Department of Computer Science

COS284 Practical and Assignment 5: Floating Point and Arrays

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

1 Introduction

This document contains both practical 5 and assignment 5. In general the assignments will build upon the work of the current practical.

1.1 Submission

The Practical will be fitchfork marked but the tutors will be available for assistance during the practical sessions on 17 and 18 September. Upload your practical before submissions close at 23:59 on the 19th of September.

The Assignment will be fitchfork only and is due at 17:00 on Friday 28th of September. You may use the practical sessions to work on your assignment in the labs and ask for assistance if it is required.

1.2 Plagiarism policy

It is in your own interest that you, at all times, act responsible and ethically. As with any work done for the purpose of your university degree, remember that the University of Pretoria will not tolerate plagiarism. Do not copy a friend's work or allow a friend to copy yours. Doing so constitutes plagiarism, and apart from not gaining the experience intended, you may face disciplinary action as a result.

For more on the University of Pretoria's plagiarism policy, you may visit the following webpage: `http://www.library.up.ac.za/plagiarism/index.htm`

1.3 Practical component [25%]

There are 3 tasks for this practical and it will be fitchfork marked only.

1.3.1 Task 1: Stirlings Approximation of the Gamma Function [5]

For this task you must create an assembler program that implements a function called st_gamma which does the following:

- Receives a single float parameter.
- Calculates
$$n! \approx \sqrt{2\pi n} \times (\frac{n}{e})^n$$
where n is the parameter given to the function.

Your function must simply return the approximation of $n!$. For instance, st_gamma(5.5) should return 283.5609.

Notes:

- You should use the following approximations for π and e:
$$\pi = 3.1415926536$$
$$e = 2.7182818285$$
- You should also use the C function powf by declaring **extern** powf in your assembly file. However, you must use sqrtss code in assembly for the square root calculations.

An example of an implementation in C Code:

```
float st_gamma(float n) {
        return sqrt(2 * M_PI * n) * powf(n / M_E, n);
}
```

For more information about Stirling's approximation please visit the following website:
`https://en.wikipedia.org/wiki/Stirling%27s_approximation`

When you are finished, upload your source code file named **task1.asm** to the assignments.cs.up.ac.za website, under the **Practical 5 Task 1** submission slot.

1.3.2 Task 2: Sample Standard Deviation [10]

For this task you must calculate the sample standard deviation of a list of floating point numbers. You must implement the following function in assembly:

- Create a function called stdev that takes for its first parameter an array of floats. And the second parameter must take an int that represents the size of the array in the first parameter.

- Your assembly program must then calculate the sample standard deviation using the following definition:

$$s = \sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} (x_i - \mu)^2}$$

where s is the sample standard deviation that should be returned, N is the number of items (the 2^{nd} parameter value), μ is the average of the array (the 1^{st} parameter) and x_i is the i^{th} item in that array.

For more information about the standard deviation visit the following site:
`https://www.mathsisfun.com/data/standard-deviation-formulas.html`

When you are finished, upload your source code file named **task2.asm** to the assignments.cs.up.ac.za website, under the **Practical 5 Task 2** submission slot.

1.3.3 Task 3: Taylor Series Approximation for e^x [10]

In this task you will be approximating the value for e^x. To do this you must implement an assembly program that has the function taylor_ex which must do the following:

- Receive a float parameter x and an int parameter n.
- Calculate and return the following:

$$\sum_{i=0}^{n} \frac{x^i}{i!}$$

For example if the call taylor_ex (3.4, 3) is made you must calculate and return the following:

$$e^{3.4} \approx \frac{3.4^0}{0!} + \frac{3.4^1}{1!} + \frac{3.4^2}{2!} + \frac{3.4^3}{3!} = 15.7307$$

For more information on the taylor series visit the following site:
`http://tutorial.math.lamar.edu/Classes/CalcII/TaylorSeries.aspx`

When you are finished, upload your source code file named **task3.asm** to the assignments.cs.up.ac.za website, under the **Practical 5 Task 3** submission slot.

1.4 Assignment component [75%]

1.4.1 Linear Equation Solver [75]

For this assignment, you will be required to implement a function that solves a system of linear equations. You must implement a function that will take an array of floats in the first parameter and the number of variables (n) to solve for in the second parameter. You may assume that you will always be given n equations. You may also assume that any input will also give a distinct answer. Your implementation must do the following:

- The function has to be called solve_linear_eq.
- The function must solve the system of equations. You may want to use the method of converting a matrix into its reduced row echelon form.
- You must then return the values that solve the system in an array of floats (Hint: call malloc)

For example given the following system of linear equations where x_{mn}, y_m are constants and variables $a, b \in \mathbb{R}$:

$$ax_{11} + bx_{12} = y_1$$
$$ax_{21} + bx_{22} = y_2$$

You must find the values for a and b to make the above equations valid. You may also think about the equations in the form of a matrix equation:

$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

The above values will be translated into the code below as input:

```
float *values[6];
values = {
        x11, x12, y1,
        x21, x22, y2
};
 // result contains the values for a and b as {a, b}
float *result = solve_linear_eq(values, 2);
```

Your function should also cater for higher orders such as 3, 4 and above. Subsequently follows a system that requires you to find 3 variables:

$$ax_{11} + bx_{12} + cx_{13} = y_1$$
$$ax_{21} + bx_{22} + cx_{23} = y_2$$
$$ax_{31} + bx_{32} + cx_{33} = y_3$$

```
float *values[12];
values = {
        x11, x12, x13, y1,
        x21, x22, x23, y2,
        x31, x32, x33, y3
};
// result contains the values for a, b and c as {a, b, c}
float *result = solve_linear_eq(values, 3);
```

Another example using 4 equations and 4 variables:

$$ax_{11} + bx_{12} + cx_{13} + dx_{14} = y_1$$
$$ax_{21} + bx_{22} + cx_{23} + dx_{24} = y_2$$
$$ax_{31} + bx_{32} + cx_{33} + dx_{34} = y_3$$
$$ax_{41} + bx_{42} + cx_{43} + dx_{44} = y_4$$

```
float *values[20];
values = {
        x11, x12, x13, x14, y1,
        x21, x22, x23, x24, y2,
        x31, x32, x33, x34, y3,
        x41, x42, x43, x44, y4,
};
// result contains the values for a, b, c and d as {a, b, c, d}
float *result = solve_linear_eq(values, 4);
```

A step by step procedure of how one may go about solving this problem follows:

- Given
$$2a + 3b = 5$$
$$2a + 4b = 5$$
- Convert to matrix form
$$\begin{bmatrix} 2 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$
- Convert to reduced row echelon form using a couple rules. You may subtract all the values of a row from another row (the subtracting row may be scaled by a factor). You may scale all the values in one row by a single constant. Both these rules are applied below demonstrating what you are allowed to do.

$$\begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix} Row_2 = Row_2 - Row_1$$

$$\begin{bmatrix} 1 & 1.5 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2.5 \\ 0 \end{bmatrix} Row_1 = \frac{1}{2} \times Row_1$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 2.5 \\ 0 \end{bmatrix} Row_1 = Row_1 - \frac{3}{2} \times Row_2$$

- Convert back to linear equations

$$1a + 0b = 2.5$$

$$0a + 1b = 0$$

- Solve for a and b

$$a = 2.5$$

$$b = 0$$

For more information on row echelon form and reduced row echelon form:
`https://en.wikipedia.org/wiki/Row_echelon_form`

When you are finished, upload your source code file named **equ.asm** to the assignments.cs.up.ac.za website, under the **Assignment 5** submission slot.

2 Mark Distribution

Activity	**Mark**
Prac Task 1	5
Prac Task 2	10
Prac Task 3	10
Assignment	75
Total	**100**