Department of Computer Science

COS132 2015
Practical 3

Ms Vreda Pieterse

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

0 Introduction

0.1 Uploads and Deadline

The uploads will become available on Monday 16 March.

The submission deadline for this practical is **23 March 2015 at 07:30 AM**. Make sure that you have submitted your code to Fitchfork by then. **No late submissions will be accepted**.

0.2 Challenges

Note that the challenges is **NOT** part of the assignment. It is given as a challenge to those who find our usual assignments too trivial. It does not contribute to your marks. It is not expected from the Teaching Assistants to assist you to complete these assignments. It is intended for you to figure it out on your own and stimulate you to use the opportunities you have to develop your programming skills to the fullest. There will also not be any uploads on these.

0.3 Plagiarism Policy

All submissions for this practical and all future practical should be documented as specified in our plagiarism policy. Refer to the notes about our plagiarism policy which can be found on the COS132 website on ClickUP.

0.4 Download and extract

Create a directory called Practical3. Download the archive called `Practical3.tar.gz` and save it in this directory. Extract the archive.

After extracting this archive the directory you have created for this practical should contain the files and sub-directories shown in the following hierarchy. You are advised to create sub-directories for other tasks.

```
Practical3
├── Practical3.tar.gz
├── OddsAndEvens
│   ├── data00.txt
│   ├── data01.txt
│   └── data02.txt
├── Claims
│   └── claims.dat
└── Convert
    ├── convert.cpp
    ├── inchToMill.h
    ├── inchToMill.cpp
    └── makefile
```

1 The cost of failing

Write a program to calculate the cost of repeating COS 132 next year. Express this cost as the number of beers, movies or burgers that a student will have to give up. You need to ask the user which option he/she wants to see.

The cost of the course is R 5000. You will need to come to campus one extra day per week which will increase your travel costs by R50 a week. The semester is 14 weeks long. The cost of a beer is R15, a burger R25 and a movie R30.

The following shows a test run of the program:

```
Enter the 'currency' of your choice (beer / burger / movie):  burger
COS 132 will cost you 228 burgers.
```

Save your program in a file named **cost.cpp**, put it in a tarball and submit it on the CS website in the slot named **Cost of failing**.

1.1 Standard version

For the automatic assessed version you may assume that the user will use only lower case when entering data and will enter valid data.

1.2 Challenge version

Create a version of your program that can deal with any mix of upper and lower case characters. It must also display a message like the following when another 'currency' is entered. The input must be changed to be initial caps.

The following shows a test run of the program complying with the challenge specs:

```
Enter the 'currency' of your choice (beer / burger / movie):  daNce cLass
Sorry, I cannot express the cost of COS 132 in terms of Dance class.
```

2 Colour Mixer

The colours red, blue and yellow are known as the primary colours because they cannot be made by mixing other colours. When you mix two primary colours, you get a secondary colour.

The secondary colour you get when mixing equal parts of two primary colours are shown in the following table:

Primary colours mixed	**Resulting secondary colour**
red and blue	purple
red and yellow	orange
blue and yellow	green

Write a program that prompts the user to enter the names of two primary colours in two separate prompts.

If the user enters anything to any one of these prompts that is not “red”, “blue” or “yellow” the program should display an error message and terminate immediately.

If the user enters the same colour to both these prompts the program should display an error message and terminate immediately.

If the input was successful, the program should display the result in a full sentence. The following is a sample test run of this program:

```
Enter the first colour:  blue
Enter the second colour:  red
When mixing blue and red you will get purple
```

Save your program in a file named **colour.cpp**, put it in a tarball and submit it on the CS website in the slot named **Colour mixer**.

2.1 Standard version

For the automatic assessed version you may assume that the user will use only lower case when entering data.

2.2 Challenge version

Create a version of your program that can deal with the users entering the data using upper and lower case characters as they like.

Extend the mixer to allow the user to input secondary as well as primary colours and show the correct tertiary colours.

See `http://www.color-wheel-artist.com/primary-colors.html` for detail about these colours and what colours you get when mixing them.

3 Magic dates

The date June 10, 1960 is special because when we write it in the format specified by the template `YY/MM/DD`, the month times the day equals the year.

Write a program that uses three consecutive prompts to input the day, the month and the year as integers (in this order). The program should then output the date in the format as specified by the above template.

The program should then determine if the month times the day is equal to the displayed year. The output of the calculation and the result should be in the next line. First show the calculation in a formula that matches the following template.

```
MM * DD <op> YY
```

When using the template, `<op>` should be replaced by one of the following three relational operators: `< > ==`

If the month times the day equals the year, the line containing the formula should be appended with tab character followed by the word **Magic** after the expression.

The following are two a sample test runs:

```
Enter the day:  10
Enter the month:  6
Enter the year:  1960
You entered 60/06/10
6 * 10 == 60 Magic
```

```
Enter the day:  1
Enter the month:  1
Enter the year:  2000
You entered 00/01/01
1 * 1 > 0
```

Test your program with today's date and some other dates. The program should work irrespective if the year is entered as a four digit number or as a two digit number.

Save your program in a file named **date.cpp**, put it in a tarball and submit it on the CS website in the slot named **Magic dates**.

3.1 Standard version

For the automatic assessed version you may assume that the user will input valid dates.

3.2 Challenge version

Create a version of your program that gives error messages when the user enters invalid data. For example when the day is out of range. The ranges are not the same for all the months – 31 July is valid, but 31 August not.

Also cater for leap years. See `https://www.timeanddate.com/date/leapyear.html` and `http://www.timeanddate.com/date/february-30.html`.

4 Sum of series

Write a program to calculate the sum of the first n terms in the series

$$\frac{2}{1^2} + \frac{4}{2^2} + \frac{6}{3^2} + \frac{8}{4^2} + \frac{10}{5^2} + \dots$$

Your program should ask the user to enter the number of terms to be used and then display the entered number of terms in the series as fractions showing their numerators and their calculated denominators as integer values.

Your program should also find the sum of the series and display the answer. The following is a sample test run of the program:

```
Enter the number of terms:7

      2        4        6        8        10       12       14
   ------ + ------ + ------ + ------ + ------ + ------ + ------ = 5.18571
      1        4        9       16       25       36       49
```

Input validation:

- Do not accept a number less than 1 for the number of terms.

Save your program in a file named **series.cpp**, put it in a tarball and submit it on the CS website in the slot named **Sum of series**.

4.1 Standard version

For the automatic assessed version your program should limit the input value to be small enough to ensure that the output does not wrap on a screen that is 160 characters wide.

4.2 Challenge version

- Allow the series to start at a specified term. Prompt the user for the numerator of the starting term (it should be even).
- Create a version of your program that will wrap long series correctly and ensures that the result will not overflow.

5 Odds and evens

Write a program that reads positive integers from a file and then display “success" if the sum of the odd numbers read equals the product of the even numbers read.

To illustrate what is required, note that if the file contains the numbers $2, 1, 15, 3, 2, 9, 1, 3, 8$ the program should output “success" because

$$1 + 15 + 3 + 9 + 1 + 3 = 2 \times 2 \times 8$$

Let the user specify the name of the file. The program should read values from the specified file and echo them to the screen while processing them. Processing should end when the whole file is processed or if zero, a negative number, a non-integer or a character is encountered before the end of the file. For your convenience we have provided files with test data that you may use to test your program.

The following is a sample test run of this program:

```
Enter the file name:  data01.txt
Input:  2 1 15 3 2 9 1 3 8
Result:  success
```

- Save your program in a file named **odds.cpp**.
- Test your program with the given data files.
- Create at least two more data files containing test data. The content of these files will be evaluated for coverage and accuracy.
- Create a tarball that contains your .cpp file and all the data files. If all the data files have names data##.txt, the following command will do exactly that:

  ```
  tar -cvz *.txt odds.cpp -f odds.tar.gz
  ```

- Submit the tarball to the COS132 Prac 3 Odds and Evens slot on the CS website.

Submit **odds.tar.gz** on the CS website in the slot named **Odds and evens**.

5.1 Standard version

When a non-integer value is encountered in the file, the error is only raised after the integer part of the real number was read (and possibly processed). For the automatic assessed version the program should consider and process the integer part of such real value when doing the calculations and termnate only at the stage that the error is raised.

5.2 Challenge version

For the challenge version of the program the integer part of a real value on the file should NOT be considered part of the calculation. The entire number should be ignored in the calcualtions.

The dual of the above problem is to check whether the sum of even numbers in a sequence of integers is equal to the product of the odd numbers. You are invited to participate in a discussion on the discussion board about writing a program to solve this problem that can produce the correct answer without even opening the file.

6 Guessing game

The aim of the guessing game is to predict the colour of the last remaining ball that are randomly selected. The game starts by having an arbitrary but pre-specified number of black and white balls (for example, those illustrated in Figure 1) are mixed together in a bag.

When the game is played, two balls are randomly drawn from the bag at a time and one is placed back according to specified rules. The player wins if he/she predicts the colour of the last ball remaining in the bag correctly.

A game step consists of drawing out two random balls from the bag, examining their respective colours, and then returning one ball back into the bag according to the following rules:

1. If 2 *black* balls were drawn, return a *white* ball. (Assume that there are enough white balls in stock.)
2. If 1 *white* and 1 *black* ball were drawn, return 1 *black* ball.
3. If 2 *white* balls were drawn, return 1 *white* ball.

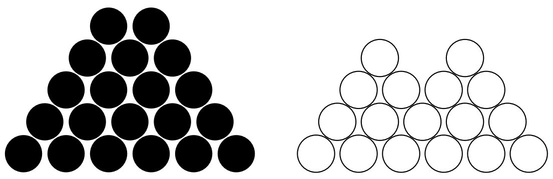

Figure 1: 20 Black and 17 White Balls

Clearly the number of balls in the bag declines by one each time a step is executed. The game stops if there is only one ball left. If we assume that there were initially `b` black balls and `w` white balls in the bag, the game stops after `b + w - 1` steps.

When playing the game, the following steps will be executed:

1. Place `b` black balls and `w` white balls in the bag.
2. Guess the colour of the last remaining ball.
3. Repeatedly execute game steps until one last ball remains.
4. Examine the colour of the last ball and check whether you have guessed its colour correctly.

Implement C++ code to simulate this guessing game.

Start by requesting the user to input a seed value for the random number generator. Read in this value and use it as a seed in the `srand` function. If the user specifies 0 as seed, use `time(0)` as seed to make the game truly random.

Request and read in from the user the starting number of white balls. Do the same for the black balls. Ask the user to guess the colour of the last ball, keying in a character 'b' for black and 'w' for white.

To simulate the execution of a game step, generate a random number between 1 and the current number of balls in the bag. If the generated number is less than or equal to the number of black balls, assume that a black ball has been drawn; otherwise assume a white ball[1]. Update the number of balls in the bag accordingly, and record the colour of the first ball drawn in some variable. Repeat these steps for drawing the second ball.

Now apply the rules to update the number of black and white balls in the bag after drawing out two and returning one. Report on the colour of balls drawn out of the bag and on the number of black and white balls left in the bag after this step.

Repeat this simulation of a game step until there is only one ball left in the bag. Compare its colour against the player's initial guess, and announce to the player whether she has won or lost.

Nest the game code (i.e. everything except the acquisition of a seed value for random number generation) in a flag-controlled while loop. The flag should initially be set to allow a first execution of the game. At the end of a game, the user should be asked whether or not to start a new game, and the flag set accordingly.

[1] Note that this makes probabilistic sense. For example, if there were 5 black balls and 95 white balls in the bag, then the chances of drawing a black ball in real life is 5%. Similarly, the chances that a random number whose range is 1 to 100 will be 5 or less, is also 5%.

The program should show the steps in the simulation in a table. The following is an example of a test run:

```
Provide a seed value for the random number generator: 13
Enter how many white balls for this game: 2
Enter how many black balls for this game: 5
Guess the colour of the last ball ('b' for black or 'w' for white): b

 -----------------------------------------------------------------
 | Iteration  | First ball | Second ball | White balls | Black balls |
 +------------+------------+-------------+-------------+-------------+
 |     0      |     -      |      -      |      2      |      5      |
 |     1      |   white    |    white    |      1      |      5      |
 |     2      |   black    |    black    |      2      |      3      |
 |     3      |   white    |    black    |      1      |      3      |
 |     4      |   black    |    white    |      0      |      3      |
 |     5      |   black    |    black    |      1      |      1      |
 |     6      |   black    |    white    |      0      |      1      |
 -----------------------------------------------------------------

Your guess was correct!
Would you like to play again (Y/N)? Y
Enter how many white balls for this game: 2
Enter how many black balls for this game: 0
Guess the colour of the last ball ('b' for black or 'w' for white): b

 -----------------------------------------------------------------
 | Iteration  | First ball | Second ball | White balls | Black balls |
 +------------+------------+-------------+-------------+-------------+
 |     0      |     -      |      -      |      2      |      0      |
 |     1      |   white    |    white    |      1      |      0      |
 -----------------------------------------------------------------

Your guess was wrong
Would you like to play again (Y/N)? n
```

Save your program in a file named **guess.cpp**, put it in a tarball and submit it on the CS website in the slot named **Guessing game**.

6.1 Standard version

There is only one version of the program.

6.2 Challenge

There is a way of predicting the colour of the last ball in the bag. Can you discover that way?

7 Insurance claims

You have been provided with a file called `claims.dat` which contains claims data for a company which insures motor cars. Each line contains the information for one claim and consists of the following fields:

- Policy number (integer)
- Claim type (string)
- Incident date (yyyy-mm-dd)
- Claim amount (real)
- Gender (string)
- Age (integer)
- Vehicle value (real)
- Year manufactured (integer)
- Colour (string)
- Immobilizer (bool)

Write a menu driven program to calculate the total value as well as the average value of all claims in respect of cars with a specified colour. The following is a template for the program output:

```
1. Blue cars
2. White cars
3. Red cars
4. Quit
Enter your choice (1-4):
-------------------------------------------------------------------------
Received <number> claims in respect of <colour> cars
The total value of claims in respect of <colour> cars is R <value>
The average value of claims in respect of <colour> cars is R <value>
-------------------------------------------------------------------------
```

If the user enters 1, the program should calculate the required information in respect of blue cars. The same should be done in respect of white and red cars when the user respectively enters 2 or 3. If the user enters 4 the program should end.

7.1 Standard version

- When an invalid option is chosen, the program should produce only one line of dashes and no other output and immediatly loop back to the menu -i.e. **No error messages**.
- All amounts should not contain spaces within the numbers and should appear rounded to the nearest cent, while integer counts should not be shown with fractional parts.

Save your program in a file named **claims.cpp**, put it in a tarball and submit it on the CS website in the slot named **Insurance claims**.

7.2 Challenge version

Formatting

Usually when large values are printed, it is customary to print them with a space before every three digits to improve the readability of the numbers and make it easier to verify if they are correct. You are required to apply this custom when displaying the results with this program. Large numbers should thus be displayed with a space separating every three digits. In modern software like spreadsheets and word processors there are often formatting specifications that automatically reformat numbers using a thin space called a *digit grouping symbol* to group the digits within a number. For this assignment you may use the usual space character (ASCII 32) for this purpose.

Delayed error message

The program must also alter the prompt when displaying the menu after the user entered an invalid menu option.

Example

The following is a sample run complying with the **challenge** specifications:

```
1. Blue cars
2. White cars
3. Red cars
4. Quit
Enter your choice (1-4): 7
----------------------------------------
1. Blue cars
2. White cars
3. Red cars
4. Quit
Invalid menu option 7 was given. Enter a valid choice (1-4): 2
----------------------------------------
Received 1130 claims in respect of white cars
The total value of claims in respect of white cars is R 69 828 346.72
The average value of claims in respect of white cars is R 61 795.00
----------------------------------------
1. Blue cars
2. White cars
3. Red cars
4. Quit
Enter your choice (1-4) : 4
```

8 Convert Table

This is a challenge question. It does not have a standard version. Although it has an upload it does not contribute to your marks.

Write a program that generates a table of conversions from feet-inches to millimeters. Ask the user the starting value for the feet-inches column and increment by 4 inches for each consecutive row. The number of rows should be 10.

The following is a sample test run of the program:

```
Starting value for feet: 5
Starting value for inches: 13
------------------------------------
  Imperial Units  |   Decimal Units
------------------+-----------------
       6’  1"     |    1854.2 mm
       6’  5"     |    1955.8 mm
       6’  9"     |    2057.4 mm
       7’  1"     |    2159.0 mm
       7’  5"     |    2260.6 mm
       7’  9"     |    2362.2 mm
       8’  1"     |    2463.8 mm
       8’  5"     |    2565.4 mm
       8’  9"     |    2667.0 mm
       9’  1"     |    2768.6 mm
```

Note that the notation ” refer to inches and ’ is used to indicate feet. Thus 7 feet 3 inches is written 7’ 3”

For the conversion, the program should use a function named `millimeters` that accepts feet and inches as arguments and returns the corresponding millimeters. The prototype for this function is given in the given file named `inchToMill.h`.

You should also use a function named `addInches` to update the values of feet and inches when incrementing the distance with a value that is passed as a value parameter to this function. The prototype for this function is given in the given file named `inchToMill.h`.

Note that 1 feet = 12 inches and 1 inch = 25.4 millimeters.

You are given a framework for the program that you should use. You only have to write code in the body of the functions in the given .cpp files. In `inchToMill.cpp` you have to write code to comply with the descriptions of the functions that is given in `inchToMill.h`. In `convert.cpp` you have to **replace** the comments describing the code with actual c++ code.

First verify that the given code is free of syntax errors

- In the terminal navigate to the `Convert` sub-directory
- Compile the given code by typing the following command:

```
make
```

- Run the program by typing

```
./convert.out
```

Knowing this, you should be convinced that when you add code, and the code does not compile anymore, the error is in the code that you added even though the compiler may indicate otherwise.

- In `convert.cpp` **replace** the comments describing the code with actual c++ code.
- In `inchToMill.cpp` insert code in the body of each of the functions to comply with the descriptions of the functions that is given in `inchToMill.h`.
- Use the following command to create a tarball with these files for upload purposes:

```
tar -cvz *.cpp *.h makefile -f convert.tar.gz
```

9 Upload specifications

Task	**Name of cpp file**	**Extras**	**Mark**
The cost of failing	cost.cpp		10%
Colour mixer	colour.cpp		10%
Magic Dates	date.cpp		15%
Sum of Series	series.cpp		15%
Odds and evens	odds.cpp	At least two more files containing test data	10% + 5%
Guessing game	guess.cpp		20%
Insurance claims	claims.cpp		15%
Conversion table	convert.cpp	makefile inchToMill.h inchToMill.cpp	-
Total			**100**

Note that the percentages given here is NOT the maximum marks possible when uploading to fitchfork.