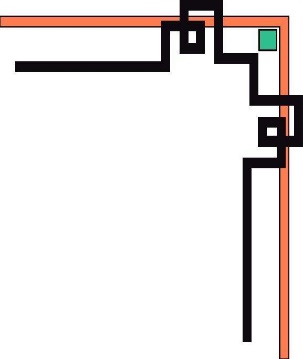
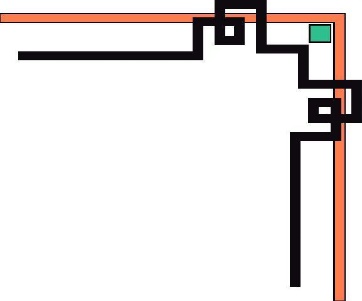
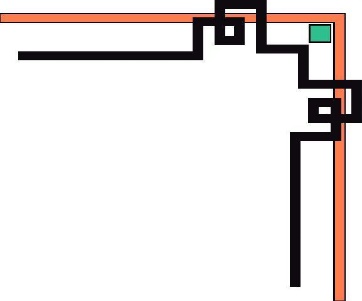
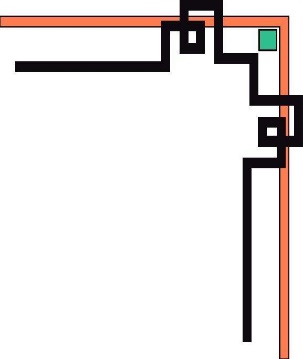
**MINISTRY OF EDUCATION AND TRAINING**



**HO CHI MINH UNIVERSITY OF SCIENCE**

**FACULTY OF INFORMANTION TECHNOLOGY \_ K20**

**Data Structures and Algorithms**

**PROJECT**



**ARITHMETIC EXPRESSION CALCULATION**

**Theory lecturer: Ngô Minh Nhựt**

**Practical lecturer: Bùi Huy Thông, Trần Thị Thảo Nhi**

**Student:**

**Trần Tiến Hoàng – 20127424**

**Lâm Kim Nhân – 20127579**

**Lê Võ Huỳnh Thanh - 20127072**

**Class: 20CLC09**

**Table of contents**

[WORK ASSIGNMENT 2](#_Toc75081066)

[CHAPTER 1: OVERVIEW 3](#_Toc75081067)

[I. Complete 3](#_Toc75081068)

[II. What we learn 3](#_Toc75081069)

[CHAPTER 2: DATA STRUCTURE 4](#_Toc75081070)

[1.1 Research 4](#_Toc75081071)

[ Converting an infix expression to a prefix expression. 4](#_Toc75081072)

[ Converting an infix expression to a postfix expression. 5](#_Toc75081073)

[ Converting a prefix expression to a postfix expression. 7](#_Toc75081074)

[1.2 Programming. 7](#_Toc75081075)

[REFERENCES 15](#_Toc75081076)

# WORK ASSIGNMENT

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | Job Description | Work progress |
| 20127424 | Trần Tiến Hoàng | Set up Stack programming.  Functional programming:   * string postfix(string s). * int Rank(char c). * float calculation(string s). | Accomplishment |
| 20127579 | Lâm Kim Nhân | * Programming main function * Presenting research in 1.1 how to convert infix expressions to suffixes. * Write project report. | Accomplishment |
| 20127072 | Lê Võ Huỳnh Thanh | * Programming fuction bool CheckExpression(string s) * Presenting research 1.1 how to convert infix expressions to prefixes and prefix expressions to suffixes * Write project report. | Accomplishment |

# CHAPTER 1: OVERVIEW

## Complete

1.1 Research

* Converting an infix expression to a prefix expression.
* Converting an infix expression to a postfix expression.
* Converting a prefix expression to a postfix expression.

1.2 Coding

* Complete program to convert expression infix to prefix expression
* The result that appears is the execution number from the expression suffix

## What we learn

* Teamwwork skill.
* Understand how to convert infix expressions to prefixes, suffixes, or prefixes to suffixes.
* Know more new libraries (stack, iomanip, ..).

# CHAPTER 2: DATA STRUCTURE

## 1.1 Research

### Converting an infix expression to a prefix expression.

Thuật toán:

Khởi tạo một stack rỗng và vị trí i ban đầu bằng 0.

Khi biểu thức đầu vào là biểu thức đúng, đảo ngược biểu thức trung tố lại sau đó bắt đầu duyệt i, lấy giá trị kí tự tại vị trí i:

* Nếu là toán hạng: output.
* Nếu là dấu đóng ngoặc “)“ thì cho vào stack
* Nếu là dấu mở ngoặc “(” ta sẽ lấy các toán tử trong stack ra và cho vào output cho đến khi gặp dấu đóng ngoặc “)“. Lúc này dâú ngoặc sẽ được đưa ra khỏi stack nhưng không xuất hiện ở ouput.
* Nếu là toán tử: Chừng nào ở đỉnh stack là toán tử và toán tử đó có độ ưu tiên lớn hơn hoặc bằng toán tử hiện tại thì lấy toán tử đó ra khỏi stack và cho vào output. Sau đó đưa toán tử hiện tại vào stack. Khi đã duyệt hết biểu thức infix, nếu trong stack còn phần tử thì lấy các giá trị kí tự trong đó ra và cho lần lượt vào output.
* Cuối cùng là đảo ngược biểu thức một lần nữa và ta sẽ thu được kết quả.

Ví dụ

* Chuyển biểu thức trung tố ((A+B)\*C)/(D-E) biểu thức hậu tố:

Đầu tiên đảo ngược biểu thức trung tố )E-D(/)C\*)B+A(( sau đỏ chuyển sang hậu tố

|  |  |  |
| --- | --- | --- |
| **Infix** | **Stack** | **Prefix** |
| **)E-D(/)C\*)B+A((** | {Empty} | **{Empty}** |
| **E-D(/)C\*)B+A((** | ) | **{Empty}** |
| **-D(/)C\*)B+A((** | ) | **E** |
| **D(/)C\*)B+A((** | )- | **E** |
| **(/)C\*)B+A((** |  | **ED-** |
| **/)C\*)B+A((** |  | **ED-** |
| **)C\*)B+A((** | / | **ED-** |
| **C\*)B+A((** | /) | **ED-** |
| **\*)B+A((** | /) | **ED-C** |
| **)B+A((** | /)\* | **ED-C** |
| **B+A((** | /)\*) | **ED-C** |
| **+A((** | /)\*) | **ED-CB** |
| **A((** | /)\*)+ | **ED-CB** |
| **((** | /)\*)+ | **ED-CBA** |
| **(** | /)\* | **ED-CBA+** |
| **{Empty}** | / | **ED-CBA+\*** |
| **{Empty}** | {Empty} | **ED-CBA+\*/** |
| **{Empty}** | {Empty} | **/\*+ABC-DE** |

Sau đó đảo ngược biểu thức đã chuyển sẽ ra được biểu thức đổi từ trung tố sang tiền tố.

### Converting an infix expression to a postfix expression.

Thuật toán:

Khởi tạo một stack rỗng và vị trí i ban đầu bằng 0.

Khi biểu thức đầu vào là biểu thức đúng, bắt đầu duyệt i, lấy giá trị kí tự tại vị trí i:

* Nếu là toán hạng: output.
* Nếu là dấu mở ngoặc “(“ thì cho vào stack
* Nếu là dấu đóng ngoặc “)” ta sẽ lấy các toán tử trong stack ra và cho vào output cho đến khi gặp dấu mở ngoặc “(“. Lúc này dâú ngoặc sẽ được đưa ra khỏi stack nhưng không xuất hiện ở ouput.
* Nếu là toán tử: Chừng nào ở đỉnh stack là toán tử và toán tử đó có độ ưu tiên lớn hơn hoặc bằng toán tử hiện tại thì lấy toán tử đó ra khỏi stack và cho vào output. Sau đó đưa toán tử hiện tại vào stack. Khi đã duyệt hết biểu thức infix, nếu trong stack còn phần tử thì lấy các giá trị kí tự trong đó ra và cho lần lượt vào output.

Ví dụ:

Chuyển biểu thức trung tố A+(B-C)\*D+E/(F\*G) sang biểu thức hậu tố:

|  |  |  |
| --- | --- | --- |
| **Infix** | **Stack** | **Postfix** |
| **A+(B-C)\*D+E/(F\*G)** | {Empty} | **{Empty}** |
| **+(B-C)\*D+E/(F\*G)** | A | **{Empty}** |
| **(B-C)\*D+E/(F\*G)** | + | **A** |
| **B-C)\*D+E/(F\*G)** | +( | **A** |
| **-C)\*D+E/(F\*G)** | +( | **AB** |
| **C)\*D+E/(F\*G)** | +(- | **AB** |
| **)\*D+E/(F\*G)** | +(- | **ABC** |
| **\*D+E/(F\*G)** | + | **ABC-** |
| **D+E/(F\*G)** | +\* | **ABC-** |
| **+E/(F\*G)** | +\* | **ABC-D** |
| **E/(F\*G)** | + | **ABC-D\*+** |
| **/(F\*G)** | + | **ABC-D\*+E** |
| **(F\*G)** | +/ | **ABC-D\*+E** |
| **F\*G)** | +/( | **ABC-D\*+E** |
| **\*G)** | +/( | **ABC-D\*+EF** |
| **G)** | +/(\* | **ABC-D\*+EF** |
| **)** | +/(\* | **ABC-D\*+EFG** |
| **{Empty}** | +/ | **ABC-D\*+EFG\*** |
| **{Empty}** | {Empty} | **ABC-D\*+EFG\*/+** |

### Converting a prefix expression to a postfix expression.

Đọc kí tự từ trái qua phải. Nếu là toán tử ta liên kết toán tử đó với số nguyên 2 (dùng pair trong c++) và đẩy pair đó vào stack. Nếu là toán hạng thì ta output toán hạng và lấy pair trên cùng của stack và giảm số nguyên đi 1. Nếu pair trên cùng stack có số nguyên = 0 thì lấy pair đó ra khỏi stack và output toán tử. Khi chạy hết toàn bộ kí tự trong biểu thức tiền tố, ta lần lượt output ra từng kí tự còn lại trong stack.

Ví dụ: chuyển biểu thức tiền tố sau sang hậu tố + x – y z

|  |  |  |
| --- | --- | --- |
| Prefix | Stack | Postfix |
| + x – y z | {Empty} | **{Empty}** |
| x – y z | {+,2} | **{Empty}** |
| – y z | {+,1} | **x** |
| Y z | {+,1} , {-,2} | **x** |
| Z | {+,1},{-,1} | **x y** |
| {Empty} | {+,1} | **x y z -** |
| {Empty} | {Empty} | **x y z - +** |

## 1.2 Programming.

Struct node ( khai báo kiểu cấu trúc node) : mỗi node gồm 2 thông tin là giá trị và con trỏ trỏ tới địa chỉ node tiếp theo

Thuật toán:

* class stack: Khởi tạo 1 node\* là p\_high lưu node cao nhất trong stack (lúc đầu p\_high lưu giá trị NULL) và khởi tạo biến đếm số lượng node hiện có trong stack lúc đầu là 0
* Hàm push : tạo node mới lưu giá trị input và con trỏ trỏ tới p\_high hiện tại, thay p\_high là node vừa nhập và cập nhật biến đếm số lượng node trong stack
* Hàm pop : nếu stack không rỗng, xóa p\_high hiện tại và cập nhật p\_high mới là node kế tiếp của node cũ và cập nhật biến đếm số lượng node trong stack
* Hàm top : nếu stack không rỗng thì trả về giá trị của node trên cùng(p\_high)
* Hàm size : trả về giá trị của biến đếm số lượng node hiện có trong stack
* Class Destructor: sẽ xóa hết các phần tử trong stack

template <class T>

struct node

{

T value;

node\* next;

};

template <class T>

class stack

{

private:

node<T>\* p\_high = NULL;

int count = 0;

public:

~stack()

{

while (count > 0)

this->pop();

}

void push(T);

void pop();

T top();

int size();

};

template <class T>

void stack<T>::push(T value)

{

node<T>\* temp;

temp = new node<T>;

temp->value = value;

temp->next = p\_high;

p\_high = temp;

count++;

}

template <class T>

void stack<T>::pop()

{

if (p\_high == NULL)

return;

node<T>\* temp;

temp = p\_high;

p\_high = p\_high->next;

delete temp;

count--;

}

template <class T>

T stack<T>::top()

{

if (p\_high != NULL)

return p\_high->value;

}

template <class T>

int stack<T>::size()

{

return count;

}

Hàm int Rank(c) trả về độ ưu tiên của toán tử, toán hạng c.

Thuật toán:

* Kiểm tra kí tự c nếu là ‘ ‘ sẽ cho độ ưu tiên 0, số nguyên từ 0 đến 9 hoặc kí tự “.” sẽ cho độ ưu tiên 1, phép cộng – trừ sẽ cho độ ưu tiên 2, phép nhân – chia sẽ cho độ ưu tiên 3, dấu “(“ sẽ cho độ ưu tiên 4 và dấu “)” sẽ cho độ ưu tiên 5.

int Rank(char c)

{

if (c == ' ')

return 0;

if ((c >= '0' && c <= '9') || c == '.')

return 1;

if (c == '+' || c == '-')

return 2;

if (c == '\*' || c == '/' || c == '^')

return 3;

if (c == '(')

return 4;

if (c == ')')

return 5;

}

Hàm bool CheckExpression(string s) trả về true nếu string s là phương trình hợp lí, false nếu không và sử dụng hàm int Rank(c) để phục vụ.

- Ta có các quy tắc sau để kiểm tra tính hợp lệ của một phép tính:

* Số lượng "mở ngoặc" luôn lớn hơn hoặc bằng số lượng "đóng ngoặc" tính từ kí tự đầu đến kí tự bất kì và bằng nhau nếu tính từ đầu đến cuối chuỗi.
* Không có trường hợp 2 toán tử hoặc 2 toán hạng nằm cạnh nhau.
* Sau "mở ngoặc" phải là toán hạng.
* Sau "đóng ngoặc" phải là toán tử hoặc kết thúc chuỗi.
* Kí tự đầu tiên không được là toán tử

- Thuật toán:

* Vòng lặp chạy từ 0 đến độ dài của chuỗi s.
* Biến check lưu rank của toán tử/toán hạng trước đó.
* So sánh rank toán tử/toán hạng trước đó với toán tử/toán hạng hiện tại, nếu vi phạm các quy tắc ở trên thì trả về false.
* Kết thúc mỗi vòng lặp, cập nhật biến check bằng rank của toán tử/toán hạng hiện tại.
* Trong suốt vòng lặp, nếu gặp toán tử "mở ngoặc" thì push vào stack, nếu gặp toán tử "đóng ngoặc" thì so sánh nó với toán tử trên cùng của stack, nếu là 1 cặp ngoặc hợp lí thì xóa toán tử "mở ngoặc" trên cùng của stack, ngược lại return false.
* Trả về true.

bool CheckExpression(string s)

{

if (Rank(s[0]) == 2 || Rank(s[0]) == 3)

return false;

int check = 0;

stack<char> temp;

for (int i = 0; i < s.length(); i++)

{

if (s[i] == '(' || s[i] == '[' || s[i] == '{')

{

temp.push(s[i]);

s[i] = '(';

}

if (s[i] == ')' || s[i] == ']' || s[i] == '}')

{

if (temp.size() < 1)

return false;

if (s[i] == ')' && temp.top() != '(')

return false;

if (s[i] == ']' && temp.top() != '[')

return false;

if (s[i] == '}' && temp.top() != '{')

return false;

temp.pop();

s[i] = ')';

}

if (check == 1)

{

if (Rank(s[i]) == 1 && Rank(s[i - 1]) == 0)

return false;

if (Rank(s[i]) == 4)

return false;

}

if (check == 2 || check == 3)

{

if (Rank(s[i]) == 2 || Rank(s[i]) == 3)

return false;

}

if (check == 4)

{

if (Rank(s[i]) == 2 || Rank(s[i]) == 3 || Rank(s[i]) == 5)

return false;

}

if (check == 5)

{

if (Rank(s[i]) == 1 || Rank(s[i]) == 4)

return false;

}

if (Rank(s[i]) > 0)

check = Rank(s[i]);

}

if (temp.size() != 0)

return false;

return true;

}

Hàm string postfix(string s) trả về chuỗi là hậu tố từ phương trình trung tố s.

Thuật toán đã trình bày ở 1.1

string postfix(string s)

{

string s\_new = "";

stack<char> temp;

for (int i = 0; i < s.length(); i++)

{

if (Rank(s[i]) == 0 && s\_new[s\_new.length() - 1] != ' ')

{

s\_new += s[i];

continue;

}

if (Rank(s[i]) == 1)

{

s\_new += s[i];

continue;

}

if (Rank(s[i]) == 2)

{

if (temp.size() > 0 && Rank(temp.top()) <= 3 && Rank(temp.top()) >= 2)

{

if (s\_new[s\_new.length() - 1] != ' ')

s\_new += ' ';

s\_new += temp.top();

temp.pop();

}

temp.push(s[i]);

continue;

}

if (Rank(s[i]) == 3)

{

if (temp.size() > 0 && Rank(temp.top()) == 3)

{

if (s\_new[s\_new.length() - 1] != ' ')

s\_new += ' ';

s\_new += temp.top();

temp.pop();

}

temp.push(s[i]);

continue;

}

if (Rank(s[i]) == 4)

temp.push(s[i]);

if (Rank(s[i]) == 5)

{

while (temp.top() != '(')

{

if (s\_new[s\_new.length() - 1] != ' ')

s\_new += ' ';

s\_new += temp.top();

temp.pop();

}

temp.pop();

}

}

while (temp.size() > 0)

{

if (s\_new[s\_new.length() - 1] != ' ')

s\_new += ' ';

s\_new += temp.top();

temp.pop();

}

return s\_new;

}

Hàm float calculation(string s) trả về giá trị của biểu thức từ chuỗi s là phép tính hậu tố. Ý tưởng:

* Tạo stack rỗng.
* Tạo vòng lặp với điều kiện dừng là đọc hết chuỗi s.
* Nếu gặp toán hạng thì đẩy vào stack.
* Nếu gặp toán tử thì lấy 2 toán hạng từ stack và tính toán sau đó đẩy giá trị tính được vào stack.
* Thực hiện đến hết chuỗi, nếu hàm trả về false thì in ra "E", nếu true thì kết quả nằm ở biến output.

bool calculation(string s, float& output)

{

stack<float> temp;

stringstream ss(s);

float f, float\_temp1, float\_temp2;

char c;

while (!ss.eof())

{

ss >> c;

if (c >= '0' && c <= '9')

{

ss.seekg(-1, ios::cur);

ss >> f;

temp.push(f);

continue;

}

if (temp.size() >= 2)

{

float\_temp2 = temp.top();

temp.pop();

float\_temp1 = temp.top();

temp.pop();

if (c == '+')

temp.push(float\_temp1 + float\_temp2);

if (c == '-')

temp.push(float\_temp1 - float\_temp2);

if (c == '\*')

temp.push(float\_temp1 \* float\_temp2);

if (c == '/')

if (float\_temp2 != 0)

temp.push(float\_temp1 / float\_temp2);

else

return false;

if (c == '^')

temp.push(pow(float\_temp1, float\_temp2));

}

}

output = temp.top();

return true;

}

Hàm main cho phép người dùng nhập đường dẫn đến file nhập - xuất và hiển thị ra màn hình: ( -c ) giá trị tính toán của các biểu thức từ tệp đầu vào lấy 2 chữ số thập phân hoặc ( -t ) phép tính ở dạng hậu tố. Nếu biểu thức lỗi cho kết quả là E.

int main()

{

ifstream input;

ofstream output;

string s, s1, PostFix;

string choice;

int n;

cout << "nhap file txt input: " << endl;

cin >> s;

input.open(s);

cout << "nhap so luong phep tinh: " << endl;

cin >> n;

cout << "chon hanh dong: " << endl;

cout << "-c: tinh toan " << endl;

cout << "-t: chuyen doi" << endl;

cin >> choice;

cout << "nhap file txt output: " << endl;

cin >> s1;

output.open(s1);

string\* a = new string[n];

float\* cal = new float[n];

if (choice == "-c")

{

for (int i = 0; i < n; i++)

{

getline(input, a[i]);

if (CheckExpression(a[i]) == true)

{

PostFix = postfix(a[i]);

calculation(PostFix, cal[i]);

output << setprecision(3) << cal[i] << endl;

}

else

{

output << "E" << endl;

}

}

}

else if (choice == "-t")

for (int i = 0; i < n; i++)

{

getline(input, a[i]);

if (CheckExpression(a[i]) == true)

{

PostFix = postfix(a[i]);

output << PostFix << endl;

}

else

{

output << "E" << endl;

}

}

else

cout << "nhap khong hop le " << endl;

delete[] a;

delete[] cal;

}

# REFERENCES

<https://yinyangit.wordpress.com/2011/01/26/algorithm-chuy%E1%BB%83n-bi%E1%BB%83u-th%E1%BB%A9c-trung-t%E1%BB%91-sang-ti%E1%BB%81n-t%E1%BB%91-va-h%E1%BA%ADu-t%E1%BB%91-b%E1%BA%B1ng-stack/>

<https://www.youtube.com/watch?v=MyCBpybscNo>

<https://www.youtube.com/watch?v=OPmN_YYQmro>

<https://www.stdio.vn/giai-thuat-lap-trinh/ung-dung-stack-bieu-thuc-tien-to-prefix-hspu1>