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

COS132 - Imperative Programming

Practical 8 Specifications

Release Date: 22-04-2025 at 06:00

Due Date: 02-05-2025 at 23:59

Total Marks: 76

Read the entire specification before starting with the practical.

Contents

1 General Instructions

- *Read the entire assignment thoroughly before you begin coding.*
- This assignment should be completed individually.
- **Every submission will be inspected with the help of dedicated plagiarism detection software.**
- 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 structure).
- 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 to generate submitted code is strictly forbidden and will be considered as plagiarism.
- For the functions in the `Recursive namespace` you may **ONLY** use recursion. This means no `for`, `while` or `do while` loops.

2 Overview

In this practical, you will practise interacting with pointers in the form of `char pointers`. You will create two versions of a series of functions, one using iterative loops and the other using recursive loops. Note, you will be submitting two separate files to two separate upload submissions. Ensure you submit the correct files to the correct submission. The functions in this practical are used to manipulate, calculate and extract data from the `string`, which has been converted to a `char*`. You will also interact with functions where the parameter and the return type are `const`.

3 Background

Before recess, you were introduced to iterative and recursive loops. You were also introduced to pointers, references and interacted with these using `strings` and `char pointers`. In class, it was discussed that `strings` can be converted to `char pointers` as illustrated below:

Task	Iterative	Recursive
1_LenFindContains	10	10
2_PrintReversePrint	8	8
3_OccFreqBetween	9	6
4_FindLastPalindrome	9	6
Testing	5	5

Table 1: Mark Distribution

```
string str = "star␣wars";
char* c = &str[0];
```

Hint: revise the study material to see which character indicates the end of a string.

4 Your Task:

You will be implementing functions in two namespaces: `Iterative` and `Recursive`. All the functions in the `Recursive` namespace are also found in the `Iterative` namespace. The `Iterative` namespace has some additional functions that you need to implement. For the functions in the `Recursive` namespace you may **ONLY** use recursion. This means no `for`, `while` or `do while` loops. For the `Iterative` namespace you may use `for`, `while` or `do while` loops.

4.1 Shared Functions

The functions listed in this section are the functions shared by both `Iterative` and `Recursive` namespaces. In both namespaces, these functions perform the exact same task, it is just the repetition structure used that differs. In all of the functions, a `string` was passed-in as a `const char *`. As such, if the function description states that a `string` was passed-in, it is referring to the `char*` in which the `string` was converted to.

4.1.1 Functions

- `length`
  - This function needs to calculate and return the length of the `string`.
  - For example: the length of “Yoda” would be 4, and the length of “” would be 0.
- `find`
  - This function needs to find the character that is found at the passed-in `index` of the passed-in `string`.
  - If the `index` is invalid (negative or outside the bounds of the `string`, the function should return `'\0'`.
  - ***Hint for recursion**: The idea is that as you move through the string, the index which you are trying to find will get closer (smaller).*

- `contains`
  - This function needs to determine if the passed-in `letter` is in the passed-in `string`.
  - If it is, the function should return `true`, else the function should return `false`.
- `printout`
  - This function needs to print each character of the passed-in `string` out, and at the end it should print a `newline`.
  - ***Hint for recursion:*** *Use a recursive helper function that will print out all the characters, and then at the end, the* `printout` *function just adds the* `newline`*.*
- `reverseWord`
  - This function needs to reverse the passed-in `string` and store the result in the passed-in `result`.
  - *Hint: remember you can change the value stored at a pointer.*
  - ***Hint for recursion:*** *Using a helper function that uses a reference which you change (increment) as the recursive loop "unwinds" itself may be useful..*
- `countOccurrences`
  - This function needs to count the number of occurrences of the passed-in parameter, `c`, in the passed-in `string`.
  - If `c` never occurred in the passed-in `string`, the function should return 0.
- `printInReverse`
  - This function needs to print the passed-in `string` in reverse with a `newline` added at the end.
  - ***Hint for recursion:*** *This function combines ideas from* `printout` *and* `reverseWord`*..*
- `findLastLetter`
  - Two overloaded versions of this function are provided, one with constness and one without.
  - Both utilise the same algorithm.
  - This function needs to return the last letter in the passed-in `string`.
- `charactersBetween`
  - This function needs to determine how many characters there are between the two `character pointers` belonging to the same string. The first character is excluded and the last letter is included.
  - Note, due to easily being able to go into an infinite loop, the function should stop once the distance between two `character pointers` is more than 100.

- For example:

```
string s1 = "tomcat";
string s2 = "warthog";
cout << charactersBetween(&s1[0], &s1[4]) << endl;
cout << charactersBetween(&s1[0], &s2[0]) << endl;
```

  The output should be:

```
4
100
```

- ***Hint for recursion:*** *The suggested strategy is to use a recursive helper function that takes in a reference which stores the distance between the two `character pointers`. Move the one pointer along until you either reach 100 or the other pointer. Then store the result in the passed-in reference.*

4.2 `Iterative` Exclusive Functions

These functions discussed in this section are exclusive to the `Iterative` namespace.

- `mostFrequentLetter`
  - This function needs to return the most frequently occurring letter in the passed-in `string`.
  - If there are multiple letters occurring with the same max frequency, return the first occurring letter.
  - Example:
    - The function call: `mostFrequentLetter(‘‘Mississippi’’)` should return `‘i’`.
    - The function call: `mostFrequentLetter(‘‘one’’)` should return `‘o’`.
- `isPalindrome`
  - This function needs to determine if the passed-in `string` is a palindrome or not.
  - A palindrome is a word that is spelt the same in reverse as it is spelt normally.
  - Examples of palindromes:
    - “a”
    - “madam”
    - “deed”
  - If the passed-in `string` is a palindrome, the function should return `true`, else `false`.

5 Warning about `Recursive namespace`

You will submit the `Recursive` file to a different submission from the `Iterative` file. This is due to the `Recursive` file being checked for any iterative loops like `for`, `while`, and `do while`. To avoid false flags, ensure that no variables, comments or helper function names contain the phrases:

- **for**
- **while**
- **do while**

If **iterative loops** are detected in your implementation of the functions in the `Recursive` namespace, you **will be** awarded a mark of **0**.

6 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [1] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using Table 2:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 2: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing 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.

[1] For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

7 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:
  - <iostream>

8 Upload Checklist

For each of the sections below, the C++ files listed should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number. 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. Ensure you **do not** include the `Iterative.cpp` in the files for the **Recursive** task

8.1 `Iterative`

- `Iterative.cpp`
- `main.cpp`
- Any textfiles used by your `main.cpp`

8.2 `Recursive`

- `Recursive.cpp`
- `main.cpp`
- Any textfiles used by your `main.cpp`

9 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++ -std=C++98 -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 for each submission slot, and your final submission's mark will be your final mark. Upload the correct archive to the relevant Practical 8 submission slot on the FitchFork website. That means upload the a Submit your work before the deadline. **No late submissions will be accepted!**