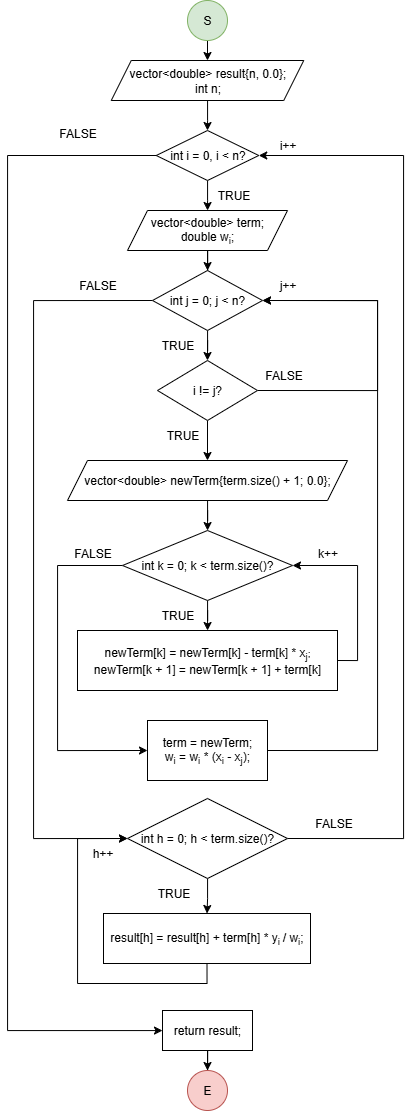
**CODE REPORT**

1. Tổng quan
   * 1. Chương trình được xây dựng dựa trên phương pháp nội suy Lagrange với hai chức năng cơ bản sau:
2. Đọc được các test case từ file dữ liệu cho trước;
3. Tự kiểm tra được dữ liệu và trả về dạng hoàn chỉnh của hàm nội suy theo phương pháp Lagrange;
4. Có giao diện người dùng.
   * 1. *Ngôn ngữ lập trình: C++.*
5. Cơ sở lý thuyết
   * 1. Dựa trên công thức Lagrange:
     2. Với là **đa thức cơ sở Lagrange** được xác định bởi:
     3. Mỗi có giá trị bằng 1 tại và bằng 0 ở các điểm còn lại.
     4. Ta gọi:
     5. Từ đó suy ra:
     6. Ta đưa về dạng đa thức tổng quát:
     7. Từ (1) và (2), ta nhận thấy:
     8. Ngoài ra, có một quy tắc khác có quan hệ mật thiết với công thức này, đó là khi ta nhân một đa thức bất kì với , ta được kết quả như sau:
     9. Ta nhận ra quy tắc:

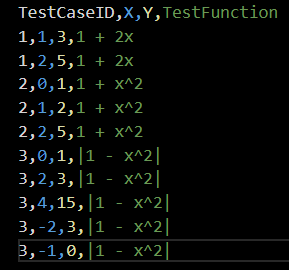
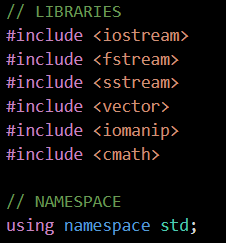
* Hệ số của :
* Hệ số của :
* Hệ số của :
  + 1. ...
* Hệ số của :
* Hệ số của :
  + 1. Để hiểu rõ mối liện hệ giữa quy tắc này với công thức (3), ta xét ví dụ:
    2. Giả sử có một đa thức bất kì:
    3. Nếu ta nhân nó với , ta được:
    4. Thực chất, cũng chỉ là một hàm **“”** bất kì nào đó khác.
    5. Từ đó suy ra, nếu ta coi:
    6. ...
    7. Thì:
    8. Từ đây, ta có thể tạo thuật toán tìm hệ số cho từng bậc của ẩn .

1. Sơ đồ thuật toán:
2. Xây dựng chương trình:
   1. Hệ thống dữ liệu
      1. Các test case được lưu vào file csv lần lượt với các thuộc tính sau:

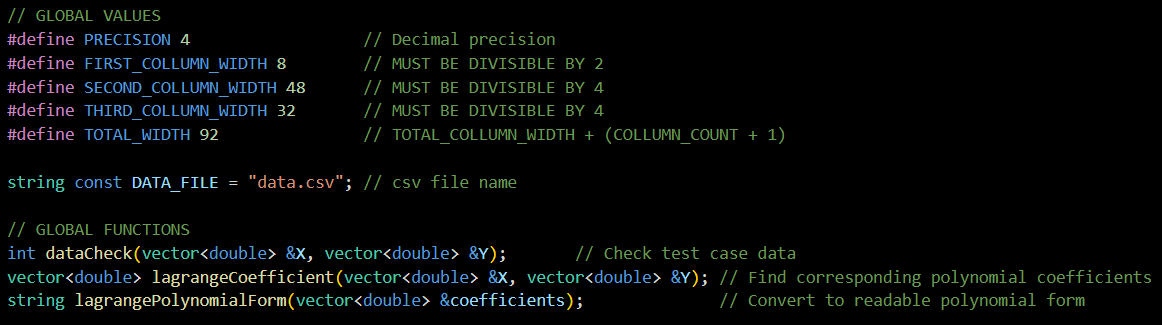
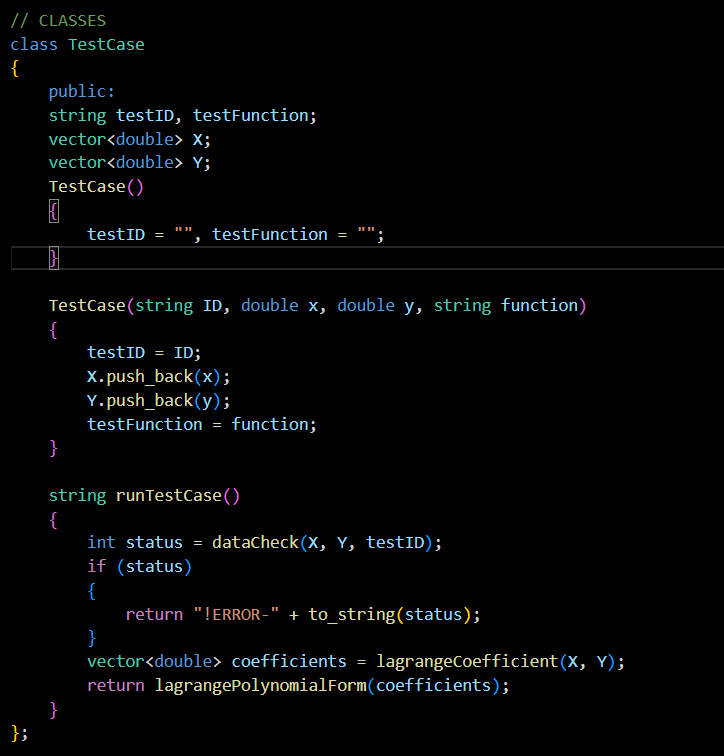
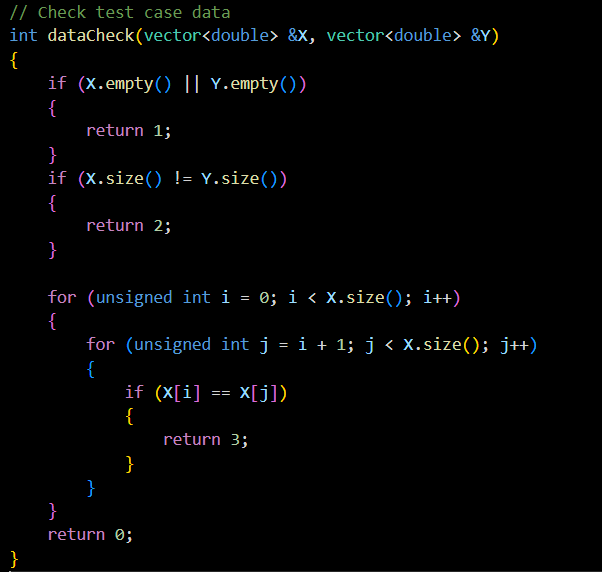
|  |  |  |  |
| --- | --- | --- | --- |
| * + 1. Test ID | * + 1. x | * + 1. y | * + 1. Test Function |

Các thuộc tính được ngăn cách nhau bằng dấu phẩy.

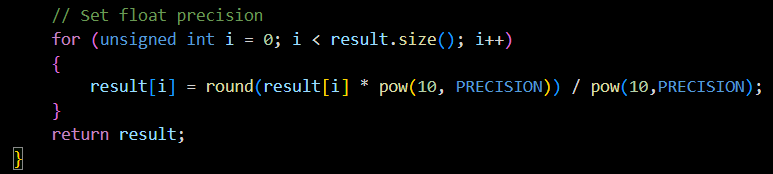
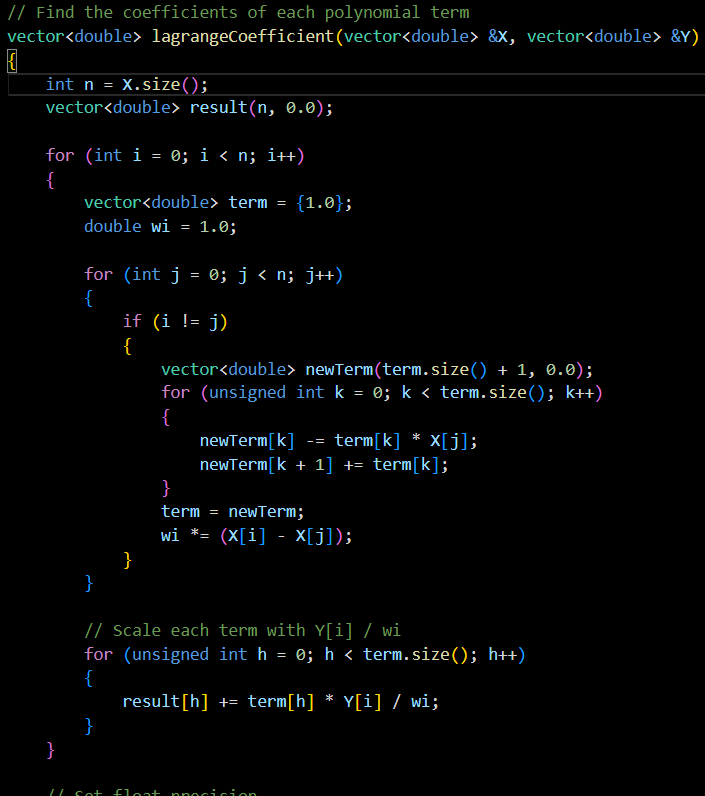
Ví dụ:

* + 1. 
  1. Chức năng chương trình
     1. **Thứ nhất, chương trình sẽ có 3 hàm cơ bản:**
     2. (1) int dataCheck(): Kiểm tra dữ liệu, trả về giá trị khác 0 nếu dữ liệu có trục trặc;
     3. (2) vector<double> lagrangeCoefficient(): Tìm và trả về vector chứa hệ số tương ứng với từng bậc của đa thức;
     4. (3) string lagrangePolynomialForm(): Xâu chuỗi từ kết quả của hàm (2), trả về dạng đa thức của hàm nội suy Lagrange.
     5. **Thứ hai, chương trình sẽ có lớp đối tượng** “TestCase”:
     6. (1) Cho phép lưu trữ các thuộc tính test case: Test ID; các tọa độ x, y tương ứng và hàm test.
     7. (2) Các đối tượng của lớp sẽ có hành vi chạy test case bằng cách sử dụng các hàm cơ bản.
     8. **Thứ ba, chương trình sẽ có UI cho phép người dùng chọn 2 chức năng:**
     9. (1) Lựa chọn và chạy 1 test case được lưu trữ;
     10. (2) Chạy toàn bộ các test case.
  2. Code
     1. Ta bắt đầu với thư viện và namespace:
     2. 

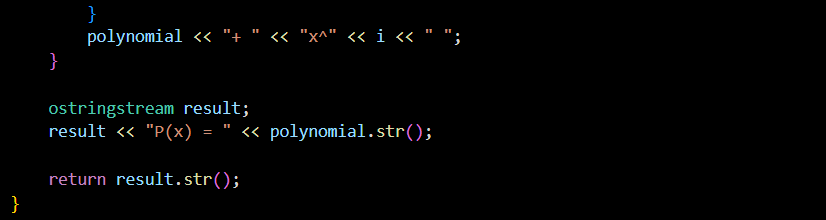
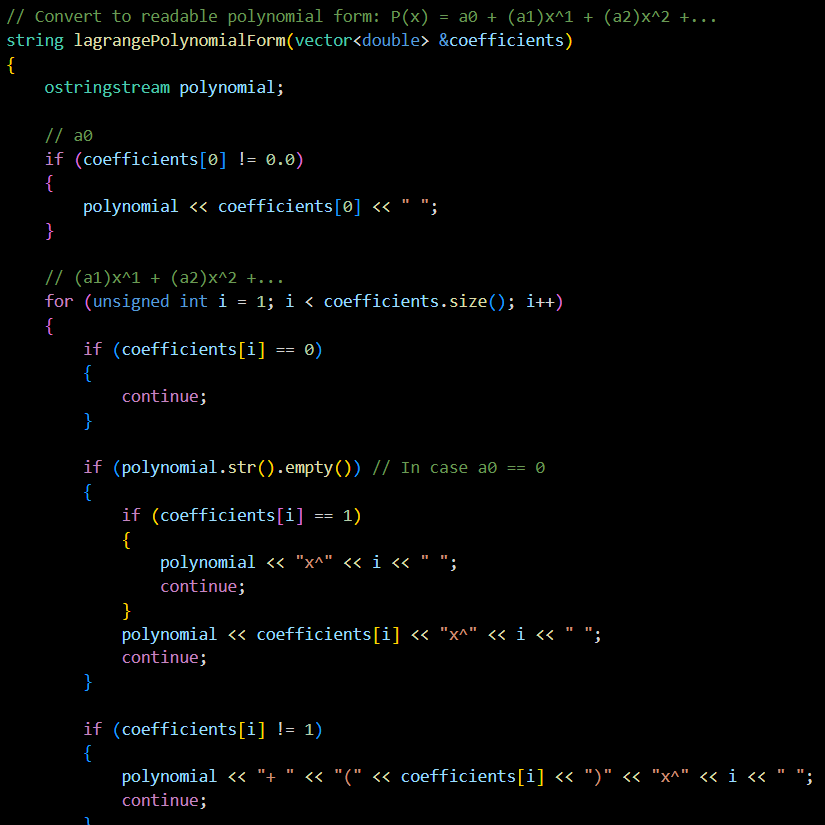
Sau đó, khởi tạo các giá trị và hàm cơ bản:

* + 1. 
    2. Tạo lớp TestCase bao gồm những thuộc tính và hành vi cơ bản của test case:
    3. 
    4. Xây dựng chi tiết chức năng của hàm cơ bản:
    5. (1) Hàm kiểm tra dữ liệu test case:
    6. 

(2) Hàm tìm hệ số:

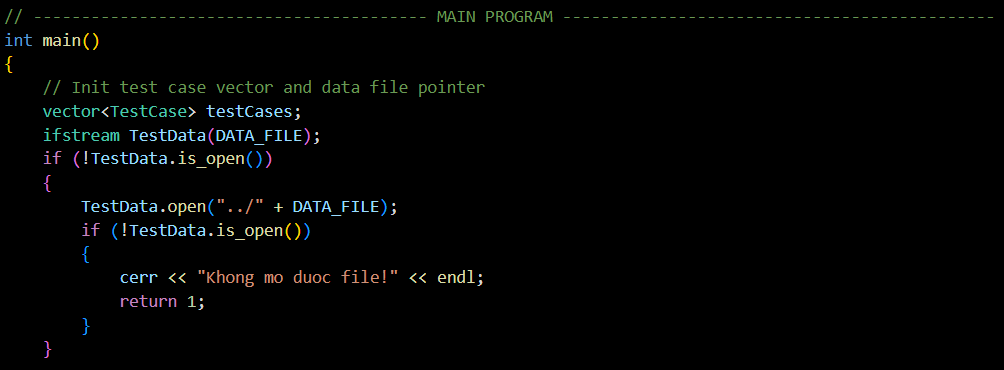


(3) Hàm xâu chuỗi:

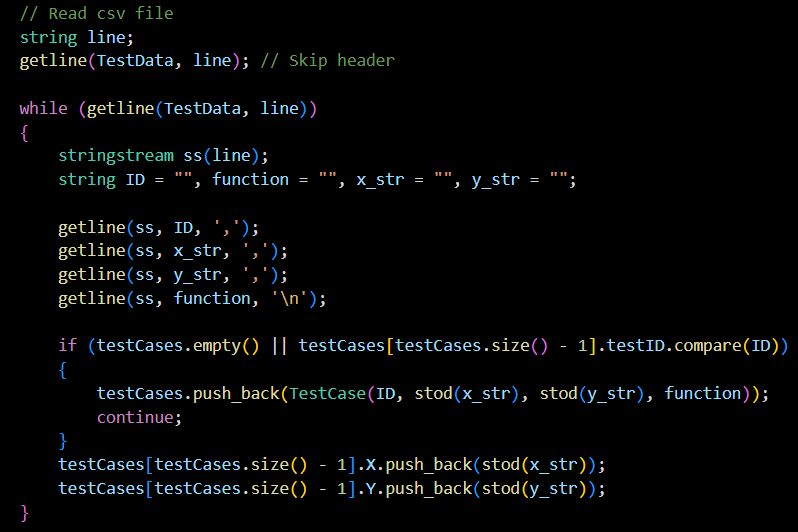


Ta chuyển sang xây dựng chương trình chính:

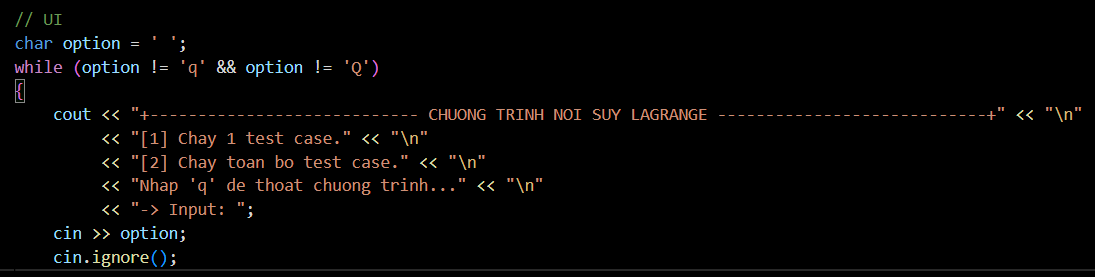
1. Bắt đầu với việc mở file data và tạo vector lưu trữ các test case:



2. Đọc và lưu dữ liệu tương ứng từ data file vào vector:

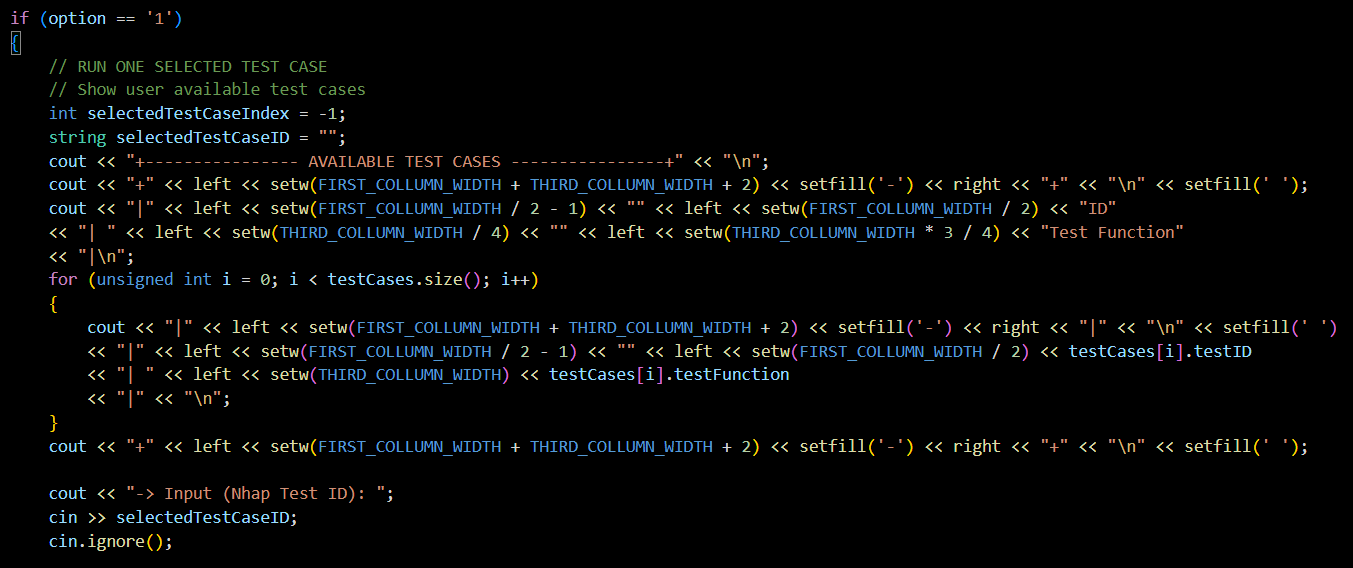


3. Xây dựng hệ thống UI đơn giản:

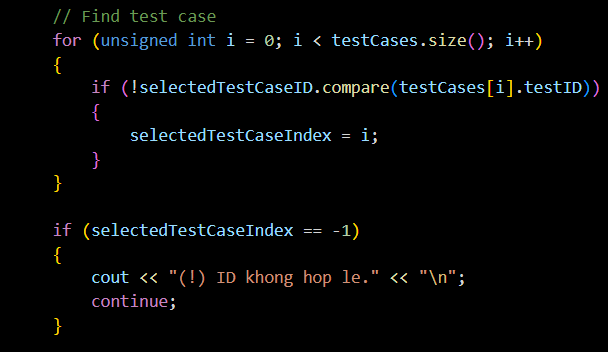


4. Xây dựng chức năng UI đầu tiên: Lựa chọn và chạy 1 test case

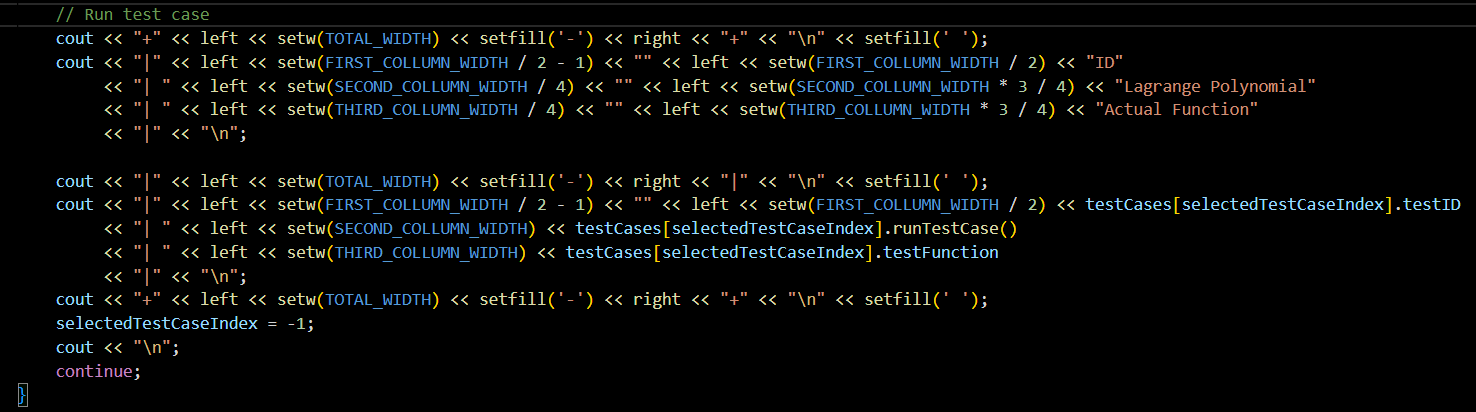
*4.1. Lựa chọn test case:*



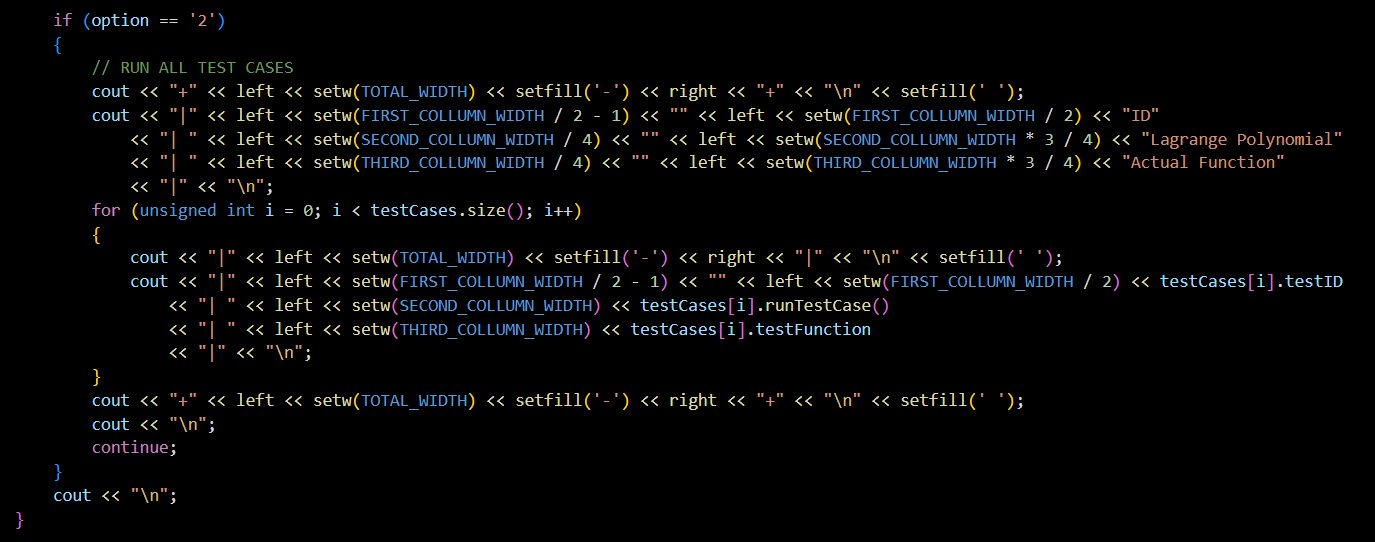
*4.2. Tìm test case:*

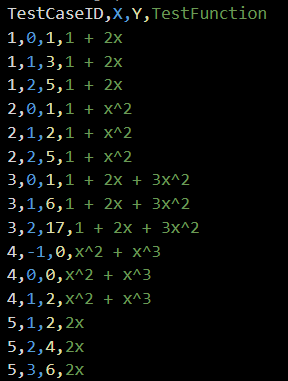
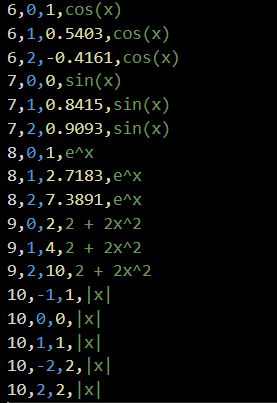
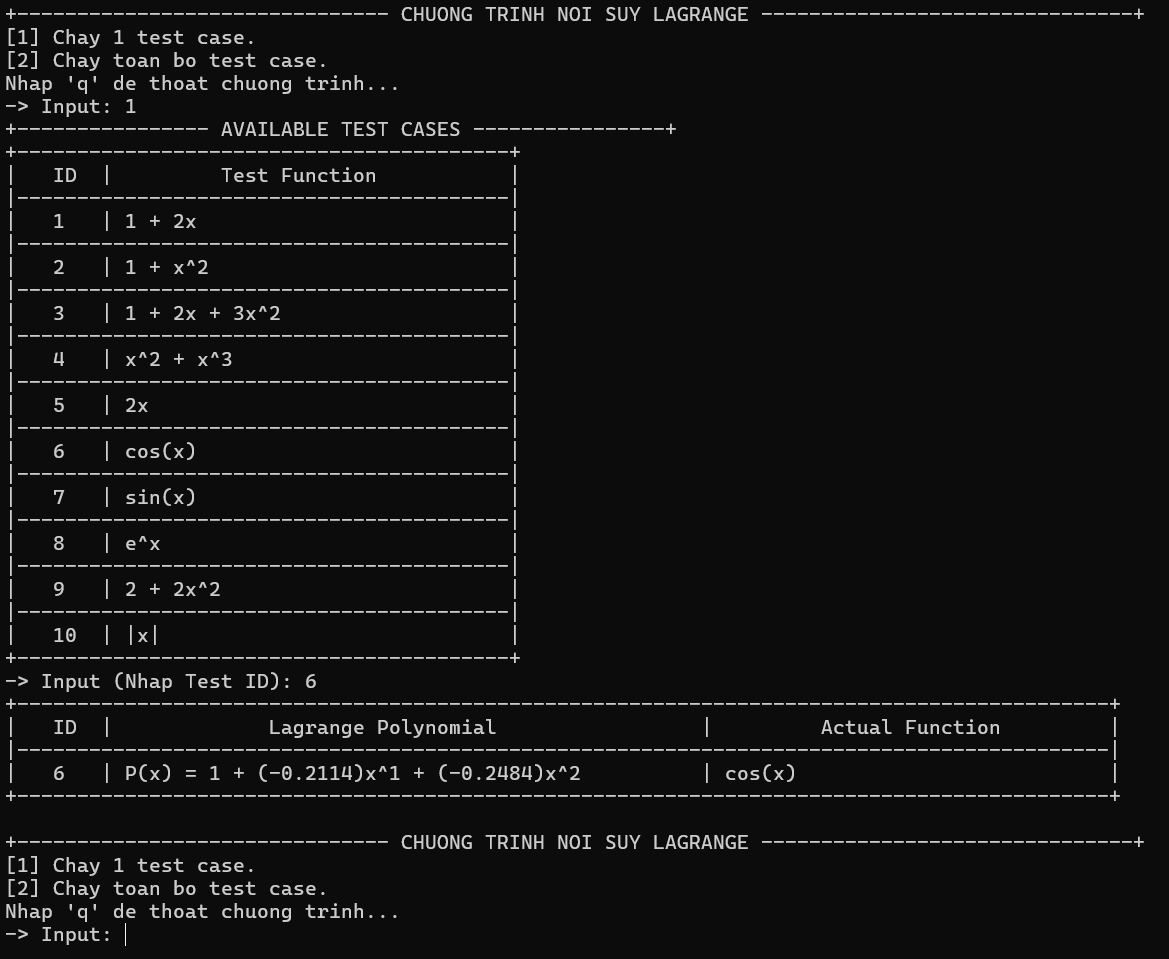
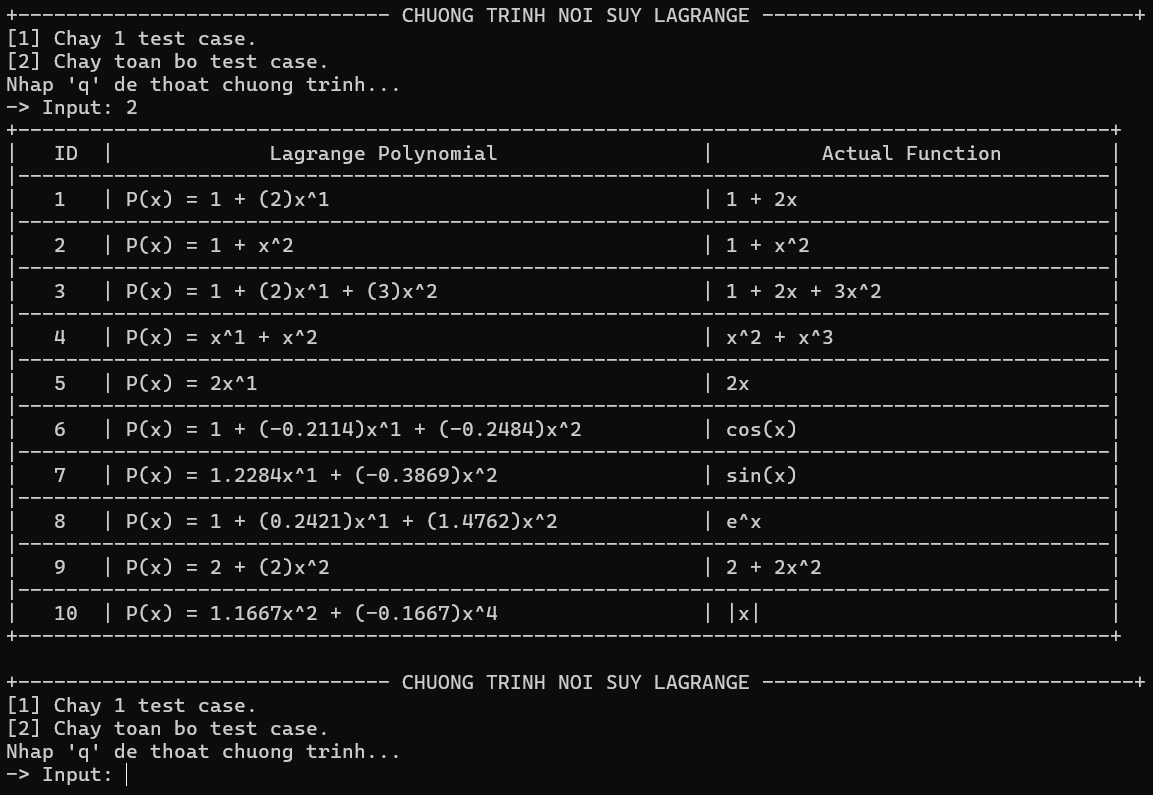
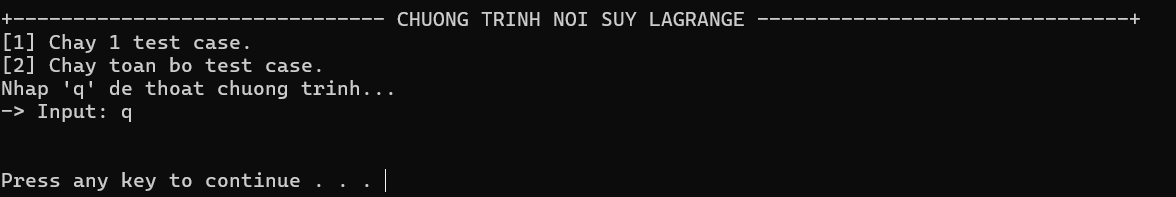


*4.3. Chạy và hiển thị test case:*



5. Xây dựng chức năng còn lại: Chạy toàn bộ test case:



1. Chạy thử chương trình
   * 1. Ta chạy thử chương trình với bộ test case sau:
     2.  
   1. Chức năng đầu tiên: Chạy 1 test case tùy chọn
      1. 
      2. Chạy thành công!
   2. Chức năng còn lại: Chạy toàn bộ các test case
      1. 
      2. Cả hại chức năng đều chạy thành công!
   3. Thoát chương trình
      1. 
2. Đánh giá:
   1. Độ phức tạp
      1. Vì chương trình đơn giản, ta xét theo độ phức tạp của chức năng tốn nhiều tài nguyên nhất:
      2. (1) Về thời gian: chương trình có tổng cộng **T** test case, trong đó độ phức tạp của mỗi lần chạy test case là => Độ phức tạp thời gian là:
      3. (2) Về bộ nhớ: Vector testCases có tổng cộng **T** test case, trong đó, mỗi test có 2 vector X, Y với độ dài **n** => Độ phức tạp bộ nhớ là:
   2. Nhận xét
      1. Nhìn chung, chương trình đáp ứng được các yêu cầu cơ bản. Tuy vậy, do độ phức tạp thời gian quá cao, chương trình hiếm khi đáp ứng được nhu cầu thực tế.