Practical 8

COS132

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

Department of Computer Science

Official Deadline: 24/05/2024 at 17:30

Extended Deadline: 26/05/2024 at 23:59

Marks: 100

1 General instructions:

- This assignment should be completed individually; no group effort is allowed.
- Be ready to upload your practical well before the deadline, as no extension will be granted.
- **The extended deadline has been put in place in case of loadshedding or other unforeseen circumstances. No further extension will be granted.**
- **Note that no tutor or lecturer will be available after the official deadline**
- You may not import any libraries that have not been imported for you, except `<iostream>`.
- You may `use namespace std;`
- If your code does not compile, you will be awarded a zero mark. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire practical before you start coding.
- You will be afforded 20 upload opportunities per task.

- **You should no longer be using the online compiler. You should compile, run and test your code locally on your machine using terminal commands or makefiles**.
- **You have to use C++98 in order to get marks**

2 Plagiarism

The Department of Computer Science considers plagiarism 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 permission) and copying material 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. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.**

3 Outcomes

Upon successful completion of this practical, students will be able to:

- Manipulate and iterate through strings using pointers.
- Declare and initialize arrays.
- Use pointers or indexes to access and modify array elements.
- Implement functions that utilise pointers and references to manipulate data.
- Understand and use double pointers for pointer manipulation.
- Utilise pointers for dynamic memory allocation and deallocation.
- Develop logical strategies for solving complex problems using arrays and pointers.

4 Structure

This practical consists of two tasks. Each task is self-contained and all of the code you will require to complete it will be provided in the appropriate Task folder. Each task will require you to submit a separate archive to an individual upload slot. That is, each separate task will require its own answer archive upload. You will upload Task 1 to Practical 8 Task 1 and so on. You can assume that all input will be valid (i.e. of the correct data type and within range)

5 Mark Distribution

Activity	Mark
Task 1 - Basic Pointers and References	20
Task 2 - String manipulation and 1D arrays	80
Total	**100**

Table 1: Mark Distribution

6 Resources

Here are some additional resources you can use to help you complete the practical

- **1D Arrays**: A 1D array is a collection of elements of the same data type, stored in contiguous memory locations. The elements are accessed using indices. The syntax for declaration and initialization is

  ```
  int arr[5] = {1, 2, 3, 4, 5};
  ```

  Here, `arr` is a 1D array of integers with 5 elements. For further details and examples, visit: `https://www.geeksforgeeks.org/cpp-arrays/`.

- **Pointers**: A pointer is a variable that stores the memory address of another variable. Pointers are used for dynamic memory allocation, arrays, and function arguments. The syntax for declaration is

  ```
  int* ptr;
  ```

  Here, `ptr` is a pointer to an integer.

  To initialize a pointer with the address of a variable, use the following syntax:

  ```
  int var = 10;
  int* ptr = &var;
  ```

  Here, `ptr` stores the address of `var`.

  To create a variable on the heap with a pointer to the value, use the following syntax:

  ```
  int* ptr = new int(10);
  ```

  Here, `ptr` stores the address of an integer on the heap, with value 10. For further details and examples, visit: `https://www.geeksforgeeks.org/cpp-pointers/`.

- For information on how to iterate though a string character by character, see the study unit notes on memory management.

PLEASE NOTE: You should write a main that tests all the functions that you have implemented with different inputs.

7 Task 1

Your task is to implement the functions in the task1.h and task1.cpp files.

7.0.1 Task1 Header

The `task1.h` header file includes the declarations for functions that sorts and swaps numbers using pointers.

```
#ifndef TASK_1_H
#define TASK_1_H
#include <iostream>

void sortNumbers(int &a, int &b, int &c);
void swap(int** ptrA, int** ptrB)

#endif
```

Create `task1.cpp` and include the `task1.h` header.

```
#include "task1.h"

// Function implementations will go here
```

You will implement the functions in `task1.cpp`

7.1 void sortNumbers(int &a, int &b, int &c)

This function sorts three integers and assigns the largest value to `a`, the smallest to `c`, and the middle value to `b`.

- **Parameters:**
  - **a:** Reference to the first integer.
  - **b:** Reference to the second integer.
  - **c:** Reference to the third integer.
- **Process:**
  1. Declare temporary variables `tempA`, `tempB`, and `tempC` to hold the initial values of `a`, `b`, and `c`.
  2. Assign the largest value to `a`, the smallest to `c`, and the middle value to `b`.
  3. Hint:
     - Compare `tempA` with `tempB`.
     - If `tempA` is greater than `tempB`, further compare `tempA` with `tempC`.
     - Otherwise, compare `tempB` with `tempC`.

7.2 void swap(int** ptrA, int** ptrB)

This function swaps two integer pointers using double pointers. The integer pointers that `ptrA` and `ptrB` are pointing to should be swapped, not just the integers themselves.

- **Parameters:**
  - **ptrA:** Double pointer to the first integer.
  - **ptrB:** Double pointer to the second integer.
- **Process:**
  1. Declare a temporary pointer `temp` to hold the value of `ptrA`.
  2. Assign the value of `ptrB` to `ptrA`.
  3. Assign the value of `temp` to `ptrB`.

8 Task 2

After successfully decrypting the messages, you discovered a chilling revelation: CodeBrew would be at the G20 summit. Your heart pounded as you realized the implications.

You had received this anonymous tip from a shadowy figure warning of a bomb planted by CodeBrew at the summit. The figure had handed you the note with some random sentences and then, with a distorted voice, uttered, "R29vZCBMdWNrIFNvbGRpZXI=" and vanished. Taking a deep breath, you realized what had to be done. The stakes had never been higher. Your mission was clear: find the exact location of the bomb and defuse it.

8.1 Header and Source File Creation Guide

This guide outlines the structure and responsibilities of the header (‘.h‘) and source (‘.cpp‘) files for the monitoring system. Each header file contains the declarations of functions related to the environmental controls for temperature, pressure, and humidity.

8.1.1 Room Locater Header

The `RoomLocater.h` header file includes the declarations for function that will use substring analysis to find the room letter where the bomb is hidden.

```
#ifndef ROOM_LOCATER_H
#define ROOM_LOCATER_H
#include <iostream>

int substringSearch(const char* str, const char* substr);
void indexToChar(int index);

#endif // ROOM_LOCATER_H
```

Create `RoomLocater.cpp` and include the `RoomLocater.h` header.

```
#include "RoomLocater.h"

// Function implementations for finding the room will go here
```

8.1.2 Bomb Defusing Header

The `BombDefusing.h` header file contains the function declarations related to defusing the bomb.

```
#ifndef BOMB_DEFUSING_H
#define BOMB_DEFUSING_H

int calculateSize(int* arr);
void findPairs(int* arr, int size);

#endif
```

Create `BombDefusing.cpp` and include the `BombDefusing.h` header at the top:

```
#include "BombDefusing.h"

// Implement the functions to defuse the bomb
```

Each '.cpp' file should implement the functions declared in its respective '.h' file. Pay attention to ensure that each '.cpp' file includes its corresponding header file to provide function prototypes, enabling proper compilation.

8.2 Detailed Function Implementation Specifications for Room Locater

8.2.1 int substringSearch(const char* str, const char* substr)

This function searches for a substring within a string and returns the starting index of the substring, or -1 if the substring is not contained in the string.

- **Parameters:**
  - **str:** Pointer to the main string.
  - **substr:** Pointer to the substring to search for.
- **Process:**
  1. Initialize pointers `ptr1` to `str` and `ptr2` to `substr`.
  2. Use a `while` loop to iterate through `str` until the end of the string (the terminating character) using ptr1. The terminating character is `\0`.
     (a) Inside the loop, initialize a variable `start` to `ptr1` and reset `ptr2` to `substr`.

(b) Use a nested `while` loop to compare characters from `str` and `substr`.

i. The inner loop should keep comparing while the substring has not reached its end (\0) and the characters at the current positions match

ii. In the loop, move `ptr1` and `ptr2` to the next character of `str` and `substr` respectively.

(c) If a match is found, (i.e. all the characters from the `substr` were matched in the inner loop) return the starting index.
Hint: `start` does not store the starting index, it stores the address of the character at the starting index. How can you get the index? You can use a smart trick with `start` and `str` (remember `str` stores the address of the start of the string), or you can create an extra int variable to keep track of the index.

(d) If no match is found, update `ptr1` to `start`+1 and continue the search.

3. If no match is found by the time the end of the string is reached, i.e. `substr` is not contained in `str`, return -1.

8.2.2 void indexToChar(int index)

This function takes an index and converts it into a character.

- **Parameters:**
  - **index:** The integer index to convert.
- **Process:**
  1. Convert the integer `index` to a character. (i.e. convert the integer to its ASCII character equivalent).
  2. Print out the index as the floor and the character as the room.
     - std::cout « "Floor: " « index « ", Room: " « room « std::endl;

8.3 Detailed Function Implementation Specifications for Bomb Defusing

8.3.1 int calculateSize(int* arr)

This function calculates the number of elements in an array until a sentinel value (-1) is encountered.

- **Parameters:**
  - **arr:** Pointer to the first element of the array.
- **Process:**
  1. Initialize a variable `size` to 0.
  2. Use a `while` loop to iterate through the array until the sentinel value (-1) is found.
  3. Increment `size` for each element in the array.

8.3.2 void findPairs(int* arr, int size)

This function finds pairs of matching wires and indicates which wires should be cut to defuse the bomb. If a wire doesn't have a matching wire, it should be cut. The wires are numbered, and you should always try to pair as soon as possible. For example, if the wires are 1, 2, 1, 1, the first '1' wire will be paired with the '1' immediately after the '2'. We keep track of paired wires using a boolean array called "paired".

The process follows a "STEP, COMPARE, NOT THE SAME" approach:

1. ***Start***: Begin at the first wire and mark it as TRUE in the "paired" array.
2. ***Step***: Move to the next wire
3. ***Compare***: Compare the current wire's number to the number you are looking for
   - ***Not the same***: Go back to Step.
   - ***Pair***: If you find a match, mark both wires as "paired" in the "paired" array and restart the process from the earliest non-paired wire.

- **Parameters:**
  - **arr:** Pointer to the first element of the array.
  - **size:** The number of elements in the array.
- **Process:**
  1. Declare an boolean array `paired` of the same size as `arr` to keep track of paired wires.

     ```
     bool* paired = new bool[size];
     ```

  2. Initialise all elements of `paired` to false, since no pairs have been found.
  3. Print the message:

     ```
     std::cout << "Starting wire comparison...\n";
     ```

  4. Use nested loops to compare each element with every other element.
     (a) If a pair is found, mark both elements in `paired` as `true` and print:

     ```
     std::cout << "Match found: Wire " << i + 1 << " and Wire " << j + 1
     << " (Value: " << arr[i] << ") are a pair.\n";
     ```

     Where `i` is the index of the wire you are looking to match and `j` is the index of the wire it matches with

     (b) If no pair is found for an element, print:

     ```
     std::cout << "No match found for Wire " << i + 1 << " (Value: " << arr[i]
     << "). Cut this wire to defuse the bomb!\n";
     ```

     Where `i` is the index of the wire you are looking to match

     (c) Remember not to recheck wires that have already been paired.

5. Remember to delete `paired` to avoid memory leaks:

```
delete[] paired;
```

Example Walkthrough:

Consider an array of wires `arr` = {1, 2, 1, 1}. Here is a step-by-step diagram of the process:

1. Initialize the `paired` array: `paired` = {false, false, false, false}.
2. Compare the first wire (1) with the others:
   (a) Compare wire 1 (index 0) with wire 2 (index 1): no match.
   (b) Compare wire 1 (index 0) with wire 3 (index 2): match found. Update `paired` = {true, false, true, false}.
   (c) Print pair message with `i=0` and `j=2`
3. Move to the next unpaired wire (wire 2 at index 1):
   (a) Compare wire 2 (index 1) with wire 4 (index 3): no match.
   (b) No more wires to compare. `paired` = {true, false, true, false}.
   (c) Print no pair message with `i=1`
4. Move to the next unpaired wire (wire 3 at index 3):
   (a) Compare wire 4 (index 3) with the remaining wires
   (b) No more wires to compare. `paired` = {true, false, true, false}.
   (c) Print no pair message with `i=3`

9 Submission checklist

For Task 1:

- Archive (zip) all the files used for Task 1 and rename the archive uXXXXXXXX.zip where XXXXXXX is your student number. The zip should include:
  - task1.h
  - task1.cpp
- Upload the archive to FitchFork Practical 8 Task 1 before the deadline

For Task 2:

- Archive (zip) all files used for Task 2 and rename the archive uXXXXXXXX.zip where XXXXXXX is your student number. The zip should include:
  - `RoomLocater.h`
  - `RoomLocater.cpp`

  - `BombDefusing.h`
  - `BombDefusing.cpp`

- Upload the archive to FitchFork Practical 8 Task 2 before the deadline