BÁO CÁO ĐỒ ÁN 1

1. Thông tin chung

STT	MSSV	Họ và tên	Công việc	Mức độ hoàn thành
1	19126022	Lê Thiên Kim	Lập trình đổi trung tố thành hậu tố	100%
2	19126055	Trần Hoàng Thảo Ngân	Tổng hợp bài, đọc ghi file, tham số dòng lệnh Viết báo cáo	100%
3	19126031	Ngô Thiên Quang	Lập trình tính toán biểu thức trung tố	100%
4	19126027	Cao Hoàng Nhân	Giải thích từng bước 3 thuật toán chuyển đổi	100%

2. Báo cáo

2.1 Nghiên cứu:

- **Chuyển đổi từ trung tố(infix) sang hậu tố(posfix):**

Thuật toán

Bước 1: Khởi tạo string postfix để chứa biểu thức hậu tố và string stack để chứa tạm các kí tự hoạt động theo nguyên tắc ngăn xếp (vào trước ra sau).

Bước 2: Đọc từng kí tự của biểu thức trung tố từ trái sang phải và xét điều kiện của từng ký tự:

- Nếu là khoảng trắng: bỏ qua và đọc phần tử tiếp theo.
- Nếu là toán hạng (từ '1' đến '9' và '.'): push_back vào postfix

- Nếu là toán tử (‘+’ ;‘-‘ ; ‘*’ ; ‘/’)
  - Kiểm tra stack trống hay không.
  - Nếu không thì kiểm tra kí tự cuối cùng của stack có phải là là toán tử không nếu phải thì xét độ ưu tiên của toán tử. Nếu kí tự trong stack có độ ưu tiên cao hơn thì push_back vào postfix và xóa kí tự đó ở trong stack.
  - Push_back toán tử hiện hành vào stack.
- Nếu là các dấu mở ngoặc (‘(‘; ‘[‘; ‘{‘): push_back vào stack.
- Nếu là các dấu đóng ngoặc (‘)’; ‘]’; ‘}’):
  - Push_back vào postfix và pop_back khỏi stack lần lượt các toán tử có trong stack tới khi gặp phải các dấu mở ngoặt tương ứng.
  - Pop_back dấu mở ngoặt tương ứng đó ra khỏi stack.

Bước 3: Nếu đã duyệt tất các các kí tự có trong biểu thức trung tố thì lấy tuần tự các phần tử còn lại trong stack cho vào trong postfix.

Ví dụ:

Biểu thức Infix: {3 + [2 * (4 + 6 ^ 3)] / 4} * 2

Đọc	Xử lí	Postfix	Stack
{	Push_back vào Stack		{
3	Push_back vào Postfix	3	{
<space>	Bỏ qua	3	{
+	Push_back vào Stack do phần tử cuối cùng của Stack là ‘{‘không phải toán tử	3	{+
<space>	Bỏ qua	3	{+
[	Push_back vào Stack	3	{+[
2	Push_back vào Postfix	3 2	{+[
<space>	Bỏ qua	3 2	{+[
*	Push_back vào Stack do phần tử cuối cùng của Stack là ‘[‘không phải toán tử	3 2	{+[*
<space>	Bỏ qua	3 2	{+[*
(	Push_back vào Stack	3 2	{+[*(
4	Push_back vào Postfix	3 2 4	{+[*(
<space>	Bỏ qua	3 2 4	{+[*(
+	Push_back vào Stack do phần tử cuối cùng của Stack là ‘(‘không phải toán tử	3 2 4	{+[*(+
<space>	Bỏ qua	3 2 4	{+[*(+
6	Push_back vào Postfix	3 2 4 6	{+[*(+

<space>	Bỏ qua	3 2 4 6	{+[*(+
^	Push_back vào Stack do phần tử cuối cùng của Stack là '+' có độ ưu tiên thấp hơn '^'	3 2 4 6	{+[*(+^
<space>	Bỏ qua	3 2 4 6	{+[*(+^
3	Push_back vào Postfix	3 2 4 6 3	{+[*(+^
)	Push_back tuần tự các phần tử '^', '+' vào trong Postfix và pop_back các phần tử tương ứng ở trong Stack tới k hi gặp '('và pop_back '('ở trong Stack	3 2 4 6 3 ^ +	{+[*
]	Push_back phần tử '*' vào trong Postfix và pop_back phần tử tương ứng ở t rong Stack tới khi gặp '['và pop_back '['ở trong Stack	3 2 4 6 3 ^ + *	{+
<space>	Bỏ qua	3 2 4 6 3 ^ + *	{+
/	Push_back vào Stack do phần tử cuối cùng của Stack là '+' có độ ưu tiên thấp hơn '/'	3 2 4 6 3 ^ + *	{+/
<space>	Bỏ qua	3 2 4 6 3 ^ + *	{+/
4	Push_back vào Postfix	3 2 4 6 3 ^ + * 4	{+/
}	Push_back tuần tự các phần tử '/', '+' vào trong Postfix và pop_back các phần tử tương ứng ở trong Stack tới k hi gặp '{'và pop_back '{'ở trong Stack	3 2 4 6 3 ^ + * 4 / +	
<space>	Bỏ qua	3 2 4 6 3 ^ + *4 / +	
*	Push_back vào Stack	3 2 4 6 3 ^ + *4 / +	*
<space>	Bỏ qua	3 2 4 6 3 ^ + *4 / +	*
2	Push_back vào Postfix	3 2 4 6 3 ^ + *4 / + 2	*

Sau khi duyệt hết tất cả các phần tử trong biểu thức Infix thì push_back '*' vào Postfix và xóa Stack.

⇨ Biểu thức hậu tố: 3 2 4 6 3 ^ + *4 / + 2 *

- **Chuyển đổi từ trung tố(infix) sang tiền tố(prefix):**

Thuật toán

Bước 1: Đảo ngược biểu thức trung tố.

Bước 2: Đảo vị trí các dấu '('_')'; '['_']'; '{'_'}' cho nhau.

Bước 3: Thực hiện các bước chuyển trung tố thành hậu tố theo như mục 1.

Bước 4: Đảo ngược biểu thức hậu tố.

Ví dụ:

Biểu thức trung tố: {3 + [2 * (4 + 6 ^ 3)] / 4} * 2

Bước 1: Đảo ngược biểu thức trung tố: 2 * }4 /])3 ^ 6 + 4(* 2[+ 3{

Bước 2: Đảo vị trí các dấu '('_')'; '['_']'; '{'_'}' cho nhau: 2 * {4 / [(3 ^ 6 + 4) * 2] + 3}

Bước 3: Thực hiện các bước chuyển Infix thành Postfix.

Đọc	Xử lí	Postfix	Stack
2	Push_back vào Postfix	2	
<space>	Bỏ qua	2	
*	Push_back vào Stack	2	*
<space>	Bỏ qua	2	*
{	Push_back vào Stack	2	*{
4	Push_back vào Postfix	2 4	*{
<space>	Bỏ qua	2 4	*{
/	Push_back vào Stack do phần tử cuối cùng của Stack là '{'không phải toán tử	2 4	*{/
<space>	Bỏ qua	2 4	*{/
[	Push_back vào Stack	2 4	*{/[

(	Push_back vào Stack	2 4	*{/[(
3	Push_back vào Postfix	2 4 3	*{/[(
<space>	Bỏ qua	2 4 3	*{/[(
^	Push_back vào Stack do phần tử cuối cùng của Stack là ‘(‘không phải toán tử	2 4 3	*{/[(^
<space>	Bỏ qua	2 4 3	*{/[(^
6	Push_back vào Postfix	2 4 3 6	*{/[(^
<space>	Bỏ qua	2 4 3 6	*{/[(^
+	Push_back phần tử cuối cùng của Stack ‘^’ vào Postfix và pop_back phần tử tương ứng trong Stack vì toán tử ‘^’ có độ ưu tiên cao hơn ‘+’ và push_back ‘+’ vào Postfix	2 4 3 6 ^	*{/[(+
<space>	Bỏ qua	2 4 3 6 ^	*{/[(+
4	Push_back vào Postfix	2 4 3 6 ^ 4	*{/[(+
)	Push_back phần tử ‘+’ vào trong Postfix và pop_back phần tử tương ứng ở t rong Stack tới khi gặp ‘(‘và pop_back ‘(‘ở trong Stack	2 4 3 6 ^ 4 +	*{/[
<space>	Bỏ qua	2 4 3 6 ^ 4 +	*{/[
*	Push_back vào Stack do phần tử cuối cùng của Stack là ‘[‘không phải toán tử	2 4 3 6 ^ 4 +	*{/[*
<space>	Bỏ qua	2 4 3 6 ^ 4 +	*{/[*
2	Push_back vào Postfix	2 4 3 6 ^ 4 + 2	*{/[*

]	Push_back phần tử '*' vào trong Postfix và pop_back phần tử tương ứng ở trong Stack tới khi gặp '['và pop_back '['ở trong Stack	2 4 3 6 ^ 4 + 2 *	*{/
<space>	Bỏ qua	2 4 3 6 ^ 4 + 2 *	*{/
+	Push_back phần tử cuối cùng của Stack '/' vào Postfix và pop_back phần tử tương ứng trong Stack vì toán tử '/' có độ ưu tiên cao hơn '+' và push_back '+' vào Postfix	2 4 3 6 ^ 4 + 2 * /	*{+
<space>	Bỏ qua	2 4 3 6 ^ 4 + 2 * /	*{+
3	Push_back vào Postfix	2 4 3 6 ^ 4 + 2 * / 3	*{+
}	Push_back phần tử '+' vào trong Postfix và pop_back phần tử tương ứng ở trong Stack tới khi gặp '{'và pop_back '{'ở trong Stack	2 4 3 6 ^ 4 + 2 * / 3 +	*

Sau khi duyệt hết tất cả các phần tử trong biểu thức Infix thì push_back '*' vào Postfix và xóa Stack.

Biểu thức Postfix: 2 4 3 6 ^ 4 + 2 * / 3 + *

Bước 4: Đảo ngược biểu thức Postfix.

⇨ Biểu thức Prefix: * + 3 / * 2 + 4 ^ 6 3 4 2

- **Chuyển đổi từ tiền tố(prefix) sang hậu tố(postfix):**

Duyệt biểu thức lần lượt từ phải sang trái

Bước 1: nếu là toán hạng đẩy nó vào stack

Bước 2: nếu là toán tử lấy lần lượt 2 toán hạng ra (gọi là A và B)

Bước 3: ghép lại dưới dạng "A B Operator" rồi đẩy vào mảng

Bước 4: sau khi đã duyệt hết biểu thức thì khởi tạo chuỗi kết quả và gán với chuỗi trong stack và trả nó về

Vd: đề mô từng bước làm tay

Biểu thức prefix: * + 3 / * 2 + 4 ^ 6 3 4 2

Đọc	Xử lí	Stack
2	Push_back vào stack	[2]
4	Push_back vào Stack	[2 4]
3	Push_back vào Stack	[2 4 3]
6	Push_back vào Stack	[2 4 3 6]
^	Lấy string toán hạng ra (6) Lấy string toán hạng ra (3) Ghép lại dạng postfix và đưa ngược lại	[2 4 3] [2 4] [2 4 6 3 ^]
4	Push_back vào Stack	[2 4 6 3 ^ 4]
+	Lấy string toán hạng ra (4) Lấy phần tử tiếp theo ra (63^) Ghép lại dạng postfix và đưa ngược lại	[2 4 6 3 ^] [2 4] [2 4 4 6 3 ^ +]
2	Push_back vào Stack	[2 4 4 6 3 ^ + 2]
*	Lấy string toán hạng ra (2) Lấy phần tử tiếp theo ra (463^+) Ghép lại dạng postfix và đưa ngược lại	[2 4 4 6 3 ^ +] [2 4] [2 4 2 4 6 3 ^ + *]
/	Lấy phần tử tiếp theo ra (2463^+*) Lấy string toán hạng ra (4) Ghép lại dạng postfix và đưa ngược lại	[2 4] [2] [2 2 4 6 3 ^ + * 4 /]
3	Push_back vào stack	[2 2 4 6 3 ^ + * 4 / 3]
+	Lấy string toán hạng ra (3) Lấy phần tử tiếp theo ra (2463^+*4/) Ghép lại dạng postfix và đưa ngược lại	[2 2 4 6 3 ^ + * 4/] [2]

		[2 3 2 4 6 3 ^ + * 4 / +]
*	Lấy phần tử tiếp theo ra (32463^+*4/+) Lấy string toán hạng ra (2) Ghép lại dạng postfix và đưa ngược lại	[2] [] [3 2 4 6 3 ^ + * 4 / + 2 *]

2.2 Lập trình:

- **Cấu trúc dữ liệu:**

STACK:

1. Trong hàm evaluateInfix:

- Có 2 stack là values và operators
- Values là stack chứa dữ liệu kiểu float dùng để chứa những toán hạng đã được cắt và chuyển đổi từ biểu thức đẻ thực hiện tính toán
- Operators là stack chứa dữ liệu kiểu char dùng đẻ chứa những toán tử và các dấu ngoặc để phối hợp với các giá trị lưu trong stack value để giải quyết biểu thức

2. Trong hàm infixToPostfix:

- Có 1 string tempStack theo nguyên lí hoạt động của stack (LIFO) và được dùng để chứa dần các toán tử / toán hạng rồi sắp xếp

VECTOR

- Chứa kiểu dữ liệu string: vector<string> listInfix
- Sử dụng trong hàm readFile và writeFile để chứa chuỗi biểu thức được đọc từ InputPath và các biểu thức sau khi được thực hiện yêu cầu của người dùng và ghi vào OutputPath

- **Thuật toán:**

1. Tính toán biểu thức trung tố:

Bước 1: Chuẩn hóa chuỗi, đổi các dấu ngoặc {, [thành (và },] thành)

Bước 2: Tạo stack value (kiểu float) để chứa toán hạng và stack operators (kiểu char) để chứa các toán tử

Bước 3: Bắt đầu vòng lặp để đọc từng phần tử của chuỗi và xử lý các phép tính trong ngoặc

- Nếu là khoảng trắng xét đến phần tử tiếp theo.
- Nếu là '(' đẩy nó vào trong stack operators.
- Nếu là số thì chuyển đổi nó thành dạng float và đưa nó vào stack value
- Nếu là ')' thì tính toán bớt các phép toán trong ngoặc với các toán tử và toán hạng đã chứa trong stack sau đó bỏ bớt các toán tử và toán hạng đã tính toán tới khi gặp dấu '(' ứng với dấu ')' đang xét
- Nếu là toán hạng +, -, *, /, * thì dựa vào độ ưu tiên tính toán giá trị biểu thức trước đó nếu đô ưu tiên của toán hạng lần này nhỏ hơn toán hạng ở đầu stack rồi đẩy toán hạng đang xét vào stack

Bước 4: Sau khi đi hết vòng lặp thì những dấu ngoặc và phép tính trong ngoặc đã được xử lí chỉ còn lại những phép tính và toán tử thông thường ta sẽ tính toán dựa trên sự ưu tiên của nó.

Bước 5: Trả về giá trị cuối cùng sau khi tính toán trong stack và đổi sang string.

2. Chuyển đổi trung tố thành hậu tố:

Bước 1: Khởi tạo string postfix để chứa biểu thức hậu tố và string stack để chứa tạm các kí tự hoạt động theo nguyên tắc ngăn xếp (vào trước ra sau).

Bước 2: Đọc từng kí tự của biểu thức trung tố từ trái sang phải và xét điều kiện của từng ký tự:

- Nếu là khoảng trắng: bỏ qua và đọc phần tử tiếp theo.
- Nếu là toán hạng (từ '1' đến '9' và '.'): push_back vào postfix
- Nếu là toán tử ('+' ;'-' ; '*' ; '/')
  - Kiểm tra stack trống hay không.
  - Nếu không thì kiểm tra kí tự cuối cùng của stack có phải là là toán tử không nếu phải thì xét độ ưu tiên của toán tử. Nếu kí tự trong stack có độ ưu tiên cao hơn thì push_back vào postfix và xóa kí tự đó ở trong stack.
  - Push_back toán tử hiện hành vào stack.
- Nếu là các dấu mở ngoặc ('('; '['; '{'): push_back vào stack.
- Nếu là các dấu đóng ngoặc (')'; ']'; '}'):
  - Push_back vào postfix và pop_back khỏi stack lần lượt các toán tử có trong stack tới khi gặp phải các dấu mở ngoặt tương ứng.
  - Pop_back dấu mở ngoặt tương ứng đó ra khỏi stack.

Bước 3: Nếu đã duyệt tất các các kí tự có trong biểu thức trung tố thì lấy tuần tự các phần tử còn lại trong stack cho vào trong postfix.

- ## Các hàm sử dụng trong bài:

- **void standardizeString(string& S);**

Input: S - là chuỗi chứa biểu thức trung tố

Output: Trả về chuỗi S đã thay tất cả các dấu {} và [] thành ()

- **float calculateExpression(float a, float b, char opt);**

Input: a, b – hai toán tử cần thực hiện tính toán

opt - toán hạng

Output: Trả về kết quả của phép toán của hai toán tử với toán hạng đã nhập vào

- **string evaluateInfix(string a);**

Input: a - là biểu thức cần được tính toán

Output: giá trị sau khi tính toán của biểu thức dưới dạng string

```
string evaluateInfix(string a) { // functions to evaluate the expression
    string expression = a;
    int len = expression.length(); // get string length
    // STEP 1:
    standardizeString(expression); // standardize the string of expression
    // STEP 2:
    stack <float> values; // stack to store float value
    stack <char> operators; // stack to store operand such as + - * / (
    // STEP 3:
    for (int i = 0; i < len; i++) { // browse each char in the string
        if (expression[i] == ' ') { // skip the whitespace
            continue;
        }
        else if (expression[i] == '(') { // push the'(' into stack of operator
            operators.push(expression[i]);
```

```
        }
        else if (isOperand(expression[i]) == true) { // get and convert string of a float number and push it in value stack
            string temp1; // string that cointain the float number string
            for (int j = i; j < len; j++) {
                if (isOperand(expression[j]) == 0) { // escape the loop once encounter the end of an float number
                    i = j - 1;
                    break;
                }
                temp1 = temp1 + expression[j];
            }
            values.push(stof(temp1)); // convert and push float number into stack
        }
        else if (expression[i] == ')') { // calculate the sub-expression between the '()'
            while (!operators.empty() && operators.top() != '(') { // calculate with two value from value stack until encounter '(' in
operator stack
                float val2 = values.top();
                values.pop();
                float val1 = values.top();
                values.pop();
                char op = operators.top();
                operators.pop();
                values.push(calculateExpression(val1, val2, op)); // push the result back to stack once finish calculate the sub-
expression
            }
            if (!operators.empty()) // pop '(' out once finishing calculating all expression in the ()
                operators.pop();
        }
        else
        {
            while (!operators.empty() && setPriority(operators.top()) >= setPriority(expression[i])) { // calculate the sub-expression
with higher priority
                float val2 = values.top();
```

```
                values.pop();
                float val1 = values.top();
                values.pop();
                char op = operators.top();
                operators.pop();
                values.push(calculateExpression(val1, val2, op)); // push the result back to stack once finish calculate the sub-
expression
            }
            // Push current operator to operator stack.
            operators.push(expression[i]);
        }
    }
    // STEP 4:
    while (!operators.empty()) { // Here, we have eliminated all the bracket in the expression
        float val2 = values.top(); // evaluate the remaining of expression
        values.pop();
        float val1 = values.top();
        values.pop();
        char op = operators.top();
        operators.pop();
        values.push(calculateExpression(val1, val2, op));
    }
    // STEP 5:
    // Top of 'values' contains result, convert it back to string return.
    stringstream ss;
    ss << values.top();
    string s = ss.str();
    return s;
}
```

- `bool checkForValidity(string infix);`

Input: infix – string chứa biểu thức trung tố cần được kiểm tra

Output: Trả về kết quả true nếu biểu thức là trung tố và false nếu không phải

```
bool checkForValidity(string infix) { // check the expression is the expression infix
    int i = 0, prev = -2; //variable “prev” is used to store the result returned of func checkToken(char token) of the previous item
    if (isOperator(infix[0]) || isOperator(infix[infix.size() - 1])) // if the expression starts or endS with an operator
        return false;
    while (i < infix.size()) {
        int n = checkToken(infix[i]);//func checkToken(char token) return 1_is an operand; 0_is an operator; -1_white space or the
others
        if (n == -1)
            i++;
        else if (prev == 0 && n == 0) //if there are 2 operators that are adjacent then this expression is invalid.
            return false;
        else {
            prev = n;
            i++;
        }
    }
    return true;
}
```

- **string infixToPostfix(string infix);**

Input: infix – string chứa biểu thức dạng infix cần chuyển qua postfix

Output: Trả về string sau khi đã chuyển đổi thành dạng postfix

```
string infixToPostfix(string infix) {
// STEP 1:
    string postfix = "", tempStack = "";
    char space = ' ';
    int i = 0, prev = 1;
    while (i < infix.size()) {
```

```
// STEP 2: included 5 cases
// First case: item is white space
        if (infix[i] == ' ') // skip the whitespace
            i++;
// Second case: item is operand
        if (isOperand(infix[i])) { // push back the operand into postfix
            if (prev == 0)
                postfix += ' ';
            postfix += infix[i];
        }
        else {
// Third case: item is operator
            if (isOperator(infix[i])) {
                //if the last operator in the tempStack has higher priority than the current item
                while (!tempStack.empty() && isOperator(tempStack[tempStack.size() - 1]) && checkPriority(tempStack[tempStack.size() -
1], infix[i])) {
                    postfix += ' ';
                    postfix += tempStack[tempStack.size() - 1]; // then push this last operator into postfix
                    tempStack.pop_back(); //and delete it from tempstack
                }
                tempStack += infix[i]; // push back the curent item into tempStack
            }
// Fourth case: item is one of three opening brackets
            else if (infix[i] == '(' || infix[i] == '[' || infix[i] == '{') //push all '(', '[', '{' into tempStack
                tempStack += infix[i];
            else {
// Fifth case: item is one of three closing brackets// vậy xong rồi đún ko hỏi ngan cái coi sao
                // push all of operators between the '(_)', '[_]', '{_}' from tempStack into postfix.
                while (!tempStack.empty() && isOperator(tempStack[tempStack.size() - 1])) {
                    postfix += ' ';
                    postfix += tempStack[tempStack.size() - 1];
                    tempStack.pop_back();
```

```
                }
                tempStack.pop_back(); //Then, delete '(', '[', '{' from tempStack
            }
        }
        prev = checkToken(infix[i]);
        i++;
    }
// STEP 3:
    while (!tempStack.empty()) { // push all remaining items from tempStack to postfix
        postfix += ' ';
        postfix += tempStack[tempStack.size() - 1];
        tempStack.pop_back();
    }
    return postfix;
}
```

3. Tham khảo

[1]: http://www.cplusplus.com/reference/string/string/

[2]: http://www.cplusplus.com/reference/vector/vector/

[3]: http://www.cplusplus.com/reference/stack/stack/

[4]: [http://www.cplusplus.com/reference/string/stof/

[5]: https://www.stdio.vn/giai-thuat-lap-trinh/ung-dung-stack-bieu-thuc-trung-to-infix-zjru1

[6]: https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/