UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS110 - Program Design: Introduction

Practical 10 Specifications: Exceptions

Release date: 06-11-2023 at 06:00
Due date: 10-11-2024 at 23:59
Total Marks: 111

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually, no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- The following imports are allowed ("<iostream>", "<string>", "<sstream>" "ArrayWrap.h", "ArrayWrap.cpp", "ListException.h", "IndexException.h", "ConstructorException.h", "SizeException.h").
- Be ready to upload your assignment well before the deadline, as **no extension will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or classes).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offense. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at http://www.ais.up.ac.za/plagiarism/index.htm.
- Unless otherwise stated, the usage of c++11 or additional libraries outside of those indicated in the assignment, will not be allowed. Some of the appropriate files that you have submitted will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**

- We will be overriding your .h files, therefore do not add anything extra to them that is not in the spec
- If you would like to use *using namespace std*, please ensure it appears in your cpp that you upload

2 Plagiarism

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding.** Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

The goal of this practical is to gain experience with working with exceptions.

4 Introduction

Implement the UML diagram and functions as described on the following pages.

5 Class Diagram

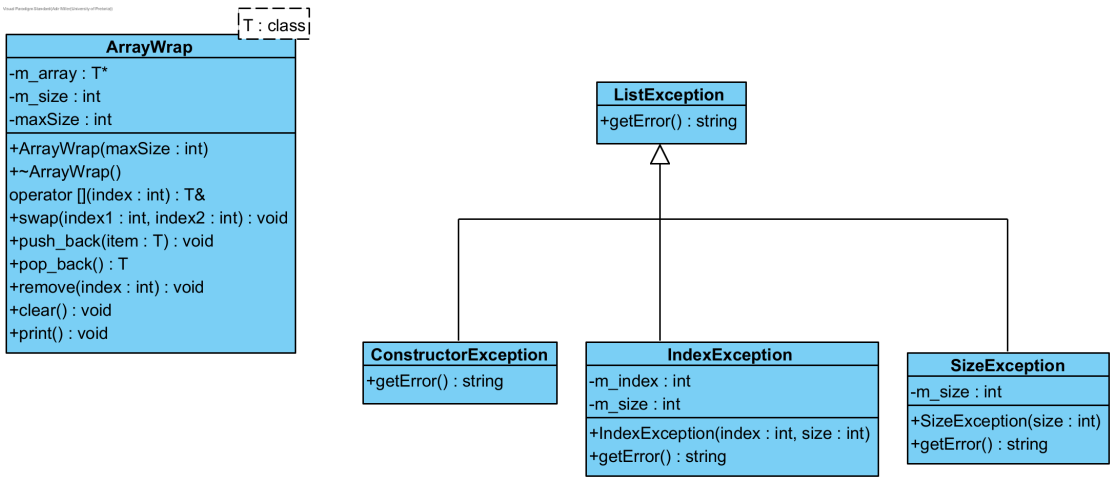

Figure 1: Class diagrams

6 Classess

6.1 ListException

- Functions
  - getError(): virtual std::string
    - Returns the string "Something went wrong!".

6.2 IndexException: public ListException

- Members
  - m_index: int
    - The invalid index.
  - m_size: int
    - The size of the list in question.
  - IndexException(index: int,size: int)
    - Constructor for the IndexException class
    - This sets the m_index and m_size variables
  - getError(): string
    - If m_index is negative, return "Index <**index**> is negative.". Replace <**index**> with the value of **m_index** .
    - Otherwise return "Index <**index**> is out of bounds for list of size <**size**>.". Replace <**index**> with the value of **m_index** and replace <**size**> with the value of and **m_size**.

6.3 ConstructorException: public ListException

- Functions
  - getError(): string
    - Returns "Invalid parameter.".

6.4 SizeException: public ListException

- Members
  - m_size: int
    - The size of the list in question.
- Functions
  - SizeException(size: int)
    - Constructor for the SizeException class.
    - Sets the m_size member variable.
  - getError(): string
    - If size is 0, return "List is empty.".
    - If size is not 0, return "List is full with <**size**> elements." Replace <**size**> with the value of **m_size**.
    - You may assume that m_size is never negative.

6.5 ArrayWrap<T>

- Members
  - m_array: T*
    - The internal array that the class wraps.
  - m_size: int
    - The current size of the array, this does not represent the actual physical size as the array is not resized. Rather it is a logical wall indicating how full the array is.
  - maxSize: int
    - The max number of elements that can be in the array, this also represents the actual size of *m_array*.
- Functions
  - ArrayWrap(size: int)
    - Sets the *maxSize* variable.
    - If size is negative or 0, throw a *ConstructorException*.
    - If size is valid, create a new T array of size *size* and saved in *m_array*.
    - Sets m_size to 0.
  - ~ArrayWrap()
    - If *m_array* is not NULL, delete the array.
  - operator[](index: int): T&
    - Indices start at 0.
    - This is the overloaded subscript operator.
    - This function will retrieve the element located at the passed in index.
    - If the index is smaller than 0, then throw an *IndexException* and pass in the value of the parameter that caused the exception to be thrown.
    - If the index is larger than or equal to the current size, then throw an *IndexException* and pass in the value of the parameter that caused the exception to be thrown.
    - You should only be able to access elements with an index smaller than *m_size*.
  - swap(index1: int, index2: int): void
    - If either of the indexes are larger than or equal to the current size, throw an *IndexException* and pass in the value of the parameter that caused the exception to be thrown.
    - If either of the indexes are smaller than 0, throw an *IndexException* and pass in the value of the parameter that caused the exception to be thrown.
    - If both indexes are valid, swap the data at the given indices.
    - If index1 is equal to index2 and they are both valid, do nothing and do not throw an exception.
    - If both indices are invalid, use the details of the first index for the exception.
  - push_back(item: T): void
    - This function appends an item to the back of the array.
    - For example: if the array has a maxSize of 10 and is currently [1,2,4], then a *push_back(5)* would result in an array of [1,2,4,5]
    - If the array is full, i.e if m_size is equal to *maxSize*, throw a *SizeException* and pass in the value of the parameter that caused the exception to be thrown.

- pop_back(): T
  - Returns the element at the end of the array.
  - Decrements *m_size*, however, you cannot actually delete or clear the element at the index, meaning it remains at that index and will be overwritten on the next push.
  - If the list is empty, throw a *SizeException* and pass in the value of the parameter that caused the exception to be thrown.
- remove(index: int): void
  - This function should remove the item at the specified index.
  - The elements of the array should then shift to cover the open space.
  - For example: if the array is [5,45,2,5] and we call *remove(1)*, then the array should look as follows: [5,2,5]
  - If the index is invalid (not in the range of 0 to m_size - 1), throw an *IndexException* and pass in the value of the parameter that caused the exception to be thrown.
- clear(): void
  - Set *m_size* to 0.
  - This function does not throw any exceptions.
- print(): void
  - This function prints each element in the list to the console, comma-separated, with no added spaces with a newline at the end.
  - Remember to only print up till m_size.
  - If the list is empty, only print a newline.

7 Memory Management

Memory management is a core part of COS110 and C++, so this practical memory management is extremely important due to its scope. Therefore each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocates *all* the memory that was allocated.

8 Testing

Coverage ratio range	% of testing mark	Final mark
0%-5%	0%	0
5%-20%	20%	6
20%-40%	40%	12
40%-60%	60%	18
60%-80%	80%	24
80%-100%	100%	30

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark. Remember that your main will be testing the Instructor Provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

9 Upload checklist

The following c++ files should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- main.cpp
- ListException.cpp
- IndexException.cpp
- SizeException.cpp
- ConstructorException.cpp
- ArrayWrap.cpp

You will notice you do not need to upload a makefile or any h files, you may if you wish, however, FitchFork will overwrite them before running your submission.

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

10 Submission

You need to submit your source files, only the cpp files (**including the main.cpp**), on the FitchFork website (https://ff.cs.up.ac.za/). All methods must be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. There is no need to include any other files or .h files in your submission. **Ensure your files are in the root directory of the zip file**. Your code should be able to be compiled with the C++98 standard

For this practical, you will have 10 upload opportunities. Upload your archive to the Practical 10 slot on the FitchFork website.