National University of Computer and Emerging Sciences

Lab Manual # 14

Course Instructor	Mr. Usama Hassan
Lab Instructor(s)	Ms. Fariha Maqbool Mr. Sohaib Ahmad
Section	BSE-2C
Semester	Spring 2023

Department of Computer Science

FAST-NU, Lahore, Pakistan

Objectives:

Topics Covered in this Lab:

- ✓ Templates
- ✓ Exception handling

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 variable type or class without repeating the code for each type. This is achieved through template parameters. A template parameter is a special kind of parameter that can be used to pass a type as parameter. These function templates can use these parameters as if they were regular types. The format for declaring function templates with type parameters is:

template <class identifier> function_declaration;

While defining a function template the body of the function definition is preceded by a statement **template <class identifier>.** The identifier can then be used as the data type of the parameters, the return type of the function, the data type of local variables and/or the data types of parameters.

Exercise 0:

Consider the definition of the following function template:

```
template <class type>
type surprise(type x, type y)
{
        return x + y;
}
```

Understand the code. What is the output of the following statements? Make a .cpp file to execute this code and add the output as comments in this file.

1. cout << surprise(5, 7) << endl;
2. string str1 = "Sunny";
   string str2 = " Day";
   cout << surprise(str1, str2) << endl;

Exercise 1:

Help: http://www.cplusplus.com/doc/oldtutorial/templates/

Write Template function for performing arithmetic operation of type int, float, double, long. Main for this function is given below.

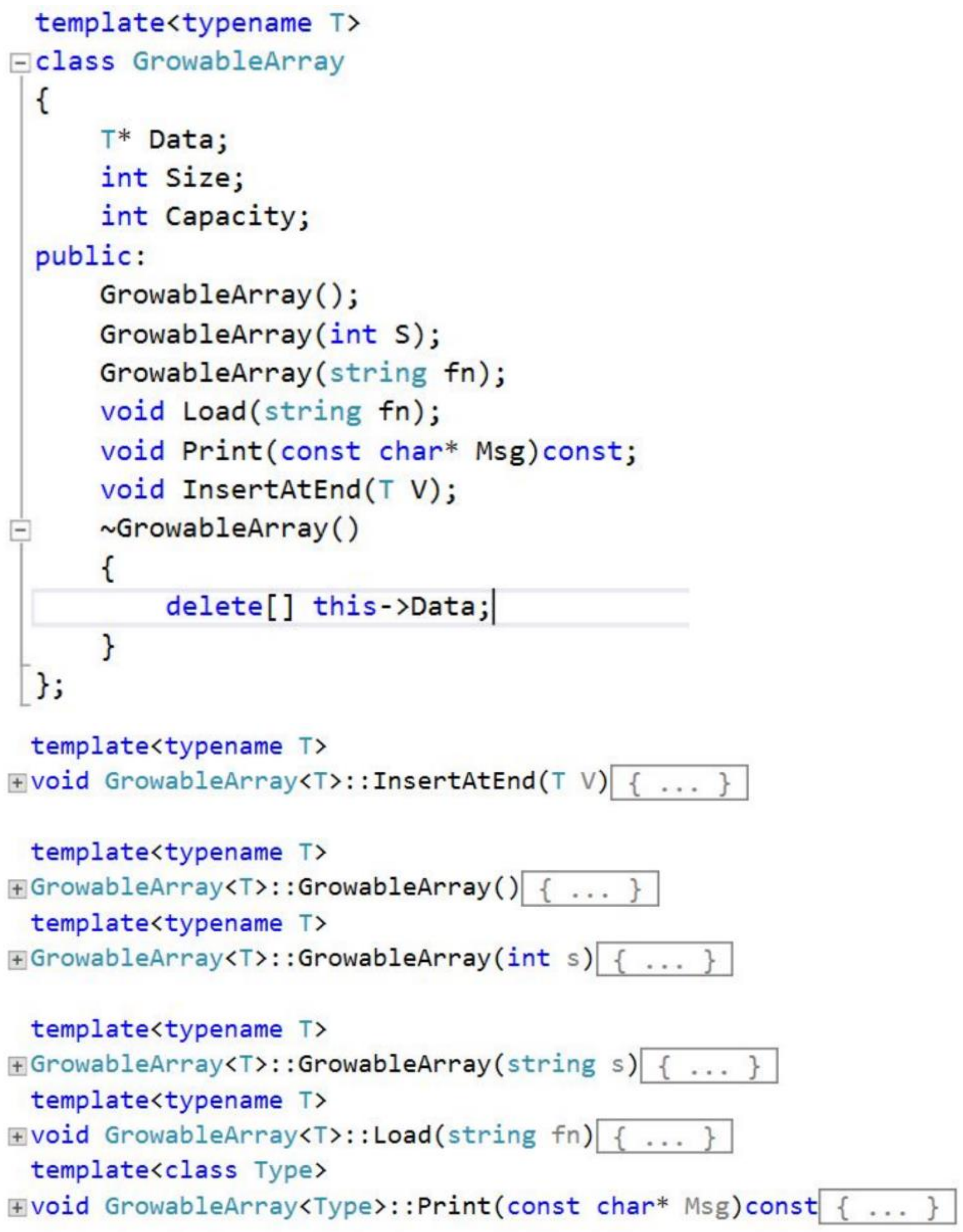
```
void main()
{
        int a, b; // this can be float, int or double too
        char op;
        cout << "Enter first operand ";
        cin >> a;
        cout << "Enter second operand ";
        cin >> b;
        cout << "Enter operation ";
        cin >> op; // op can be +, -, * or /
        if (op == '*' || op == '+' || op == '-' || op == '/')
        {
                performOperation(a, b, op);
        }
        else
        {
                cout << "Wrong operation";
        }

}
```

Exercise 2:

Implement the Growable Array using two implementations (Please create two separate Projects).

1. Without taking into account Capacity at all.
2. With Using Capacity which will grow geometrically by twice and size will hold the current data size.
3. Add the operator [.] in both the classes so that one can loop through in the GrowableArray and vector.



```cpp
template<typename T>
class GrowableArray
{
    T* Data;
    int Size;
    int Capacity;
public:
    GrowableArray();
    GrowableArray(int S);
    GrowableArray(string fn);
    void Load(string fn);
    void Print(const char* Msg)const;
    void InsertAtEnd(T V);
    ~GrowableArray()
    {
        delete[] this->Data;
    }
};

template<typename T>
void GrowableArray<T>::InsertAtEnd(T V) { ... }

template<typename T>
GrowableArray<T>::GrowableArray() { ... }
template<typename T>
GrowableArray<T>::GrowableArray(int s) { ... }

template<typename T>
GrowableArray<T>::GrowableArray(string s) { ... }
template<typename T>
void GrowableArray<T>::Load(string fn) { ... }
template<class Type>
void GrowableArray<Type>::Print(const char* Msg)const { ... }
```

Exercise 3:

Create a C++ class named "TemperatureConverter" with two private member variables: a double representing the temperature in Celsius, and a double representing the temperature in Fahrenheit. The class should have a constructor that takes a single double argument representing the temperature in Celsius, and it should also have two member functions: "getCelsius()" and "getFahrenheit()", which return the temperature in Celsius and Fahrenheit, respectively. The "getCelsius()" function should always return the original value passed to the constructor, while the "getFahrenheit()" function should calculate and return the Fahrenheit equivalent of the temperature in Celsius. If the temperature in Celsius is less than absolute zero (-273.15°C), the constructor should throw a custom exception of type "TemperatureTooLow".

Here is the basic prototype for the "TemperatureConverter" class and the "TemperatureTooLow" exception class:

```
class TemperatureTooLow {
public:
   const char* what() const {
      return "Temperature is below absolute zero (-273.15°C).";
   }
};

class TemperatureConverter {
private:
   double celsius;
   double fahrenheit;
public:
   TemperatureConverter(double c);
   double getCelsius();
   double getFahrenheit();
};
```

In the main function, create a try-catch block that catches the "TemperatureTooLow" exception and outputs the error message using the "what()" function. Within the try block, prompt the user to enter a temperature in Celsius, create a TemperatureConverter instance using that value, and output both the temperature in Celsius and the temperature in Fahrenheit using the "getCelsius()" and "getFahrenheit()" functions.

Here is some sample code for the main function:

```
int main() {
   double c;
   std::cout << "Enter a temperature in Celsius:\n";
   std::cin >> c;
   try {
      TemperatureConverter tc(c);
      std::cout << "The temperature in Celsius is " << tc.getCelsius() << "°C.\n";
      std::cout << "The temperature in Fahrenheit is " << tc.getFahrenheit() << "°F.\n";
   } catch (TemperatureTooLow& e) {
      std::cout << e.what() << "\n";
   }
   return 0;
}
```