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 1 Specifications

Release Date: 29-07-2024 at 06:00

Due Date: 02-08-2024 at 23:59

Total Marks: 75

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.**
- 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 offence. 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 `https://portal.cs.up.ac.za/files/departmental-guide/`.
- 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 submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use c++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.
- Note, UML notation for variable and function signatures are used in this assignment.
- Please note that there is a late deadline which is 1 hour after the initial deadline, but you will lose 20% of your mark.

2 Overview

Dynamic arrays are arrays that automatically grow and shrink when elements are added or removed from it. Thus, the minimum size needed to store the array is always used, and the array does not contain any NULL values.
A 2D array is a multidimensional array, which is an array of rows, where each row is also an array. C++ allows for jagged arrays, which means that the size of the rows do not need to be the same. Therefore, it is possible to create non-rectangular arrays.

3 Your Task:

You are required to implement all of the functions listed in `DynamicArrays.h`, in the `DynamicArrays.cpp` file. The functions are explained in the next subsection.

In addition to implementing the `DynamicArrays.cpp` file, you are also tasked with familiarising yourself with writing testing code. You are given a `main.cpp` and `howToTest.pdf` file which will guide you in how to create testing code. For all future practicals and assignments, you will receive marks for your testing. This practical aims to show you an example of how testing is done, so that you can use it as a guide for future assessments.

3.1 DynamicArrays.cpp

This file contains functions which can be called on a dynamic 2D jagged array. All of the functions take in at least three parameters, all of which are passed in by reference. You may assume that these parameters will always be valid and follows the following specifications. These are:

- numRows : int&
  - This is the number of rows in the array.
- numColumns : int*&
  - This is a dynamic 1D array of size *numRows*.
  - The elements in this array show how many columns each row has.
- array : int**&
  - The dynamic 2D array.
  - It has *numRows* rows.
  - The number of columns for each row is stored in *numColumns*.

3.1.1 Functions

- createArray(array: int**&, numColumns: int*&, numRows: int& , input: std::string): void
  - If no previous arrays have been created, the passed-in parameters will be NULL. If a previously created array was already stored there, the parameters will be non NULL. If the passed-in parameters are not NULL, you may assume they follow the specifications stated at the start of this section.
  - The passed-in string should be used to populate the variables.
  - Rows are split using a bar, |.
  - If no bars are present, then it means only one row in the array.
  - The values in each row are split using commas. You may assume that the values between commas will always be valid integers.
  - A row can also be an empty string. In this case, that row should be an array that has a size of 0.
  - You may assume that the input string will always follow these specifications, and you do not need to check whether the input is valid.
- destroyArray(array: int**&, numColumns: int*&, numRows: int&): void
  - This will deallocate all of the passed in variables.
  - After this function is called, *array* and *numColumns* should be NULL, and *numRows* should be 0.
- printArrayStructure(array: int**&, numColumns: int*&, numRows: int&): std::string
  - This function returns a string representing the structure of the array.
  - The string starts with the word "numRows", followed by a space, then a colon, and then another space.
  - The number of rows is then appended to this string, with a newline after this.
  - For every row, the following is then appended:
    - The word "row", followed by the row number that starts from 0 inside square brackets. After the second square bracket, there is a space, followed by a colon, followed by a space.
    - After this, the number of columns for that row is appended.
    - For all rows except the last row, a new line is added to the end.

- printArray(array: int**&, numColumns: int*&, numRows: int&): std::string
  - This function returns a string representing the array.
  - Each row should be printed on its own line. Each line should end with a new line, except the last row.
  - The elements in each row should be separated with a comma.
  - If the row has 0 columns, then an empty string should be used as the row.
- addRow(array: int**&, numColumns: int*&, numRows: int&): void
  - This function should add a row to the end of the array.
  - The new row should have 0 columns.
- addValueToRow(array: int**&, numColumns: int*&, numRows: int&, rowNumber: int, value: int): void
  - This function will attempt to add the passed-in value, to the end of the row indicated by the passed-in *rowNumber*.
  - If the *rowNumber* is invalid, then this function does nothing.
- removeRow(array: int**&, numColumns: int*&, numRows: int&, rowNumber: int): void
  - This function will attempt to remove a row from the array, indicated by the passed-in *rowNumber*.
  - If the *rowNumber* is invalid, then this function does nothing.
- removeValueFromRow(array: int**&, numColumns: int*&, numRows: int&, rowNumber: int, value: int): void
  - This function will attempt to remove values from a row in the array.
  - All occurrences of the passed-in value should be removed from the row indicated by *rowNumber*.
  - If the *rowNumber* is invalid, then this function does nothing.
- rowSum(array: int**&, numColumns: int*&, numRows: int&, rowNumber: int); int
  - This function returns the sum of the passed-in *rowNumber*.
  - If the *rowNumber* is invalid, then return -1.
- rowAvg(array: int**&, numColumns: int*&, numRows: int&, rowNumber: int): float
  - This function returns the average of the passed-in *rowNumber*.
  - If the *rowNumber* is invalid or if the number of columns for that row is 0, then return -1.

- rowMax(array: int**&, numColumns: int*&, numRows: int&, rowNumber: int): int
  - This function returns the maximum element of the passed-in *rowNumber*.
  - If the *rowNumber* is invalid or if the number of columns for that row is 0, then return -1.
- rowMin(array: int**&, numColumns: int*&, numRows: int&, rowNumber: int): int
  - This function returns the minimum element of the passed-in *rowNumber*.
  - If the *rowNumber* is invalid or if the number of columns for that row is 0, then return -1.

4 Memory Management

As memory management is a core part of COS110 and C++, 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.

5 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **c++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- Do not include using `namespace std` in any of the files.
- You may only use the following libraries:
  - sstream
  - string
  - iostream
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

6 Upload Checklist

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

- `DynamicArray.cpp`
- `main.cpp`
- Any textfiles used by your `main.cpp`
- `testingFramework.h` and `testingFramework.cpp` if you used these files.

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.

7 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ *.cpp -o main
```

and run with the following command:

```
./main
```

Remember your `h` file will be overwritten, so ensure you do not alter the provided `h` files.
You have 10 submissions and your best mark will be your final mark. Upload your archive to the Practical 1 slot on the FitchFork website. Submit your work before the deadline. **No late submissions will be accepted!**

8 Accessibility

No figures used in this practical.