Trường Đại học Khoa học Tự Nhiên - ĐHQG HCM

ĐỒ ÁN LÝ THUYẾT CẤU TRÚC DỮ LIỆU

ĐƯỢC THỰC HIỆN BỞI·

Chủ đề:

ARITHMETIC EXPRESSION CALCULATION

Danh sách thành viên:

Bùi Minh Nhật - 21127126

Trần Hà Minh Nhật - 21127658

Cao Hoài Yến Vy - 21127205

Âu Dương Khang - 21127621

Mục lục

Thông tin nhóm 2
1.1 Thông tin chung 2
1.2 Phân công công việc và kết quả đạt được 2

Thông tin đồ án 4
2.1 Research 4
Giải thích: 4
Infix to Postfix 4
Infix to Prefix 5
Prefix to Postfix 6
2.2 Programming 7
2.2.1 Hàm kiểm tra chuỗi đầu vào 7
2.2.2 Hàm thực hiện chuyển đổi 10

Reference: 14

1. Thông tin nhóm

1.1 Thông tin chung

Tên nhóm: Mã nhóm: (Nếu có)

Các thành viên có tham dự: (Nhóm trưởng in đậm)

STT	MSSV	Tên	Gmail
1	**21127205**	**Cao Hoài Yến Vy**	**chyvy21@clc.fitus.edu.vn**
2	21127658	Trần Hà Minh Nhật	thmnhat21@clc.fitus.edu.vn
3	21127621	Âu Dương Khang	adkhang21@clc.fitus.edu.vn
4	21127126	Bùi Minh Nhật	bmnhat21@clc.fitus.edu.vn

1.2 Phân công công việc và kết quả đạt được

MSSV	Họ và tên	Mô tả nội dung công việc	Kết quả đạt được	Đánh giá
21127205	Cao Hoài Yến Vy	- Tham khảo và tìm kiếm tài liệu về phần research. Chỉnh sửa và hỗ trợ cả nhóm phần research. - Code và chỉnh sửa, tìm kiếm các trường hợp input bị sai. - Tham gia viết báo cáo và chỉnh sửa bản báo cáo phần research, reference, chỉnh sửa từ ngữ toàn bản báo cáo.	- Tổng hợp thành công phần giải thích thuật toán, ví dụ step by step cho thuật toán. - Code hoàn thiện, phân tích và chỉnh sửa các lỗi phát sinh. - Tìm được các lỗi input có thể xảy ra và các test cases để test hàm, thảo luận với nhóm để mở rộng thêm các trường hợp có thể xảy ra.	
21127126	Bùi Minh Nhật	- Nghiên cứu và thực hiện phần research Prefix -> Postfix. - Code và chỉnh sửa, tìm kiếm các trường hợp input bị sai, ghép code lại thành một bản hoàn chỉnh. - Tham gia viết báo cáo và chỉnh sửa bản báo cáo phần programming (biểu diễn code và	- Tổng hợp thành công phần giải thích thuật toán, ví dụ step by step cho thuật toán. - Code hoàn thiện, phân tích và chỉnh sửa các lỗi phát sinh. - Thuật toán phù hợp với các test case đưa ra. - Tìm được các lỗi input có thể xảy ra và các test cases để test hàm, thảo luận với nhóm để mở rộng thêm các trường hợp	

		các bước phù hợp).	có thể xảy ra. - Gom code của cả nhóm thành một file .cpp hoàn chỉnh đáp ứng đủ yêu cầu của đề bài.	
21127658	Trần Hà Minh Nhật	- Nghiên cứu và thực hiện phần research Infix -> Prefix - Code và chỉnh sửa phần command. - Tham gia viết báo cáo và chỉnh sửa bản báo cáo phần programming (biểu diễn code và các bước phù hợp).	- Tổng hợp thành công phần giải thích thuật toán, ví dụ step by step cho thuật toán. - Code hoàn thiện, phân tích và chỉnh sửa các lỗi phát sinh. - Thuật toán phù hợp với các test case đưa ra. - Chỉnh sửa command prompt phù hợp với yêu cầu	
21127621	Âu Dương Khang	- Nghiên cứu và thực hiện phần research Infix -> Postfix. - Hỗ trợ, chỉnh sửa phần command. - Tham gia viết báo cáo và chỉnh sửa bản báo cáo phần research, reference, chỉnh sửa từ ngữ toàn bản báo cáo.	- Tổng hợp thành công phần giải thích thuật toán, ví dụ step by step cho thuật toán. - Code hoàn thiện, phân tích và chỉnh sửa các lỗi phát sinh. - Thuật toán phù hợp với các test case đưa ra. - Chỉnh sửa command prompt phù hợp với yêu cầu	
Tổng				100%

2. Thông tin đồ án

2.1 Research

Giải thích:

– Prefix: Biểu thức tiền tố được biểu diễn bằng cách đặt toán tử lên trước các toán hạng. Với cách biểu diễn này, thay vì viết x+y như dạng trung tố, ta sẽ viết +xy. Tùy theo độ ưu tiên của toán tử mà chúng sẽ được sắp xếp khác nhau. Một biểu thức được gọi là prefix nếu nó được biểu diễn dưới dạng (toán tử - toán hạng 1 - toán hạng 2).
Ví dụ: *+AB-CD (Infix : (A+B) * (C-D))
– Postfix: Ngược lại với cách Prefix, tức là các toán tử sẽ được đặt sau các toán hạng. Một biểu thức được gọi là prefix nếu nó được biểu diễn dưới dạng (toán hạng 1 - toán hạng 2 - toán tử)
Ví dụ: AB+CD-* (Infix : (A+B * (C-D))

Một số ví dụ:

Infix	Prefix	Postfix
x+y	+xy	xy+
x+y-z	-+xyz	xy+z –
x+y*z	+x*yz	xyz*+
x+(y-z)	+x-yz	xyz-+

Infix to Postfix

Thuật toán:
Sử dụng stack để thực hiện convert
1.Quét biểu thức infix từ trái sang phải.
2. Nếu ký tự được quét là một toán hạng, xuất ra.
3. Nếu không:

1. Nếu mức độ ưu tiên và tính liên kết của toán tử được quét lớn hơn mức độ ưu tiên và tính liên kết của toán tử trong ngăn xếp (hoặc ngăn xếp trống hoặc ngăn xếp chứa dấu ‘(‘), push nó vào ngăn xếp.

2.Toán tử ‘^’ là liên kết phải và các toán tử khác như ‘+’, ’-‘, ’*’ và ‘/’ là liên kết trái. Đặc biệt kiểm tra một điều kiện khi cả trên cùng của ngăn xếp toán tử và toán tử được quét đều là ‘^’. Trong điều

kiện này, mức độ ưu tiên của toán tử được quét cao hơn do tính liên kết đúng của nó. Vì vậy, nó sẽ được đẩy trong ngăn xếp toán tử. Trong tất cả các trường hợp khác khi trên cùng của ngăn xếp toán tử giống với toán tử được quét, chúng ta sẽ bật toán tử khỏi ngăn xếp vi tính liên kết trái do đó toán tử được quét ít được ưu tiên hơn.

3. Nếu không, pop tất cả các toán tử từ ngăn xếp lớn hơn hoặc bằng được ưu tiên hơn so với toán tử được quét. Sau khi thực hiện điều đó push toán tử đã quét vào ngăn xếp. (Nếu gặp phải dấu ngoặc đơn trong khi xuất hiện thì dừng lại ở đó và push toán tử đã quét vào ngăn xếp.)

4. Nếu ký tự được quét là dấu '(', push vào ngăn xếp.

5. Nếu ký tự được quét là dấu ')', mở ngăn xếp và xuất ra cho đến khi gặp dấu '(' và loại bỏ cả hai dấu ngoặc đơn.

6. Lặp lại các bước 2-6 cho đến khi biểu thức infix được quét hết.

7. In đầu ra

8. Pop và xuất ra từ ngăn xếp cho đến khi nó không còn trống.

Ví dụ: step-by-step với chuỗi (1+1)^3

Sr. no.	Expression	Stack	Postfix
0		(	
1	1	((	1
2	+	((+	1
3	1	((+	11
4	)	(	11+
5	^	(^	11+
6	3	(^	11+3
7	)		11+3^

Infix to Prefix

Bước 1: Đảo ngược chuỗi và đổi dấu ')' thành '(' và ngược lại.

Bước 2: Thực hiện các bước để biến đổi Infix thành Postfix, nhận được chuỗi.

Bước 3: Đảo ngược chuỗi nhận được.

Ví dụ step-by-step với chuỗi (a+b)-(c*d)/e

Sr. no.	Expression	Stack	Prefix
0		(	
1	e	(	e
2	/	(/	e
3	d	(/ (	ed
4	*	(/ (*	ed
5	c	(/ (*	edc
6	)	(/	edc*
7	-	(-	edc*/
8	b	(- (	edc*/b
9	+	(- (+	edc*/b
10	a	(- (+	edc*/ba
11	)	(-	edc*/ba+
12	)		edc*/ba+-
13	)		-+ab/*cde

Prefix to Postfix

Ý tưởng: chạy ngược mảng prefix và sử dụng cấu trúc dữ liệu stack (kiểu string) để lưu kết quả.
Các bước làm:
Bước 1: Chạy i từ cuối mảng về 0
Bước 2: Xét prefix[i]:

+ Nếu prefix[i] là 1 biến số: chuyển biến số thành 1 string và push vào stack

+ Nếu prefix[i] là 1 kí hiệu toán tử ('*','/','^','+','-'), ta sẽ pop stack và lấy 2 chuỗi đầu trong stack, gọi lần lượt là s1 và s2. Do ta chạy ngược về nên thứ tự của các phần tử sẽ bị ngược lại, do đó ta sẽ push lại vào stack 1 chuỗi (s1 + s2 + prefix[i]).

Xét lần lượt cho tới khi i = 0, kết quả sẽ là phần tử đầu của stack.

Sr. no.	Expression	Stack	Postfix
0	e	e	
1	d	"e", "d"	
2	c	"e", "d", "c"	
3	*	"e", "cd*"	
4	/	"cd*e/"	
5	b	"cd*e/", "b"	
6	a	"cd*e/", "b", "a"	
7	+	"cd*e/", "ab+"	
8	-	"ab+cd*e/-"	ab+cd*e/-

2.2 Programming

2.2.1 Hàm kiểm tra chuỗi đầu vào

1. Các hàm kiểm tra đầu vào:

Input: chuỗi đầu vào có định dạng là một infix expression.
Lần lượt đưa ra các trường hợp với các bước như sau:
Bước 1: Xây dựng hàm isNum() kiểm tra ký tự có phải là số hay không

```
bool isNum(char c)
{
    if (c >= '0' && c <= '9')
        return true;
    return false;
}
```

Thực hiện kiểm tra ký tự đó có nằm trong khoảng từ ‘0’ đến ‘9’ hay không, nếu có thì return true, không return false.
Bước 2: Xây dựng hàm isOperator() kiểm tra ký tự có phải là toán tử hay không

```
bool isOperator(char c)
{
    if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
        return true;
    return false;
}
```

Thực hiện kiểm tra ký tự đó có nằm thuộc các toán tử + - * / ^ hay không, nếu có thì return true, không return false.
Bước 3: Xây dựng 1 các hàm isBalanced() để kiểm tra lần lượt vị trí của ngoặc, số lần mở ngoặc và số lần đóng ngoặc có hợp lý và đúng logic không.

```
bool isBalanced(string s)
{
    int cnt_operator = 0;
    int cnt_operand = 0;
    stack<char> st;
    for (int i = 0; i < s.length(); i++)
    {
        if (isBracket(s[i]))
        {
            if (s[i] == '(' || s[i] == '[' || s[i] == '{')
            {
                st.push(s[i]);
            }
            else
            {
                if (st.empty())
```

```
17.                 return false;
18.             char c = st.top();
19.             st.pop();
20.             if (s[i] == ')')
21.             {
22.                 if (c != '(')
23.                     return false;
24.             }
25.             else if (s[i] == '}')
26.             {
27.                 if (c != '{')
28.                     return false;
29.             }
30.             else
31.             {
32.                 if (c != '[')
33.                 {
34.                     return false;
35.                 }
36.             }
37.         }
38.     }
39.     else if (isNum(s[i]))
40.     {
41.         if (i - 1 >= 0)
42.         {
43.             if (!isNum(s[i - 1]) && s[i - 1] != '.')
44.                 cnt_operand++;
45.         }
46.         else
47.             cnt_operand++;
48.     }
49.     else if (isOperator(s[i]))
50.     {
51.         cnt_operator++;
52.     }
53.   }
54. if (!st.empty() || (cnt_operator != cnt_operand - 1))
55.   {
56.     return false;
57.   }
58.   return true;
59. }
```

Để kiểm tra ngoặc, sử dụng cấu trúc dữ liệu stack để push vào các ngoặc mở và mỗi khi đụng 1 ngoặc đóng, lấy phần tử đầu của stack để so sánh. Ngoài ra do biểu thức có dạng 1 infix expression, ta thấy số toán tử trong mọi trường hợp luôn bằng số toán hạng - 1, đếm 2 giá trị này và so sánh.

Bước 4: Xây dựng 1 các hàm isValidSpace() để kiểm tra lần lượt vị trí của các dấu space có thiếu hoặc dư so với tiêu chuẩn hay không. Với dấu space, xét các trường hợp 2 dấu space cạnh nhau, trước ngoặc mở và sau ngoặc đóng không có dấu space, và ở 2 phần tử bên cạnh của toán tử cũng phải có dấu space.

```
bool isValidSpace(string s)
{
    int n = s.length();
    int i;
    for (i = 0; s[i] != '\0'; i++)
    {
        if (s[i] == ' ' && ((i + 1 < n && s[i + 1] == ' ') || (i - 1 >= 0 && s[i - 1] == ' ')))
        {
            return false;
        }
        int temp = isBracket(s[i]);
        if (temp)
        {
            if (temp == 2 && i + 1 < n && s[i + 1] != ' ')
            {
                return false;
            }
            if (temp == 1 && i - 1 >= 0 && s[i - 1] != ' ')
            {
                return false;
            }

        }
        if (isOperator(s[i]))
        {
            if (i + 1 < n && s[i + 1] != ' ')
            {
                return false;
            }
            if (i - 1 >= 0 && s[i - 1] != ' ')
            {
                return false;
            }
        }
    }
    return true;
}
```

Bước 5: Tổng hợp các hàm kiểm tra trên để có được hàm kiểm tra để tạo ra hàm checkString(string s) nhằm kiểm tra input hợp lệ hay không.

```
bool checkString(string s)
{
    int len = s.length();
    if (s == "" || !isBalanced(s) || !isValidSpace(s))
    {
        return false;
    }

    for (int i = 0; i < len; i++)
    {
```

```
11.      if (s[i] == '.')
12.      {
13.         if (i == len - 1 || !isNum(s[i + 1]))
14.            return false;
15.         for (int j = i + 1; j < len; j++)
16.         {
17.            if (!isNum(s[j]))
18.               break;
19.            if (isNum(s[j]) && j == i + 3)
20.               return false;
21.         }
22.      }
23.      else if (isOperator(s[i]))
24.      {
25.         if (i - 2 < 0 || i + 2 > len)
26.            return false;
27.         if ((!isNum(s[i - 2]) && !isBracket(s[i - 2]) && !isBracket(s[i + 2])) || isBracket(s[i - 2]) == 1 || isBracket(s[i
    + 2]) == 2)
28.            return false;
29.      }
30.   }
31.   return true;
32. }
```

Sau đó ta sẽ thực hiện thêm 1 vòng for kiểm tra các phần tử kề các phần tử toán tử, đồng thời do các toán hạng có thể là 1 floating point có tối đa 2 chữ số thập phân bằng cách nếu phần tử đang xét là kí tự '.' ta sẽ xét 3 phần tử s[i+1], s[i+2], s[i+3] (nếu có). Đồng thời do giữa các toán hạng phải là 1 toán tử, ta sẽ xét trường hợp khi gặp 1 toán tử, kiểm tra 2 phần tử i+2 và i-2 xem có thỏa yêu cầu trên hay không.

2.2.2 Hàm thực hiện chuyển đổi

Bước 1: Chuẩn bị cho việc tính toán và chuyển đổi thành hậu tố bằng hàm prepare: thêm khoảng trắng vào sau các dấu mở ngoặc và trước các dấu đóng ngoặc.

```
33. void prepare(string& s)
34. {
35.    for (int i = 0; i < s.length(); i++)
36.    {
37.       if (s[i] == '(')
38.       {
39.          s.insert(s.begin() + i + 1, ' ');
40.       }
41.       else if (s[i] == ')')
42.       {
43.          s.insert(s.begin() + i, ' ');
44.          i++;
45.       }
46.    }
47. }
48. void printResult(vector<string> result)
49. {
```

```
50.    for (int i = 0; i < result.size(); i++)
51.    {
52.       cout << result[i] << " ";
53.    }
54.    /*/for (string i : result) {
55.       cout << i<< " ";
56.    }*/
```

Trường hợp người dùng yêu cầu tính toán từ các infix expression, hàm calculate sẽ được gọi: hàm sẽ scan infix expressions để lấy ra các toán hạng để push vào một stack và khi gặp ,ột toán tử, hàm sẽ pop các phần tử từ stack và tính toán với các toán tử tương ugsw.

```
57. double calculate(vector<string> s)
58. {
59.    stack < double > num;
60.    double tmp;
61.    double result = 0;
62.    for (int i = 0; i < s.size(); i++)
63.    {
64.       if (s[i] != "+" && s[i] != "-" && s[i] != "*" && s[i] != "/" && s[i] != "^")
65.       {
66.          tmp = stof(s[i]);
67.          num.push(tmp);
68.       }
69.       else {
70.          double tmp1 = num.top();
71.          num.pop();
72.          double tmp2 = num.top();
73.          num.pop();
74.          if (s[i] == "+")
75.          {
76.             result = tmp1 + tmp2;
77.          }
78.          else if (s[i] == "-")
79.          {
80.             result = tmp2 - tmp1;
81.          }
82.          else if (s[i] == "*")
83.          {
84.             result = tmp2 * tmp1;
85.          }
86.          else if (s[i] == "/")
87.          {
88.             result = tmp2 / tmp1;
89.          }
90.          else if (s[i] == "^")
91.          {
92.             result = pow(tmp2, tmp1);
93.          }
94.          num.push(result);
95.       }
96.    }
97.    return num.top();
98. }
99.
```

Hàm đánh số thứ tự ưu tiên các toán tử:

```
100.        // Function to return precedence of operators
101.        int prec(string c)
102.        {
103.           if (c == "^")
104.              return 3;
105.           else if (c == "/" || c == "*")
106.              return 2;
107.           else if (c == "+" || c == "-")
108.              return 1;
109.           else
110.              return -1;
111.        }
```

Hàm chính chuyển đổi từ trung tố sang hậu tố:

Sử dụng stack để lưu từng string riêng biệt, nếu nó là toán tử thì cho vào vector result, nếu là ký tự '(' thì push vào stack phụ st lưu những toán hạng đến khi gặp ')' thì pop ra các toán tử có trong stack st. Nếu thứ tự ưu tiên của toán tử đang kiểm tra bé hơn thứ tự ưu tiên của phần tử top trong stack st vào trong vector result đến khi thứ tự đó bé hơn thứ tự ưu tiên của phần tử top. Kết thúc vòng lặp push phần tử đang xét vào trong stack st. Sau cùng pop tất cả các phần tử trong stack st vào stack result.

```
112.        // The main function to convert infix expression
113.        // to postfix expression
114.        vector<string> infixToPostfix(string s)
115.        {
116.           vector<string> vs;
117.           stringstream fi(s);
118.
119.           string tmp;
120.
121.           stack<string> st; // For stack operations, we are using
122.                             // C++ built in stack
123.           vector<string> result;
124.           while (fi >> tmp)
125.           {
126.              vs.push_back(tmp);
127.           }
128.           /*for (int i = 0; i < vs.size(); i++)
129.           {
130.              cout << vs[i] << endl;
131.           }*/
132.
133.
134.           for (int i = 0; i < vs.size(); i++) {
135.              string c = vs[i];
136.
137.              // If the scanned character is
138.              // an operand, add it to output string.
139.
140.              if (vs[i] != "+" && vs[i] != "-" && vs[i] != "*" && vs[i] != "/" && vs[i] != "^" && vs[i] != "(" &&
    vs[i] != ")")
141.                 result.push_back(c);
```

```
142.
143.             // If the scanned character is an
144.             // ‘(‘, push it to the stack.
145.             else if (c == "(")
146.                st.push("(");
147.
148.             // If the scanned character is an ‘)’,
149.             // pop and to output string from the stack
150.             // until an ‘(‘is encountered.
151.             else if (c == ")") {
152.                while (st.top() != "(") {
153.                   result.push_back(st.top());
154.                   st.pop();
155.                }
156.                st.pop();
157.             }
158.
159.             // If an operator is scanned
160.             else {
161.                while (!st.empty() && prec(vs[i]) <= prec(st.top())) {
162.                   if (c == "^" && st.top() == "^")
163.                      break;
164.                   else {
165.                      result.push_back(st.top());
166.                      st.pop();
167.                   }
168.                }
169.                st.push(c);
170.             }
171.          }
172.
173.          // Pop all the remaining elements from the stack
174.          while (!st.empty()) {
175.             result.push_back(st.top());
176.             st.pop();
177.          }
178.          //printResult(result);
179.          return result;
180.          //cout << "\n" << calculate(result) << endl;
181.       }
```

Hàm main debug chạy command theo yêu cầu đưa ra

```
182.       // Driver program to test above functions
183.       void main_debug(string input_file, int x, string action, string output_file) {
184.          ifstream input(input_file);
185.          ofstream output(output_file);
186.          string exp;
187.          for (int i = 0; i < x; i++) {
188.             getline(input, exp);
189.             if (!checkString(exp))
190.             {
191.                output << 'E' << endl;
192.                continue;
```

```
193.            }
194.            prepare(exp);
195.            vector<string> postFix = infixToPostfix(exp);
196.            if (action == "-c") {
197.               output << calculate(postFix) << endl;
198.            }
199.            else if (action == "-t") {
200.               //printResult(postFix);
201.               for (int i = 0; i < postFix.size(); i++)
202.               {
203.                  output << postFix[i] << " ";
204.               }
205.               output << endl;
206.            }
207.         }
208.      }
209.      void main_testcases(string input_file, int x, string action, string output_file) {
210.         main_debug(input_file, x, action, output_file);
211.      }
212.      int main(int argc, char* argv[])
213.      {
214.         string input_file, action, output_file; int x;
215.         if (argc > 4) {
216.            input_file = argv[1];
217.            output_file = argv[4];
218.            x = atoi(argv[2]);
219.            action = argv[3];
220.            main_testcases(input_file, x, action, output_file); return 0;
221.         }
222.         else {
223.            cout << "Enter input_file: "; cin >> input_file;
224.            cout << "Enter x: "; cin >> x;
225.            cout << "Enter action: "; cin >> action;
226.            cout << "Enter output_file: "; cin >> output_file;
227.         }
228.         main_debug(input_file, x, action, output_file);
229.         if (!system(NULL)) system("pause"); return 0;
230.         return 0;
231.      }
```

Reference:

- Infix to Postfix (algorithm and code): Infix to Postfix.
- Infix to Prefix (algorithm and code): Infix to Prefix.
- Prefix to Postfix(algorithm and code): Prefix to Postfix.
- Code command prompt: Mr. Thong Huy Bui and Command line