Department of Computer Science

COS132 2015
Practical 5

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Mr Madoda Nxumalo

0 Introduction

0.1 Uploads and Deadline

This practical is a three(5) in-lab practical, and as such students must write it at the Informatorium during the normal practical session hours. The uploads will become available at practical session hours as of **17 April - 23 April**. HONOS will be the interface to upload your tasks.

Make sure that you have submitted your code to Fitchfork all your tasks at the end of the practical session. **No late submissions will be accepted**.

0.2 Plagiarism Policy

All submissions for this 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.

1 Polynomials

You have been introduced to polynomials at some previous levels of mathematics. At this level you must be familiar with basic operations on polynomials. They can be added, subtracted, multiplied by a scalar and multiply each other. For this assignment we will limit the multiplication of polynomials with one another to polynomials of order 1.

Polynomials expressions examples are shown below.

$p(x) = x$
$q(y) = y^2 + 3y - 4$

In this in-lab practical session we want to implement a set of functions for polynomials with integer coefficients. We store the coefficient of the polynomial in an array, and the array size is initialized based on the highest order of the polynomial expression (also known as the degree of the polynomial). The degree of a polynomial is the highest power of x that occurs in the polynomial's terms. Here are examples in Table 1.

equation	array Index 0	array Index 1	array Index 2	array Index 3	degree
$q(x) = x^2 + 3x - 4$	-4	3	1		2
$p(x) = x$	0	1			1
$r(x) = 9x^3 + 5x^2 + 6$	6	0	5	9	3

Table 1: Three polynomials represented as arrays (with coefficients as array elements)

1.1 Functions

1. **int** evaluate(**int** p[], **size_t** size, **int** x)
   This function returns the integer value of the polynomial. e.g. if the polynomial is $x^2 - 2x + 4$, the following table shows the values that should be returned for some values passed in x:

x	Calculation	Return
0	$0 - 0 + 4$	4
2	$9 - 6 + 4$	7

2. **void** multiplyPoly(**const int** p[], **size_t** s1, **int** scalar, **int** r[])
   This function multiplies the coefficients of polynomial p (of order s1) by an integer scalar. r[] is the resulting polynomial. It will have the same order as p1.

3. **void** addPoly(**const int** p[], **size_t** s1, **const int** q[], **size_t** s2, **int** r[], **size_t** s3)
   The function calculates the sum of two polynomials p and q, and stores the sum in r. The corresponding indexes of p and q are added. s1, s2 and s3 are respective orders of p, q and r. Note that polynomials, p and q, may not have same order (i.e. the highest power of x), and sum r, have the same order as the highest order of the two given polynomials. For example, adding $p(x) = x + 1$ and $q(x) = x^2 + x + 2$, will result to $r(x) = x^2 + 2x + 3$.

4. **void** subtractPoly(**int** p[], **size_t** s1, **int** q[], **size_t** s2, **int** r[], **size_t** s3)

   The function subtracts polynomial q from polynomial p, corresponding indexes of the p and q arrays are subtracted. The array r will contains the result of the subtraction and its order must be equal to that of the highest order between q and p.

5. **void** multiplyPoly(**const int** p[], **size_t** s1, **const int** q[], **size_t** s2, **int** r[], **size_t** s3)
   This function multiply a polynomial p of order 1 by another polynomial p1 of order 1. r[] is the resulting polynomial. It will have an order of 2. To perform multiplication each term of p[] is multiplied with each term in q[]. The following table shows a few examples:

Problem	**Calculation**	**Resulting polynomial**
$(x+3)\times(x+1)$	$x\times x+3x+1x+3\times 1$	x^2+4x+3
$(2x+1)\times(5x-3)$	$2x\times 5x-6x+5x-3\times 1$	$10x^2-x-3$

6. printPoly(**int** p1[], **size_t** s1)
   The function prints a polynomial p1 in a single line in this general format;
   axˆn + bxˆ(n-1) + . . . +yxˆ1 + zxˆ0
   Here n is the highest order of the polynomial. An example; $r(x)=x^2-2x+3$ will be printed as **1xˆ2-2xˆ1+3xˆ0**.

Note that the mathematical expression is always written with the highest order term first. However, it is expected that the array should store the coefficient of the term in the index position corresponding with the order – i.e. the coefficient of the lowest order term first.

1.2 Sample Output

The user is expected to state the highest order of a polynomial and then input the coefficients of the polynomial. Here is a sample output and all input values are written in bold-font.

1.2.1 Test run

```
Polynomial Calculator:  Operations
1.  Value
2.  Scalar multiply
3.  Add Polynomials
4.  Subtract polynomials
5.  Multiply polynomials
6.  Print
Enter a number representing a function to test:  1
-----------------------------------------------
```

```
Enter the highest order of the first polynomial, p(x):  3
Enter the coefficients of the first polynomial, p(x):  2 3 4 5
Enter an x value for p(x):  3
The value of p(3) is:  182
```

1.2.2 Test run

```
Polynomial Calculator:  Operations
1.  Value
2.  Scalar multiply
3.  Add Polynomials
4.  Subtract polynomials
5.  Multiply
6.  Print
Enter a number representing a function to test:  2
-----------------------------------------------
Enter the highest order of the polynomial, p(x):  3
Enter the coefficients of the polynomial, p(x):  2 3 4 5
Enter the constant multiplyer (scalar):  2
The scalar multiplication of p(x) by 2 have elements:  4 6 8 10
```

1.2.3 Test run

```
Polynomial Calculator:  Operations
1.  Value
2.  Scalar multiply
3.  Add Polynomials
4.  Subtract polynomials
5.  Multiply
6.  Print
Enter a number representing a function to test:  3
-----------------------------------------------
Enter the highest order of the first polynomial, p(x):  3
Enter the coefficients of the first polynomial, p(x):  2 3 4 5
Enter the highest order of the second polynomial, q(x):  2
Enter the coefficients of the second polynomial, q(x):  4 -6 9
The sum of p(x) and q(x) have these elements:  6 -3 13 5
```

1.2.4 Test run

```
Polynomial Calculator:  Operations
1.  Value
2.  Scalar multiply
3.  Add Polynomials
4.  Subtract polynomials
5.  Multiply
```

```
6.  Print
Enter a number representing a function to test:  4
-----------------------------------------------
Enter the highest order of the first polynomial, p(x):  3
Enter the coefficients of the first polynomial, p(x):  2 3 4 5
Enter the highest order of the second polynomial, q(x):  2
Enter the coefficients of the second polynomial, q(x):  4 -6 9
The difference of p(x) and q(x) have these elements:  -2 9 -5 5
```

1.2.5 Test run

```
Polynomial Calculator:  Operations
1.  Value
2.  Scalar multiply
3.  Add Polynomials
4.  Subtract polynomials
5.  Multiply
6.  Print
Enter a number representing a function to test:  6
-----------------------------------------------
Enter the highest order of the polynomial, p(x):  3
Enter the coefficients of the polynomial, p(x):  -2 -3 4 5
The printed polynomial is:  5x^3+4x^2-3x^1-2x^0
```

1.2.6 Comments

- This given test suite does not have full test coverage. You are encouraged to test your program with more test cases.
- **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] = ...;
  ```

1.3 Submission

You have to upload the following files:

- **main.cpp** - You are given an incomplete version of this program. You have to complete the program by replacing the comments with code.

- **polynomials.h** - This file is given. It contains the prototypes of the functions. You should not change this file.
- **polynomials.cpp** - You have to create this file. It should contain the implementations of the functions.
- **makefile** - you are provided with a makefile containing all the needed commands.

The tarball containing the above mentioned files can be created by the following command.

```
tar -cvz polynomials.h polynomials.cpp main.cpp makefile -f
                    polynomials.tar.gz
```

1.4 Marks Allocation

Task	**Percentage**	**Maximum mark**
evaluate	17.5%	7
Scalar multiply	15%	6
Add Polynomials	20%	8
Subtract polynomials	17.5%	7
Multiply Polynomial	15%	6
Print Polynomial	15%	6
Total	**100**	**40**