

Report

Trang bìa

- Tên học phần: Kỹ thuật lập trình
- Giảng viên hướng dẫn: Nguyễn Thị Thanh Huyền
- Nhóm sinh viên thực hiện: Nhóm 2 lớp
 - An Việt Trung 20195936
 - Lê Thị Kiều Trang 20195930
- Logo Trường + Viện

Mục lục

Mục lục

I. Quá trình phát triển Chương trình

Bước 0: Phân tích thiết kế chương trình

Thiết kế chức năng tìm nghiệm

Thiết kế chức năng in

Bước 1: Mô tả chương trình chính

Các cấu trúc dữ liệu

Hàm main

Các chức năng trong menu

Bước 2: Chi tiết các Module chính

Tìm cận trên của miền nghiệm

Đổi hàm số từ dạng $f(x)$ sang $f(-x)$

Tìm miền chứa nghiệm

Thu hẹp khoảng bằng phương pháp chia đôi

Tìm khoảng phân ly nghiệm

Công thức sai số

Tính giá trị hàm số $f(x)$ và đạo hàm $f'(x)$ tại x

Thuật toán tiếp tuyến (gốc)

II. Mã nguồn

system.h

adv_print.h

Polynomial.h

main.cpp

III. Giao diện và demo chương trình

Thông tin các file dữ liệu demo

project-settings.txt

input.txt

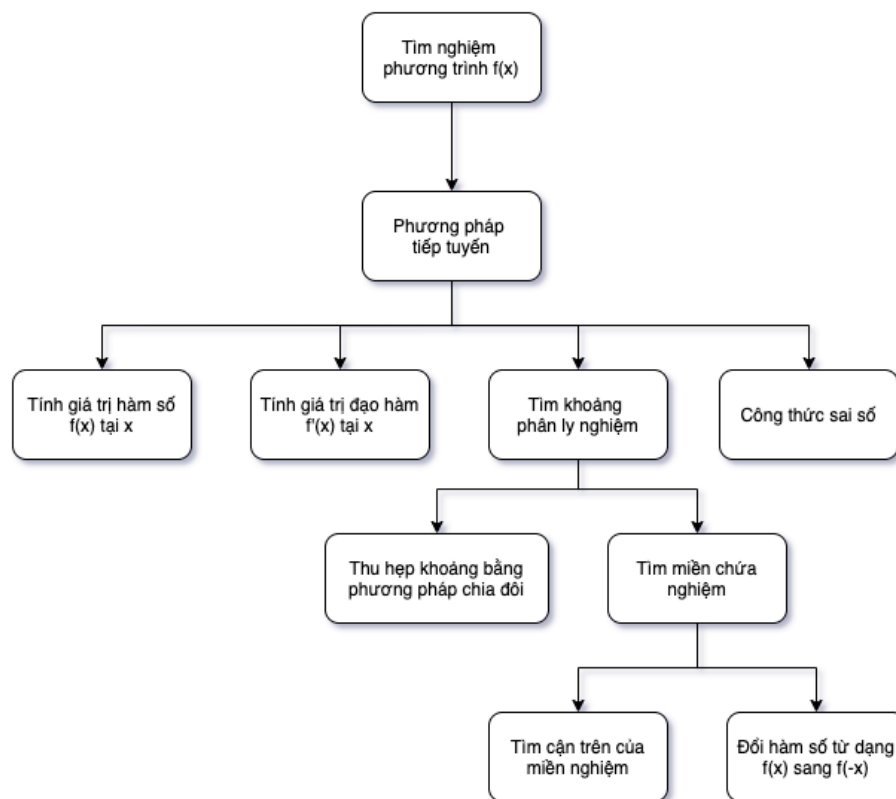
Demo và giao diện

File quá trình và kết quả (output.txt)

I. Quá trình phát triển Chương trình

Bước 0: Phân tích thiết kế chương trình

Thiết kế chức năng tìm nghiệm



Thiết kế chức năng in

- Gửi nội dung xâu
 - In ra màn hình xâu
 - In ra file xâu

Bước 1: Mô tả chương trình chính

Các cấu trúc dữ liệu

```

// Kiểu miền
typedef struct Interval {
    double left; // Giới hạn trái
    double right; // Giới hạn phải
} Interval;

// Hạng tử trong đa thức
typedef struct PNode {
    double coeff; // Hệ số
    int exponent; // Bậc

    PNode* next; // Hạng tử tiếp theo
} PNode;

// Đa thức
class Polynomial {
private:
    PNode* head; // Danh sách các hạng tử
    Interval root_interval; // Miền chứa nghiệm

    bool found_root_interval; // đánh dấu để không tìm lại miền nghiệm nhiều lần
}
  
```

Hàm main

- Thực thi hàm đọc file "**project-settings.txt**" và khởi tạo các giá trị global cho chương trình, gồm có:
 - Đường dẫn đến **file input**: chứa thông tin về đa thức
 - Đường dẫn đến **file output**: chứa quá trình thực hiện chương trình và các kết quả ra
 - Các hằng số mặc định
 - Số chữ số thập phân sau dấu phẩy
 - Hằng số Epsilon
- Khởi tạo đa thức $f(x)$ từ file input
- Thực thi thân chương trình chính (vòng lặp):
 - In lên màn hình các lựa chọn chức năng hiện tại của chương trình (chi tiết các chức năng phía dưới)
 - Đợi người dùng chọn chức năng muốn sử dụng
 - In ra tên chức năng người dùng lựa chọn
 - Thực thi chức năng người dùng lựa chọn
- Thực thi hàm kết thúc chương trình (đóng file, giải phóng vùng nhớ,...)

Các chức năng trong menu

1. Tìm miền chứa nghiệm
 - Sử dụng hàm **Get_Root_Interval**
 - In kết quả
2. Tìm các khoảng phân ly (a, b) thoả mãn $|b - a| \leq 0,5$
 - Sử dụng hàm **Find_KPL** với giá trị 0,5
 - In kết quả
3. Tìm nghiệm gần đúng với số lần lặp n
 - Nhập n
 - Nhập khoảng và kiểm tra xem có phân ly
 - Sử dụng hàm **Newton_Raphson** với số lần lặp n
 - In kết quả
4. Tìm nghiệm gần đúng với sai số ϵ
 - Nhập ϵ
 - Nhập khoảng và kiểm tra xem có phân ly
 - Sử dụng hàm **Newton_Raphson_Err_Formula** với sai số ϵ
 - In kết quả
5. Tìm nghiệm gần đúng x_n thoả mãn $|x_n - x_{n-1}| \leq \epsilon$
 - Nhập ϵ
 - Nhập khoảng và kiểm tra xem có phân ly
 - Sử dụng hàm **Newton_Raphson** với sai số ϵ
 - In kết quả

Bước 2: Chi tiết các Module chính

Tìm cận trên của miền nghiệm

```
double Polynomial::Find_Root_Interval_Upper_Edge() {
    if (head == NULL) return 0;

    // sign of a0
    double sign = head->coeff < 0 ? -1.0f : 1.0f;
    int m = -1;
    double max;
    PNode* cur = head->next;

    while (cur != NULL) {
        if (cur->coeff * sign < 0) {
            m = head->exponent - cur->exponent;
            max = cur->coeff;
            break;
        }
        cur = cur->next;
    }

    // opposite sign coefficient is available
    if (m >= 0) {
        cur = head->next;
        while (cur != NULL) {
            if (cur->coeff * sign < 0) {
                if (fabs(cur->coeff) > fabs(max))
                    max = cur->coeff;
            }
            cur = cur->next;
        }

        return 1 + pow(-max / head->coeff, 1.0f / m);
    } else {
        return 0;
    }
}
```

Đổi hàm số từ dạng $f(x)$ sang $f(-x)$

```
void Polynomial::Reverse_Odd_Exponent_Sign() {
    PNode* cur = head;
    while (cur != NULL) {
        if (cur->exponent % 2 == 1)
            cur->coeff = -cur->coeff;
        cur = cur->next;
    }
}
```

Tìm miền chứa nghiệm

```
void Polynomial::Find_Root_Interval() {
    root_interval.right = Find_Root_Interval_Upper_Edge();

    // Temporary change  $f(x)$  to  $f(-x)$ 
    Reverse_Odd_Exponent_Sign();

    root_interval.left = -Find_Root_Interval_Upper_Edge();

    // Change back to  $f(x)$ 
    Reverse_Odd_Exponent_Sign();

    found_root_interval = true;
}
```

Thu hẹp khoảng bằng phương pháp chia đôi

```
Interval Bisection(Polynomial* f, double a, double b, double dist) {
    Interval res;
    double c = 0;
    while (fabs(a - b) > dist) {
        c = (a + b) / 2;
        if ( sign(f->F(c)) == sign(f->F(b)) )
            b = c;
        else
            a = c;
    }

    res.left = a;
    res.right = b;

    return res;
}
```

Tìm khoảng phân ly nghiệm

```
void Polynomial::Find_KPL(double e) {
    Interval ri = Get_Root_Interval();
    double k = (ri.right - ri.left) / head->exponent;
    double A = ri.left;
    double B = ri.right;

    Interval sInter;
    int count = 0;

    while (A <= B) {
        if (F(A) * F(A + k) < 0) {
            print("\tKPL %d ban đầu là (%lf, %lf)\n", ++count, A, A + k);
            sInter.left = A;
            sInter.right = A + k;
            Bisection(this, &sInter, e);
            print("\tKPL %d sau khi thu hẹp bằng PP chia đôi là (%lf, %lf)\n\n", count, sInter.left, sInter.right);
        }
        A = A + k;
    }
}
```

Công thức sai số

```
double Polynomial::Err_Target_Form(double x, double m) {
    return fabs(F(x) / m);
}

double Polynomial::Err_DA_Form(double x, double px, double m, double M) {
    return (M - m) * fabs(x - px) / m;
}
```

Tính giá trị hàm số $f(x)$ và đạo hàm $f'(x)$ tại x

```
double Polynomial::F(double x) {
    if (head == NULL) return 0;

    double res = head->coeff;
    int degree = head->exponent;
    PNode* cur = head->next;

    while (cur != NULL) {
        while (cur->exponent < degree-1) {
            res *= x;
            degree--;
        }
        res = cur->coeff + res * x;
    }
}
```

```

        degree--;

        cur = cur->next;
    }

    return res;
}

double Polynomial::dF(double x) {
    if (head == NULL) return 0;

    double res = head->coeff * head->exponent;
    int degree = head->exponent-1;
    PNode* cur = head->next;

    while (cur != NULL) {
        while (cur->exponent < degree-1) {
            res *= x;
            degree--;
        }
        res = cur->coeff * cur->exponent + res * x;
        degree--;

        cur = cur->next;
    }

    return res;
}

```

Thuật toán tiếp tuyến (gốc)

```

double Newton_Raphson(Polynomial* f, double a, double b, double e) {
    double x = a;
    double px = 0;

    do {
        px = x;
        x = x - f->F(x) / f->dF(x);
    } while (fabs(x - px) > e);

    return x;
}

```

II. Mã nguồn

system.h

```

#ifndef AVT_SYSTEM
#define AVT_SYSTEM

#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <unistd.h>
#include <ctime>
#include <map>

#ifdef _WIN32
#include <conio.h>
#include <windows.h>
#endif

using namespace std;

namespace avt_sys {

// =====

```

```

// SYSTEM CALL
// =====
void ScrollScreen();
void CursorToHome();
void ClearScreen();
char GetInput();
string avtGetCurrentTime();

// =====
// IMPLEMENT
// =====
void ScrollScreen()
{
    #ifdef _WIN32
        system("clear");
    #else
        cout << "\x1b[2J";
    #endif
};
void CursorToHome()
{
    #ifdef _WIN32
        COORD p = {0, 0};
        SetConsoleCursorPosition( GetStdHandle( STD_OUTPUT_HANDLE ), p);
    #else
        cout << "\x1b[H";
    #endif
}

void ClearScreen()
{
    CursorToHome();

    int lines = 26;
    for(int i = 0; i <= lines * 80; i++) {
        putchar(i % 80 ? ' ' : 10); // 10 = [Line Feed]
    }

    CursorToHome();
}

char GetInput()
{
    #ifdef _WIN32
        char input = getch();
    #else
        // Set terminal to raw mode
        system("stty raw");

        // Wait for single character
        char input = getchar();

        // Reset terminal to normal "cooked" mode
        system("stty cooked");
    #endif

    return input;
}

char* GetCurrentTime()
{
    {
        time_t now = time(0);
        return ctime(&now);
    }
}

#endif

```

adv_print.h

```

#ifndef ADV_PRINT
#define ADV_PRINT

#include <stdio.h>
#include <stdarg.h>

```

```

#include "system.h"

#define MAX_N 256

namespace adv_print {
// =====
// Khai báo
// =====
// FILE output
static FILE * fout = NULL;

// Mở file
void init_output_file(char * fileName, const char * mode);
// Đóng file
void close_output_file();

// In ra màn hình và file (nếu có)
void print(const char* format, ...);

// In kí tự 'c' n lần ra màn hình và file (nếu có)
void print_multiple_char(char c, int n);

// In thời gian và mệnh lệnh ra màn hình và file (nếu có)
void print_command(const char* command);

// =====
// Khai triển
// =====

void init_output_file(char * fileName, const char * mode) {
    fileName[strcspn(fileName, "\n")] = 0; // remove '\n' character
    fout = fopen(fileName, mode);
}

void close_output_file() {
    fclose(fout);
}

void print(const char* format, ...) {
    va_list args;

    va_start(args, format);
    vprintf(format, args);
    va_end(args);

    if (fout != NULL) {
        va_start(args, format);
        vfprintf(fout, format, args);
        va_end(args);
    }
}

void print_multiple_char(char c, int n) {
    char s[MAX_N];
    memset(s, c, n);
    s[n] = 0;

    printf("\t%s\n", s);

    if (fout != NULL)
        fprintf(fout, "\t%s\n", s);
}

void print_command(const char* command) {
    printf("=== %s ===\n", command);

    if (fout != NULL)
        fprintf(fout, "%s\t%s\n", avt_sys::GetCurrentTime(), command);
}
}

#endif

```

Polynomial.h


```

#ifndef POLYNOMIAL
#define POLYNOMIAL

#include <stdio.h>
#include <math.h>
#include "adv_print.h"

using namespace adv_print;

static int p_precision = 4;

// =====
// Cấu trúc dữ liệu
// =====

// Kiểu miền
typedef struct Interval {
    double left; // Giới hạn trái
    double right; // Giới hạn phải
} Interval;

// Hạng tử trong đa thức
typedef struct PNode {
    double coeff; // Hệ số
    int exponent; // Bậc

    PNode* next; // Hạng tử tiếp theo
} PNode;

// Đa thức
class Polynomial {
private:
    PNode* head; // Danh sách các hạng tử
    Interval root_interval; // Miền chứa nghiệm

    bool found_root_interval; // đánh dấu để không tìm lại miền nghiệm nhiều lần
public:
    Polynomial();
    ~Polynomial();

    // Thêm một hạng tử dạng  $cx^e$  vào đa thức
    void Insert_Node(int e, double c);

    // In hàm số
    void Print();

    // Tính giá trị đa thức tại điểm x
    double F(double x);

    // Tính giá trị đạo hàm tại điểm x
    double dF(double x);

    // Tính giá trị đạo hàm cấp 2 tại điểm x
    double ddF(double x);

    // Đổi đa thức từ  $f(x)$  sang  $f(-x)$ 
    void Reverse_Odd_Exponent_Sign();

    // Tìm cận trên của miền nghiệm
    double Find_Root_Interval_Upper_Edge();

    // Tìm miền nghiệm
    void Find_Root_Interval();
    Interval Get_Root_Interval();

    // Tìm khoảng phân ly
    void Find_KPL(double e);

    // Kiểm tra xem có phải khoảng phân ly
    bool Is_KPL(double a, double b);
    bool Is_KPL(Interval interval);

    // Các công thức sai số
    double Err_Target_Form(double x, double m);
    double Err_DA_Form(double x, double px, double m, double M);
};

```

```

// Lặp n
double Newton_Raphson(Polynomial* f, double a, double b, int n, bool print_state = false);

// Sai số
double Newton_Raphson(Polynomial* f, double a, double b, double e, bool print_state = false);

// Sai số theo công thức
double Newton_Raphson_Err_Formula(Polynomial* f, double a, double b, double e, bool print_state = false);

// Các Hàm phụ trợ
// -----
int sign(double x);
double min(double a, double b);
double max(double a, double b);

Interval Read_Interval();

// Phương pháp chia đôi
Interval Bisection(Polynomial* f, double a, double b, double dist);
void Bisection(Polynomial* f, Interval* interval, double dist);

// =====
// Khai triển class Polynomial
// =====

Polynomial::Polynomial() {
    head = NULL;
    found_root_interval = false;
}

Polynomial::~Polynomial() {
    PNode* cur = head;

    while (cur != NULL) {
        head = head->next;
        delete cur;
        cur = head;
    }
}

void Polynomial::Insert_Node(int e, double c) {
    PNode* new_node = (PNode*)malloc(sizeof(PNode));
    PNode* prev = NULL;
    PNode* cur = head;

    if ( new_node == NULL ) return; // space isn't available

    // Setup value
    new_node->exponent = e;
    new_node->coeff = c;

    // Loop to find the correct location in the list (descent)
    while ( cur != NULL && cur->exponent >= e ) {
        prev = cur;
        cur = cur->next;
    }

    // Insert new node into list
    if (prev == NULL) {
        new_node->next = head;
        head = new_node;
    } else {
        if (prev->exponent == e) {
            prev->coeff += new_node->coeff;
            delete new_node;
        } else {
            prev->next = new_node;
            new_node->next = cur;
        }
    }

    found_root_interval = false;
}

void Polynomial::Print() {
    PNode* cur = head;

```

```

while (cur != NULL) {
    if (cur != head) // not first node
        printf("%.1f", cur->coeff);
    else
        printf("%.1f", cur->coeff);

    switch (cur->exponent) {
        case 0: break;

        case 1:
            printf("x ");
            break;

        default:
            printf("x^%d ", cur->exponent);
            break;
    }

    cur = cur->next;
}
}

double Polynomial::F(double x) {
    if (head == NULL) return 0;

    double res = head->coeff;
    int degree = head->exponent;
    PNode* cur = head->next;

    while (cur != NULL) {
        while (cur->exponent < degree-1) {
            res *= x;
            degree--;
        }
        res = cur->coeff + res * x;
        degree--;

        cur = cur->next;
    }

    return res;
}

double Polynomial::dF(double x) {
    if (head == NULL) return 0;

    double res = head->coeff * head->exponent;
    int degree = head->exponent-1;
    PNode* cur = head->next;

    while (cur != NULL) {
        while (cur->exponent < degree-1) {
            res *= x;
            degree--;
        }
        res = cur->coeff * cur->exponent + res * x;
        degree--;

        cur = cur->next;
    }

    return res;
}

double Polynomial::ddF(double x) {
    if (head == NULL) return 0;

    double res = head->coeff * head->exponent * (head->exponent - 1);
    int degree = head->exponent-2;
    PNode* cur = head->next;

    while (cur != NULL) {
        while (cur->exponent < degree-2) {
            res *= x;
            degree--;
        }
        if (cur->exponent > 1)
            res = cur->coeff * cur->exponent * (cur->exponent - 1) + res * x;
    }
}

```

```

        else
            res = res * x;
            degree--;

        cur = cur->next;
    }

    return res;
}

void Polynomial::Reverse_Odd_Exponent_Sign() {
    PNode* cur = head;
    while (cur != NULL) {
        if (cur->exponent % 2 == 1)
            cur->coeff = -cur->coeff;
        cur = cur->next;
    }
}

double Polynomial::Find_Root_Interval_Upper_Edge() {
    if (head == NULL) return 0;

    // sign of a0
    double sign = head->coeff < 0 ? -1.0f : 1.0f;
    int m = -1;
    double max;
    PNode* cur = head->next;

    while (cur != NULL) {
        if (cur->coeff * sign < 0) {
            m = head->exponent - cur->exponent;
            max = cur->coeff;
            break;
        }
        cur = cur->next;
    }

    // opposite sign coefficient is available
    if (m >= 0) {
        cur = head->next;
        while (cur != NULL) {
            if (cur->coeff * sign < 0) {
                if (fabs(cur->coeff) > fabs(max))
                    max = cur->coeff;
            }
            cur = cur->next;
        }

        return 1 + pow(-max / head->coeff, 1.0f / m);
    } else {
        return 0;
    }
}

void Polynomial::Find_Root_Interval() {
    root_interval.right = Find_Root_Interval_Upper_Edge();

    // Temporary change f(x) to f(-x)
    Reverse_Odd_Exponent_Sign();

    root_interval.left = -Find_Root_Interval_Upper_Edge();

    // Change back to f(x)
    Reverse_Odd_Exponent_Sign();

    found_root_interval = true;
}

Interval Polynomial::Get_Root_Interval() {
    if (found_root_interval == false)
        Find_Root_Interval();

    return root_interval;
}

void Polynomial::Find_KPL(double e) {
    Interval ri = Get_Root_Interval();

```

```

double k = (ri.right - ri.left) / head->exponent;
double A = ri.left;
double B = ri.right;

Interval sInter;
int count = 0;

while (A <= B) {
    if (F(A) * F(A + k) < 0) {
        print("\tkPL %d ban đầu là (%lf, %lf)\n", ++count, A, A + k);
        sInter.left = A;
        sInter.right = A + k;
        Bisection(this, &sInter, e);
        print("\tkPL %d sau khi thu hẹp bằng PP chia đôi là (%lf, %lf)\n\n", count, sInter.left, sInter.right);
    }
    A = A + k;
}

bool Polynomial::Is_KPL(double a, double b) {
    return F(a) * F(b) < 0;
}

bool Polynomial::Is_KPL(Interval interval) {
    return Is_KPL(interval.left, interval.right);
}

double Polynomial::Err_Target_Form(double x, double m) {
    return fabs(F(x) / m);
}

double Polynomial::Err_DA_Form(double x, double px, double m, double M) {
    return (M - m) * fabs(x - px) / m;
}

// =====
// Khai triển Thuật toán tiếp tuyến
// =====
double Newton_Raphson(Polynomial* f, double a, double b, int n, bool print_state) {
    double x = a;
    double px = 0;
    int count = 0;
    double m = min( fabs(f->dF(a)), fabs(f->dF(b)) );
    double M = max( fabs(f->dF(a)), fabs(f->dF(b)) );
    double err;

    // CT sai số 1
    if (print_state) {
        print("\t1. Sai số theo công thức mục tiêu\n");
        print_multiple_char('-', 2 * p_precision + 20);
        print("\t|%-12s|%-*s|%-*s|\n", "Lần lặp", p_precision+4, "x", p_precision+6, "Sai số");
        print_multiple_char('-', 2 * p_precision + 20);
    }

    do {
        x = x - f->F(x) / f->dF(x);
        err = f->Err_Target_Form(x, m);
        count++;

        if (print_state) {
            print("\t|%-8d|%.*lf|%.*lf|\n", count, p_precision+4, p_precision, x, p_precision+4, p_precision, err);
            print_multiple_char('-', 2 * p_precision + 20);
        }
    } while (count < n);

    // CT sai số 2
    x = a;
    count = 0;
    if (print_state) {
        print("\t1. Sai số theo công thức hai xấp xỉ liên tiếp\n");
        print_multiple_char('-', 2 * p_precision + 20);
        print("\t|%-12s|%-*s|%-*s|\n", "Lần lặp", p_precision+4, "x", p_precision+6, "Sai số");
        print_multiple_char('-', 2 * p_precision + 20);
    }

    do {
        px = x;

```

```

    x = x - f->F(x) / f->dF(x);
    err = f->Err_DA_Form(x, px, m, M);
    count++;

    if (print_state) {
        print("\t|%-8d|%. *lf|%. *lf|\n", count, p_precision+4, p_precision, x, p_precision+4, p_precision, err);
        print_multiple_char('-', 2 * p_precision + 20);
    }

} while (count < n);

return x;
}

double Newton_Raphson_Err_Formula(Polynomial* f, double a, double b, double e, bool print_state) {
    double x = a;
    double px = 0;
    int count = 0;
    double m = min( fabs(f->dF(a)), fabs(f->dF(b)) );
    double M = max( fabs(f->dF(a)), fabs(f->dF(b)) );
    double c = fabs((m * e) / (M - m));
    double err;

    // CT sai số 1
    if (print_state) {
        print("\t1. Sai số theo công thức mục tiêu\n");
        print_multiple_char('-', 2 * p_precision + 20);
        print("\t|%-12s|%- *s|%- *s|\n", "Lần lặp", p_precision+4, "x", p_precision+6, "Sai số");
        print_multiple_char('-', 2 * p_precision + 20);
    }

    do {
        x = x - f->F(x) / f->dF(x);
        err = f->Err_Target_Form(x, m);
        count++;

        if (print_state) {
            print("\t|%-8d|%. *lf|%. *lf|\n", count, p_precision+4, p_precision, x, p_precision+4, p_precision, err);
            print_multiple_char('-', 2 * p_precision + 20);
        }

    } while (err > e);

    // CT sai số 2
    x = a;
    count = 0;
    if (print_state) {
        print("\t1. Sai số theo công thức hai xấp xỉ liên tiếp\n");
        print_multiple_char('-', 2 * p_precision + 20);
        print("\t|%-12s|%- *s|%- *s|\n", "Lần lặp", p_precision+4, "x", p_precision+6, "Sai số");
        print_multiple_char('-', 2 * p_precision + 20);
    }

    do {
        px = x;
        x = x - f->F(x) / f->dF(x);
        err = f->Err_DA_Form(x, px, m, M);
        count++;

        if (print_state) {
            print("\t|%-8d|%. *lf|%. *lf|\n", count, p_precision+4, p_precision, x, p_precision+4, p_precision, err);
            print_multiple_char('-', 2 * p_precision + 20);
        }

    } while (fabs(x - px) >= c);

    return x;
}

double Newton_Raphson(Polynomial* f, double a, double b, double e, bool print_state) {
    double x = a;
    double px = 0;
    int count = 0;

    if (print_state) {
        print_multiple_char('-', 2 * p_precision + 20);
        print("\t|%-12s|%- *s|%- *s|\n", "Lần lặp", p_precision+4, "x", p_precision+6, "Sai số");
    }

```

```

        print_multiple_char('-', 2 * p_precision + 20);
    }

    do {
        px = x;
        x = x - f->F(x) / f->dF(x);
        count++;

        if (print_state) {
            print("\t|%-8d|%. *lf|%. *lf|\n", count, p_precision+4, p_precision, x, p_precision+4, p_precision, fabs(x - px));
            print_multiple_char('-', 2 * p_precision + 20);
        }

    } while (fabs(x - px) > e);

    return x;
}

// =====
// Khai triển Các hàm phụ trợ
// =====

int sign(double x) {
    return x < 0 ? 0 : 1;
}

double min(double a, double b) {
    return a < b ? a : b;
}
double max(double a, double b) {
    return a > b ? a : b;
}

Interval Read_Interval() {
    Interval res;
    double a, b;

    scanf("%lf %lf", &a, &b);
    if (a > b) {
        double t = a;
        a = b;
        b = t;
    }
    res.left = a;
    res.right = b;

    return res;
}

Interval Bisection(Polynomial* f, double a, double b, double dist) {
    Interval res;
    double c = 0;
    while (fabs(a - b) > dist) {
        c = (a + b) / 2;
        if ( sign(f->F(c)) == sign(f->F(b)) )
            b = c;
        else
            a = c;
    }

    res.left = a;
    res.right = b;

    return res;
}

void Bisection(Polynomial* f, Interval* interval, double dist) {
    *interval = Bisection(f, interval->left, interval->right, dist);
}

#endif

```

main.cpp

```

#include <stdio.h>
#include <iostream>
#include "system.h"
#include "adv_print.h"
#include "Polynomial.h"

using namespace avt_sys;
using namespace adv_print;
using namespace std;

// =====
// DEFINITION
// =====
#define PROJECT_SETTINGS_FILE "project-settings.txt"

// =====
// CONSTANT
// =====
#define MAX_FILE_NAME_SIZE 256
static double default_epsilon = 0.001f;
static int precision = 4;

// =====
// GLOBAL VARIABLE
// =====
FILE * ps_file;
FILE * inp;

Polynomial poly;
char ans = '0';

// =====
// MAIN FUNCTION
// =====
int Initialization();
void Destruction();

void MainMenu_Layout();
void Function_Layout(char id);

int Read_Interval_Layout(Interval* res);
double Read_Epsilon_Layout();

// =====
// MAIN PROGRAM
// =====
int main()
{
    // Nếu khởi tạo không thành công thì thoát chương trình
    int err = Initialization();
    if ( err > 0 ) {
        printf("Khởi tạo thất bại. error code: %d\n", err);
        Destruction();
        return 0;
    }

    ScrollScreen();

    while (ans != 'x') {
        if ('0' <= ans && ans <= '5')
            ClearScreen();
        switch (ans) {
            case '0':
                MainMenu_Layout();
                break;
            default:
                Function_Layout(ans);
                break;
        }

        ans = GetInput();
    }

    print_command("Kết thúc chương trình");

    ClearScreen();
}

```



```

    Destruction();
    return 0;
}

// =====
// IMPLEMENT
// =====
int Initialization()
{
    char buff[MAX_FILE_NAME_SIZE];
    int n, e;
    double c;

    // Đọc file project settings
    ps_file = fopen(PROJECT_SETTINGS_FILE, "r");
    if (ps_file == NULL)
        return 1;

    // Đọc file input
    fgets(buff, MAX_FILE_NAME_SIZE, ps_file);
    buff[strcspn(buff, "\n")] = 0; // remove '\n' character
    inp = fopen(buff, "r");
    if (inp == NULL)
        return 2;

    // Khởi tạo đa thức
    fscanf(inp, "%d", &n);
    for (int i = 0; i < n; i++) {
        fscanf(inp, "%d %lf", &e, &c);
        poly.Insert_Node(e, c);
    }

    // Mở file output
    fgets(buff, MAX_FILE_NAME_SIZE, ps_file);
    init_output_file(buff, "w");
    if (adv_print::fout == NULL)
        return 3;

    // Khởi tạo hằng số
    fscanf(ps_file, "%d", &precision);
    p_precision = precision;
    fscanf(ps_file, "%lf", &default_epsilon);

    print_command("Khởi tạo chương trình thành công");
    return 0;
}

void Destruction()
{
    fclose(ps_file);
    fclose(inp);
    close_output_file();
}

void MainMenu_Layout()
{
    cout << "=== Menu Chính ===" << endl
        << "Đa thức: "; poly.Print();
    cout << endl;
    cout << "[1] Tìm miền chứa nghiệm" << endl
        << "[2] Tìm các khoảng phân ly (a,b) thoả mãn |b-a| <= 0.5" << endl
        << "[3] Tìm nghiệm gần đúng với số lần lặp n" << endl
        << "[4] Tìm nghiệm gần đúng với sai số e" << endl
        << "[5] Tìm nghiệm gần đúng x_n thoả mãn |x_n - x_{n-1}| <= e" << endl
        << "[x] Thoát";
}

void Function_Layout(char id)
{
    Interval interval;
    double e, x0;
    int n;

    switch (id) {
        case '1':
            print_command("Tìm miền chứa nghiệm");

            interval = poly.Get_Root_Interval();

```

```

        print("\tMiền chứa nghiệm là (%.*lf, %.*lf)\n",
            precision, interval.left,
            precision, interval.right);

        break;
    case '2':
        print_command("Tìm các khoảng phân ly");
        poly.Find_KPL(0.5f);
        break;
    case '3':
        print_command("Tìm nghiệm gần đúng với số lần lặp n");
        printf("Nhập số lần lặp n = ");
        scanf("%d", &n);
        print("\tSố lần lặp n là %d\n", n);

        if (Read_Interval_Layout(&interval)) {
            x0 = Newton_Raphson(&poly, interval.left, interval.right, n, true);
            print("\n\tNghiệm x là %.*lf\n", precision, x0);
        }

        break;
    case '4':
        print_command("Tìm nghiệm gần đúng với sai số e");
        e = Read_Epsilon_Layout();

        if (Read_Interval_Layout(&interval)) {
            x0 = Newton_Raphson_Err_Formula(&poly, interval.left, interval.right, e, true);
            print("\n\tNghiệm x là %.*lf\n", precision, x0);
        }
        break;
    case '5':
        print_command("Tìm nghiệm gần đúng x_n");
        e = Read_Epsilon_Layout();

        if (Read_Interval_Layout(&interval)) {
            x0 = Newton_Raphson(&poly, interval.left, interval.right, e, true);
            print("\n\tNghiệm x_n là %.*lf\n", precision, x0);
        }
        break;

    default:
        return;
}

cout << endl << endl
    << "[0] Về menu chính" << endl
    << "[x] Thoát";
}

int Read_Interval_Layout(Interval* res)
{
    bool valid;

    printf("Nhập khoảng phân ly (a, b) = ");
    *res = Read_Interval();

    print("\tKhoảng phân ly đã nhập là (%.*lf, %.*lf)\n", precision, res->left, precision, res->right);
    if (poly.Is_KPL(*res) == false) {
        print("\tKhoảng đã nhập không phải khoảng phân ly!\n");
        return 0;
    }
    return 1;
}

double Read_Epsilon_Layout()
{
    double e;

    printf("\nNhập e = 0 nếu muốn sử dụng hằng số epsilon mặc định!\n");
    printf("Nhập sai số e = ");
    scanf("%lf", &e);
    e = e != 0 ? e : default_epsilon;

    print("\tSai số epsilon đã nhập là %.*lf\n", precision, e);

    return e;
}

```

III. Giao diện và demo chương trình

Thông tin các file dữ liệu demo

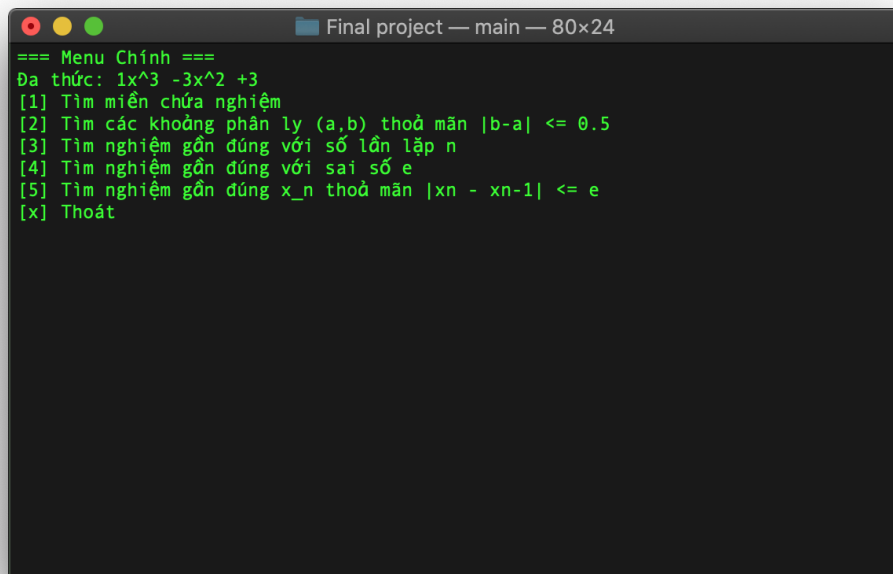
project-settings.txt

```
input.txt  
output.txt  
9  
0.001
```

input.txt

```
3  
0 3  
2 -3  
3 1
```

Demo và giao diện



```
Final project — main — 80x24  
=== Menu Chính ===  
Đa thức:  $1x^3 - 3x^2 + 3$   
[1] Tìm miền chứa nghiệm  
[2] Tìm các khoảng phân ly (a,b) thỏa mãn  $|b-a| \leq 0.5$   
[3] Tìm nghiệm gần đúng với số lần lặp n  
[4] Tìm nghiệm gần đúng với sai số e  
[5] Tìm nghiệm gần đúng  $x_n$  thỏa mãn  $|x_n - x_{n-1}| \leq e$   
[x] Thoát
```

```
Final project — main — 80x24
=== Tìm miền chứa nghiệm ===
    Miền chứa nghiệm là (-2.442249586, 4.000000000)

[0] Về menu chính
[x] Thoát_
```

```
Final project — main — 80x24
=== Tìm các khoảng phân ly ===
    KPL 1 ban đầu là (-2.442250, -0.294833)
    KPL 1 sau khi thu hẹp bằng PP chia đôi là (-1.100114, -0.831687)

    KPL 2 ban đầu là (-0.294833, 1.852583)
    KPL 2 sau khi thu hẹp bằng PP chia đôi là (1.315729, 1.584156)

    KPL 3 ban đầu là (1.852583, 4.000000)
    KPL 3 sau khi thu hẹp bằng PP chia đôi là (2.389438, 2.657865)

[0] Về menu chính
[x] Thoát_
```

```
Final project — main — 80x24
=== Tìm nghiệm gần đúng với số lần lặp n ===
Nhập số lần lặp n = 5
Số lần lặp n là 5
"Nhập (a, b) = (0, 0) nếu muốn sử dụng các khoảng phân ly mặc định!"
Nhập khoảng phân ly (a, b) = 1 1.8
Khoảng phân ly đã nhập là (1.000000000, 1.800000000)
1. Sai số theo công thức mục tiêu

|Lần lặp |x          |Sai số          |
|-----|-----|-----|
|1      | 1.333333333| 0.034293553|
|-----|-----|-----|
|2      | 1.347222222| 0.000181093|
|-----|-----|-----|
|3      | 1.347296353| 0.000000005|
|-----|-----|-----|
|4      | 1.347296355| 0.000000000|
|-----|-----|-----|
|5      | 1.347296355| 0.000000000|
|-----|-----|-----|
1. Sai số theo công thức hai xấp xỉ liên tiếp

|Lần lặp |x          |Sai số          |
|-----|-----|-----|
```

```
Final project — main — 80x24
=== Tìm nghiệm gần đúng với sai số e ===
"Nhập e = 0 nếu muốn sử dụng hằng số epsilon mặc định!"
Nhập sai số e = 0.000001
Sai số epsilon đã nhập là 0.000001000
"Nhập (a, b) = (0, 0) nếu muốn sử dụng các khoảng phân ly mặc định!"
Nhập khoảng phân ly (a, b) = 2.2 3
Khoảng phân ly đã nhập là (2.200000000, 3.000000000)
1. Sai số theo công thức mục tiêu

|Lần lặp |x          |Sai số          |
|-----|-----|-----|
|1      | 2.860606061| 1.408583371|
|-----|-----|-----|
|2      | 2.608854148| 0.255920005|
|-----|-----|-----|
|3      | 2.537962701| 0.018106127|
|-----|-----|-----|
|4      | 2.532127701| 0.000118857|
|-----|-----|-----|
|5      | 2.532088888| 0.000000005|
|-----|-----|-----|
1. Sai số theo công thức hai xấp xỉ liên tiếp

|Lần lặp |x          |Sai số          |
|-----|-----|-----|
```

```
Final project — main — 80x24
=== Tìm nghiệm gần đúng x_n ===
"Nhập e = 0 nếu muốn sử dụng hằng số epsilon mặc định!"
Nhập sai số e = 0
    Sai số epsilon đã nhập là 0.001000000
"Nhập (a, b) = (0, 0) nếu muốn sử dụng các khoảng phân ly mặc định!"
Nhập khoảng phân ly (a, b) = -1 -0.5
    Khoảng phân ly đã nhập là (-1.000000000, -0.500000000)



| Lần lặp | x            | Sai số      |
|---------|--------------|-------------|
| 1       | -0.888888889 | 0.111111111 |
| 2       | -0.879451567 | 0.009437322 |
| 3       | -0.879385245 | 0.000066322 |



Nghiệm x_n là -0.879385245

[0] Về menu chính
[x] Thoát
```

File quá trình và kết quả (output.txt)

```
Mon Jun 14 02:21:53 2021
    Khởi tạo chương trình thành công
Mon Jun 14 02:21:59 2021
    Tìm miền chứa nghiệm
    Miền chứa nghiệm là (-2.442249586, 4.000000000)
Mon Jun 14 02:22:05 2021
    Tìm các khoảng phân ly
    KPL 1 ban đầu là (-2.442250, -0.294833)
    KPL 1 sau khi thu hẹp bằng PP chia đôi là (-1.100114, -0.831687)

    KPL 2 ban đầu là (-0.294833, 1.852583)
    KPL 2 sau khi thu hẹp bằng PP chia đôi là (1.315729, 1.584156)

    KPL 3 ban đầu là (1.852583, 4.000000)
    KPL 3 sau khi thu hẹp bằng PP chia đôi là (2.389438, 2.657865)
Mon Jun 14 02:22:10 2021
    Tìm nghiệm gần đúng với số lần lặp n
    Số lần lặp n là 5
    Khoảng phân ly đã nhập là (1.000000000, 1.800000000)
    1. Sai số theo công thức mục tiêu



| Lần lặp | x           | Sai số      |
|---------|-------------|-------------|
| 1       | 1.333333333 | 0.034293553 |
| 2       | 1.347222222 | 0.000181093 |
| 3       | 1.347296353 | 0.000000005 |
| 4       | 1.347296355 | 0.000000000 |
| 5       | 1.347296355 | 0.000000000 |


    1. Sai số theo công thức hai xấp xỉ liên tiếp



| Lần lặp | x           | Sai số      |
|---------|-------------|-------------|
| 1       | 1.333333333 | 0.034293553 |
| 2       | 1.347222222 | 0.000181093 |
| 3       | 1.347296353 | 0.000000005 |
| 4       | 1.347296355 | 0.000000000 |
| 5       | 1.347296355 | 0.000000000 |


```

1		1.333333333		0.592592593	
2		1.347222222		0.024691358	
3		1.347296353		0.000131788	
4		1.347296355		0.000000004	
5		1.347296355		0.000000000	

Nghiệm x là 1.347296355

Mon Jun 14 02:22:43 2021

Tìm nghiệm gần đúng với sai số e

Sai số epsilon đã nhập là 0.000001000

Khoảng phân ly đã nhập là (2.200000000, 3.000000000)

1. Sai số theo công thức mục tiêu

Lần lặp x	Sai số	
1		2.860606061 1.408583371
2		2.608854148 0.255920005
3		2.537962701 0.018106127
4		2.532127701 0.000118857
5		2.532088888 0.000000005

1. Sai số theo công thức hai xấp xỉ liên tiếp

Lần lặp x	Sai số	
1		2.860606061 3.843526171
2		2.608854148 1.464738402
3		2.537962701 0.412459330
4		2.532127701 0.033949089
5		2.532088888 0.000225821
6		2.532088886 0.000000010

Nghiệm x là 2.532088886

Mon Jun 14 02:23:29 2021

Tìm nghiệm gần đúng x_n

Sai số epsilon đã nhập là 0.001000000

Khoảng phân ly đã nhập là (-1.000000000, -0.500000000)

Lần lặp x	Sai số	
1		-0.888888889 0.111111111
2		-0.879451567 0.009437322
3		-0.879385245 0.000066322

Nghiệm x_n là -0.879385245

Mon Jun 14 02:24:06 2021

Kết thúc chương trình