Practical 4

COS132

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

Department of Computer Science

Official Deadline: 19/04/2024 at 17:30

Extended Deadline: 21/04/2024 at 23:59

Marks: 510

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.**
- You may not import any libraries that have not been imported for you.
- 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 15 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:

- Implement ternary operators to replace traditional conditional statements for code simplification.
- Develop and use functions for code modularization, enhancing readability and maintainability.
- Apply systematic search strategies like range reduction and digit-based analysis for problem-solving.
- Manipulate ASCII values for character conversions and comparisons, handling edge cases like character wrapping.

4 Structure

This practical consists of three 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 4 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 - Semester Mark Calculation	52
Task 2 - Crack Numeric Password	278
Task 3 - Crack Special Password	180
Total	**510**

Table 1: Mark Distribution

6 Resources

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

- **pow**: The `pow` function is used to raise a base number to the power of an exponent. It is included in the `<cmath>` library and is called using two arguments, `pow(base, exponent)`. `https://www.geeksforgeeks.org/power-function-c-cpp/`
- **ternary if**: The ternary operator is a concise way to perform conditional checks and assign values based on the condition. It follows the syntax `condition ? expression1 : expression2`, where `expression1` is executed if `condition` is true, otherwise `expression2` is executed. `https://www.geeksforgeeks.org/cpp-ternary-or-conditional-operator/`

7 Task 1

Your task is to fix the code given to you in the Task 1 archive. The code attempts to calculate a COS132 student's progress mark so far, however there are various syntax and semantic errors. Fix these by following the instructions below.

You are given this Task1.cpp file, it contains your main loop:

```cpp
#include <iostream>
#include "Calculate.h"
#include "Total.h"
#include "Message.h"
using namespace std;

int main(){
    double q1M = 10.5;
    int q2M = 12;
    int q3M = 11;
    int q4M = 8;

    int q1T = 15;
    int q2T = 13;
    int q3T = 15;
    int q4T = 10;

    int p1M = 10;
    int p2M = 12;
    int p3M = 13;

    int p1T = 10;
    int p2T = 15;
    int p3T = 20;

    double totalPracWeight = 30;
    int numPracs = 10;
```

```
    double totalQuizWeight = 20;
    int numQuizzes = 10;

    double pracWeight = calculateIndividualWeight(totalPracWeight, numPracs);
    double quizWeight = calculateIndividualWeight(totalQuizWeight,numQuizzes);

    double pracSubTotalVal = pracSubTotal(pracWeight, p1M, p1T, p2M, p2T,
       p3M, p3T);

    double quizSubTotalVal = quizSubTotal(quizWeight, q1M, q1T, q2M, q2T,
       q3M, q3T, q4M, q4T);

    double subTotalVal = subTotal(4,quizWeight,3,pracWeight);

    double actualTotal = pracSubTotalVal+quizSubTotalVal;
    cout<<actualTotal<<"/"<<subTotalVal<<endl;

    double result = getResult(actualTotal,subTotalVal);
    cout<<"Result: "<<result<<endl;

    result<0.5? printFailMessage() : handlePass(result);

}
```

1. The **q1M-q4M** and **p1M-p4M** variables contain the mark that the student received for the practicals and quizzes.
2. The **q1T-q4T** and **p1T-p4T** variables contain the total mark that can be acquired for each practical and quiz.

Instructions to fix faulty code:

1. Implement the following functions in your **Calculate.cpp** file:
   (a) The **calculateContribution** function that calculates the contribution of a single assessment using the mark that the student received / the total mark of the assessment * by the weight of the assessment.
   (b) The **calculateIndividualWeight** function that calculates the weight per assessment using the total weight of the assessments / the number of assessments.
2. Fix **Total.h** and **Total.cpp** such that it compiles. Ensure that you fix the guards and includes of **Total.h** and **Total.cpp** where applicable.
3. Implement the **Message.h** file using the **Message.cpp** as a guideline.
4. Implement the following fixes in your **Message.cpp** file:
   (a) The cout statements for **printPassMessage()** and **printFailMessage()** have been swapped around

(b) The ternary if statment in **handlePass(double result)** is incorrect, (semantic error). Fix this to make sure that it goes to the correct function every time

8 Task 2

You have been sent a mysterious email with an interesting coding challenge. You have to develop two methods to unlock a numerical password. First, by efficiently narrowing down the range of possible passwords, until you guess the password. Then, by guessing the password one digit at a time, using math to isolate and verify each digit.

The code you will use to solve this task can be found in the Task2 folder in the Student Files archive. You have been given all the header files, as well as the full implementation of Printer.cpp. DO NOT CHANGE THESE FILES. You have also been provided with a sample main.cpp. We strongly encourage you to write a more comprehensive main, that tests your functions individually as well as the overall functionality.

You are not allowed to add any additional imports (i.e. include), unless you need to include a ".h" in the relevant ".cpp" file. Any other imports may be penalised.

8.1 Sub-Task 1: Comparisons

You have been provided with `Comparisons.h` in which three functions are declared. You are responsible to implement these function in `Comparisons.cpp`.

8.1.1 `bool isEqual(int a, int b)`

This function takes two integers as parameters, `a` and `b`. The function should compare `a` and `b` and return `true` if they are equal, and `false` if they are not.

8.1.2 `bool isLessThan(int a, int b)`

This function takes two integers as parameters, `a` and `b`. The function should compare `a` and `b` and return `true` if the `a` is less than `b`, and `false` if `a` is not less than `b`.

8.1.3 `bool isGreaterThan(int a, int b)`

This function takes two integers as parameters, `a` and `b`. The function should compare `a` and `b` and return `true` if the `a` is greater than `b`, and `false` if `a` is not greater than `b`.

8.2 Sub-Task 2: Range Based Password Search

You have been provided with `RangeBasedPasswordCracker.h`, `Printer.h` and `Printer.cpp`. Do not change these files. You will implement the functions declared in `RangeBasedPasswordCracker.h` in `RangeBasedPasswordCracker.cpp`.

These functions will crack the password by systematically narrowing down the possible values that the password can be.

8.2.1 `int crackPassword(int low, int high, int password)`

This function takes a lower bound (`low`), an upper bound (`high`), and the target `password` as inputs. It searches for the password by calculating the midpoint of the current range as the guess, and narrowing the search space based on the guess relative to the answer.

The function should do the following:

1. Calculate the midpoint of the current range as the guess. The midpoint should be "floored", ie. rounded down when the result is not integer
   - If `low = 1` and `high = 10`, the actual midpoint is 5.5, thus you should use 5. (Truncate or Floor 5.5)
   - If `low = 1` and `high = 9`, the actual midpoint is 5, thus you should use 5.
2. Call `printGuess`, defined in `Printer.cpp` with the current guess (This is really important as your output will be marked)
3. Use the `isEqual` function (that you defined in Sub-Task 1) to check whether your guess is correct.
   (a) If your guess is correct, call `foundPassword` with your guess.
   (b) If your guess is incorrect, call `decideAndSearch` (to be implemented by you) with appropriate parameters.
4. Return the correct password

8.2.2 `int decideAndSearch(int guess, int password, int low, int high)`

This function takes the current `guess`, the `password`, as well as the current lower bound (`low`), and upper bound (`high`) as input. This function then decides the direction of the search after each guess.

The function should do the following:

1. Use the correct comparison operator (defined in Sub Task 1) to determine whether your `guess` is greater or smaller than the target `password`.
   (a) If your `guess` is smaller than `password` you should call `crackPassword` with new bounds:
      - The lower bound should be 1 more than the current `guess`
      - The upper bound should remain unchanged
   (b) If your `guess` is greater than `password` you should call `crackPassword` with new bounds:
      - The lower bound should remain unchanged
      - The upper bound should be one smaller than the current `guess`
2. Return the result of the appropriate `crackPassword` call.

8.3 Sub-Task 3: Digit Based Password Search

You have been provided with DigitBasedPasswordCracker.h, Printer.h and Printer.cpp. Do not change these files. You will implement the functions declared in DigitBasedPasswordCracker.h in DigitBasedPasswordCracker.cpp

Together these functions will crack a password, one digit at a time, by going through all possible values for a digit until a match is found. This process cracks a password digit by digit, starting from the most significant digit and moving towards the least significant one. Each step involves guessing a digit in the current position (starting from 0 through 9) and verifying if it matches the corresponding digit in the actual password. The process is illustrated for a example password "213" in the table below:

Actual Password	2	1	3
No match	0	_	_
No match	1	_	_
Match Move to second digit	2	_	_
No match	2	0	_
Match Move to third digit	2	1	_
No match	2	1	0
No match	2	1	1
No match	2	1	2
Match Password found	2	1	3

8.3.1 `int crackPasswordPerDigit(int password, int numDigits)`

A "wrapper" function to start looking for the correct password. It should call `crackDigit`, with `guessedDigit` = 0, `numDigitsLeftToGuess` = `numDigits` and `currentPasswordGuess` = 0, and return the result.

8.3.2 `int crackDigit(int password, int guessedDigit, int numDigitsLeftToGuess, int currentPasswordGuess)`

This function accepts the following parameters as input:

- `password`: The target password we are looking for
- `guessedDigit`: The value of the digit you are guessing (start from 0, max 9).
- `numDigitsLeftToGuess`: The number of digits you still need to guess. This will start at the length of password, and decrease every time a match is found.

- `currentPasswordGuess`: Our current "guess". For any digit not yet guessed we have "0" as placeholder. Thus for a 3 digit password we start at 0, then 100, 200 etc.

The function should do the following:

1. Call `printCurrentGuess(password, guessedDigit, numDigitsLeftToGuess, currentPasswordGuess)` defined in `Printer`
2. Test whether the number of digits left to guess is 0 (then the password is found)
   (a) If `numDigitsLeftToGuess` is 0, return the current guess
   (b) If `numDigitsLeftToGuess` is not yet 0, call `processDigitGuess` (with the matching parameters) and return the result.

8.3.3 `int processDigitGuess(int password, int guessedDigit, int numDigitsLeftToGuess, int currentPasswordGuess)`

This function also accepts the following parameters as input:

- `password`: The target password we are looking for
- `guessedDigit`: The value of the digit you are guessing (start from 0, max 9).
- `numDigitsLeftToGuess`: The number of digits you still need to guess. This will start at the length of password, and decrease every time a match is found.
- `currentPasswordGuess`: Our current "guess". For any digit not yet guessed we have "0" as placeholder. Thus for a 3 digit password we start at 0, then 100, 200 etc.

The function should do the following:

1. Extract the digit to be compared against from the expected password
   - Eg. For a 4 digit password 1234:
     - If `numDigitsLeftToGuess` = 4, extract "1".
     - If `numDigitsLeftToGuess` = 3, extract "2"
     - If `numDigitsLeftToGuess` = 2, extract "3"
     - If `numDigitsLeftToGuess` = 1, extract "4"
2. Compare the extracted digit with the `guessedDigit`, to see whether the guess is correct
3. Call `crackDigit` with updated parameters based on whether the `guessedDigit` is correct
   (a) If `guessedDigit` is correct, call `crackDigit` with updated parameters that restarts the search for the next digit (`numDigitsLeftToGuess` - 1).
   Remember to update `currentPasswordGuess` to include the updated digit.
   - Eg. If `password` = 1234 and `currentPasswordGuess` at entry to the function was 1000, if the digit "2" was "discovered" update `currentPasswordGuess` to 1200 for the next call to `crackDigit`

(b) If `guessedDigit` is incorrect, continue the search for the correct digit by calling `crackDigit`. Do not update `numDigitsLeftToGuess` or `currentPasswordGuess`

- Remember to update `guessedDigit` to the next digit (if `guessedDigit` = 1 at function entry, call `crackDigit` with `guessedDigit` = 2.

4. Return the result of the call to `crackDigit`

9 Task 3

It turns out the email was from the elite Cyber Sleuths League (CSL). Since you passed with flying colours you have just been recruited to join their team. Your first mission is to infiltrate the highly secure servers of Code Brew—the company you were unjustly fired from due to a mysterious conspiracy.

You have a personal vendetta, yes, but the stakes are higher: inside Code Brew's servers lies critical evidence that can expose a cyber syndicate's misdeeds. You must use your coding skills to crack the password and retrieve the evidence before it's too late.

You vaguely remember the password format, and it's up to you to put the pieces together. Think of it as a game of Wordle, where each correct guess brings you closer to the full password.

You are given this Task3.cpp file, it contains your main loop:

```cpp
#include <iostream>
#include "PasswordCracker.h"
#include "Int.h"
#include "Char.h"
#include "SpecialChar.h"
int main()
{
    initializePasswordCracker("y83MG$#Fjf7?b");

    checkOutput(findChar('z'), 0);
    checkOutput(convertChar(findInt(8)), 1);
    checkOutput(convertChar(findInt(3)), 2);
    checkOutput(findChar('M'), 3);
    checkOutput(findSpecialChar('$'), 5);
    checkOutput(convertChar(findInt(7)), 10);

    return 0;
}
```

9.1 Sub-Task 1: Char

You have been given a **Char.h** and **Char.cpp** file, it contains the function **char findNextChar(char current)** and **char findChar(char target, char current = 'a')**

9.1.1 char findNextChar Function:

This function returns the next ASCII character. It also checks if the current character is the last in its respective case range (‘z‘ or ‘Z‘).

- If `current` is ‘z‘, it wraps around and returns ‘A‘ to start with uppercase letters.
- If `current` is ‘Z‘, it wraps around and returns ‘a‘ to cycle back to lowercase letters.
- If neither, it simply progresses to the next character in the ASCII sequence and returns that character.

This must be done using **ternary if statements**

9.1.2 char findChar Function:

This function finds a target character starting from the given current character.

1. It checks if the `current` character is the `target` character.
   (a) If they match, the `current` character is returned as the result.
   (b) If no match, the function calls itself with the `target` and the result of **findNextChar(current)**.

This implementation uses **ternary if statements** to determine progression or match conditions.

9.2 Sub-Task 2: Int

You have been given a **Int.h** and **Int.cpp** file, it contains the function **char convertChar(int character)** and **int findInt(int target, int current = 0)**

9.2.1 findInt Function:

This function finds a target integer starting from the given current integer.

1. It checks if the `current` integer is the `target` integer.
   (a) If they match, the `current` integer is returned as the result.
   (b) If they do not match, the function calls itself with the `target` and `current + 1`.

This implementation uses **ternary if statements** to increment the current integer until the target is matched.

9.2.2 convertChar Function:

This function converts an integer to its corresponding ASCII character.

1. The function takes an integer `character` which represents a digit from 0 to 9.
2. It returns the ASCII character equivalent by adding `'0'` to the integer. This converts the integer to a character ranging from '0' to '9'.

This conversion is straightforward and does not involve conditional logic but relies on the ASCII standard where the digits 0-9 are sequentially ordered.

9.3 Sub-Task 3: SpecialChar

You have been given a **SpecialChar.h** and **Special.cpp** file, it contains the function **char findSpecialChar(char target, int current)**

9.3.1 findSpecialChar Function:

This function finds a target special character in the ASCII table, starting from a given current ASCII value.

1. It checks if the current ASCII value matches the target character's ASCII code.
   (a) If they match, the current ASCII character is returned as the result, effectively identifying the target character.
   (b) If they do not match, the function calls itself with the target character and the next ASCII value (`current + 1`).

This method utilizes **ternary if statements** to increment the current value until the target character's ASCII code is matched.

10 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:
  - Calculate.h
  - Calculate.cpp
  - Message.h
  - Message.cpp
  - Total.h
  - Total.cpp
- Upload the archive to FitchFork Practical 4 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:
  - Comparisons.h
  - Comparisons.cpp
  - DigitBasedPasswordCracker.h
  - DigitBasedPasswordCracker.cpp

  - Printer.h
  - Printer.cpp
  - RangeBasedPasswordCracker.h
  - RangeBasedPasswordCracker.cpp
- Upload the archive to FitchFork Practical 4 Task 2 before the deadline

For Task 3:

- Archive (zip) all files used for Task 3 and rename the archive uXXXXXXXX.zip where XXXXXXX is your student number. The zip should include:
  - Char.h
  - Char.cpp
  - Int.h
  - Int.cpp
  - SpecialChar.h
  - SpecialChar.cpp
  - PasswordCracker.h
  - PasswordCracker.cpp
- Upload the archive to FitchFork Practical 4 Task 3 before the deadline