BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC KINH TẾ TP HỒ CHÍ MINH (UEH)

TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ

ĐỒ ÁN MÔN HỌC

ĐỀ TÀI

STACK VÀ TÍNH TOÁN BIỂU THỨC TOÁN HỌC

Học Phần: Cấu Trúc Dữ Liệu & Giải Thuật

Danh Sách Nhóm:

1. NGUYỄN DUY THÀNH VŨ
2. PHẠM TẤN TRUNG NAM
3. PHẠM TRẦN MINH THƯ
4. NGÔ ĐỖ NHẬT MINH

Chuyên Ngành: KHOA HỌC MÁY TÍNH

Khóa: K50

Giảng Viên: TS. Đặng Ngọc Hoàng Thành

Tp. Hồ Chí Minh, Ngày 18 tháng 05 năm 2025

MỤC LỤC

MỤC LỤC **2**

CHƯƠNG 1. STACK VÀ CÁC THUẬT TOÁN **3**

1.1. Các Khái Niệm Liên Quan 3

1.1.1 Cài đặt Stack trong C# 4

a. Cài đạt lớp các ô ngăn Node: 4

1.1.2 Infix (Biểu thức trung tố) 4

1.2. Cấu trúc và cài đặt các thuật toán 5

1.2.1. Thuật Toán Chuyển biểu thức Infix sang Postfix 5

1.2.2. Thuật Toán Tính toán biểu thức đưới dạng Postfix 7

a. Ý tưởng thuật toán: 8

b. Mã giả (Pseudocode): 8

..... 8

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ LỚP **9**

2.1. Phân Tích Bài Toán Tính toán biểu thức toán học bằng Stack 9

2.2. Sơ Đồ Lớp 10

2.3. Cài Đặt Lớp 10

CHƯƠNG 3. THIẾT KẾ GIAO DIỆN **13**

3.1. Giao Diện Máy Tính 13

3.2. Chi Tiết Chức Năng 13

CHƯƠNG 4. THẢO LUẬN & ĐÁNH GIÁ **15**

5.1. Các Kết Quả Nhận Được 15

5.2. Một Số Tồn Tại 15

5.1. Hướng Phát Triển 15

PHỤ LỤC **16**

TÀI LIỆU THAM KHẢO **21**

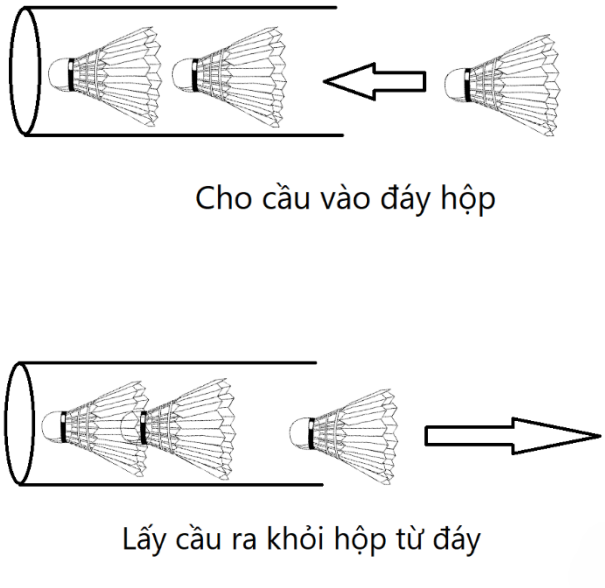
CHƯƠNG 1. STACK VÀ CÁC THUẬT TOÁN

1.1. Các Khái Niệm Liên Quan

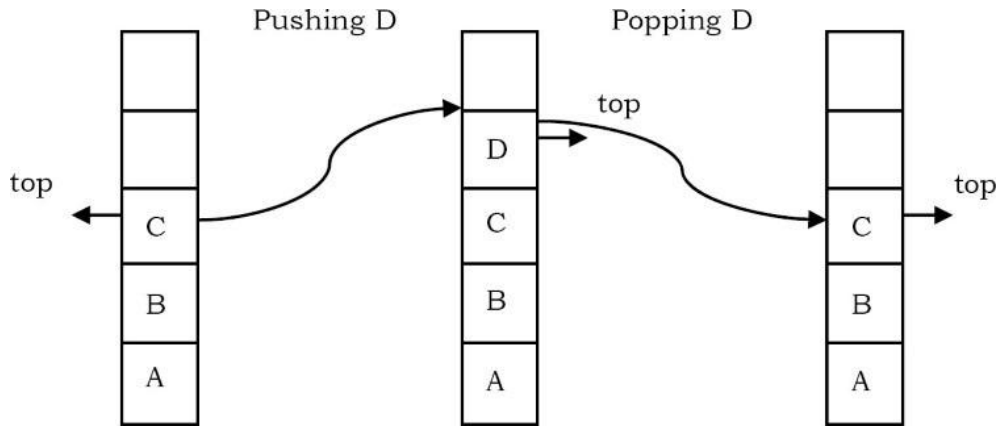
Stack là một danh sách có thứ tự trong đó việc chèn và xóa được thực hiện ở một đầu, được gọi là top - đỉnh. Phần tử cuối cùng được chèn là phần tử đầu tiên sẽ bị xóa. Do đó, nó được gọi là Last in First out (LIFO) hoặc First in Last out (FILO) list.

Khi một phần tử được chèn vào một ngăn xếp, khái niệm được gọi là push và khi một phần tử bị xóa khỏi ngăn xếp, khái niệm được gọi là pop. Việc cố gắng pop một stack trống được gọi là underflow và cố gắng đẩy một phần tử trong một stack đầy được gọi là overflow.

Một ví dụ trong thực tế của stack có thể dễ hình dung được đó chính là hộp cầu lông. Khi muốn cho cầu vào trong hộp ta sẽ cho cầu vào đáy hộp cầu và khi muốn lấy cầu ra thì ta sẽ lấy quả cầu gần nhất được cho vào.



Hình 1: Ví dụ về stack từ hộp cầu lông.



Hình 2: Ví dụ về push và pop trong stack.

Trong thực tế về stack, nó giống như việc bạn đang làm một dự án dài hạn nhưng xếp lại đưa cho bạn một nhiệm vụ quan trọng hơn phải thực hiện và bạn thực hiện nó nhưng lúc này một cuộc gọi lại vang lên, lúc này ưu tiên cao nhất là nghe cuộc gọi đó, sau khi nghe xong bạn lại xử lý công việc quan trọng, thực hiện tiếp dự án.

1.1.1 Cài đặt Stack trong C#

a. Cài đạt lớp các ô ngăn Node:

Cài đặt lớp ô ngăn Node: Khởi tạo thuộc tính object data để lưu giá trị dữ liệu vào và khởi tạo Node next để tạo con trỏ chỉ đến phần tử tiếp theo.

b. Cài đặt lớp Các thao tác của Stack (MyStack)

Trong lớp này cần có thuộc tính Node top để làm giá trị đỉnh của ngăn xếp. Tiếp theo đó cần triển khai các phương thức thao tác cho MyStack như sau:

- Push(object data): Thêm phần tử mới vào đỉnh stack. Tạo node mới, gán next là node hiện tại.
- object Pop(): Loại bỏ phần tử ở đỉnh stack và trả về giá trị. Nếu stack rỗng, trả về null hoặc thông báo lỗi.
- object Peek(): Trả về phần tử ở đỉnh stack mà không loại bỏ nó.
- bool IsEmpty(): Trả về true nếu stack không có phần tử nào.
- void Clear(): Xóa toàn bộ stack bằng cách đặt top = null.
- bool Contains(object data): Kiểm tra xem phần tử cụ thể có tồn tại trong stack không.
- void Reverse(): Đảo ngược toàn bộ stack bằng cách tạo lại liên kết các node.
- void Sort(): Sắp xếp các phần tử trong stack theo thứ tự tăng hoặc giảm. Có thể dùng thuật toán sắp xếp đơn giản như bubble sort kết hợp thao tác push/pop.

1.1.2 Infix (Biểu thức trung tố)

Infix là dạng biểu thức mà toán tử được đặt giữa hai toán hạng, ví dụ: A + B. Đây là dạng ký pháp quen thuộc và dễ hiểu đối với con người vì nó giống cách viết trong toán

học phổ thông. Tuy nhiên để máy tính xử lý chính xác biểu thức Infix ta phải xác định rõ thứ tự ưu tiên toán tử và sử dụng dấu ngoặc để nhóm các phần tử để tránh gây mơ hồ. Việc này làm tăng độ phức tạp khi đánh giá biểu thức trực tiếp từ dạng Infix.

Ký pháp trung tố (Infix) có ưu điểm dễ đọc và dễ hiểu với con người vì các toán tử được đặt giữa các toán hạng giống cách viết thông thường trong toán học. Tuy nhiên, nhược điểm của ký pháp này là nó yêu cầu quy tắc ưu tiên toán tử và dấu ngoặc để xác định thứ tự thực hiện các phép toán. Điều này có thể gây nhầm lẫn và khó xử lý đối với máy tính. Do đó, ký pháp này thường được sử dụng trong các giao diện người dùng của các ứng dụng tính toán, như máy tính trên các hệ điều hành hoặc các chương trình toán học đơn giản.

1.1.3 Postfix (Biểu thức hậu tố)

Postfix hay còn gọi là Reverse Polish Notation (RPN) là dạng biểu thức mà toán tử được đặt sau hai toán hạng. Ví dụ: A B + thay vì A + B. Dạng hậu tố được thiết kế để đơn giản hóa quá trình tính toán cho máy. Khi sử dụng Postfix ta không cần đến dấu ngoặc hay luật ưu tiên – chỉ cần đọc từ trái sang phải và xử lý toán tử ngay khi gặp đủ toán hạng.

Ký pháp hậu tố (Postfix) có ưu điểm là không cần dấu ngoặc hay quy tắc ưu tiên toán tử, giúp việc tính toán trở nên đơn giản và dễ dàng hơn cho máy tính, đặc biệt khi sử dụng với cấu trúc dữ liệu Stack. Nhược điểm là nó khó đọc và khó hiểu đối với con người vì các toán tử được đặt sau các toán hạng, khác với cách viết thông thường. Postfix thường được sử dụng trong các máy tính bỏ túi, trình biên dịch, và các hệ thống tính toán mà yêu cầu sự hiệu quả cao trong việc xử lý các phép toán phức tạp.

1.2. Cấu trúc và cài đặt các thuật toán

1.2.1. Thuật Toán Chuyển biểu thức Infix sang Postfix

Thuật toán chuyển Infix sang Postfix là một thuật toán dùng để chuyển đổi biểu thức ở dạng truyền thống (Infix) sang dạng hậu tố (Postfix), giúp việc tính toán trở nên đơn giản và tuần tự hơn. Thuật toán này được xây dựng dựa trên cấu trúc dữ liệu Stack để xử lý dấu ngoặc và quản lý độ ưu tiên của toán tử một cách hiệu quả.

a.

- Khi gặp toán hạng → thêm trực tiếp vào danh sách kết quả.
- Khi gặp toán tử: so sánh với toán tử trên đỉnh Stack:
- Nếu có độ ưu tiên cao hơn → push vào Stack.
- Nếu thấp hơn hoặc bằng → pop các toán tử có độ ưu tiên cao hơn hoặc bằng, đưa vào kết quả, rồi mới push toán tử hiện tại.
- Khi gặp dấu ngoặc mở ‘(’: push vào Stack.
- Khi gặp dấu ngoặc đóng ‘)’: pop và thêm vào kết quả cho đến khi gặp dấu mở ‘(’.

b. Mã giả (Preudocode):

```
1:  Tạo hàm ConvertInfix(Infix)
2:      Tạo ngăn xếp Stack mystack rỗng
3:      Tạo danh sách Postfix rỗng
4:      Tạo một chuỗi  number rỗng
5:      for prev in Infix
6:          Nếu prev là số hoặc dấu “.” thì
7:              Thêm prev vào number
8:          Ngược lại
9:              Nếu number không rỗng thì
10:                 Thêm number vào Postfix
11:                 Chuyển chuỗi number thành rỗng
12:             Kết thúc điều kiện (9)
13:             Nếu prev = '(' thì
14:                 mystack.push(prev)
15:             Nếu prev = ')' thì
16:                 while mystack.Peek() ≠ '(' do
17:                     Postfix.Add(mystack.Pop())
18:                 Kết thúc vòng lặp while (16)
19:                 mystack.Pop() // bỏ dấu '('
20:             Nếu prev là toán tử
21:                 while mystack không rỗng và độ ưa tiên mystack.Peek() ≥ độ
ưa tiên prev thì
22:                     Postfix.Add(mystack.Pop())
23:                 Kết thúc vòng lặp while (21)
24:                 mystack.Push(prev)
25:             Kết thúc điều kiện (20)
```

```
26:         Kết thúc điều kiện (8)
27:     Kết thúc vòng lặp for (5)
28:     Nếu number không rỗng thì
29:         Postfix.Add(number)
30:     Kết thúc điều kiện (28)
31:     while mystack không rỗng thì
32:         Postfix.Add(mystack.Pop())
33:     Kết thúc vòng lặp while (31)
34:     Trả về kết quả Postfix
```

1.2.2. Thuật Toán Tính toán biểu thức đưới dạng Postfix

Biểu thức hậu tố (Postfix expression), hay còn gọi là biểu thức dạng RPN (Reverse Polish Notation), là một cách biểu diễn toán học trong đó các toán tử được đặt sau các toán hạng. Không giống như biểu thức trung tố (Infix), biểu thức hậu tố không cần dấu ngoặc để xác định thứ tự thực hiện phép toán, vì thứ tự đã được xác định sẵn thông qua vị trí của toán tử trong chuỗi.

Ví dụ:

Biểu thức Infix: (3 + 4) × 5

Biểu thức Postfix tương ứng: 3 4 + 5 ×

Ưu điểm chính của biểu thức hậu tố là loại bỏ sự mơ hồ trong thứ tự thực hiện phép toán, do đó nó rất phù hợp để xử lý bằng máy tính hoặc các chương trình biên dịch, tính toán.

a. Ý tưởng thuật toán:

1. Duyệt qua từng phần tử (token) trong biểu thức hậu tố từ trái sang phải.
2. Nếu phần tử là toán hạng (số), đưa nó vào stack.
3. Nếu phần tử là toán tử, lấy ra hai toán hạng gần nhất từ stack (toán hạng thứ hai được lấy ra chính là toán hạng đầu tiên trong phép toán), thực hiện phép tính, và đưa kết quả trở lại vào stack.
4. Lặp lại quá trình cho đến khi duyệt hết chuỗi biểu thức.
5. Kết quả cuối cùng là phần tử duy nhất còn lại trong stack.

b. Mã giả (Pseudocode):

```
1:   Tạo hàm CalPostfix(List<string> Postfix)
2:       Tạo ngăn xếp Stack mystack rỗng
3:       foreach (string value in Postfix)
4:             Nếu value là số thì
5:                   mystack.Push(value)
6:             Nếu value là toán tử thì
7:                   num1 = mystack.Pop()
8:                   num2 = mystack.Pop()
9:                   result = kết quả của phép tính toán tử value và 2 toán hạng
là num1 và num2
10:                    mystack.Push(result)
11:              Kết thúc điều kiện (6)
12:       Kết thúc vòng lặp foreach (3)
13:       Trả về kết quả cuối cùng mystack.Pop()
```

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ LỚP

2.1. Phân Tích Bài Toán Tính toán biểu thức toán học bằng Stack

Yêu cầu bài toán:

Thiết kế một hệ thống cho phép đánh giá biểu thức toán học theo hướng tuần tự từ trái sang phải mà không cần quay lại các bước trước đó. Điều này đòi hỏi chuyển đổi biểu thức trung tố (Infix) sang hậu tố (Postfix), giúp đơn giản hóa việc xử lý và loại bỏ nhu cầu xác định lại độ ưu tiên phép toán trong quá trình đánh giá.

Ý tưởng giải pháp:

Biểu thức trung tố sẽ được chuyển sang hậu tố để đảm bảo thứ tự thực hiện phép toán là chính xác và rõ ràng. Sau đó, biểu thức hậu tố sẽ được tính toán sử dụng cấu trúc dữ liệu Stack. Điều này đảm bảo việc duyệt và xử lý chỉ cần thực hiện một lần, từ trái sang phải, với độ chính xác và hiệu quả cao.

Triển khai thuật toán:

Thiết kế lớp Solve

Lớp Solve đảm nhận toàn bộ quá trình xử lý biểu thức từ đầu vào đến kết quả đầu ra. Các thành phần chính bao gồm:

Biến thành viên:

- Infix: Biểu thức đầu vào (dạng chuỗi).
- Ans: Kết quả của phép tính trước đó.
- OpCheck: Cờ đánh dấu trạng thái âm/dương của toán hạng.
- ErrorCheck: Kiểm tra lỗi xảy ra trong quá trình tính toán.

Hàm Input(string input):

Tiền xử lý biểu thức đầu vào: loại bỏ khoảng trắng và thay thế "Ans" thành ký tự "A" để tiện xử lý.

Hàm Priority(char value):

Chuyển đổi biểu thức trung tố sang hậu tố bằng cách sử dụng Stack để lưu trữ tạm thời các toán tử và xử lý độ ưu tiên của chúng. Phần này xử lý cả các ký hiệu đặc biệt như √, %, !, và biến Ans.

Hàm Factorial(double value):

Tính giai thừa của một số nguyên dương bằng phương pháp đệ quy.

Hàm CalPostfix(List<string> Postfix):

Đánh giá biểu thức hậu tố bằng cách duyệt qua từng phần tử:

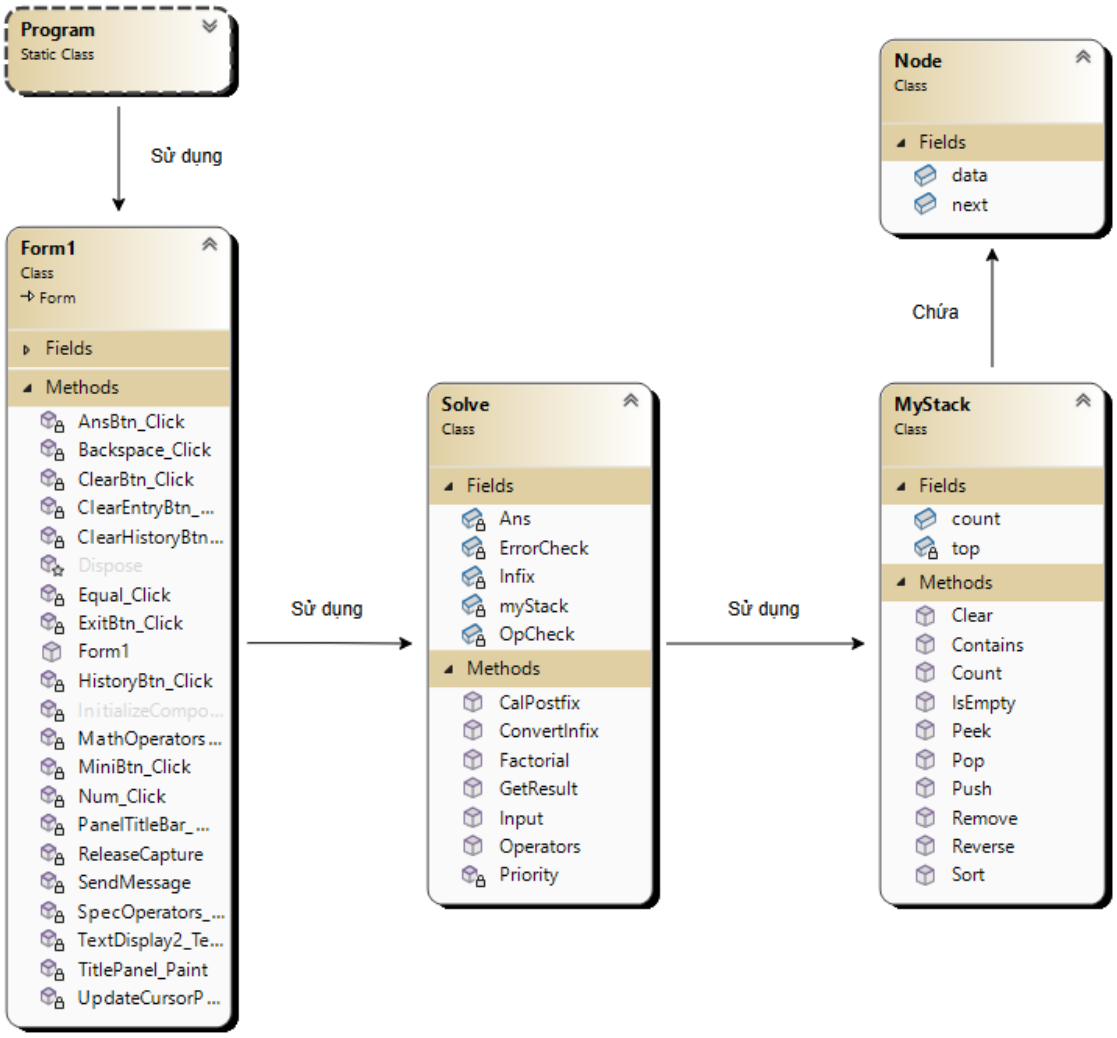
- Nếu là toán hạng: đẩy vào Stack.
- Nếu là toán tử một ngôi: lấy một phần tử từ Stack và thực hiện phép tính.

- Nếu là toán tử hai ngôi: lấy hai phần tử từ Stack, thực hiện phép tính, rồi đẩy kết quả trở lại.

Hàm GetResult():

Gọi các hàm chuyển đổi và tính toán, đồng thời kiểm tra lỗi và trả về kết quả cuối cùng, đã được làm tròn đến 8 chữ số thập phân.

2.2. Sơ Đồ Lớp



2.3. Cài Đặt Lớp

Lớp Solve: đóng vai trò trung tâm trong quá trình xử lý biểu thức toán học, từ việc tiếp nhận đầu vào, chuyển đổi biểu thức sang dạng hậu tố (Postfix), đến việc tính toán kết quả và phát hiện lỗi. Đây là một lớp xử lý nghiệp vụ với trọng trách lớn trong kiến trúc của hệ thống.

Lớp bao gồm các thành phần sau:

Thuộc tính:

- string Infix: lưu biểu thức đầu vào.
- double Ans: lưu kết quả của phép tính gần nhất, hỗ trợ tính tiếp theo.
- bool OpCheck: kiểm tra trạng thái âm/dương của toán hạng.
- bool ErrorCheck: cờ báo lỗi xảy ra trong quá trình tính toán.
- MyStack myStack: Stack để hỗ trợ thao tác trong cả hai thuật toán chính.

Phương thức chính:

- void Input(string input): Chuẩn hóa biểu thức đầu vào, loại bỏ khoảng trắng và xử lý từ khóa đặc biệt "Ans".
- bool Operators(char value): Kiểm tra xem một ký tự có phải là toán tử hợp lệ.
- int Priority(char value): Trả về mức độ ưu tiên của toán tử, được định nghĩa theo thứ tự toán học chuẩn.
- List<string> ConvertInfix(string Infix): Chuyển đổi biểu thức từ trung tố sang hậu tố. Thuật toán xử lý từng ký tự, phân biệt giữa số, toán tử, dấu ngoặc, và áp dụng quy tắc ưu tiên toán tử để đưa ra chuỗi hậu tố hợp lệ.
- double CalPostfix(List<string> Postfix): Duyệt danh sách hậu tố và thực hiện phép tính bằng Stack. Khi gặp toán hạng, đẩy vào Stack; khi gặp toán tử, lấy toán hạng ra, thực hiện phép toán tương ứng và đưa kết quả trở lại Stack. Phép toán một ngôi được xử lý riêng biệt với kiểm tra điều kiện chặt chẽ (ví dụ: không cho phép căn bậc hai số âm hoặc giai thừa của số âm).
- int Factorial(double value): Hàm đệ quy để tính giai thừa, với điều kiện đầu vào là số nguyên dương.
- string GetResult(): Hàm tổng hợp: thực hiện chuyển đổi biểu thức, tính toán kết quả và trả về chuỗi kết quả dưới dạng chuỗi được làm tròn (precision: 8 chữ số thập phân). Nếu có lỗi xảy ra, hàm trả về thông báo "Lỗi".

Lớp MyStack: xây dựng toàn bộ logic thao tác với stack dựa trên cấu trúc liên kết đơn

Thuộc tính:

Node top: Phần tử đầu stack (đỉnh stack).

int count: Đếm số lượng phần tử hiện có trong stack.

Phương thức chính:

- Push(object data): Thêm phần tử mới vào đỉnh stack. Tạo node mới, gán next là node hiện tại, cập nhật lại top và tăng count.

- object Pop(): Loại bỏ phần tử ở đỉnh stack và trả về giá trị. Nếu stack rỗng, trả về null hoặc thông báo lỗi.
- object Peek(): Trả về phần tử ở đỉnh stack mà không loại bỏ nó.
- bool IsEmpty(): Trả về true nếu stack không có phần tử nào.
- int Count(): Trả về số lượng phần tử trong stack.
- void Clear(): Xóa toàn bộ stack bằng cách đặt top = null và count = 0.
- bool Contains(object data): Kiểm tra xem phần tử cụ thể có tồn tại trong stack không.
- void Reverse(): Đảo ngược toàn bộ stack bằng cách tạo lại liên kết các node.
- void Sort(): Sắp xếp các phần tử trong stack theo thứ tự tăng dần.

Lớp Node: đóng vai trò là phần tử cơ bản của stack. Mỗi node đại diện cho một đơn vị dữ liệu và liên kết đến phần tử tiếp theo trong stack.

Thuộc tính:

object data: Lưu trữ giá trị của phần tử. Kiểu object giúp lưu linh hoạt nhiều loại dữ liệu khác nhau (toán hạng, toán tử, v.v.).

Node next: Trỏ đến phần tử tiếp theo trong stack.

Phương thức:

Hàm khởi tạo Node(object value): Gán giá trị và khởi tạo liên kết.

CHƯƠNG 3. THIẾT KẾ GIAO DIỆN

3.1. Giao Diện Máy Tính

Giao diện người dùng được thiết kế nhằm đảm bảo trải nghiệm trực quan, dễ sử dụng cho sinh viên hoặc người dùng phổ thông. Giao diện không chỉ là công cụ nhập biểu thức mà còn hiển thị kết quả, quản lý lịch sử, và hỗ trợ thao tác nhanh qua các nút chức năng.

Ứng dụng sử dụng giao diện Windows Forms (WinForms) trong ngôn ngữ C#, tận dụng khả năng kéo-thả trực quan, đồng thời kết hợp xử lý sự kiện linh hoạt thông qua các hàm tương ứng.

Thành phần chính trong giao diện:

- Thanh tiêu đề (Title Panel): Hỗ trợ kéo cửa sổ ứng dụng bằng chuột.
- Ô nhập biểu thức (TextDisplay2): Cho phép người dùng nhập biểu thức toán học.
- Ô hiển thị kết quả (TextDisplay1): Hiển thị kết quả phép tính gần nhất.
- Bảng lịch sử (HistoryDisplay): Ghi lại các phép tính trước đó.
- Các nút số (0-9), toán tử (+, -, ×, ÷, ^), và đặc biệt (√, %, !, Ans, =)
- Nút chức năng khác: Xóa (Clear), xóa từng ký tự (Backspace), xóa toàn bộ (Clear All), thu nhỏ cửa sổ (Minimize), thoát (Exit).

Chức năng kéo cửa sổ, thu nhỏ và đóng ứng dụng được thực hiện thông qua các lệnh gọi đến thư viện user32.dll, nhằm tối ưu trải nghiệm người dùng trên nền tảng Windows.

3.2. Chi Tiết Chức Năng

Nhập biểu thức và toán tử: Các hàm như Num_Click, MathOperators_Click, SpecOperators_Click đảm nhiệm xử lý khi người dùng bấm nút số, toán tử cơ bản hoặc nâng cao. Tự động làm mới biểu thức nếu có kết quả cũ để tránh xung đột biểu thức.

Thực hiện phép tính: Hàm Equal_Click là nơi kết nối giao diện và thuật toán. Khi người dùng nhấn "=", biểu thức từ TextDisplay2 được chuyển sang lớp Solve để xử lý. Kết quả sau đó hiển thị ở TextDisplay1 và được thêm vào bảng lịch sử.

Lưu và xử lý lịch sử

- HistoryBtn_Click: Thu gọn hoặc mở rộng vùng lịch sử.
- ClearHistoryBtn_Click: Xóa toàn bộ lịch sử với thông báo mặc định "There is no history yet."

Các chức năng hỗ trợ khác

- Backspace_Click: Xóa ký tự cuối cùng theo ngữ cảnh đặc biệt.
- ClearEntryBtn_Click và ClearBtn_Click: Xóa nội dung biểu thức hoặc toàn bộ giao diện.

AnsBtn_Click: Thêm kết quả trước vào biểu thức hiện tại, giúp người dùng tiếp tục tính toán nhanh chóng.

CHƯƠNG 4. THẢO LUẬN & ĐÁNH GIÁ

4.1. Các Kết Quả Nhận Được

Hệ thống đã được xây dựng thành công, thực hiện chính xác các phép tính toán học theo yêu cầu đề bài. Các kết quả tính toán được lưu lại để sử dụng cho các phép toán tiếp theo, đồng thời người dùng có thể truy cập và xem lại lịch sử tính toán. Giao diện được thiết kế trực quan, dễ sử dụng, và cho phản hồi nhanh chóng.

4.2. Một Số Tồn Tại

- Chưa hỗ trợ giải các phương trình bậc hai trở lên, giải ma trận, xử lý đa thức,...
- Khi người dùng nhấn nút "=" nhiều lần với cùng một biểu thức chỉ trả về kết quả cũ mà không lặp lại phép toán trước đó.
- Giao diện lịch sử mới chỉ cho phép xem kết quả, chưa hỗ trợ thao tác tiếp tục từ kết quả cũ.
- Chưa hỗ trợ phóng to thu nhỏ máy tính.
- Một số lỗi của phép tính vẫn chưa phát hiện ra được.

4.1. Hướng Phát Triển

- Mở rộng khả năng xử lý các biểu thức toán học phức tạp hơn như phương trình hoặc biểu thức đại số.
- Bổ sung tính năng thao tác trực tiếp với kết quả trong lịch sử (nhấn để sử dụng lại).
- Bổ sung khả năng phóng to thu nhỏ máy tính.
- Cải tiến chức năng "=" để lặp lại phép tính gần nhất.
- Tách riêng giao diện tính toán cơ bản và tính toán khoa học nhằm tối ưu trải nghiệm người dùng.

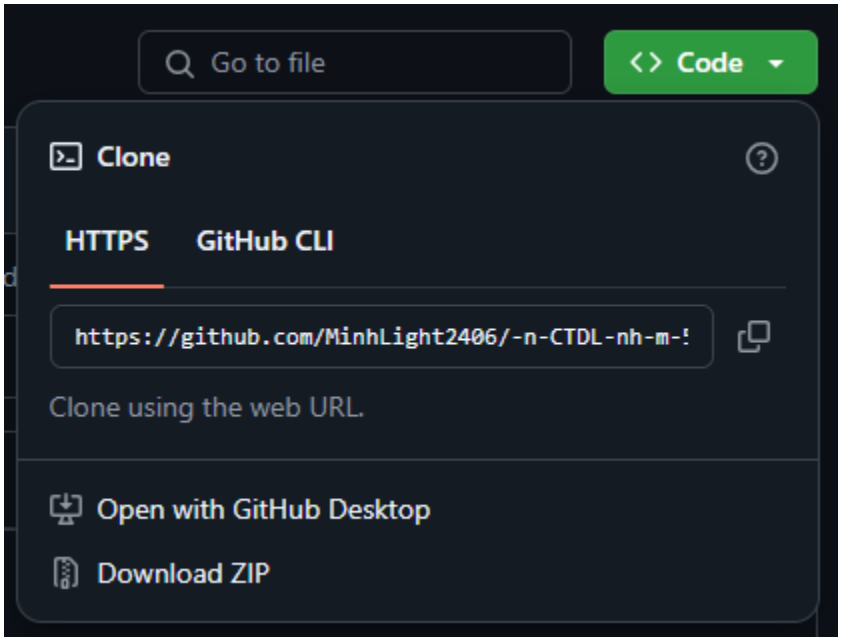
PHỤ LỤC

- Github: https://github.com/MinhLight2406/-n-CTDL-nh-m-5.git

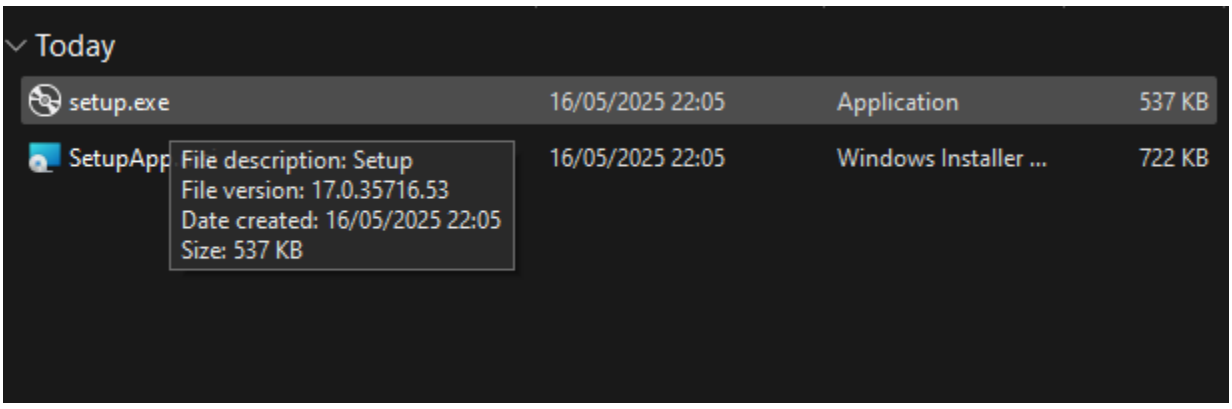
- Hướng dẫn cài đặt:

Bước 1: Vào đường link: https://github.com/MinhLight2406/-n-CTDL-nh-m-5.git

Bước 2: Bấm "< > Code" chọn "Download Zip"



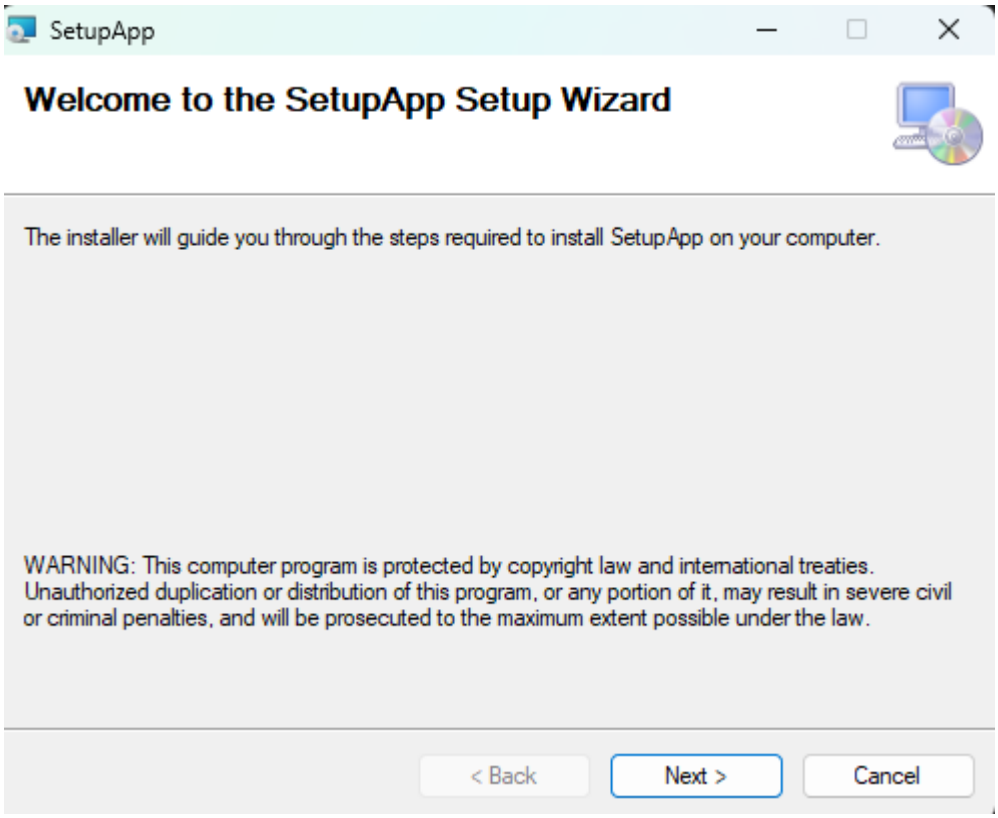
Bước 3: Chờ tải về và mở lên, mở thư mục "Đồ án cuối kì CTDL" -> "Debug" -> "setup.exe"



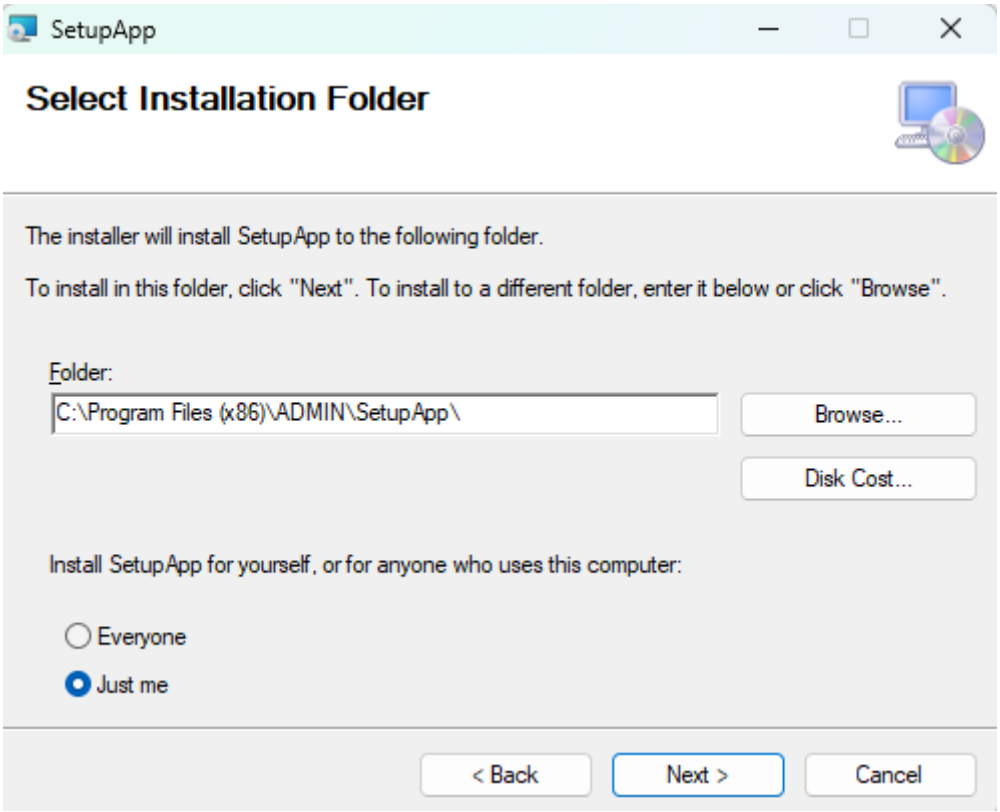
Bước 4: Chọn More Info

Bước 5: Chọn Run anyway

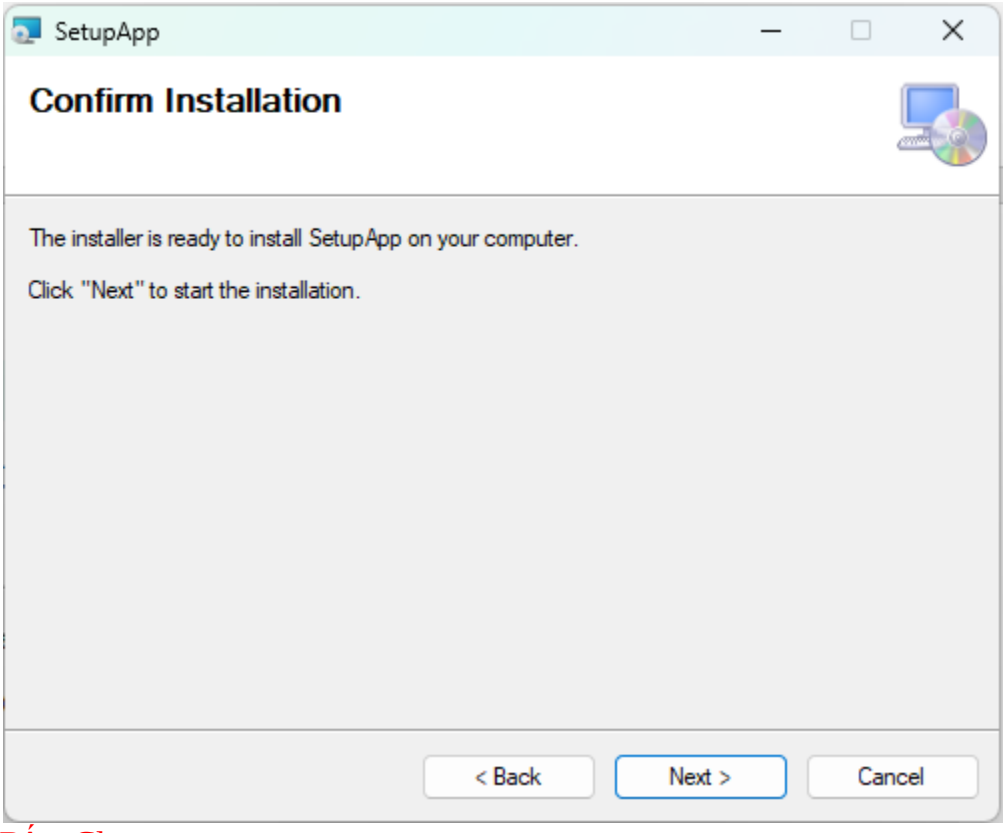
Bước 6: Chọn “Next”



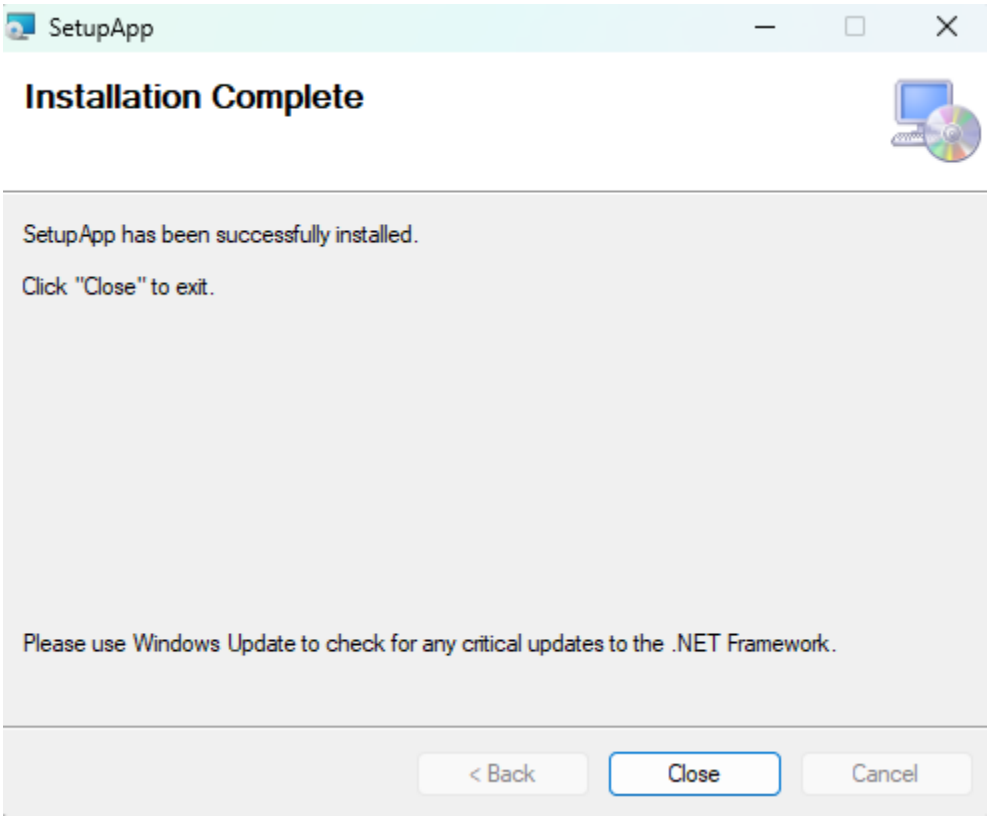
Bước 7: Bấm “Browse” để chọn địa chỉ muốn lưu



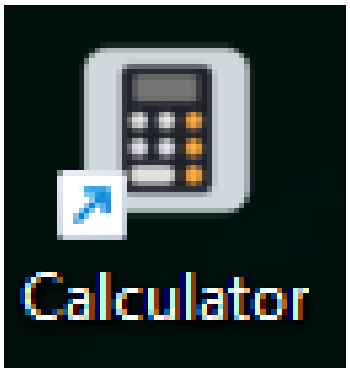
Bước 8: Bấm next để tải



Bước 9: Bấm Close



Bước 10: Trên màn hình Desktop có ứng dụng “Calculator” mở ứng dụng lên và sử dụng



- Phân Công Công Việc

Thành Viên	Nhiệm Vụ
NGUYỄN DUY THÀNH VŨ	Thiết kế lớp cho thuật toán chuyển đổi biểu thức Infix sang biểu thức Postfix Kiểm tra và tìm lỗi bug của máy tính
PHẠM TẤN TRUNG NAM	Thiết kế giao diện WinForm Thiết kế lớp Mystack Làm báo cáo nội dung chương III
PHẠM TRẦN MINH THƯ	Cài đặt các thao tác nút bấm của giao diện Kết nối thuật toán Solve với nút bấm giao diện Làm báo cáo nội dung chương I
NGÔ ĐỖ NHẬT MINH	Thiết kế lớp cho thuật toán tính toán kết quả biểu thức dạng Postfix Làm báo cáo nội dung chương II Làm báo cáo nội dung chương IV

TÀI LIỆU THAM KHẢO

1. INTRODUCTION TO ALGORITHMS (4th Edition) - THOMAS H. CORMEN, CHARLES E. LEISERSON, RONALD L. RIVEST, CLIFFORD STEIN

2. https://www.youtube.com/watch?v=6x0bUWt26z0

3. Slide bài giảng của Giảng Viên: TS. Đặng Ngọc Hoàng Thành