CSc10300 - Assignment 5

Inheritance/Exceptions/Templates/Containers - Fall 2021

Motahare Mounesan

* YOU NEED TO UNIT TEST YOUR CODES.

1

Define a class named **CheckedArray**. The objects of this class are like regular arrays but have range checking. If **a** is an object of the class **CheckedArray** and **i** is an illegal index, then use of $a[i]$ will cause your program to throw an exception. You need to implement the code with 3 different types of exceptions and test them separately. To test each of them you can simply comment the other two.

1. Throw an exception(an object) of class **ArrayOutOfRangeError**. Defining a class named ArrayOutOfRangeError inside CheckedArray class is part of this assignment. Also, your CheckedArray class must have a suitable overloading of the [ ] operator.

2. Throw an exception (an object) of one of the subclasses of the C++'s exception class we used in the lecture. Choose the most relevant exception subclass.

3. Throw an exception of ArrayOutOfRange class as described in what follows:
   Considering the definition of the C++'s exception class:

```
class exception {
    public:
        exception () noexcept;
        exception (const exception&) noexcept;
        exception& operator= (const exception&) noexcept;
        virtual ~exception();
        virtual const char* what() const noexcept;
}
```

   The logic_error class:

```
class logic_error : public exception {
    public:
        explicit logic_error (const string& what_arg);
        explicit logic_error (const char* what_arg);
};
```

And the out_of_range class:

```
class out_of_range : public logic_error {
    public:
        explicit out_of_range (const string& what_arg);
        explicit out_of_range (const char* what_arg);
};
```

Derive a class ArrayOutOfRange from out_of_range class:

- add two data members to keep the size of the array and the attempted out of range index. These data members should be accessible from sub-classes of this class but neither by objects of this class nor by objects of its sub-classes.
- write a constructor that initializes the two data members you defined in the previous step and the error message. Using the following code, you can set the value of message. You just need to complete it to set other data members.

```
explicit ArrayOutOfRange (const string& errorMsg, *rest of the arguments* ):
out_of_range(errorMsg)
{
//*body of constructor*
}
```

- override what() function to return the error message, the attempted index and size of the array in the following format

```
"Error Message (attempted index,size of array)"
```

The return type of *what* function is **char***(cstring), you can convert your string to a csting using c_str() function. Also, you can access the error message you passed to the constructor by out_of_range::what().

2

Write a class template that uses a vector to implement a stack data structure with these member functions:

1. (constructor) : Construct stack (public member function)

2. empty : Test whether container is empty (public member function)
3. size : Return size (public member function)
4. top : Access next element (public member function)
5. push : Insert element (public member function)
6. pop : Remove top element (public member function)

Put function definitions outside the definition of the class (i.e. inline functions are not allowed).

3

We want to store the information of different computers.

(a) Create a class named Computer with five data member named CPU type, clock speed, RAM, storage and price.

(b) Create its two subclasses:

- Desktop with data members to store tower size, warranty (by years), and PSU wattage.
- Laptop with data members to store screen size, weight, and battery life.

(c) Make another two subclasses Dell and HP for Desktop, each having a data member to store the model type.

(d) Next, make two subclasses Apple and Lenovo for Laptop, each having a data member to store the make-type.

(e) Add a print function to the base class (Computer) and override it in all the subclass. The print function of each class should print a list of all its data members, each on a separate line.

(f) Now, store all the information of a Dell and a HP computer , and print them using print function.

(g) Do the same for a Apple and a Lenovo laptop.

(h) Draw the UML of this program.

NOTE: Data members of all of the classes should be private.

Overloading the Array index Square Brackets

You can overload the square brackets, [ ], for a class so that they can be used with objects of the class. If you want to use [ ] in an expression on the left-hand side of an assignment operator, then the operator must be defined to return a reference, which is indicated by adding to the returned type. (This has some similarity to what we discussed for overloading the I/O operators << and >>.) When overloading [ ], the operator [ ] must be a member function; the overloaded [ ] cannot be a friend operator.
For example, the following defines a class called Pair whose objects behave like arrays of characters with the two indexes 1 and 2 (not 0 and 1):

```
class Pair {
    public:
        Pair();
        Pair(char first_value, char second_value);
        char& operator[](int index);
    private:
        char first;
        char second;
};
```

The definition of the member function[] can be as follows:

```
char& Pair::operator[](int index) {
    if (index == 1)
        return first;
    else if (index == 2)
        return second;
    else
    {
        cout << "Illegal index value.\n";
        exit(1);
    }
}
```

Objects are declared and used as follows:

```
    Pair a;
    a[1] = 'A', a[2] = 'B';
    cout << a[1] << a[2] << endl;
```

Note that in a[1], a is the calling object and 1 is the argument to the member function [].

Submission:

1. Unit-test your programs.
2. For questions 1 and 2, you need a separate file for the class you are defining. You can upload a .hpp file or a pair of .h and .cpp files for the class.
3. You can put all the code for question 3 in a single main.cpp file.