Department of Computer Science

COS132 2016
Practical 5

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Submission

The submission deadline for this practical is **23 May 2016 at 07.00am**. Note that this is after the weekend. You should aim to complete this assignment before the weekend because staff support is not available during the weekend.

size_t data type

size_t is an alias of one of the fundamental unsigned integer types, defined in `<cstddef>`. It is a type that is guaranteed to be able to represent any array index, or the size of any object in bytes. **size_t** is the type returned by the `sizeof` operator and is widely used in the standard library to represent sizes and counts. You should *prefer* using **`size_t`** instead of **`int`** or **`unsigned`** for variables that hold the size of an array, and for loop counters that represent array indexes. For example:

```
for (size_t i = 0; i < len; i++)
    array[i] = ...;
```

h-files and the #include pre-processor

Typically an **h-file** is the interface specification for a set of functions. It contains the function prototypes and descriptions of the functions as well as the declaration of constants. It should have as little as possible code and should avoid including other files. The name of the file should be descriptive of the purpose of the functions. It should have **.h** as its file name extension. The following is a minimal example of an **h-file** called **generalTool.h** defining a single function:

```
#ifndef GENERAL_TOOL
#define GENERAL_TOOL
/**
*  @brief Create a string consisting of a specified number of dashes
*  @param len represents the number of dashes
*  @return a string with consisting of the specified number of dashes
*/
string dashes(size_t len);
#endif
```

The implementation of the functions are usually implemented in a **cpp-file** with the same name as the **h-file** but with **.cpp** as its file name extension. the **cpp-file** and any other implementation files (such as the main program of the system that wants to use these functions) should use the #include pre-processor to include the **h-file**. The libraries that may be needed for the implementation of the functions should be included in the **cpp-file**. These are usually

listed above the #include pre-processor to include the **h-file** especially when the functions use data types and constants that are declared in these libraries. The following is the **cpp-file** called **generalTool.cpp** implementing the function defined in the above **h-file**:

```
#include <string>

using namespace std;

#include "generalTool.h"

string dashes(size_t number)
{
    string lineOfDashes;
    lineOfDashes.assign(len, '-');
    return lineOfDashes;
}
```

A statement such as `using namespace std;` to indicate that the implementation code access entities whose names are part of the namespace called std does not belong in an **h-files**. If needed, it should be in the implementation files **ccp-files**.

Given code and data

Extract the content of the `Prac5.tar.gz` archive. After extracting this archive in a directory named `COS132/Prac5`, this directory should contain the files and directories shown in the following hierarchical structure.

```
COS132
└── Prac5
    ├── Prac5.tar.gz
    ├── Task1
    │   ├── makefile
    │   ├── testVector.cpp
    │   └── vectorFunctions.h
    ├── Task2
    │   ├── makefile
    │   ├── viewer.cpp
    │   ├── array8.txt
    │   ├── array20.txt
    │   └── array3.txt
    ├── Task3
    │   ├── names.txt
    │   └── nameSort.h
    ├── Task5
    │   ├── chequeTools.h
    │   └── chequeFacts.txt
    └── Task6
        ├── makefile
        ├── raindata.txt
        ├── rainfall.cpp
        └── arrayUtilities.h
```

Task 1: Vectors

You have been introduced to vectors at some previous level of mathematics. At this level you must be familiar with basic operations on vectors. They can be added, subtracted, multiplied by a scalar and have a dot product etc. Vectors are represented as a column matrix as shown below.

$$\begin{pmatrix} 2 \\ -9 \end{pmatrix} \quad \begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 1 \\ -2 \\ -3 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 2 \\ -3 \\ 4 \end{pmatrix}$$

You are given a **makefile**, a h-file called **vectorFunctions.h** and a program called **testVector.cpp**. You are expected to implement a set of functions for vectors with integer elements. We store the vector elements in arrays. In general, the array $\texttt{v} = \{\texttt{x}_0, \texttt{x}_1, \ldots, \texttt{x}_\texttt{n}\}$ stores the elements for the vector;

$$v^- = \begin{pmatrix} x_0 \\ \vdots \\ x_n \end{pmatrix}$$

Implement these functions in a file called **vectorFunctions.cpp**. If you need to use a library (like <cmath>) when implementing a function, you should add a preprocessor directive to include the library in **vectorFunctions.cpp**.

`void printVector(int v[], size_t s)` - print the elements of a vector to `cout` using this general format: $[v_1, v_2, \ldots, v_3]$. For example, if

$$v = \begin{pmatrix} 1 \\ -2 \\ -3 \end{pmatrix}$$

then `printVector(v, 3)` outputs `[1, -2, -3]`

`printVector` should print out angular brackets (not preceded or followed by any spaces). `printVector` must not output any linebreaks. Also note that the final element should not have a comma after it.

`double magnitude(const int v[], size_t dim)` calculates the magnitude of vector v, as follows:

$$magnitude = \sqrt{(x_0^2 + \ldots + x_n^2)}$$

for a given vector

$$v^- = \begin{pmatrix} x_0 \\ \vdots \\ x_n \end{pmatrix}$$

where n is the size of the vector.

`int* addVectors(const int v1[], const int v2[], size_t dim)` - this function adds the corresponding indices of the two vectors `v1` and `v2` without changing the contents of v1 or v2. The function returns a pointer to the resulting vector. Note that the memory for the resulting vector should be allocated dynamically i.e. on the heap. v1, v2 as well as the resulting vector have the same size namely the value of `dim`.

`int* subtractVectors(const int v1[], const int v2[], size_t dim)` - this function subtracts the corresponding indices of the two vectors `v1` and `v2` without changing the contents of v1 or v2. The function returns a pointer to the resulting vector. Again the memory for the resulting vector should be allocated on the heap. v1, v2 as well as the resulting vector have the same size namely the value of `dim`.

`bool equalVectors(const int v1[], const int v2 [], size_t dim)` - returns `true` if two vectors have equal corresponding values at all indices, otherwise returns `false`.

`int dotProduct(const int v1[], const int v2[], size_t dim)` - returns the dot product of $v1$ and $v2$ notated as $v1 \bullet v2$ for two vectors with an equal size. This is calculated as the sum of product of corresponding elements. For example, if:

$$v1 = \begin{pmatrix} 5 \\ -2 \\ 3 \end{pmatrix} \quad v2 = \begin{pmatrix} 1 \\ -2 \\ -3 \end{pmatrix}$$

then $v1 \bullet v2 = (5 \times 1) + (-2 \times -2) + (3 \times -3) = 0$

Sample Output

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

```
Enter the dimension of the vectors: 4
Enter the elements of the vector p: 2 3 4 5
Enter the elements of the vector q: 4 -4 3 7
The sum of vectors p and q is [6, -1, 7, 12]
The difference of vectors q from p is [-2, 7, 1, -2]
The magnitude of vector p is 7.348
The vectors p and q are NOT EQUAL
The dot product of p and q is 43
```

Create a tarball containing all the source code of the system and upload your tarball using the upload slot called **Prac 5 - vectors**.

Task 2: Arrays of numbers

You are given a test program called **viewer.cpp**. It reads numbers from a file and stores the numbers in an array. It also records the actual number of values that was read from the file. Each number in the file is on its own line. We assume that the file contains a maximum of 200 numbers. The program can reverse, rotate, shift and display the arrays of numbers using the functions declared in an **h-file**. The program prompts the user for a file name.

You are given a **makefile**. It requires the same **h-file** that is used by **viewer.cpp**. It also requires a **cpp-file**. You should create these files. The **h-file** should be called **numberArray.h** and contain the prototypes of the functions listed on the next page. The **cpp-file** should be called **numberArray.cpp** and contain the implementations of these functions.

The following explains the functions that you should implement:

- `void reverseOrder(int[], size_t size)`
  The function takes the array and the size of the array as its parameters and reverses the order of elements in the array.

- `void rotate(int[], size_t size, int dist = 0)`
  The parameter **array** is the array to be rotated, **size** is the size of the array and **dist**, which defaults to 0, is the number of cells by which the values must rotate to the left or right depending on whether the value of dist is positive or negative.

- `bool shift(int array[], size_t size, int startInd, int stopInd , int dist = 0)`
  The function takes five parameters. The parameter **array** is the array that is to be shifted, **size** is the number of elements in the array, **startInd** is the index of the first cell that must be shifted, **stopInd** is the index of the last cell that must be shifted and **dist**, which defaults to 0, is the number of cells by which the values must be shifted to the left or right depending on whether the value of dist is positive or negative. The values of the specified cells are copied to the new position without changing the content of any other cells in the array. If the value of **startInd** is larger than the value of **stopInd**, the function should not perform any alterations and return false. Otherwise the function should perform the shift and return true. Note that when the shift is to a cell outside the bounds of the array, the shift is ignored – nothing is written outside the bounds of the array, but the function still returns true.

- `void displayArray(int [], size_t size)`
  The function takes an array and the size of the array as its arguments and prints out the elements of the array separated by commas and enclosed in square brackets. The output should not be preceded or followed by any spaces or line breaks.

You are given different input files to test your program. You should create more test files to test your functions for more cases.

Examples of various functions are listed below:

Input array	Function call	Resulting array	Return value
`[5,2,8,6,1,3]`	reverseOrder(arr,6)	[3,1,6,8,2,5]	
`[14,25,87,23,78]`	rotate(arr, 5, 2)	[23,78,14,25,87]	
`[14,25,87,23,78]`	rotate(arr, 5, -1)	[25,87,23,78,14]	
`[14,25,87,23,78]`	shift(arr, 5, 0, 1, 1)	[14,14,25,23,78]	true
`[5,2,8,32,6,1,3,12]`	shift(arr,8,4,6,-5)	[1,3,8,32,6,1,3,12]	true
`[5,2,8,32,6,1,3,12]`	shift(arr,8,6,4,-5)	[5,2,8,32,6,1,3,12]	false

Create a tarball containing all the source code. Upload your tarball using the upload slot called **Prac 5 - Arrays of numbers**.

Task 3: Name Sorting

You are given an **h-file** called **nameSort.h** containing the prototypes and description of the functions which must be implemented in a file called **nameSort.cpp**. The names should be sorted strictly alphabetical regardless of case. The names should, however, be stored in the original case as it was read from the file.

1. **sortSurnameFirst**: The names are sorted in ascending order of the surnames and then in ascending order of the first names. If there are two people with the same surname, they should appear in first names order. i.e. one with the lower order of first name must be listed first.
2. **sortFirstnameFirst**: The names are sorted in ascending order of the first names and then in ascending order of the surnames. If there are two people with the same first names, the one with the lower order of surname must be listed first.
3. **displayNames**: Display a heading, a line of dashes and then the names in the array in the order they are stored. Display each name and surname on its own line, surname first and then first name.

Create a tarball containing only the **nameSort.cpp** file. Upload it to the upload slot called **Prac 5 - Name Sorting Function**.

Main Program

Write a program be called **names.cpp** that reads the surnames and names of people from a file named **names.txt** and display them in a preferred sorted order. The program should read the names into parallel string arrays. You may assume that the file will not contain more than 200 names. For simplicity reasons you may assume that the names and surnames do not contain punctuation and that they are single words. The file contains the names and the surnames separated by a **tab**. The names in the file may be mixed case.

The program should display a menu to allow the user to select the preferred sorting order, sort the names in the selected order and display them using the functions you have implemented. The following are sample test runs of the program:

```
Names Sorting
-------------
1. Sort names by surnames then first names
2. Sort names by first names then surnames
Enter your sorting choice: 1
The sorted names by surnames then first names are:
Surnames            First Names
----------------------------------------
Adams               Grace
Adams               Hein
Adams               Samantha
Babajide            Pretty
chadwick            billy
deWet               Abel
Olivier             Vreda
Venter              Charlie
```

Sorting names by first names then surnames:

```
Names Sorting
-------------
1. Sort names by surnames then first names
2. Sort names by first names then surnames
Enter your sorting choice: 2
The sorted names by first names then surnames are:
Surnames            First names
---------------------------------------
deWet               Abel
chadwick            billy
Venter              Charlie
Adams               Grace
Adams               Hein
Babajide            Pretty
Adams               Samantha
Olivier             Vreda
```

Create a **makefile** to compile and run your system. Create a tarball containing all the source code. Upload your tarball using the upload slot called **Prac 5 - Name Sorting**.

Task 4: Pig Latin

An accountant was given a task to submit the regular report to his superior from another branch. The superior was concerned about their rival's competition and scared about the interception of these reports by the rivals during the transportation, hence he instructed the accountant to encrpyt the report using pig latin. Write a program that translates a message to Pig Latin.

The translate function

Implement the following overloaded versions of the function `translate`:

1. `string translate(const string& word)`: Translates a word to Pig Latin and returns the result. If the first letter is a consonant, it must be removed and placed at the end of the word. If it is a vowel, it should remain at the start of the word. In either case, append the string "OB" to the end of the word.
2. `string translate(int number)`: Returns a string of the form "X.Y", where X=number/61 (integer division) and Y=number%61.
3. `string translate(double number)`: Returns a string of the form "X.Y", where X=(integral part of number)/61 (integer division) and Y=(fraction part of number).

Declare these versions of the function in a header file called **translate.h**, and implement them in a file called **translate.cpp**. Create a tarball containing only these two files. Upload it to the upload slot called **Prac 5 - Pig Latin functions**.

Main Program

Write a program called **pigLatin.cpp** that reads a single line of input and translates it to Pig Latin. It should use the functions from the first part of this task.

Words and numbers in the input string may be separated by whitespace or punctuation. Leading and trailing whitespace in the input should be ignored. Multiple whitespace between words should be reduced to one space. All punctuation should remain in place when translating the message. A negative sign must be interpreted as punctuation, and not as part of a number.

The following is a sample test run of the program:

```
Enter a line of text below:
PLAGIARISM IS A CRIME, hence it is NOT ACCEPTABLE. 1357/4 = 339.25
--- The Pig Latin for the above line of text is given below:
LAGIARISMPOB ISOB AOB RIMECOB, encehOB itOB isOB OTNOB ACCEPTABLEOB.  22.15/0.4 = 5.25
```

Note: Unlike the usual convention to enter the input on the same line of the prompt, this program requires the user to enter the text below the prompt.

Create a tarball containing all the source files. Upload it to the upload slot called **Prac 5 - Pig Latin**.

Task 5: Cheque Writing

Write a program that displays a simulated paycheque. The program should ask the payee's name, and the amount of the cheque (up to R10,000) and then display the simulated cheque on the screen.

Functions

You are given a **h-file** called **chequeTools.h**. You are required to implement a set of functions described in the **h-file**. The requirements of the functions are included in the **h-file**.

Create a file called **chequeTools.cpp** that should contain the implementation of the functions specified in the **h-file**. If you need to create and implement more functions, you should declare and implement such helper functions in **chequeTools.cpp**. Typically the prototypes of the helper functions should be at the top of this **.cpp** file.

Create a tarball containing only the **chequeTools.cpp** file. Upload it to the upload slot called **Prac 5 - Cheque Writing Function**.

Main

The main program should use the given functions to display the simulated paycheque.

Obtain information about the routing number and account number as well as the cheque number from a text file called **chequeFacts.txt**. An example of such file is given. The program should declare variables to hold these values. The program should update the cheque number for the next cheque to be written and write it back to the file after obtaining the number. The cheque number should appear at the top of the cheque on the right as well as at the bottom. At the top only the sequence number is shown. At the bottom it is shown in full i.e. routing number, account number and cheque number separated with colons.

The program should prompt the user to enter his name and surname using a single prompt. Both the name and surname may contain spaces and punctuation. If the entered name contains special characters, they should be displayed appropraitely.

Format the numeric value of the cheque in fixed-point notation with two decimal places of precision. Be sure the decimal place always displays, even when the number is zero or has no fractional part.

It should display a simulated cheque with the Rand amount spelled out, exactly as shown in the following sample test run of the program:

```
Enter the name of the recipient: Jason Van
Enter the amount to be paid in rand: R 6725.85
************************************************************************ 00132 *
*************************** University of Pretoria ***************************
*******************************************************************************
*                                                                    2016-05-06 *
*  Pay to the Order of: Jason Van                                               *
*                                                                     R 6725.85 *
*  Six thousand seven hundred twenty five Rand and 85 cents                     *
*******************************************************************************
******************************** CHEQUE NUMBER ********************************
************************** 123456789:6255658688:00132 **************************
```

Input Validation: Do not accept negative Rand amounts, or amounts more than R10000.00

If an invalid value is enter the program should display an error message containing the unacceptable value and terminate immediately.

Create a tarball containing all the source code for the system as well as a makefile for compiling and running the system. Upload your tarball using the upload slot called **Prac 5 - Cheque Writing**.

Task 6: Rainfall Statistics

An automatic logging rain guage collects and records rainfall data and sends that data to a server. Rainfall statistics are computed and used for visualization tasks and in the seasonal prediction of rain and crop production. You are tasked with creating a module that takes the raw rainfall data from the server and populates a three-dimensional array with the data for further processing.

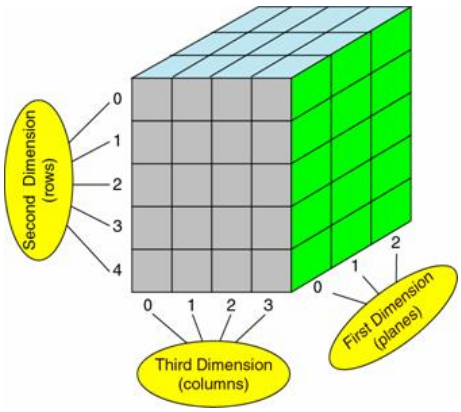
You are given a **makefile**, a main file named **rainfall.cpp**, a **h-file** called **arrayUtilities.h** as well as a data file called **raindata.txt**. You are required to implement a set of functions described in the **h-file**. The requirements of the functions are included in the **h-file**.

The given program should read the data from the given file. The following is a snippet of the data file illustrating its format:

```
1990/08/06-07:31    0
1990/08/07-07:32    0
1990/08/08-07:35    15
1990/08/09-07:34    8
1990/08/12-07:31    20
```

Each line contains a time stamp followed by a number. The number represents the number of millimeters rainfall measured for that day. The dates are in a sequential order i.e they follow in a logical order of occurence. Note, however, that data for some days may be missing. In the above example the data for 10 and 11 Aug 1990 is missing.

The data in the file must be stored in a 3D array of shorts (smallest data size). The following is a possible visualisation of the 3D array with three planes, five rows and four columns representing `array[Planes][Rows][Columns]`



In this program you will create a 3D array `rainData[year][month][day]` with a plane for each year, a row for each month and a column for each day. When populating the 3D array, the cells representing the days for which the data is missing should be set to **-1**.

Sample Output

The following is a sample test run of the program:

```
Enter exit any time to quit
Enter the date in the form YYYY/MM/DD to display the rainfall on that day: 1990/08/08
It rained 15mm on 1990/08/08
--------------------------------------------------------------------------------
Enter the date in the form YYYY/MM/DD to display the rainfall on that day: 1990/08/11
No data found for input "1990/08/11"
--------------------------------------------------------------------------------
Enter the date in the form YYYY/MM/DD to display the rainfall on that day: 1993/08/08
It rained 7mm on 1993/08/08
--------------------------------------------------------------------------------
Enter the date in the form YYYY/MM/DD to display the rainfall on that day: 1997/08/08
No data found for input "1997/08/08"
--------------------------------------------------------------------------------
Enter the date in the form YYYY/MM/DD to display the rainfall on that day: exit
--------------------------------------------------------------------------------
```

Create a tarball containing all the source code for the system. Upload your tarball using the upload slot called **Prac 5 - Rainfall Statistics**.