Department of Computer Science

COS132 2016
Practical 4

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1 Introduction

1.1 Submission

The submission deadline for this practical is **9 May 2016 at 07:00 AM**.

1.2 Marking

Tasks are marked using fitchfork. Fitchfork marks by comparing the output of your program with specified expected output on a line by line basis. For this reason you should pay close attention to the instructions for the output of your program. Also remember that names of files are case sensitive.

Your code will also be assessed for compliance with coding standards.

1.3 A Serious Warning

It is in your own interest that you, at all times, act responsible and ethically.

1.4 Given code and data

1. If you do not already have a sub-directory called `COS132`, create such directory.
2. Create a subdirectory called `Prac4` in the `COS132` directory.
3. Download `Prac4.tar.gz` from the CS website and save it in the `COS132/Prac4` directory.
4. Extract the content of the `Prac4.tar.gz` archive.
5. After extracting this archive, your `COS132/Prac4` directory should contain the files and directories shown in the following hierarchical structure.

```
COS132
└── Prac4
    ├── Prac4.tar.gz
    ├── Task2
    │   └── raindata.txt
    ├── Task4
    │   └── makefile
    ├── Task5
    │   ├── makefile
    │   └── test.cpp
    └── Task6
        ├── ArrayOperations.h
        ├── data00.txt
        ├── data01.txt
        ├── data02.txt
        └── data03.txt
```

Create subdirectories for the tasks for which there are no subdirectories.

Task 1: Hours worked

Write a program called **`Payroll.cpp`** that asks for the number of hours worked by any number of employees and calculate the average hours these employees worked. It should store the values in an integer array.

Declare a named constant for the maximum size of the array and assign it the value 10000.

The program should use a **sentinal** to end. When the user enters 999, the program should end and display the required output. The number of hours each employee worked should be listed and their average should be calculated and displayed as shown in the following is a sample test run of the program (user input is shown in bold):

```
Enter 999 to stop input
Enter the hours worked by employee #1: 22
Enter the hours worked by employee #2: 33
Enter the hours worked by employee #3: 35
Enter the hours worked by employee #4: 999
-----------------------------------------------------------
The hours you entered are: 22 33 35
The employee(s) worked an average of 30.0 hours this week.
-----------------------------------------------------------
```

The program should terminate without showing the final results when invalid input is entered. Values above **50 hours** as well as **negative input** should be treated as invalid. The following is a sample test run of the program when the program ends with an error message (user input is shown in bold):

```
Enter 999 to stop input
Enter the hours worked by employee #1: 10
Enter the hours worked by employee #2: -12
-----------------------------------------------------------
You have entered -12. It is invalid. Program terminated.
-----------------------------------------------------------
```

The program should check for abnormal input. Values above 40 as well as 0 should be treated as abnormal input. When the user enters an abnormal value the program should prompt for confirmation. If the user confirms the program should accept the value and continue. If the user does not confirm, the program should terminate after showing an error message. The following is a sample test run of the program that includes rejection of an abnormal value (user input is shown in bold):

```
Enter 999 to stop input
Enter the hours worked by employee #1: 8
Enter the hours worked by employee #2: 17
Enter the hours worked by employee #3: 21
Enter the hours worked by employee #4: 0
You have entered 0, which is an abnormal input. Please confirm input (Y or N): N
-----------------------------------------------------------
You have entered 0 and indicated that it is invalid. Program terminated.
-----------------------------------------------------------
```

The following is a sample test run of the program that includes positive confirmation of an abnormal value (user input is shown in bold):

```
Enter 999 to stop input
Enter the hours worked by employee #1: 10
Enter the hours worked by employee #2: 12
Enter the hours worked by employee #3: 33
Enter the hours worked by employee #4: 45
You have entered 45, which is an abnormal input. Please confirm input (Y or N): Y
Enter the hours worked by employee #5: 13
Enter the hours worked by employee #6: 20
Enter the hours worked by employee #7: 25
Enter the hours worked by employee #8: 22
Enter the hours worked by employee #9: 23
Enter the hours worked by employee #10: 24
Enter the hours worked by employee #11: 999
-----------------------------------------------------------
The hours you entered are: 10 12 33 45 13 20 25 22 23 24
The employee(s) worked and average of 22.7 hours this week.
-----------------------------------------------------------
```

Create a tarball containing the **`Payroll.cpp`** and upload it using the active fitchfork assignment called **Prac 4 - Hours worked**.

Task 2: Rainfall graph

You are given a text file called **`raindata.txt`**. It contains 364 integer values representing the rainfall on each day of an entire year measured in millimeters. Write a program that reads these values into an array and can show a graph of the rainfall of a number of consecutive weeks chosen by the user.

The program should prompt the user to specify a starting week number and the number of weeks for which a bar chart should be displayed.

If the user enters a week's index number that is outside the specified range, the program should display an error message and terminate.

If the user enters a week's index number that is in the specified range, the program should prompt the user for the number of weeks that should be displayed. This prompt should specify valid values based on the starting week number. If the user enters a number that is outside the specified range, the program should display an error message and terminate.

The following is a sample test with invalid input:

```
Enter the starting week's index [1 - 52]: 50
Enter the number of weeks [1 - 3]: 4
You entered 4 where only values [1 - 3] are valid.
```

If the user entered valid values, the program should calculate the total rainfall for the weeks within the specified range an then calculate the % of this total rainfall that was recorded for each of the weeks in the range. For each week draw a line of = signs to visualize the calculated value. For 100% the program should draw 80 = signs and for 50% it should draw 40. Round the number of signs to be printed to the nearest integer on this scale.

The following is a sample successful test run.

```
Enter the starting week's index [1 - 52]: 22
Enter the number of weeks [1 - 31]: 4
--------------------------------------------------------------------------------
                                Relative Weekly Rainfall (%)
    0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90  95 100
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
  22|================================
  23|
  24|============================================
  25|
--------------------------------------------------------------------------------
```

Create a tarball containing all the source code for the system as well as a makefile for compiling and running the program. Upload your tarball using the upload slot called
Prac 4 - Rainfall graph.

Task 3: Tic-Tac-Toe Game

Write a program that allows two players to play a game of tic-tac-toe. Use a two- dimensional char array with three rows and three columns as the game board. Each element of the array should be initialized with an asterisk (*). The program should run a loop that

- Displays the contents of the board array
- Allows player 1 to select a location on the board for an X by entering the row and column number.
- Allows player 2 to select a location on the board for an O by entering the row and column number.
- Determines whether the game ends or not.

You should prevent the user from selecting an occupied element or column or row numbers outside the bounds of the board. If a user specifies an invalid or occupied cell, the player should be required to enter his choice again **without displaying any error messages**. The particular player's turn must simply be repeated.

The following is part of a sample test run that contains an example of such behaviour (user input is shown in bold):

```
### TIC TAC TOE GAME ###

* * *
* * *
* * *

Player 1: Enter your move in the format <row> <column>: 1 2

* X *
* * *
* * *

Player 2: Enter your move in the format <row> <column>: 1 2

* X *
* * *
* * *

Player 2: Enter your move in the format <row> <column>: 2 2

* X *
* O *
* * *

Player 1: Enter your move in the format <row> <column>: 0 7

* X *
* O *
* * *

Player 1: Enter your move in the format <row> <column>: 2 2

* X *
* O *
* * *

Player 1: Enter your move in the format <row> <column>: -1 -1

* X *
* O *
* * *

Player 1: Enter your move in the format <row> <column>:
```

It is important that your program should have output matching this example exactly. The introduction line as well as the open lines and prompts should be as shown here.

The game ends when a player has won, or a tie has occurred. A player wins when there are three of his tokens in a row/column/diagonal on the game board. If a player has won, the program should declare that player as the winner and end. A tie occurs when all of the locations on the board are full, but there is no winner. If a tie has occurred, the program should say so and end. The following is a sample test run with a winner (user input is shown in bold):

```
### TIC TAC TOE GAME ###

* * *
* * *
* * *

Player 1: Enter your move in the format <row> <colum>: 1 2

* X *
* * *
* * *

Player 2: Enter your move in the format <row> <colum>: 2 2

* X *
* O *
* * *

Player 1: Enter your move in the format <row> <colum>: 2 3

* X *
* O X
* * *

Player 2: Enter your move in the format <row> <colum>: 1 1

O X *
* O X
* * *

Player 1: Enter your move in the format <row> <colum>: 3 2

O X *
* O X
* X *

Player 2: Enter your move in the format <row> <colum>: 3 3

O X *
* O X
* X O

Player 2 has won!
```

Create a tarball containing all the source code for the system as well as a makefile for compiling and running the game. Upload your tarball using the upload slot called
Prac 4 - Tic-Tac-Toe Game.

Task 4: Chips and Salsa

Write a program that lets a maker of chips and salsa keep track of sales for six different types of salsa sold. The types are: mild, medium, sweet, hot, zesty, and SUPA hot.

You are given a makefile. You have to create the following files that will successfully compile with the given makefile.

1. **`ChipsAndSalsa.h`**
   – This file should contain definitions of the prescribed constants and arrays.
2. **`ChipsAndSalsa.cpp`**
   – This file should contain a main function producing the prescribed output

You will notice that the makefile expect only these two files. You should declare the prototypes of the functions you use above the main function in **`ChipsAndSalsa.cpp`** and include their implementations after the main function in **`ChipsAndSalsa.cpp`**.

ChipsAndSalsa.h

1. Declare a constant named **`NUM_OF_SALSA_TYPES`**. Assign the value **6** to this constant. This constant should be used in the declaration of the arrays as well as in the main function where needed.
2. Declare an array of strings named **`salsaName`** that holds the six salsa names. The given salsa names should be stored in this array using an initialization list.
3. Declare a parallel array of integers named **`numOfJars`** to hold the number of jars sold during the past month for each of the salsa types in the corresponding element in **`salsaName`**.

Create a tarball containing only **`ChipsAndSalsa.h`** and upload your tarball using the upload slot called **Prac 4 - Chips and Salsa header**.

ChipsAndSalsa.cpp

This file should have the pre-processor directive `#include "ChipsAndSalsa.h"`.

Declare the prototypes of the functions you use above the main function in this file and include their implementations after the main function in this file.

The program should prompt the user to enter the number of jars sold for each type. Each number should be read as a real value. Cast the input to an integer before storing it in the array of integers. Once this sales data has been entered, the program should produce a report that displays sales for each salsa type, total sales, and the name of the maximum selling and minimum selling products.

If an equal number of jars were sold for more than one salsa type and this value is the lowest. The last type in the list should be reported as the minimum selling product.

If an equal number of jars were sold for more than one salsa type and this value is the highest. The first type in the list should be reported as the maximum selling product.

Error handling

The program should not crash when invalid input is given. Rather when invalid input is given, the program should merely cause a warning to be displayed after the program result is displayed. Deal as follows with the different types of faulty input:

- Zero -> accept
- Negative -> make 0
- Fractions -> ignore fractional part

If one ore more input values were faulty, a warning containing a count of the errors should appear as the very last line of output. Otherwise the following message should appear:

```
No errors were encountered during input.
```

The following is a sample test run. Your program should produce the same – pay special attention to positioning of open lines and lines of dashes and lines of stars as they are used by fitchfork to find the lines where the output is produced.

```
********************
* CHIPS AND SALSA  *
********************
Enter the number of jars of mild sold: 5
Enter the number of jars of medium sold: 8
Enter the number of jars of sweet sold: -9
Enter the number of jars of hot sold: 3.897
Enter the number of jars of zesty sold: 4
Enter the number of jars of SUPA hot sold: 1

*************************
* CHIPS AND SALSA REPORT *
*************************
Name:           Amount:
mild                5
medium              8
sweet               0
hot                 3
zesty               4
SUPA hot            1
----------------------
Total:             21
----------------------

The maximum sold was medium at 8 jars.
The minimum sold was sweet at 0 jars.
Warning: A total of 2 errors were encountered during input.
```

Create a tarball containing the given makefile, `ChipsAndSalsa.h` and `ChipsAndSalsa.cpp`. Upload it using the active fitchfork assignment called **Prac 4 - Chips and Salsa program**.

Task 5: Reducing a fraction

Write a program that can be used to express a fraction in its lowest form.

Functions

In order to complete this task, you are expected to complete the implementation of the three functions described below.

- `void displayFraction(int a, int b)`
  Takes two integer numbers and display them in a fraction format such as a/b. Use default values for the parameters of this function so that if the function is called without parameters 0/1 is displayed and when called with one parameter (say x), then x/1 is displayed. Note that this function should simply display what it is given even if it is invalid.

- `int gcd(int a, int b)`
  Takes two positive integers and returns the greatest common divisor of the two numbers. The greatest common divisor of any two numbers a and b is the largest positive integer c that is a divisor of both a and b. The number c is a divisor of b if the remainder after dividing b by c is zero. When passing 0 or a negative value as any one of these parameters, the function should return -1.

- `bool reduceFraction(int &num, int &denom)`
  Takes two integers representing the numerator and denominator of a fraction. Reduces the fraction to its lowest form. For example, the result of reducing $\frac{20}{4}$ is $\frac{5}{1}$. When passing 0 or a negative value as any one of these parameters, the function should return **`false`**, otherwise it should return **`true`**.

Create a header file called **reduce.h** that contains the definition of the functions and an implementation file called **reduce.cpp** that contains the implementation of the functions described above.

Create a tarball containing only **`reduce.h`**. Upload this tarball using the upload slot called **Prac 4 - Fraction header**.

You are given a makefile to compile and run a test program as well as the production program. The given test program is called **`test.cpp`**. Give the following command to use the test program to interactively test the implementation of your functions.

```
make test
```

Create a tarball containing only **`reduce.h`** and **`reduce.cpp`**. Upload this tarball using the upload slot called **Prac 4 - Fraction functions**.

Main program

Write a program called **fractions.cpp** that uses the above functions. The program should prompt the user for a numerator and denominator. When given valid input, it should display the GCD of the two input values and then display the given fraction, the fraction in its lowest form as well as the fraction in decimal format.

The following is a sample test run (user input is shown in bold):

```
Fraction Reduction
------------------------------
Enter the numerator: 4
Enter the denominator: 20
The GCD of 4 and 20 is 5
The lowest form of 4/20 is 1/5 = 0.2
```

When the user enters 0 as the denominator the program should notify the user with a phrase containing "division by zero" instead of displaying the GCD and then exit.

When negative values are entered, the program should deal with it as follows:

- To display the GCD, display the absolute values (positive equivalents) and their GCD.
- To display the given fraction, display negative values as given.
- To display the fraction in its lowest format, display the negative in the numerator if the fraction is negative and without signs when it is positive. Recall that a negative divided by a negative is positive.
- To display the fraction in decimal format, display it with a negative sign if the fraction is negative and without a sign when it is positive.

Give the following command to verify that your program compiles and run as required when using the given makefile for compiling and running the production program.

```
make run
```

Create a tarball containing only `fraction.cpp`. Upload it using the active fitchfork assignment called **Prac 4 - Fraction reduction**.

Task 6: 2D Array Operations

Functions

You are given a header file called **`ArrayOperations.h`** that contains the definitions of functions you have to implement.

Create a cpp file called **`ArrayOperations.cpp`** that contains the implementations of the functions defined in the given header file.

Create a driver program to test these functions and a makefile with a custom test target to compile and test your functions with this driver.

You will notice that `getTotal()` is overloaded to calculate the overall total as well as row totals and column totals. Overload the other thee functions in a similar way to add functionality to perform their operations on a specified row or specified column. Extend your driver program to test the additional overloaded functions.

Create a tarball containing only **`ArrayOperations.h`** and **`ArrayOperations.cpp`**. Upload this tarball using the upload slot called **Prac 4 - 2D Array functions**.

Main program

Write a production program called **`ArrayViewer.cpp`** that creates a two-dimensional **`double`** array and populate the array with data from a given text file with test data. Use the constants in the given header file for the array dimensions and use user-input to specify the data file. The program should display error messages instead of the complete output if the given input file does not exist or does not have enough data to fill the array with the specified dimensions.

The following is an example test run showing an error message:

```
Please enter file name: data03.txt
This program needs 40 real values to fill an array with 8 rows and 5 columns.
Only 17 real values could be read from the file called data03.txt.
```

If the program is able to fill the entire specified array, it must display the content of the array as shown in the example test run below. All values should be displayed accurate to 2 decimal places. Use the constant in the given header file for the column width in the output.

The program should use the functions you have written in the first part of this task to display the row totals and other detail about the elements in each row as shown in the example test run below.

The program should use the functions you have written in the first part of this task to display the column totals, the grand total as well as the overall maximum and minimum in the final output line as shown in the example test run below.

```
Please enter file name: data01.txt
                                                  |     Total       Max       Min
--------------------------------------------------|------------------------------
      1.77      4.22      6.64      3.76      7.61 |     24.00      7.61      1.77
      1.08      7.13      8.04      1.34      1.26 |     18.85      8.04      1.08
      4.91      4.51      8.47      6.62      4.81 |     29.32      8.47      4.51
      6.06      3.37      5.00      0.70      3.62 |     18.75      6.06      0.70
      6.32      5.61      0.46      5.02      0.33 |     17.74      6.32      0.33
      0.90      9.15      9.10      7.50      9.40 |     36.05      9.40      0.90
      7.35      9.33      8.67      6.24      1.62 |     33.21      9.33      1.62
--------------------------------------------------|------------------------------
     28.39     43.32     46.38     31.18     28.65 |    177.92      9.40      0.33
```

Test your program with the given data files. You should also test it with other data. The program output should change accordingly if the values of the constants in the header file are altered.

Create a tarball containing only **`ArrayViewer.cpp`**. Upload it using the active fitchfork assignment **Prac 4 - 2D Array Operation**.