.

```
ptrDog->Animal::speak();
```

Modify the last line of your program to use the syntax above, execute it and paste the output in the box below.

Task 6:

Now we will see what happens if we change this. Change the main function so that `ptrAnimal` is pointing

to the object of `Dog`, execute your program and paste the output below

Now modify the code so that `ptrDog` points to `objAnimal` and compile your program. It will not compile successfully. Paste the error in the space below. What does this show?

Error: a value of type “Animal *” cannot be used to initialize an entity of type “Dog *”

You can see that there was no problem pointing to an object of a derived class by a pointer of the base class but when calling a function through this pointer, the definition of the base class is used. In the other case, there was a compilation error which showed that it is not possible to point a derived class pointer to an object of base class,then downcasting comes up.Now change the main functions for so that derive class pointer points to the base class object.