UCSD 167113 Data Structure And Algorithms in C/C++
HW2
Cheng FEI

Source Code:

```
//
//  main.c
//  polynomials
//
//  Created by Cheng FEI on 2022/10/2.
//

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "list.h"

/*
 Interface of polynomial functions.
 */
// Part (a)
void appendTerm(List *pPolynomial, double constant);
// Part (b)
void display(List *pPolynomial);
// Part (c)
double evaluate(List *pPolynomial, double x);

/*
 Interface of test function.
 */
void test(double *arr, int number, double variable);

/*
 Main function.
 */
int main(int argc, const char * argv[]) {
    // Test case 1: x + 1.0
    printf("Test case 1:\n\n");
    printf("Expected coefficient of polynomial: 1.0, 1.0\n");
    printf("Expected output: x + 1.0\n");
    printf("Expected value of polynomial: 2.0\n\n");
    double arr1[] = {1.0, 1.0};
    test(arr1, 2, 1.0);

    // Test case 2: x^2 - 1.0
    printf("Test case 2:\n\n");
    printf("Expected coefficient of polynomial: 1.0, 0.0, -1.0\n");
    printf("Expected output: x^2 - 1.0\n");
    printf("Expected value of polynomial: 3.1209\n\n");
    double arr2[] = {1.0, 0.0, -1.0};
    test(arr2, 3, 2.03);

    // Test case 3: -3.0x^3 + 0.5x^2 - 2.0x
    printf("Test case 3:\n\n");
    printf("Expected coefficient of polynomial: -3.0, 0.5, -2.0, 0.0\n");
    printf("Expected output: -3.0x^3 + 0.5x^2 - 2.0x\n");
    printf("Expected value of polynomial: -372.5\n\n");
    double arr3[] = {-3.0, 0.5, -2.0, 0.0};
    test(arr3, 4, 5.0);

    // Test case 4: -0.3125x^4 - 9.915x^2 - 7.75x - 40.0
    printf("Test case 4:\n\n");
    printf("Expected coefficient of polynomial: -0.3125, 0.0, -9.915, -7.75, -40.0\n");
    printf("Expected output: -0.3125x^4 - 9.915x^2 - 7.75x - 40.0\n");
    printf("Expected value of polynomial: -72731671.7\n\n");
    double arr4[] = {-0.3125, 0.0, -9.915, -7.75, -40.0};
    test(arr4, 5, 123.45);
```

```
    return EXIT_SUCCESS;
}

/*
 Implementation of polynomial functions.
 */
/*
 Part (a)
 Add new term with new double number to the tail of polynomial.
 Params: pPolynomial -- pointer to List object into which new constant will be inserted.
 Params: constant -- double number representing the coefficient of the new term.
 Returns: nothing.
 Updates: update pPolynomial by adding a new ListElmt object with value constant to its tail.
 */
void appendTerm(List *pPolynomial, double constant) {
    // Declare a new pointer to double object.
    double *newConstant;
    // Allocate dynamic memory for new pointer to double.
    if ((newConstant = (double *) malloc(sizeof(double))) == NULL) {
        // If out of memory, exit program.
        fprintf(stderr, "Out of memory!");
        exit(EXIT_FAILURE);
    }
    // If not out of memory, assign new value to newConstant.
    *newConstant = constant;
    // Insert newConstant to polynomial linked list.
    if (list_ins_next(pPolynomial, list_tail(pPolynomial), newConstant) != 0) {
        // If insertion fails, exit program.
        fprintf(stderr, "Append term fails, exiting!");
        exit(EXIT_FAILURE);
    }
}

/*
 Part (b)
 Display the linked list in the format of polynomial.
 Params: pPolynomial -- a pointer of List object which is displayed.
 Returns: nothing, but print out the polynomial to console.
 */
void display(List *pPolynomial) {
    // Initialize a pointer to list element.
    ListElmt *pElmt;
    pElmt = list_head(pPolynomial);
    // Get order of the polynomial.
    int order = list_size(pPolynomial) - 1;
    int curOrder = order;
    // Initialize a mark.
    // If 0, don't print operands.
    int printOperand = 0;

    // Loop through coefficients of polynomial's terms.
    while (pElmt != NULL) {
        // Format the output based on different conditions.
        // Get the coefficient of current term.
        double coefficient = *(double *) list_data(pElmt);
        // If coefficient is 0, skip current term.
        if (coefficient != 0.0) {
            // First part: operand +/-.
            if (printOperand) {
                if (coefficient > 0) printf(" + ");
                else printf(" - ");
            }
            // Print coefficient.
            // If the coefficient is 1.0 or -1.0, don't print coefficient, except the constant term.
            if (curOrder == 0 || fabs(coefficient) != 1.0) {
                // If current order does not belong to the first term to print.
                if (printOperand) {
                    if (coefficient == (int) coefficient) {
                        printf("%.1f", fabs(coefficient));
                    }
                    else {
                        printf("%.6g", fabs(coefficient));
```

```
                    }
                }
                // If current order belongs to the first term to print.
                else {
                    if (coefficient == (int) coefficient) {
                        printf("%.1f", coefficient);
                    }
                    else {
                        printf("%.6g", coefficient);
                    }
                }
            }
            // Print the term.
            if (curOrder == 0) printf("");
            else if (curOrder == 1) printf("x");
            else printf("x^%d", curOrder);
            // Update printOperand.
            printOperand = 1;
        }

        // Update pointer to ListElmt.
        pElmt = list_next(pElmt);
        // Update current order.
        curOrder--;
    }
}

/*
 Part (c)
 Evaluate value of polynomial with entered double number x.
 Params: pPolynomial -- a pointer to List object which is evaluated.
 Params: x -- double number to calculate polynomial's value.
 Returns: result -- double number: value of the polynomial with x.
 */
double evaluate(List *pPolynomial, double x) {
    // Initialize double number result.
    double result = 0.0;
    // Initialize a pointer to list element.
    ListElmt *pElmt = list_head(pPolynomial);
    // Get higest order of the polynomial.
    int order = list_size(pPolynomial) - 1;
    int curOrder = order;
    // Loop through the polynomial's all terms.
    while (pElmt != NULL) {
        // Get current coefficient.
        double coefficient = *(double *) list_data(pElmt);
        // Update computed result.
        result += coefficient * pow(x, curOrder);
        pElmt = list_next(pElmt);
        curOrder--;
    }
    return result;
}

/*
 Implementation of test function.
 */
void test(double *arr, int number, double variable) {
    // Initialize a polynomial linked list.
    List polynomial;
    list_init(&polynomial, free);

    printf("Creating polynomial: \n");
    // Append constants to polynomial.
    for (int i = 0; i < number; i++) appendTerm(&polynomial, arr[i]);
    printf("Actual coefficient of polynomial: ");
    ListElmt *pElmt = list_head(&polynomial);
    while (pElmt != NULL) {
        printf("%g ", *(double *) list_data(pElmt));
        pElmt = list_next(pElmt);
    }
    printf("\n");
```

```
    // Display polynomial.
    printf("Displaying polynomial: \n");
    printf("Actual output: ");
    display(&polynomial);
    printf("\n");

    printf("Evaluating polynomial: \n");
    // Evaluate the polynomial with variable.
    double result = evaluate(&polynomial, variable);
    // Display value of the polynomial with entered variable.
    // Format the output based on different conditions.
    printf("Actual value of polynomial: ");
    display(&polynomial);
    if (variable == (int) variable) {
        printf(" with x = %.1f", variable);
    }
    else {
        printf(" with x = %g", variable);
    }
    if (result == (int) result) {
        printf(" equals %.1f", result);
    }
    else printf(" equals %g", result);

    // Destroy the polynomial with all dynamically allocated memory for its properties.
    list_destroy(&polynomial);

    printf("\n-----------------------------------\n\n");
}
```

Program Output:

Test case 1:

```
Expected coefficient of polynomial: 1.0, 1.0
Expected output: x + 1.0
Expected value of polynomial: 2.0

Creating polynomial:
Actual coefficient of polynomial: 1 1
Displaying polynomial:
Actual output: x + 1.0
Evaluating polynomial:
Actual value of polynomial: x + 1.0 with x = 1.0 equals 2.0
-----------------------------------
```

Test case 2:

```
Expected coefficient of polynomial: 1.0, 0.0, -1.0
Expected output: x^2 - 1.0
Expected value of polynomial: 3.1209

Creating polynomial:
Actual coefficient of polynomial: 1 0 -1
Displaying polynomial:
Actual output: x^2 - 1.0
Evaluating polynomial:
Actual value of polynomial: x^2 - 1.0 with x = 2.03 equals 3.1209
-----------------------------------
```

Test case 3:

```
Expected coefficient of polynomial: -3.0, 0.5, -2.0, 0.0
Expected output: -3.0x^3 + 0.5x^2 - 2.0x
Expected value of polynomial: -372.5

Creating polynomial:
Actual coefficient of polynomial: -3 0.5 -2 0
Displaying polynomial:
Actual output: -3.0x^3 + 0.5x^2 - 2.0x
```

```
Evaluating polynomial:
Actual value of polynomial: -3.0x^3 + 0.5x^2 - 2.0x with x = 5.0 equals -372.5
-------------------------------------

Test case 4:

Expected coefficient of polynomial: -0.3125, 0.0, -9.915, -7.75, -40.0
Expected output: -0.3125x^4 - 9.915x^2 - 7.75x - 40.0
Expected value of polynomial: -72731671.7

Creating polynomial:
Actual coefficient of polynomial: -0.3125 0 -9.915 -7.75 -40
Displaying polynomial:
Actual output: -0.3125x^4 - 9.915x^2 - 7.75x - 40.0
Evaluating polynomial:
Actual value of polynomial: -0.3125x^4 - 9.915x^2 - 7.75x - 40.0 with x = 123.45 equals -7.27317e+07
-------------------------------------

Program ended with exit code: 0
```