ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA TOÁN - TIN HỌC

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (MTH10407)

ỨNG DỤNG CẤU TRÚC DỮ LIỆU STACK ĐỂ TÍNH GIÁ TRỊ BIỂU THỨC TRUNG TỐ

Sinh viên thực hiện: Nguyễn Ngọc Sơn – 20280080
Nguyễn Văn Sơn – 20280081

Tp. Hồ Chí Minh, Tháng 06/2022

Mục lục

1 Cấu trúc dữ liệu Stack

1.1 Khái niệm Stack

Stack là một cấu trúc dữ liệu trừu tượng có 2 phương thức chính là thêm một phần tử vào đầu Stack (push) và loại bỏ một phần tử ở đầu Stack (pop). Vị trí đầu của Stack được gọi là đỉnh (top).

Có thể hình dung Stack như một chồng sách. Việc đặt một quyển sách lên trên cùng chính là thao tác thêm phần tử, và lấy ra quyển sách ở trên đầu là thao tác loại bỏ phần tử. Như vậy, quyển sách được đặt vào sau cùng sẽ luôn được lấy ra trước tiên. Vì thế, Stack sẽ hoạt động theo quy tắc LIFO (Last In - First Out hay vào sau - ra trước).

1.2 Template

1.2.1 Khái niệm

Template (khuôn mẫu) là một từ khóa trong C++, và là một kiểu dữ liệu trừu tượng tổng quát hóa cho các kiểu dữ liệu như int, float, double, bool...

Template giúp người lập trình định nghĩa tổng quát cho hàm và lớp thay vì phải nạp chồng (overloading) cho từng hàm hay phương thức với những kiểu dữ liệu khác nhau.

Template trong C++ có 2 loại đó là function template và class template.

1.2.2 Lý do nên dùng template để cài đặt Stack

Một Stack có thể được dùng với nhiều kiểu dữ liệu khác nhau. Do đó, thay vì viết overloading từng phương thức cho các kiểu dữ liệu khác nhau thì ta chỉ cần viết một phương thức template cho tất cả. Đó là lý do ta sử dụng template cho việc cài đặt Stack.

1.3 Cài đặt Stack bằng Singly Linked List

1.3.1 Khai báo cấu trúc Node

Một Node trong một Singly Linked List sẽ bao gồm 2 thành phần:

- **Thành phần dữ liệu (data)**: Lưu dữ liệu về bản thân của chính Node đó
- **Thành phần liên kết (next)**: Lưu địa chỉ phần tử đứng sau Node đó

```
template <class T>
struct Node
{
    T val;
    Node* next;
    Node() {}
    Node(T data, Node* next = nullptr): val(data), next(next){}
};
```

Listing 1: Minh họa khai báo cấu trúc của một Node

1.3.2 Khai báo lớp Stack

Sau khi khai báo cấu trúc của một Node, ta tiến hành khai báo lớp Stack với các phần tử sẽ là kiểu dữ liệu cấu trúc Node

```
template <class T>
class Stack
{
private:
    Node<T>* pHead;
    int count;
public:
    Stack(); // constructor
```

```
    Stack(const Stack&); // copy constructor
    ~Stack(); // destructor
    void push(T);
    void pop();
    T top();
    bool empty();
    int size();
};
```

Listing 2: Minh họa khai báo lớp Stack

1.3.3 Các chức năng của các hàm thành phần trong Stack

- **push**: Thêm phần tử vào đỉnh Stack
- **pop**: Xóa phần tử ở đỉnh Stack
- **top**: Trả về giá trị ở đỉnh của Stack
- **empty**: Kiểm tra xem liệu Stack có rỗng hay không
- **size**: Trả về số phần tử đang có trong Stack

2 Bài toán tính giá trị của biểu thức trung tố (Infix)

2.1 Ý tưởng

Để tính giá trị một biểu thức trung tố ta sẽ chuyển biểu thức trung tố (Infix) sang biểu thức hậu tố (Postfix). Biểu thức hậu tố là thuật toán được biểu diễn bằng cách đặt các toán tử ra sau các toán hạng.

Ví dụ: Biểu thức trung tố: $2 * 3 + 5/10 + 7$ và biểu thức hậu tố: $2\,3 * 5\,10\,/ + 7+$

Sau đó, ta áp dụng nguyên lý của Stack vào để tính toán biểu thức hậu tố. Kết quả cuối cùng là kết quả của bài toán.

2.2 Toán tử số học và một số hàm toán được áp dụng trong bài toán

2.2.1 Toán tử số học

Cộng	Trừ	Nhân	Chia	Chia lấy dư	Lũy thừa
$+$	$-$	$*$	$/$	$\%$	$\wedge$

2.2.2 Một số quy ước

$\pi = pi = 3.141593$

$e = 2.718282$

Số thập phân không có phần nguyên đằng trước sẽ được xem như phần nguyên là 0. Ví dụ: $0.967 = .967$

Số thập phân không có phần thập phân đằng sau sẽ được xem như phần thập phân là 0. Ví dụ: $5. = 5.0 = 5$

2.2.3 Hàm toán

Hàm	Chức năng
$\cos(x)$	Tính cos của x
$\sin(x)$	Tính sin của x
$\tan(x)$	Tính tan của x
$\text{acos}(x)$	Tính arccos của x
$\text{asin}(x)$	Tính arcsin của x
$\text{atan}(x)$	Tính arctan của x
$\cosh(x)$	Tính hyperbolic cos của x
$\sinh(x)$	Tính hyperbolic sin của x
$\tanh(x)$	Tính hyperbolic sin của x
$\text{acosh}(x)$	Tính diện tích hyperbolic cos của x
$\text{asinh}(x)$	Tính diện tích hyperbolic sin của x
$\text{atanh}(x)$	Tính diện tích hyperbolic tan của x
$\exp(x)$	Tính e mũ x
$\text{abs}(x)$	Tính giá trị tuyệt đối của x
$\ln(x)$	Tính logarithm cơ số e của x
$\log(x)$	Tính logarithm cơ số 10 của x
$\text{sqrt}(x)$	Tính căn bậc 2 của x
$\text{cbrt}(x)$	Tính căn bậc 3 của x

2.3 Kiến thức bổ sung khi giải quyết bài toán

2.3.1 Con trỏ hàm

Trong C++, giống như các loại con trỏ dữ liệu thông thường (int*, char*, float*...), ta cũng có một con trỏ trỏ tới những hàm. Một con trỏ hàm là một con trỏ mà giữ địa chỉ của một hàm nào đó.

Không giống những con trỏ khác, một con trỏ hàm chỉ trỏ đến một đoạn code chứ không trỏ đến một data nào vì hàm là một nội dung của một đoạn code chứ không phải là một giá trị dữ liệu. Thông thường, một con trỏ hàm sẽ lưu trữ phần bắt đầu của code thực thi.

Cú pháp khai báo của một con trỏ hàm: <Kiểu trả về> (* <Tên con trỏ>) (* <Danh sách tham số>)

```
    // The function pointer points to a function whose return type is int and takes 1 int
     argument
    int (*ptr)(int);

    void (*ptr)(int, int);
    // The function pointer points to a function whose return type is void and takes 2
    int argument
```

Listing 3: Minh họa cú pháp khai báo một con trỏ hàm

Con trỏ hàm có thể được sử dụng để gọi hàm mà nó trỏ đến

```
#include <iostream>
using namespace std;

int add(int x, int y){
    return x+y;
}

int main()
{
    // Assign to the pointer variable function of add function
    int (*ptr)(int, int) = &add;
    int x = 5, y = 5;

    // Call function add
    cout << ptr(x,y);
    return 0;
}
```

2.3.2 Ứng dụng con trỏ hàm để để gọi hàm thông qua một chuỗi

Để có thể gọi được hàm thông qua một chuỗi ta sẽ áp dụng cấu trúc dữ liệu map trong C++, nó tương tự như một bảng băm là cấu trúc dữ liệu để ánh xạ từ giá trị xác định, được gọi là khóa (key) thành giá trị tương ứng (value).

Ta sẽ truyền vào một key có giá trị là một chuỗi vào map từ đó sẽ ánh xạ đến một value là một con trỏ hàm mà con trỏ hàm này sẽ lưu địa chỉ của hàm cần gọi.

```
map<string, float (*)(float)> mp;
```

Listing 4: Một map có key là một string và ánh xạ đến một con trỏ hàm

Vậy ta chỉ cần truyền vào một chuỗi bất kì vào map như vậy ta sẽ gọi được hàm mà ta mong muốn. Tận dụng điều này ta sẽ áp dụng vào bài toán chính để tính giá trị của biểu thức trung tố khi trong biểu thức có chứa những hàm toán thông dụng như sqrt, exp, log, sin, cos...

Ví dụ: Tính giá trị $cos(3)$ khi truyền vào một chuỗi "*cos*"

```
#include<iostream>
#include<map>
#include<cmath>
using namespace std;

float cal_cos(float x){
    return cos(x);
}

int main()
{
   map<string, float (*)(float)> mp;
   mp["cos"] = &cal_cos;
   cout << mp["cos"](3) << endl;
   return 0;
}
```

2.4 Giải quyết bài toán

Tạo lớp **Solution** để minh họa các bước thực hiện:

```
class CalculateInfix_Solution {
public:
  bool isDigit(char c);
  bool isLetter(char c);
  int precedence(string c);
  vector<string> StringHandling(string& str);
  vector<string> InfixToPostfix(vector<string> s);
  float calculatePostfix(vector<string> s);
};
```

- **isDigit**: Trả về true nếu ký tự c thuộc dạng số từ '0' đến '9'
- **isLetter**: Trả về true nếu ký tự c thuộc dạng chữ từ 'a' đến 'z'
- **precedence**: Trả về mức độ ưu tiên

Toán tử	(	+	−	∗	/	%	∧	Hàm
Độ ưu tiên	0	1	1	2	2	2	3	4

- **StringHandling**: Hàm xử lý chuỗi
- **InfixToPostfix**: Hàm chuyển biểu thức từ trung tố sang hậu tố
- **calculatePostfix**: Hàm tính giá trị hậu tố

2.4.1 Xử lý chuỗi

Khi người dùng nhập vào một chuỗi bất kì ta phải xử lý chuỗi đó mục đích là để thực hiện các bước tiếp theo dễ dàng hơn.

Ví dụ: str = "2+3.5*3+5.6" sau khi xử lý chuỗi ta được output = ["2", "+", "3.5", "*", "3", "+", "5.6"]

Dữ liệu đầu vào: biểu thức trung tố do người dùng nhập vào

2.4.1.a Các trường hợp xảy ra và cách xử lý

1. Nếu trong chuỗi có tồn tại các trường hợp như các ví dụ dưới đây:
   - "5 4"
   - " 5 . 4"
   - ". 5 4"
   - "5 4 ."
   - " *sq r t* (4)"

   ⇒ **Thông báo lỗi và kết thúc chương trình**
2. Nếu chuỗi có khoảng trắng thì xóa khoảng trắng của chuỗi str mà người dùng nhập vào bằng cách dùng hàm **erase** trong **STL** <**string**> và kết hợp với hàm **remove** trong thư viện **algorithm** để xóa đi những khoảng trắng của toàn bộ chuỗi.

```
    str.erase(remove(str.begin(), str.end(), ' '), str.end());

```

3. Nếu phần tử đầu là kí tự giai thừa '!' thì **thông báo lỗi và kết thúc chương trình**
4. Nếu như ở trong chuỗi có một chuỗi con là "()" ở vị trí bất kì thì **thông báo lỗi và kết thúc chương trình**

```
    size_t found = str.find("()");
    if (found != string::npos)
        throw "Something wrong with expression!";

```

5. Nếu như ở vị trí đầu chuỗi có tồn tại một dãy dấu '+' hoặc '−'
   - Duyệt cho đến khi không còn gặp dấu '+' hoặc '−' và đếm số dấu '−' xuất hiện
   - Số dấu '−' là số lẻ thì thêm số 0 và dấu '−' lần lượt vào output.

   Ví dụ:

   Chuỗi "− − + − + − − + + 2 + 5" sau khi duyệt đến số 2 ta đếm được số dấu '−' bằng 5 là số lẻ nên output = ["0", "−"]

   Minh họa ý tưởng: $- - + - + - - + +2 + 5 = 0 - 2 + 5$
6. Nếu như ở vị trí giữa chuỗi bất kỳ có tồn tại một dãy dấu '+' hoặc '−'
   - Duyệt cho đến khi không còn gặp dấu '+' hoặc '−' và đếm số dấu '−' xuất hiện
   - Tiếp tục bước tiếp theo bên dưới
7. Nếu như kí tự đang xét là dấu '+' hoặc '−', và ở phần tử cuối của output là dấu '*' hoặc '/'
   - Thêm dấu '*' vào vị trí trước phần tử cuối của output
   - Nếu kí tự đang xét là dấu '+' thì thêm số 1 vào vị trí trước phần tử cuối của output. Còn nếu kí tự đang xét là dấu '+'thì thêm số −1

Ví dụ:

Chuỗi "$2+3/-2$" có phần tử đang xét str[i] = '$-$' và output = ["2", "+", "3", "/"]

Sau khi thực hiện bước trên ta được output = ["2", "+", "3", "$*$", "-1", "/"]

Minh họa ý tưởng: $2+3/-2 = 2+3*-1/2$

8. Nếu như kí tự đang xét là dấu '+' hoặc '$-$', và ở phần tử cuối của output là dấu '('
   - Thêm số 0 trước rồi mới thêm dấu '+' hoặc '$-$' vào output

   Ví dụ:

   Chuỗi "$5+(-2+3)$" có phần tử đang xét str[i] = '$-$' và output = ["5", "+", "("]

   Sau khi thực hiện bước trên ta được output = ["5", "+", "(", "0", "$-$"]

   Minh họa ý tưởng: $5+(-2+3) = 5+(0-2+3)$

9. Nếu như kí tự đang xét là dấu '$-$', và ở phần tử cuối của output là dấu '$\wedge$'
   - Thêm dấu '(' vào vị trí cách phần tử cuối của output 2 vị trí
   - Thêm dấu '1' vào vị trí cách phần tử cuối của output 2 vị trí
   - Thêm dấu '/' vào vị trí cách phần tử cuối của output 2 vị trí
   - Thêm dấu ')' vào vị trí trước phần tử cuối của output

   Ví dụ:

   Chuỗi "$2\wedge-2$" có phần tử đang xét str[i] = '$-$' và output = ["2", "$\wedge$"]

   Sau khi thực hiện các bước trên ta được output = ["(", "1", "/", "2", ")", "$\wedge$"]

   Minh họa ý tưởng: $2^{-2} = \left(\frac{1}{2}\right)^2$

10. Nếu như kí tự đang xét là kí tự giai thừa '!', và ở phần tử cuối output là 1 trong các dấu '(', '+', '$-$', '$*$', '/', '%', '$\wedge$' hoặc là 1 hàm toán thì **thông báo lỗi và kết thúc chương trình**

    Ví dụ:

    $2+!5 \Rightarrow Sai$

    $(!5+3)-2 \Rightarrow Sai$

    $8+abs!5 \Rightarrow Sai$

11. Nếu như kí tự đang xét là kí tự dạng chữ từ 'a' đến 'z'
    - Khởi tạo chuỗi rỗng và thêm vào cuối của output
    - Duyệt cho đến khi không còn gặp kí tự dạng chữ từ 'a' đến 'z' và thêm lần lượt vào phần tử cuối của output

    Ví dụ:

    Chuỗi "$2+sqrt(4)$" có phần tử đang xét str[i] = 's' và output = ["2", "+"]

    Sau khi thực hiện các bước trên ta được output = ["2", "+", "$sqrt$"]

12. Nếu như kí tự đang xét là kí tự dạng số từ '0' đến '9'
    - Thêm chuỗi rỗng vào cuối của output
    - Duyệt cho đến khi không còn gặp kí tự dạng số từ '0' đến '9' và thêm lần lượt những kí tự đó vào phần tử cuối của output

- Nếu như sau khi duyệt xong phần tử hiện tại là dấu '.' ta lại thực hiện duyệt như trên với vị trí bắt đầu duyệt là vị trí tiếp theo của dấu '.'

Ví dụ:

Chuỗi "4 + 25.35 + 7" có phần tử đang xét str[i] = '2' và output = ["4", " + "]

Sau lần duyệt đầu tiên ta được output = ["4", " + ", "25"].Vì sau lần duyệt đầu tiên ta được phần tử hiện tại là str[i] = '.' nên ta tiếp tục thực hiện duyệt lần 2 với phần tử tiếp theo str[i] = '3' ta được output = ["4", " + ", "25.35"]

13. Nếu như kí tự đang xét là dấu '.'

- Thêm chuỗi rỗng vào cuối của output
- Thêm dấu '.' vào phần tử cuối của output
- Xét tiếp phần tử tiếp theo dấu '.' và duyệt cho đến khi không còn gặp kí tự dạng số từ '0' đến '9' và thêm lần lượt những kí tự đó vào phần tử cuối của output

Ví dụ:

Chuỗi "2 + .54 + 3" có phần tử đang xét str[i] = '.' và output = ["2", " + "]

Sau khi thực hiện các bước trên ta được output = ["2", " + ", ".54"]

Minh họa ý tưởng: $2 + .54 + 3 = 2 + 0.54 + 3$

2.4.1.b Minh họa thuật toán

Input: " + −5.6 + 2/ − 3 + −4"

String Handling

i	str[i]	Output
0	+	[]
1	−	0 −
2	5	0 − 5
3	.	0 − 5.
4	6	0 − 5.6
5	+	0 − 5.6 +
6	2	0 − 5.6 + 2
7	/	0 − 5.6 + 2 /
8	−	0 − 5.6 + 2 ∗ −1 /
9	3	0 − 5.6 + 2 ∗ −1 / 3
10	+	0 − 5.6 + 2 ∗ −1 / 3
11	−	0 − 5.6 + 2 ∗ −1 / 3 −
12	4	0 − 5.6 + 2 ∗ −1 / 3 − 4

Final output: 0 − 5.6 + 2 ∗ −1 / 3 − 4

2.4.2 Chuyển biểu thức trung tố sang hậu tố

Sau khi thực hiện xong bước **xử lý chuỗi** ta tiến hành bước chuyển biểu thức trung tố sang hậu tố

Dữ liệu đầu vào: Biểu thức trung tố sau khi đã được xử lý

2.4.2.a Các trường hợp xảy ra và cách xử lý

1. Khởi tạo Stack rỗng chứa các phần tử có kiểu dữ liệu là string
2. Nếu phần tử đang xét s[i] là một chuỗi chứa dạng 1 số
   - Thêm s[i] vào output

3. Nếu phần tử đang xét s[i] là "("
   - Push s[i] vào Stack
4. Nếu phần tử đang xét s[i] là ")" và trong Stack có chứa những hàm toán học ("*sqrt*", "*log*", "*exp*"...)
   - Lấy giá trị top của Stack thêm vào output và pop, lặp lại thao tác như vậy cho đến khi giá trị top là dấu "("
   - Pop dấu "(" ra khỏi Stack
   - Nếu đỉnh của Stack đang là những hàm toán học thì thêm vào output. Sau đó dừng và tiếp tục kiểm tra điều kiện lặp tiếp theo.
   - Nếu phần tử tiếp theo là s[i+1] = "$\wedge$" thì đưa vào output đỉnh của Stack. Sau đó dừng và tiếp tục kiểm tra điều kiện của lần lặp tiếp theo.
5. Nếu phần tử đang xét s[i] là ")" và trong Stack **không** chứa những hàm toán học
   - Lấy giá trị top của Stack thêm vào output và pop, lặp lại thao tác như vậy cho đến khi giá trị top là dấu "("
   - Nếu Stack rỗng có nghĩa là biểu thức thiếu dấu "(" nên kết thúc chương trình và thông báo lỗi. Ngược lại pop dấu "(" ra khỏi Stack
6. Độ ưu tiên của phần tử top trong Stack nhỏ hơn độ ưu tiên của phần tử s[i] hoặc cùng là dấu "$\wedge$"
   - Push s[i] vào Stack
7. Độ ưu tiên của phần tử top trong Stack lớn hơn hoặc bằng độ ưu tiên của phần tử s[i]
   - Lấy giá trị top của Stack thêm vào output và pop, lặp lại thao tác như vậy cho đến khi Stack rỗng hoặc độ ưu tiên của phần tử top trong Stack nhỏ hơn độ ưu tiên của phần tử s[i].
   - Push s[i] vào Stack
8. Nếu đã duyệt xong mảng và Stack vẫn chưa rỗng
   - Lấy giá trị top của Stack thêm vào output và pop, lặp lại thao tác như vậy cho đến khi Stack rỗng

2.4.2.b Minh họa thuật toán

Input: (5 + 2 * 2) − sqrt (4)

Infix to Postfix

i	s[i]	Stack	Output
0	(	(	Empty
1	5	(	5
2	+	(+	5
3	2	(+	5 2
4	*	(+ *	5 2
5	2	(+ *	5 2 2
6	)	Empty	5 2 2 * +
7	−	−	5 2 2 * +
8	sqrt	− sqrt	5 2 2 * +
9	(	− sqrt (	5 2 2 * +
10	4	− sqrt (	5 2 2 * + 4
11	)	−	5 2 2 * + 4 sqrt
12		Empty	5 2 2 * + 4 sqrt −

Final output: 5 2 2 * + 4 sqrt −

2.4.3 Tính biểu thức hậu tố

Bước cuối cùng của bài toán là tính giá trị biểu thức hậu tố sau khi đã chuyển biểu thức trung tố sang hậu tố

Dữ liệu đầu vào: Biểu thức hậu tố

2.4.3.a Các trường hợp xảy ra và cách xử lý

1. Nếu trong Stack chỉ có 1 phần tử và s[i] là các toán tử 2 ngôi như "+", "−"', "∗", "/", "%", "∧"
   - Kết thúc chương trình và thông báo lỗi
2. Nếu phần tử đang xét s[i] là một chuỗi chứa dạng 1 số
   - Push s[i] vào Stack
3. Nếu phần tử đang xét s[i] là một hàm toán
   - Lấy giá trị top của Stack truyền vào hàm đó rồi sau đó lưu kết quả mới vào top.
4. Nếu trong Stack chỉ có 1 phần tử và s[i] là các toán tử 2 ngôi như "+", "−"', "∗", "/", "%", "∧"
   - Pop Stack và lưu giá trị mà vừa pop ra bên ngoài
   - Lấy giá trị đó thực hiện phép tính với giá trị top hiện tại của Stack và lưu kết quả mới vào top đó
5. Nếu phần tử đang xét s[i] là các kí tự đặc biệt
   - Kết thúc chương trình và thông báo lỗi
6. Nếu đã duyệt xong mảng.
   - Nếu trong Stack vẫn còn nhiều hơn 1 phần tử hoặc là Stack rỗng thì kết thúc chương trình và thông báo lỗi. Ngược lại, trả về kết quả là 1 phần tử duy nhất còn lại trong Stack.

2.4.3.b Minh họa thuật toán

Input: 5 2 2 ∗ + 4 sqrt −			
Calculate Postfix			
i	**s[i]**	**Action**	**Stack**
0	5	Push 5	5
1	2	Push 2	5 2
2	2	Push 2	5 2 2
3	∗	Pop $(2, 2)$ và tính $2 * 2 = 4$. Sau đó Push 4	5 4
4	+	Pop $(4, 5)$ và tính $5 + 4 = 4$. Sau đó Push 9	9
5	4	Push 4	9 4
6	sqrt	Pop 4 và tính sqrt$(4) = 2$. Sau đó Push 2	9 2
7	−	Pop $(2, 9)$ và tính $9 - 2 = 7$. Sau đó Push 7	7
Final output: 7			

3 Tổ chức các file chính

- **MathFunctions.h**: Khai báo các hàm toán

```
#ifndef MathFunctions_h
#define MathFunctions_h
#include<cmath>
#include<vector>

float cal_cos(float);
float cal_sin(float);
```

```
float cal_tan(float);
float cal_acos(float);
float cal_asin(float);
float cal_atan(float);
float cal_cosh(float);
float cal_sinh(float);
float cal_tanh(float);
float cal_acosh(float);
float cal_asinh(float);
float cal_atanh(float);
float cal_exp(float);
float cal_abs(float);
float cal_log(float);
float cal_log10(float);
float cal_sqrt(float);
float cal_cbrt(float);
float cal_factorial(float);

#endif

```

- **MathFunctions.cpp**: Cài đặt chi tiết các hàm đã khai báo trong file **MathFunctions.h**
- **Stack.h**: Khai báo và cài đặt chi tiết cấu trúc dữ liệu Stack

```
#ifndef Stack_h
#define Stack_h
#include<iostream>

template <class T>
struct Node
{
    T val;
    Node* next;
    Node() {}
    Node(T data, Node* next = nullptr): val(data), next(next){}
};

template <class T>
class Stack {
private:
    Node<T>* pHead;
    int count;
public:
    Stack();
    Stack(const Stack&);
    ~Stack();
    void push(T);
    void pop();
    T top();
    bool empty();
    int size();
};

template <class T>
Stack<T>::Stack() : pHead(nullptr), count(0) {}

template <class T>
Stack<T>::Stack(const Stack& scr)
{
    this->count = scr.count;
    pHead = new Node<T>(scr.pHead->val);
    Node<T>* temp = scr.pHead->next;
    while(temp)
    {
        push(temp->val);
        temp = temp->next;
    }
}

template <class T>
```

```cpp
bool Stack<T>::empty()
{
    return (pHead == nullptr);
}

template <class T>
void Stack<T>::push(T x)
{
    Node<T>* p = new Node<T>(x,pHead);
    pHead = p;
    count++;
}

template <class T>
void Stack<T>::pop()
{
    if (empty())
    {
        std::cout << "Empty Stack!" << std::endl;
        exit(0);
    }
    Node<T>* temp = pHead;
    pHead = pHead->next;
    delete temp;
    count--;
}

template <class T>
T Stack<T>::top()
{
    if (empty())
    {
        std::cout << "Empty Stack!" << std::endl;
        exit(0);
    }
    return pHead;
}

template <class T>
Stack<T>::~Stack()
{
    while (!empty())
        pop();
    pHead = nullptr;
}

template <class T>
int Stack<T>::size()
{
    return count;
}

#endif

```

- **CalculateInfix_Solution.h**: Khai báo các hàm là các bước để giải quyết bài toán

```cpp
#ifndef CalculateInfix_Solution_h
#define CalculateInfix_Solution_h

#include "MathFunctions.h"
#include "Stack.h"
#include<string>
#include<vector>
#include<map>
#include<algorithm>
using namespace std;

class CalculateInfix_Solution {
public:
  bool isDigit(char c);
```

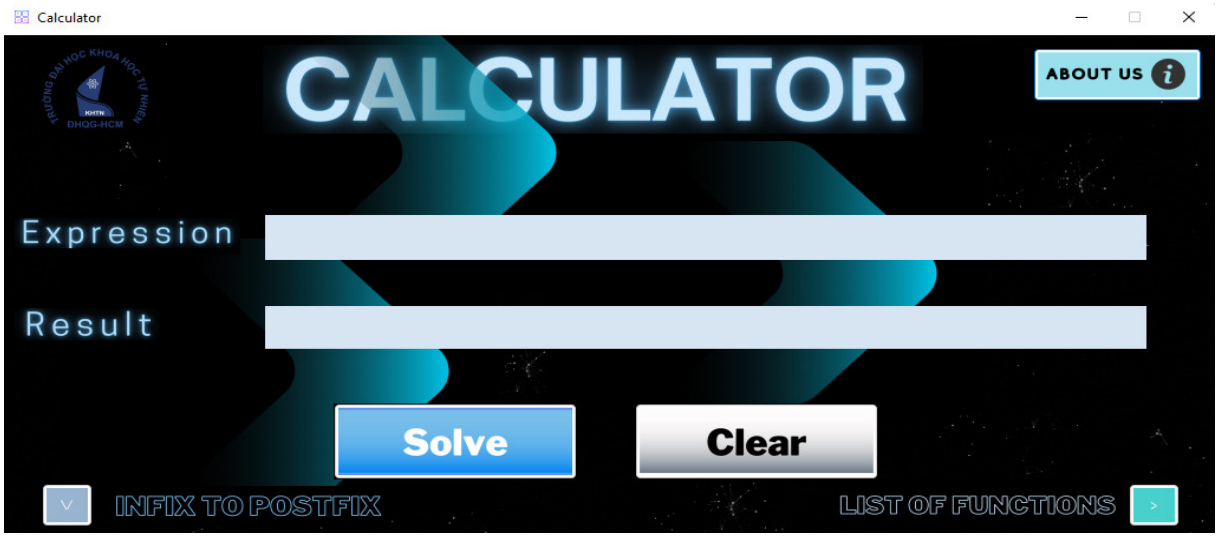
```
15    bool isLetter(char c);
16    int precedence(string c);
17    vector<string> StringHandling(string& str);
18    vector<string> InfixToPostfix(vector<string> s);
19    float calculatePostfix(vector<string> s);
20 };
21
22 #endif
```

- **CalculateInfix_Solution.cpp**: Cài đặt chi tiết các hàm đã khai báo trong file **CalculateInfix_Solution.h**

4 Thiết kế ứng dụng Calculator bằng Windows Form

4.1 Giao diện ban đầu

- Chạy file thực thi **.exe** ta được giao diện ban đầu

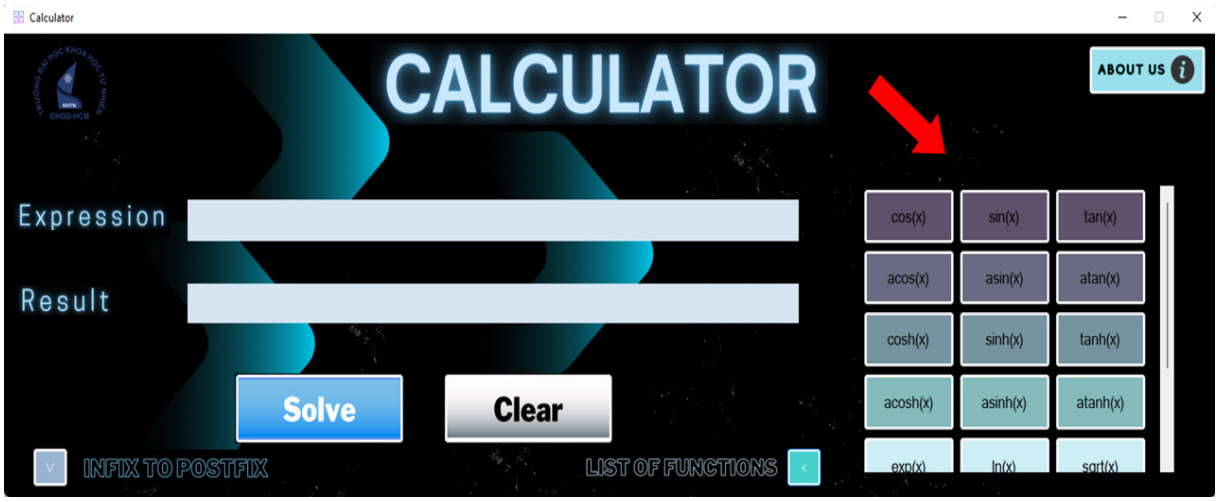


- Để biết thông tin thành viên ta click vào ô ở góc trên bên phải

4.2 Danh sách các hàm và hằng số

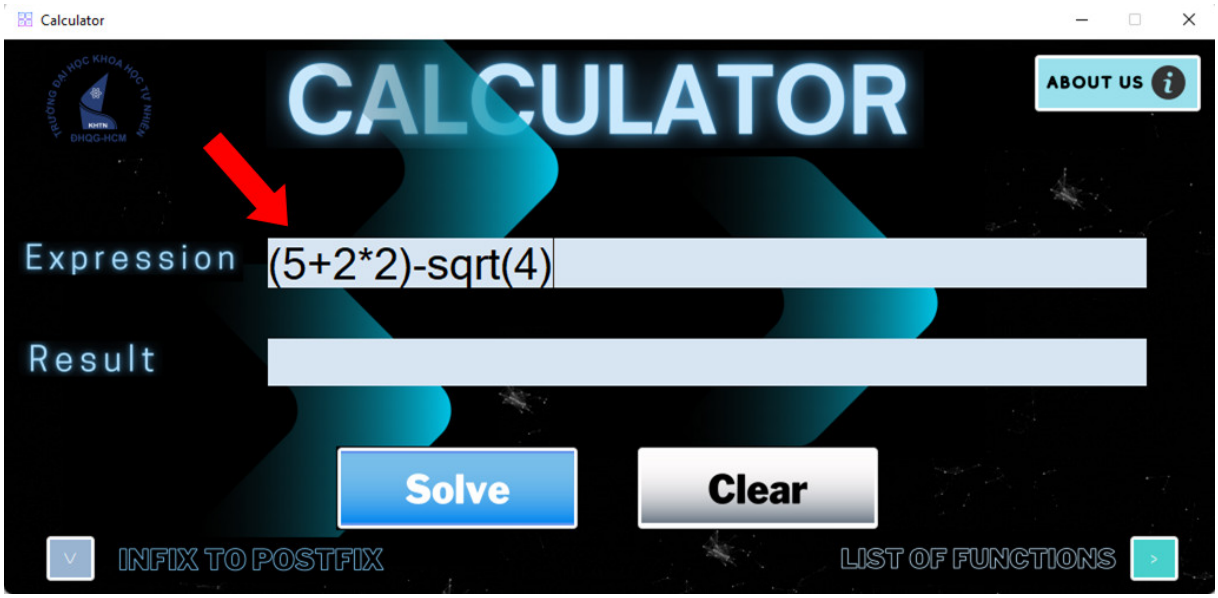
- Nếu người dùng click vào ô ở góc dưới bên phải, cạnh chữ **LIST OF FUNCTIONS**

- Ứng dụng sẽ hiện ra một danh sách các hàm và hằng số để có thể click chọn

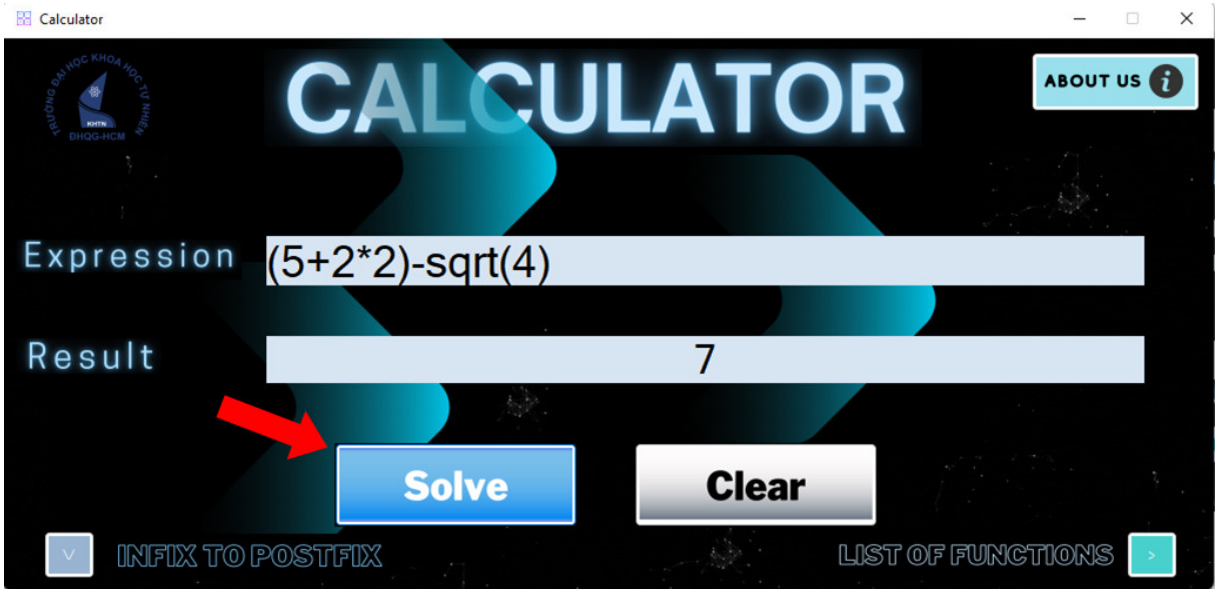


4.3 Tính giá trị biểu thức trung tố

- Để nhập biểu thức ta nhập vào ô **Expression**



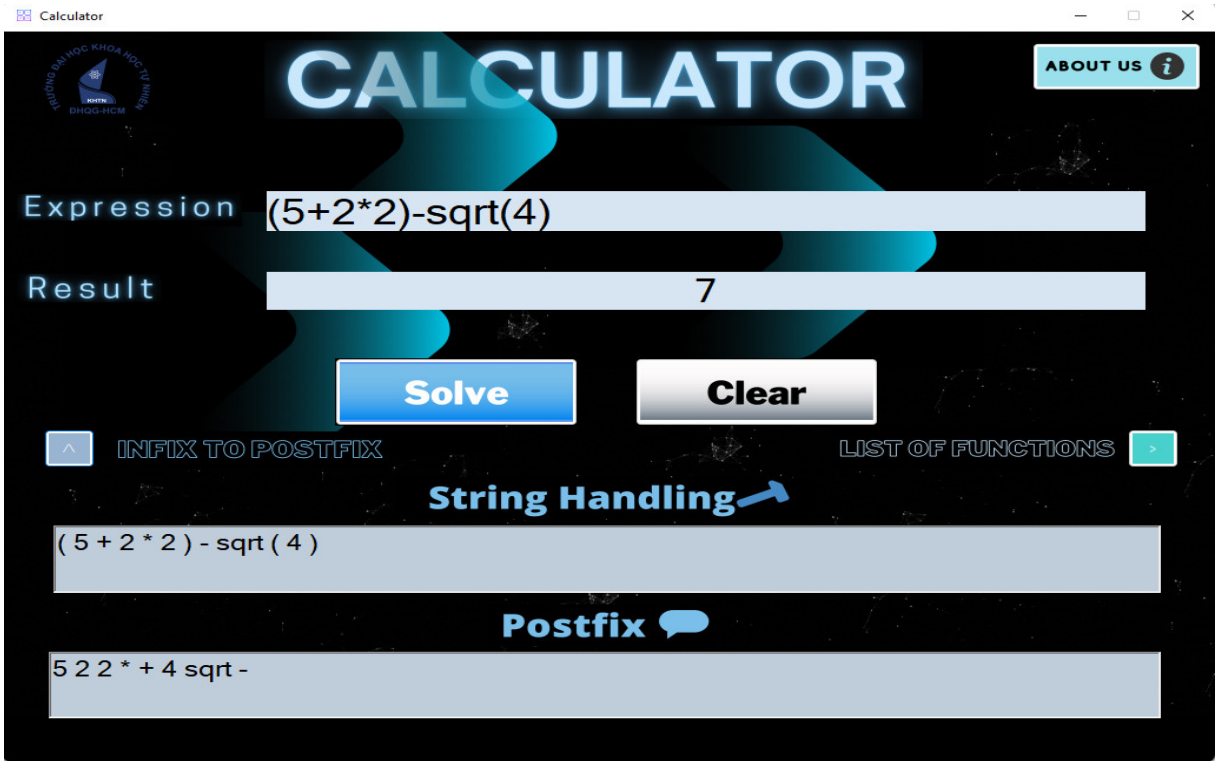
- Để tính kết quả ta click vào ô **Solve** hoặc bấm **Enter** và kết quả sẽ hiện ở ô **Result**



4.4 Kết quả các bước giải quyết bài toán

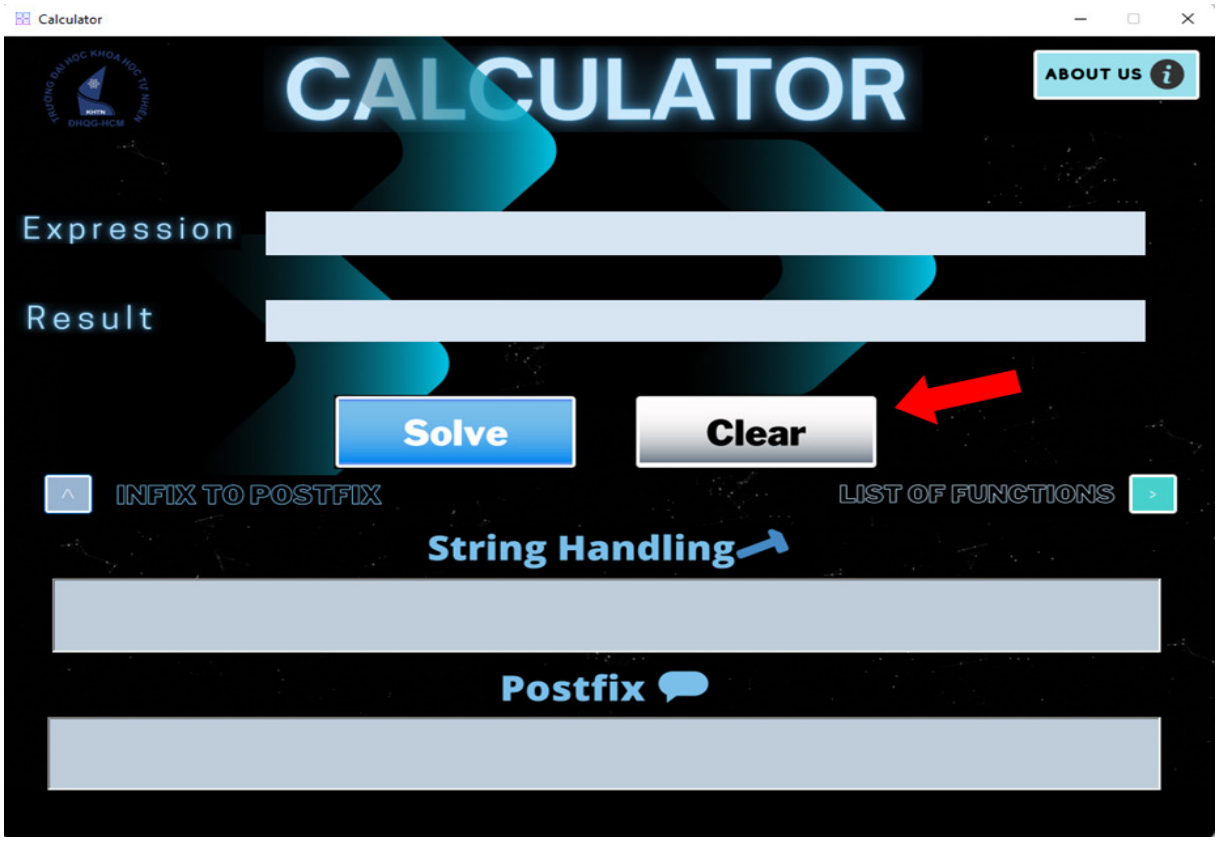
- Nếu người dùng click vào ô ở góc dưới bên trái cạnh chữ **INFIX TO POSTFIX**

- Ứng dụng sẽ cho ra kết quả của quá trình xử lý chuỗi **(String Handling)** và kết quả của biểu thức trung tố sang hậu tố **(Postfix)**



4.5 Xóa biểu thức và kết quả hiển thị

- Để xóa hết tất cả ta click vào ô **Clear**



5 Kết luận

5.1 Đánh giá

Ứng dụng tính giá trị biểu thức trung tố cơ bản đã hoàn thành được những công việc sau:

- Tính được các biểu thức trung tố từ cơ bản đến phức tạp
- Xử lý được các trường hợp khi người dùng nhập một biểu thức không hợp lệ
- Đưa được một số các hàm toán thông dụng vào ứng dụng
- Xây dựng ứng dụng bằng Windows Form để trực quan hóa bài toán hơn

5.2 Nhận xét

Ứng dụng không đánh mạnh về đồ họa, hiệu ứng mà chủ yếu tập trung vào thuật toán, cách tổ chức các hướng tiếp cận bài toán và đưa ra hướng giải quyết một cách hợp lý, cũng như là biết vận dụng được cấu trúc dữ liệu Stack vào bài toán. Nhóm em đã tận dụng tất cả những kiến thức của môn Lập trình hướng đối tượng cùng với môn Cấu trúc dữ liệu và giải thuật để đưa ra hướng giải quyết bài toán một cách tối ưu nhất có thể.

Nhìn chung là một ứng dụng có cách sử dụng vì đơn giản là tính toán giá trị biểu thức nhưng tìm ra hướng giải không phải là một việc dễ dàng. Nhóm em đã phải trải qua một quá trình khó khăn mới đạt được đích đến cuối cùng. Hi vọng rằng ứng dụng của bọn em có thể giúp ích được cho mọi người.