Assignment 2

CS 1037
Computer Science Fundamentals II

Due date: October 10, 2024, at 23:59

Learning Outcomes

Upon completion of this assignment, students will be able to:

- Apply pointer arithmetic and dynamic memory allocation in C
- Implement arrays for polynomial, vector, and matrix operations
- Develop custom string manipulation functions without relying on standard libraries
- Solve mathematical problems using C programming techniques

Introduction

This assignment focuses on developing your skills in C programming by working with pointers, arrays, and algorithm implementation. You will create functions for polynomial operations, vector and matrix manipulations, and string handling. This assignment will enhance your problem-solving abilities and provide hands-on experience with fundamental C programming principles.

Question 1 – Polynomial Operations (6 points)

Implement functions to perform operations on polynomials, including evaluation, derivative calculation, and root finding using Newton's method.

Files Provided

- **polynomial.h**: Header file containing function declarations
- **polynomial_ptest.c**: Main function for testing your implementation

Implementation Details

Create a C file named polynomial.c that implements the following functions:

1. `float horner(float *p, int n, float x)`

   - Compute and return the value of an (n-1)-th degree polynomial using Horner's algorithm.
   - The polynomial is of the form:

   $$P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0$$

   More about Horner's algorithm can be found in Appendix A.

2. `void derivative(float *p, float *d, int n)`

   - Compute the derivative of an input (n-1)-th degree polynomial.
   - Output the coefficients of the derivative polynomial in the array pointed to by `d`.

Refer to **polynomial.h** for detailed function specifications.

Compilation

```
gcc -o q1 polynomial.c polynomial_ptest.c
./q1
```

Testing

Test your implementation using the provided **polynomial_ptest.c** file. Compile and run your program using appropriate terminal commands. The output should match the expected format as shown in the public test results.

Question 2 – Vector and Matrix Operations (7 points)

Implement functions to perform operations on vectors and matrices represented as arrays.

Files Provided

- **matrix.h**: Header file containing function declarations
- **matrix_ptest.c**: Main function for testing your implementation

Implementation Details

Create a C file named matrix.c that implements the following functions:

1. `float norm(float *v, int n)`
   - Compute and return the norm of a vector.
2. `float dot_product(float *v1, float *v2, int n)`
   - Compute and return the dot product of two vectors.
3. `void matrix_multiply_vector(float *m, float *v, float *vout, int n)`
   - Compute the multiplication of a matrix and a vector.

Refer to matrix.h for detailed function specifications.

Compilation

```
gcc -o q2 matrix.c matrix_ptest.c
./q2
```

Testing

Test your implementation using the provided **matrix_ptest.c** file. Compile and run your program using appropriate terminal commands. The output should match the expected format as shown in the public test results.

Question 3 – String Manipulation (7 points)

Implement custom string manipulation functions without using any existing library functions.

Assignment 2

CS 1037
Computer Science Fundamentals II

Files Provided

- **mystring.h**: Header file containing function declarations
- **mystring_ptest.c**: Main function for testing your implementation

Implementation Details

Create a C file named **mystring.c** that implements the following functions declared in **mystring.h**:

1. `int str_wc(char *s)`
   - Count the number of words in a given simple string.
2. `int str_lower(char *s)`
   - Change every uppercase English letter to its lowercase equivalent.
3. `void str_trim(char *s)`
   - Remove unnecessary space characters in a simple string.

Implementation Requirements

- It is not allowed to use other library functions.
- Implement all string manipulations manually.
- Ensure proper handling of edge cases and null-terminated strings.
- Use pointer arithmetic for efficient string traversal and manipulation.

Compilation

```
gcc -o q3 mystring.c mystring_ptest.c

./q3
```

Testing

Test your implementation using the provided **mystring_ptest.c** file. Compile and run your program using appropriate terminal commands. The output should match the expected format as shown in the public test results.

Submission Instructions

- Submit your completed C files (**polynomial.c, matrix.c,** and **mystring.c**) through the designated course submission system
- Do not submit any other files, including header files or executables
- Ensure your code compiles and runs without errors on the university lab computers

Grading Criteria

Total: 40 points

Question 1: Polynomial Operations (6 points)

1. horner function implementation: 3 points
2. derivative function implementation: 3 points

Question 2: Vector and Matrix Operations (7 points)

1. norm function implementation: 2 points
2. dot_product function implementation: 2 points
3. matrix_multiply_vector function implementation: 3 points

Question 3: String Manipulation (7 points)

1. str_wc function implementation: 2 points
2. str_lower function implementation: 2 points
3. str_trim function implementation: 3 points

Code Quality (5 points)

1. Proper use of C programming conventions and best practices: 2 points
2. Code organization and readability: 2 points
3. Appropriate comments and documentation: 1 point

Correctness and Efficiency (5 points)

1. Code compiles without errors: 1 point
2. Passes all provided test cases: 2 points
3. Efficient implementation (e.g., proper use of pointers, avoiding unnecessary computations): 2 points

This grading scheme totals 30 points.

This grading scheme will be used to evaluate your submission. Ensure that your implementation addresses all aspects outlined above to maximize your score.

Submission Instructions (due date: Oct 10 at 11:59 pm)

Assignments must be submitted through OWL Brightspace.

Rules

- Please only submit the files specified below.
- Do not attach other files, even if they were part of the assignment.
- Do not upload the .exe files! Penalties will be applied for this.
- Submit the assignment on time. Late submission policy will be applied.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through OWL Brightspace.
- If your code runs on your IDE but not on the university lab computers, you will NOT get the marks!
- Assignment files must NOT be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit the code as many times as you wish. However, re-submissions after the assignment deadline will receive a late penalty.

Files to submit

Remember **you must do** all the work on your own. **Do not copy** or even look at the work of another student. All submitted code will be run through similarity-detection software.

This grading criteria will be used to evaluate your submission. To maximize your score, ensure that your implementation addresses all aspects outlined above.

Compilation and Output for Question 1

```
compile command: gcc -o q1 polynomial.c polynomial_ptest.c
test run command: q1

------------------
Test: horner

p(x): 1.00*x^3+2.00*x^2+3.00*x^1+4.00
horner(p 0.00): 4.00
horner(p 1.00): 10.00
horner(p 10.00): 1234.00

------------------
Test: derivative

p'(x): 3.00*x^2+4.00*x^1+3.00
```

Compilation and Output for Question 2

```
compile command: gcc matrix.c matrix_ptest.c -o q2
test run command: q2

------------------
Test: norm

v1:
1.0
2.0
2.0

norm(v1): 3.0
------------------
Test: dot_product

v1:
1.0
1.0
1.0

v2:
1.0
2.0
```

```
3.0

dot_product(v1 v2): 6.0
------------------
Test: matrix_multiply_vector

m:
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0

v:
1.0
1.0
1.0

matrix_multiply_vector(m v v1)
v1:
6.0
15.0
24.0
```

Compilation and Output for Question 3

```
compile command: gcc mystring.c mystring_ptest.c -o q3
test run command: q3

------------------
Test: str_wc

str_wc('Abc  DEF'): 2
str_wc('  aBc   Def   '): 2
str_wc('  Toonie is the Canadian $2 coin. '): 5
str_wc('  Binary has 10 digits.  '): 3

------------------
Test: str_lower

str_lower('Abc  DEF'): 'abc  def'
str_lower('  aBc   Def   '): '  abc   def   '
str_lower('  Toonie is the Canadian $2 coin. '): '  toonie is the canadian $2
coin. '
str_lower('  Binary has 10 digits.  '): '  binary has 10 digits.  '

------------------
Test: str_trim

str_trim('Abc  DEF'): 'Abc DEF'
str_trim('  aBc   Def   '): 'aBc Def'
str_trim('  Toonie is the Canadian $2 coin. '): 'Toonie is the Canadian $2
coin.'
str_trim('  Binary has 10 digits.  '): 'Binary has 10 digits.'
```

Good Luck ☺

Appendix A

Horner's algorithm is an efficient method for evaluating polynomials that reduce the number of multiplications required. It transforms a polynomial from its standard form to a nested form.

The Polynomial Form:
For an $(n-1)$-th degree polynomial of the form:

$$P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0$$

Horner's Algorithm Representation:
Using Horner's algorithm, the polynomial can be rewritten in a nested manner as:

$$P(x) = (\ldots((a_{n-1}x + a_{n-2})x + a_{n-3})x + \cdots + a_1)x + a_0$$

Steps of Horner's Algorithm:

1. Start with the leading coefficient a_{n-1} as the initial value.
2. Multiply the current value by xxx and add the next coefficient.
3. Repeat this process until all coefficients are processed.

This method minimizes the number of multiplications and additions needed, making it computationally efficient.