Practical 7

COS132

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

Department of Computer Science

Official Deadline: 10/05/2024 at 17:30

Extended Deadline: 12/05/2024 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>` and `cmath`.
- 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:

- Understand and apply appropriate looping constructs (for, while, and do-while loops) to various problem-solving contexts, discerning when each type of loop is most effective based on the structure and requirements of the data manipulation task.
- Translate algorithms and process flows from flowcharts into executable C++ code.
- Employ problem-solving skills to integrate mathematical concepts with programming techniques, enhancing their ability to conceptualize, design, and debug programs that solve complex problems.

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 7 Task 1 and so on. You can assume that all input to functions will be valid (i.e. of the correct data type and within range)

5 Mark Distribution

Activity	**Mark**
Task 1 - Basic Loops	25
Task 2 - Decoding a Message	75
Total	**100**

Table 1: Mark Distribution

6 Resources

- You may use the `pow` function to calculate powers. For more info, visit `https://www.geeksforgeeks.org/power-function-c-cpp/`
- For more information on the mathematical concepts used in this practical you can visit:
  - Perfect Numbers: `https://www.britannica.com/science/perfect-number`
  - Collatz Conjecture: `https://science.howstuffworks.com/math-concepts/collatz-conjectu htm#:~:text=The%20Collatz%20Conjecture%2C%20also%20known,lead%20to%20the%20number 20one.`
  - Armstrong numbers: `https://en.wikipedia.org/wiki/Narcissistic_number`
  - Palindromes: `https://en.wikipedia.org/wiki/Palindrome`

7 Task 1

You have been given a `task1.cpp` with a main function, as well as `loop.cpp` and `loop.h`. Your task is to complete the function implementations in `loop.cpp` according to the instructions.

7.1 `void printCollatz(int num)`

Complete the function so that it implements the flowchart below:

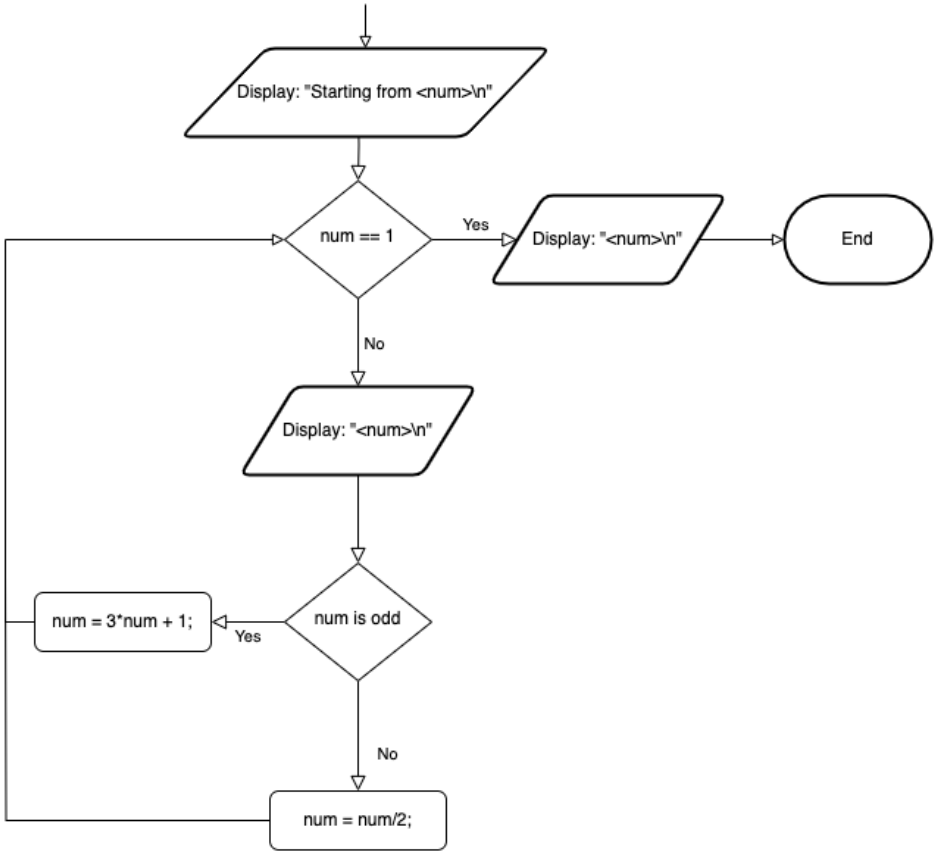

Figure 1: Collatz Flowchart

Ensure that your print statements match the flowchart exactly. Replace <num> with the variable `num`.

7.2 `void printSquares(int n)`

This function should print all "perfect square" numbers from 1, up to the n'th square number. Use a for loop to implement this.

Your output should follow the format: "Square of <index> is <square>\n";

Example: If printSquares is called with n = 3, your output should match:

```
Square of 1 is 1
Square of 2 is 4
Square of 3 is 9
```

7.3 `void printDiamond(int halfSize)`

In this function you should use for loops to print a diamond of stars "*". The `halfSize` parameter describes how many rows should be from the top to the middle of the diamond.

Example: If printDiamond is called with n = 3, your output should match:

```
  *
 ***
*****
 ***
  *
```

There should be no spaces after or between the stars in a row.

Hint: You will need more than one for loop, and you will need to use nested loops. Start by figuring out how to print the right number of spaces per row, then the stars.

8 Task 2

Success! "Operation Codebreaker" has yielded better results than you could have hoped for. Your skills and determination have enabled you to intercept a series of encrypted and corrupted messages from CodeBrew's main server. These cryptic fragments may very well contain the damning evidence needed to unveil the true extent of CodeBrew's manipulations.

The world is on the brink, and the information you hold could tip the scales towards justice. As a key member of "Freedom Bytes," it is now your mission to decrypt and restore these messages fully. Revealing their contents will expose CodeBrew's hidden agenda and could be pivotal in safeguarding the principles of freedom and democracy worldwide.

Time is of the essence. Together with your skilled team at "Freedom Bytes," you must work to decode these secrets. The truth you uncover will illuminate the shadowy machinations of CodeBrew, potentially altering the course of history. Are you ready to crack the code and bring their dark deeds into the light? The fate of our free world might just depend on 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 number utilities and deciphering.

8.1.1 Number Utils

The `NumberUtils.h` header file includes the declarations for functions that perform basic checks on numbers, which will be used in your `Decipher` functions.

```
// NumberUtils.h
#ifndef NUMBER_UTILS_H
#define NUMBER_UTILS_H
```

```
int reverseInteger(int number);
bool isPalindrome(int number);
int sumOfDigits(int number);
int getNumberOfDigits(int number);
bool isArmstrong(int number);
bool isPerfect(int number);

#endif
```

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

```
//NumberUtils.cpp
#include "NumberUtils.h"
#include <cmath>

// Function implementations for number utilities will go here
```

8.1.2 Decipher

The `Decipher.h` header file includes the functions directly involved in deciphering the message.

```
// Decipher.h
#ifndef DECIPHER_H
#define DECIPHER_H
#include <string>
using namespace std;

char shiftCharacter(char input, int shift);
string decipher(int num1, string str1, int num2, string str2);

#endif
```

Create `Decipher.cpp` and include the `Decipher.h`, `NumberUtils.h` and `StringUtils.h` header files.

```
//Decipher.cpp
#include <iostream>
#include "NumberUtils.h"
#include "StringUtils.h"
#include "Decipher.h"

using namespace std;

// Function implementations for deciphering will go here
```

8.1.3 StringUtils

Note that you have been given a `StringUtils.h` and `StringUtils.cpp`. You should not change these files. The functions in these files will be called from some of your functions and will also call some of your functions.

8.2 Detailed Function Implementation Specifications for NumberUtils

8.2.1 int reverseInteger(int number)

- **Purpose:** Reverses the digits of an integer.
- **Parameters:**
  - **int number:** The integer whose digits are to be reversed.
- **Process:**
  1. Initialize an integer `reversed` to 0.
  2. To get the reverse of `number`:
     - Until `number` is 0:
     - Extract the least significant `digit` of `number`, eg. 1234 –> 4. *Hint: Use modulus.*
     - Multiply `reversed` by 10 (to move existing digits one space to the left)
     - Add the extracted `digit` to `reversed`.
     - Get rid of the last digit in `number`, eg 1234–> 123. *Hint: Use integer division.*
  3. Returns `reversed`.

Number	Reversed
1234	0
123	0*10 + 4 = 4
12	4*10 + 3 = 43
1	43*10 + 2 = 432
0	432*10 + 1 = 4321

Figure 2: Illustrating process to reverse a number

8.2.2 bool isPalindrome(int number)

- **Purpose:** Checks if an integer is a palindrome. A number is considered a palindrome if it reads the same forwards and backwards.
- **Parameters:**
  - **int number:** The integer to check for palindrome properties.
- **Process:**
  1. Call the `reverseInteger` function to obtain the reverse of `number`.
  2. Compare the reversed number with the original number.
  3. Return `true` if they are the same, `false` otherwise.

8.2.3 int sumOfDigits(int number)

- **Purpose:** Calculates the sum of the digits of an integer.
- **Parameters:**
  - **int number:** The integer whose digits are to be summed.
- **Process:**
  1. Initializes an integer `sum` to 0.
  2. Uses a `while` loop to add each digit of `number` to `sum`:
     - Extract the last digit of `number` using modulus.
     - Add the digit to `sum`
     - Set `number` to `number` divided by 10.
  3. Returns `sum` when `number` becomes 0.

8.2.4 int getNumberOfDigits(int number)

- **Purpose:** Returns the number of digits a number has.
- **Parameters:**
  - **int number:** The integer whose number of digits should be returned.
- **Process:**
  1. Initialise `digits` to 0.
  2. While number is not 0:
     - Increment `digits`
     - Set `number` equal to `number/10`.
  3. Return `digits`

8.2.5 bool isArmstrong(int number)

- **Purpose:** Checks if a number is an Armstrong number. An Armstrong number is an integer such that the sum of its own digits raised to the power of the number of digits equals the number itself.
  For example: 153 has 3 digits. $1^3 + 5^3 + 3^3 = 153$. Thus 153 is an Armstrong number
- **Parameters:**
  - **int number:** The integer to check.
- **Process:**
  1. Saves the original value of `number` in `original`.
  2. Calls `getNumberOfDigits` to get the number of digits in `number` and saves it in a variable named `digits`.
  3. Computes the `sum` of each digit raised to the power of `digits`.
     Hint: Use the same method to extract each digit as you did in sumOfDigits. You may use the pow() function to compute the powers..
  4. Returns `true` if the `sum` equals the `original` number, `false` otherwise.

8.2.6 bool isPerfect(int number)

- **Purpose:** Checks if a number is perfect. A perfect number is an integer that is equal to the sum of its proper divisors, excluding itself.
  For Example: 6 has divisors 1,2,3: 1+2+3 = 6. Thus 6 is a perfect number.
- **Parameters:**
  - **int number:** The integer to check.
- **Process:**
  1. Initializes `sum` to 0.
  2. Uses a `for` loop to sum all divisors of `number` excluding `number`.
     Hint: If a number divides `number` completely, i.e. there is no remainder, it is a proper divisor and is added to `sum`.
  3. After the loop, compares `sum` with `number`.
  4. Returns `true` if `sum` equals `number`, `false` otherwise.

8.3 Detailed Function Implementation Specifications for Decipher

8.3.1 char shiftCharacter(char input, int shift)

- **Purpose:** Shifts an alphabetic character by a specified number of positions in the alphabet, wrapping around if necessary. This function handles both uppercase and lowercase letters.

- **Parameters:**
  - **char input:** The character to be shifted.
  - **int shift:** The number of positions to shift, which can be positive (right shift) or negative (left shift).
- **Process:**
  1. Initialise a character variable `base` to 'a' if the input character is lowercase, or 'A' if the input character is uppercase.
     Hint: You can use comparison operators >, < etc to check ranges on characters as you would on integers
  2. Calculate the integer `offset` of `input` from `base`. Eg. if `input = 'b'` then `base = 'a'` and `offset = 1`.
     Hint: You can subtract characters.
  3. Calculate the integer `shiftedIndex` by adding `shift` to `offset`.
  4. Make sure that the final `shiftedIndex` is between 0 and 25 (an index per letter) using modulus.
  5. The final `char result` is `shiftedIndex+base`
  6. Return `result`.

8.3.2 string decipher(int num1, string str1, int num2, string str2)

- **Purpose:** Applies decryption transformations to two strings based on the properties of two corresponding numbers. Combines the results into a single string.
- **Parameters:**
  - **int num1:** The first integer used to determine the transformation for `str1`.
  - **string str1:** The first string to be possibly transformed.
  - **int num2:** The second integer used to determine the transformation for `str2`.
  - **string str2:** The second string to be possibly transformed.
- **Process:**
  1. Checks if `num1` is a palindrome.
     - If true, call `returnDecodedString` with str1 and the sum of the digits of `num1` as parameters. Save the result into `str1`.
     - If false, `str1` remains unchanged
  2. Checks if `num2` is either an Armstrong number or a perfect number.
     - If true, call `returnDecodedString` with str2 and the sum of the digits of `num2` as parameters. Save the result into `str2`.
     - If false, `str2` remains unchanged

3. Concatenates the possibly transformed `str1` and `str2` with a hyphen (`"-"`) in between and returns the result.

9 Submission checklist

For Task 1:

- Archive (zip) the file used for Task 1 and rename the archive uXXXXXXXX.zip where XXXXXXX is your student number. The zip should include:
  - loop.cpp
- Upload the archive to FitchFork Practical 7 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:
  - `NumberUtils.h`
  - `NumberUtils.cpp`
  - `Decipher.h`
  - `Decipher.cpp`
- Upload the archive to FitchFork Practical 7 Task 2 before the deadline