**Lab Sections**

1. Objectives
2. Introduction
3. Declaration Syntax
4. Experiments

|  |
| --- |
|  |

Class Templates

**Class Templates**

1. **Objectives**

**After you complete this experiment you will be able to implement and know when it is appropriate to implement a class template.**

1. **Introduction**

Templates allow classes to use and operate on generic data types. They provide a mechanism to parameterize data types. Whenever you notice that two or more classes have identical code but operate on data of different types, you should consider templates if possible.

1. **Declaration Syntax**

The class definition and all its member functions must be prefaced with the following:

template <class Type\_Parameter>

The implementation for a class template is the same as the implementation for ordinary classes. However, you must remember to preface all member functions with the template heading, even those functions that do not use the generalized type specified in the type parameter.

Consider the following class definition:

template <class New\_Type>

class Array\_Class

{

public:

Array\_Class();

~Array\_Class();

void Add(New\_Type item);

int Search(New\_Type item);

void Print();

private:

New\_Type \*A;

int count;

};

Carefully observe the locations of the “New\_Type” parameter.

Now consider the following program that includes a main function and the definitions of the member functions for the class Array\_Class:

#include <iostream>

#include <string>

using namespace std;

const int SIZE=5;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Class declaraton for Array\_Class

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <class New\_Type>

class Array\_Class

{

public:

Array\_Class();

~Array\_Class();

void Add(New\_Type item);

int Search(New\_Type item);

void Print();

private:

New\_Type \*A;

int count;

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Class definitions for the member function of Array\_Class

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <class New\_Type>

Array\_Class<New\_Type>::Array\_Class()

{

cout<<"You are inside the default constructor\n";

cout<<"New\_Type has a size of "<<sizeof(New\_Type)<<" bytes\n\n";

count=0;

A = new New\_Type[SIZE];

}

template <class New\_Type>

Array\_Class<New\_Type>::~Array\_Class()

{

cout<<"The Destructor has been called\n\n";

delete [] A;

count = 0;

A = 0;

}

template <class New\_Type>

void Array\_Class<New\_Type>::Add(New\_Type item)

{

if (count<SIZE)

{

A[count++] = item;

}

else

{

cout<<"Array is Full\n";

}

}

template <class New\_Type>

int Array\_Class<New\_Type>::Search(New\_Type item)

{

int i;

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

{

if (item == A[i])

{

return i;

}

}

return -1;

}

template <class New\_Type>

void Array\_Class<New\_Type>::Print()

{

int i;

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

{

cout<<"A[i"<<i<<"] = "<<A[i]<<endl;

}

}

int main()

{

return 0;

}

Notice that every member function is prefaced with the template heading, even functions that do not use the type parameter.

More information on templates can be found in your course textbook and on the web.

1. **Experiments**

**Step 1: In this experiment you will investigate a template array class. Enter, save, compile and execute the following program in MSVS. Call the new project “ClassTemplatesExp” and**

**Questionthe program “ClassTemplates.cpp”. Answer the questions below:**

Create a source file which contains the program defined in the previous section (Declaration Syntax). Call the source file “template\_tester.cpp”. Add code to the main function for the following:

1. declaring an Array\_Class object of strings called “my\_String”;
2. declaring an Array\_Class object of integers called “my\_Ints”;
3. declaring an Array\_Class object of characters called “my\_Chars”;
4. adding the strings “Hello”, “GoodBye”, “ComeHere”, “SayNo” and “SayYes” to my\_String;
5. adding the integers 1,2,3,4 and 5 to my\_Ints;
6. adding the characters ‘a’, ‘b’, ‘c’ and ‘d’ to my\_Chars;
7. printing the Array\_Class objects my\_String, my\_Ints and my\_Chars;
8. searching my\_String for the string “SayYes” and “No”.

**Execute the program and explain its output by answering the following questions:**

1. Why do the output statements which contain “New\_Type” have different sizes printed?
2. How many times was the destructor called in the program?
3. What happened when parts (d), (e) and (f) were performed by the program?
4. How many items were printed when part (g) was performed by the program?
5. What happened when part (h) was performed by the program?
6. Why was the type parameter included in the class name but not in the function name? Consider the following: void Array\_Class<New\_Type>::Print( );
7. Why does the Search function return an integer type and not a New\_Type type?
8. Are there any types that cannot be parameterized by New\_Type? Explain.
9. What happens if you add code to the main function to add ‘e’, ‘f’ and ‘g’ to the “my\_Chars” array? Explain.
10. Please implement the following functions for Array\_Class.
11. A Boolean function called “Is\_Empty” that returns true if the array A is empty; otherwise it returns false.
12. A Boolean function called “Is\_Full” that returns true if the array A is full; otherwise it returns false.
13. A void function called “Remove” that deletes an item from the array A. Remove has one formal parameter of type New\_Type which holds a copy of the item to be deleted.
14. Add code to the main function to test these functions.