a)

```
void appendTerm(List *pPolynomial, double constant) {
    double *data = (double *)malloc(sizeof(double));
    if (data == NULL) {
        printf("Error: Failed to allocate memory for new term.\n");
        exit(EXIT_FAILURE);
    }
    *data = constant;
    if (list_ins_next(pPolynomial, list_tail(pPolynomial), data) != 0) {
        printf("Error: Failed to append new term.\n");
        exit(EXIT_FAILURE);
    }
}
```

b)

```
void display(List *pPolynomial) {
    int degree = list_size(pPolynomial) - 1;
    ListElmt *current = list_head(pPolynomial);
    int isFirstTerm = 1;  // A flag to check if it's the first term being printed

    while(current) {
        double coefficient = *(double *)list_data(current);
        if(coefficient) {
            if (!isFirstTerm) {
                if (coefficient > 0) printf(" + ");
                else if (coefficient < 0) printf(" - ");
            } else {
                if (coefficient < 0) printf("-");
                isFirstTerm = 0;  // reset the flag once the first term is
processed
            }

            if (fabs(coefficient) != 1.0 || degree == 0) {
                printf("%.1lf", fabs(coefficient));
            }

            if (degree > 1) printf("x^%d", degree);
            else if (degree == 1) printf("x");
        }
        degree--;
        current = list_next(current);
    }
    printf("\n");
}
```

c)

```
double evaluate(List *pPolynomial, double x) {
    double result = 0.0;
    int degree = list_size(pPolynomial) - 1;
    ListElmt *current = list_head(pPolynomial);

    while(current) {
        double coefficient = *(double *)list_data(current);
        result += coefficient * pow(x, degree);
        degree--;
        current = list_next(current);
    }

    return result;
}
```

d)

```
int main() {
    List polynomial;

    // Test 1: x + 1.0 with x = 1.0
    list_init(&polynomial, free);
    appendTerm(&polynomial, 1.0);
    appendTerm(&polynomial, 1.0);
    display(&polynomial);
    printf("Evaluation: %.2f\n", evaluate(&polynomial, 1.0));
    list_destroy(&polynomial);

    // Test 2: x^2 - 1.0 with x = 2.03
    list_init(&polynomial, free);
    appendTerm(&polynomial, 1.0);
    appendTerm(&polynomial, 0.0);
    appendTerm(&polynomial, -1.0);
    display(&polynomial);
    printf("Result: %g\n", evaluate(&polynomial, 2.03));
    list_destroy(&polynomial);
    printf("\n");

    // Test 3: -3.0x^3 + 0.5x^2 - 2.0x with x = 5.0
    list_init(&polynomial, free);
    appendTerm(&polynomial, -3.0);
    appendTerm(&polynomial, 0.5);
```

```
    appendTerm(&polynomial, -2.0);
    appendTerm(&polynomial, 0.0);
    display(&polynomial);
    printf("Result: %g\n", evaluate(&polynomial, 5.0));
    list_destroy(&polynomial);
    printf("\n");

    // Test 4: -0.3125x^4 - 9.915x^2 - 7.75x - 40.0 with x = 123.45
    list_init(&polynomial, free);
    appendTerm(&polynomial, -0.3125);
    appendTerm(&polynomial, 0.0);
    appendTerm(&polynomial, -9.915);
    appendTerm(&polynomial, -7.75);
    appendTerm(&polynomial, -40.0);
    display(&polynomial);
    printf("Result: %g\n", evaluate(&polynomial, 123.45));
    list_destroy(&polynomial);
    printf("\n");

    return 0;
}
```

Output:

x + 1.0

Evaluation: 2.00

x^2 - 1.0

Result: 3.1209

-3.0x^3 + 0.5x^2 - 2.0x

Result: -372.5

-0.3x^4 - 9.9x^2 - 7.8x - 40.0

Result: -7.27317e+07

Whole program:

```c
#include <stdio.h>
#include "list.h"
#include <math.h>

//Prototype
void appendTerm(List *pPolynomial, double constant);
void display(List *pPolynomial);
double evaluate(List *pPolynomial, double x);

//main funct
int main() {
    List polynomial;

    // Test 1: x + 1.0 with x = 1.0
    list_init(&polynomial, free);
    appendTerm(&polynomial, 1.0);
    appendTerm(&polynomial, 1.0);
    display(&polynomial);
    printf("Evaluation: %.2f\n", evaluate(&polynomial, 1.0));
    list_destroy(&polynomial);

    // Test 2: x^2 - 1.0 with x = 2.03
    list_init(&polynomial, free);
    appendTerm(&polynomial, 1.0);
    appendTerm(&polynomial, 0.0);
    appendTerm(&polynomial, -1.0);
    display(&polynomial);
    printf("Result: %g\n", evaluate(&polynomial, 2.03));
    list_destroy(&polynomial);
    printf("\n");

    // Test 3: -3.0x^3 + 0.5x^2 - 2.0x with x = 5.0
    list_init(&polynomial, free);
    appendTerm(&polynomial, -3.0);
    appendTerm(&polynomial, 0.5);
    appendTerm(&polynomial, -2.0);
    appendTerm(&polynomial, 0.0);
    display(&polynomial);
    printf("Result: %g\n", evaluate(&polynomial, 5.0));
    list_destroy(&polynomial);
    printf("\n");
```

```
    // Test 4: -0.3125x^4 - 9.915x^2 - 7.75x - 40.0 with x = 123.45
    list_init(&polynomial, free);
    appendTerm(&polynomial, -0.3125);
    appendTerm(&polynomial, 0.0);
    appendTerm(&polynomial, -9.915);
    appendTerm(&polynomial, -7.75);
    appendTerm(&polynomial, -40.0);
    display(&polynomial);
    printf("Result: %g\n", evaluate(&polynomial, 123.45));
    list_destroy(&polynomial);
    printf("\n");

    return 0;
}

//Append_function
void appendTerm(List *pPolynomial, double constant) {
    double *data = (double *)malloc(sizeof(double));
    if (data == NULL) {
        printf("Error: Failed to allocate memory for new term.\n");
        exit(EXIT_FAILURE);
    }
    *data = constant;
    if (list_ins_next(pPolynomial, list_tail(pPolynomial), data) != 0) {
        printf("Error: Failed to append new term.\n");
        exit(EXIT_FAILURE);
    }
}

//display funt
void display(List *pPolynomial) {
    int degree = list_size(pPolynomial) - 1;
    ListElmt *current = list_head(pPolynomial);
    int isFirstTerm = 1;  // A flag to check if it's the first term being printed

    while(current) {
        double coefficient = *(double *)list_data(current);
        if(coefficient) {
            if (!isFirstTerm) {
                if (coefficient > 0) printf(" + ");
                else if (coefficient < 0) printf(" - ");
            } else {
                if (coefficient < 0) printf("-");
                isFirstTerm = 0;  // reset the flag once the first term is
processed
```

```
            }

            if (fabs(coefficient) != 1.0 || degree == 0) {
                printf("%.1lf", fabs(coefficient));
            }

            if (degree > 1) printf("x^%d", degree);
            else if (degree == 1) printf("x");
        }
        degree--;
        current = list_next(current);
    }
    printf("\n");
}


//evaluate funct
double evaluate(List *pPolynomial, double x) {
    double result = 0.0;
    int degree = list_size(pPolynomial) - 1;
    ListElmt *current = list_head(pPolynomial);

    while(current) {
        double coefficient = *(double *)list_data(current);
        result += coefficient * pow(x, degree);
        degree--;
        current = list_next(current);
    }

    return result;
}
```