Experiment –Week 14

**Template Classes**

Templates are a very powerful feature of C++. They allow you to write a single code segment for a set of related functions, called a function template, and for a set of related classes, called a class template. The syntax we use for templates is:

template <class Type> declaration;

**Function templates:** A function template behaves like a function except that the template can have arguments of many different types In other words; a function template represents a family of functions. The general format of a function template is as follows:

template<class T>

return\_type functionname(argument T )

{

// body of function with Type T

};

**Class templates:** A class template provides a specification for generating classes based on parameters. Class templates are generally used to implement [containers.](http://en.wikipedia.org/wiki/Container_(data_structure))

template <class type>

class class\_name

{

//(Body of the class)

};

Let’s take an example, just declare an “integer” generic class. This class contains two member variables which are of Type T, and a member function “greater” to return greater number.

template<class T>

class number

{

T no\_1, no\_2; public:

number (T n1, T n2)

{ no\_1=n1;

no\_2=n2;

}

T greater();

};

The class that we have just defined serves to store two elements of any valid type. For example, if we wanted to declare an object of this class to store two integer values of type “int” with the values 115 and 36 we would write:

number<int> myobject1(1,2);

This same class could also be used to create an object to store any other type, such as:

number<double> myobject2(1.2,2.2);

If a member function is defined outside the definition of the class template, it shall be preceded with the template <...> prefix:

Notice the syntax of the definition of a member function “greater” is:

template<class T>

T number<T>::greater()

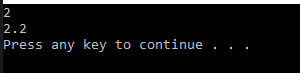
{

// body of greater definition

}

**Exercise – 1 Write Output of the following code**

|  |
| --- |
| template<class T>  class number  {  T no\_1, no\_2;  public:  number (T n1, T n2) { no\_1=n1; no\_2=n2; }  T greater();  };  template<class T>  T number<T>::greater()  {  if(no\_1>no\_2)  return no\_1;  else  return no\_2;  }  void main()  {  number<int> myobject1(1,2);  cout<<myobject1.greater()<<endl;  number<double> myobject2(1.2,2.2);  cout<<myobject2.greater()<<endl;  }  **Output:** |



### Exercise 2: (10 points)

Study the myMAXfunction provided below. You are required to create a C++ template based

myMAXfunction and test it on different built-in data types.

//Make a template out of this function. Don't forget the return type. int myMax(int one, int two)

{

int bigger; if(one < two)

bigger = two;

else

bigger = one; returnbigger;

}

int main()

{

int i\_one = 3, i\_two = 5;

cout <<"The max of "<< i\_one <<" and "<< i\_two <<" is "

<< myMax(i\_one, i\_two) << endl;

//Test your template on float and string types return 0;

}

#include <iostream>

#include <string>

using namespace std;

template <typename T>

T myMax(T one, T two) {

   if (one < two) {

      return two;

   }

   else {

      return one;

   }

}

int main() {

   int i\_one = 3, i\_two = 5;

   cout << "The max of " << i\_one << " and " << i\_two << " is "

      << myMax(i\_one, i\_two) << endl;

   float f\_one = 4.5, f\_two = 6.7;

   cout << "The max of " << f\_one << " and " << f\_two << " is "

      << myMax(f\_one, f\_two) << endl;

   string s\_one = "apple", s\_two = "banana";

   cout << "The max of " << s\_one << " and " << s\_two << " is "

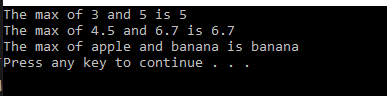
      << myMax(s\_one, s\_two) << endl;

   system("pause");

   return 0;

}

**Output:**



**Exercise – 3:**

Consider the class of points in the *xy* plane. The location of each point is determined by the real numbers (x, y) specifying the cartesian coordinates. The class definition is:

#include<iostream>

using namespace std;

class point{

public:

point();

point(double value\_x, double value\_y);

double get\_x() const;

double get\_y() const;

void print() const;

void move(double dx, double dy);

private:

double x, y;

};

point::point(){

x = 0.0; y = 0.0;

}

point::point(double a, double b){

x = a; y = b;

}

void point::print() const{

cout<<x<<""<<y<< endl;

}

double point::get\_x() const{

return x;

}

double point::get\_y() const{

return y;

}

void point::move(double dx, double dy){

x = x+dx;

y = y+dy;

}

**Generalize the class Point into a template and test your code using following main function.**

int main()

{

point<int> A = point<int>(1, 2);

A.print();

A.move(4, -5);

A.print();

point<float>B(3.2, 4.9);

cout << B.get\_x() <<""<< B.get\_y() << endl ;

point<string> C("day", "young");

C.print();

C.move("s","ster");

C.print();

return 0;

}

#include<iostream>

#include<string>

using namespace std;

template <typename T>

class point{

public:

   point();

   point(T value\_x, T value\_y);

   T get\_x();

   T get\_y();

   void print();

   void move(T dx, T dy);

private:

   T x, y;

};

template <typename T>

point<T>::point(){

   x = 0.0; y = 0.0;

}

template <typename T>

point<T>::point(T a, T b){

   x = a; y = b;

}

template <typename T>

void point<T>::print(){

   cout << x << " " << y << endl;

}

template <typename T>

T point<T>::get\_x(){

   return x;

}

template <typename T>

T point<T>::get\_y(){

   return y;

}

template <typename T>

void point<T>::move(T dx, T dy){

   x = x + dx;

   y = y + dy;

}

int main()

{

   point<int> A(1, 2);

   A.print();

   A.move(4, -5);

   A.print();

   point<float> B(3.2, 4.9);

   cout << B.get\_x() << " " << B.get\_y() << endl;

   point<string> C("day", "young");

   C.print();

    C.move("s","ster");

    C.print();

    system("pause");

   return 0;

}

**Output:**

