TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

----------o0o----------

BÁO CÁO BÀI TẬP LỚN

Môn học: CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

Đề tài: Xây dựng và ứng dụng của Stack - Ngăn xếp

Giảng viên hướng dẫn: ThS. Phạm Xuân Tích

Nhóm thực hiện: Nhóm 2 sinh viên - Lớp CNTT4 K60

Danh sách sinh viên tham gia đóng góp vào báo cáo:

1. Đặng Thị Ngọc Anh - 191213566
2. Nguyễn Thanh Hằng - 191200383

Hà Nội, tháng 12 năm 2020

Phần 1:
Giới thiệu về ngăn xếp

1.Khái niệm:

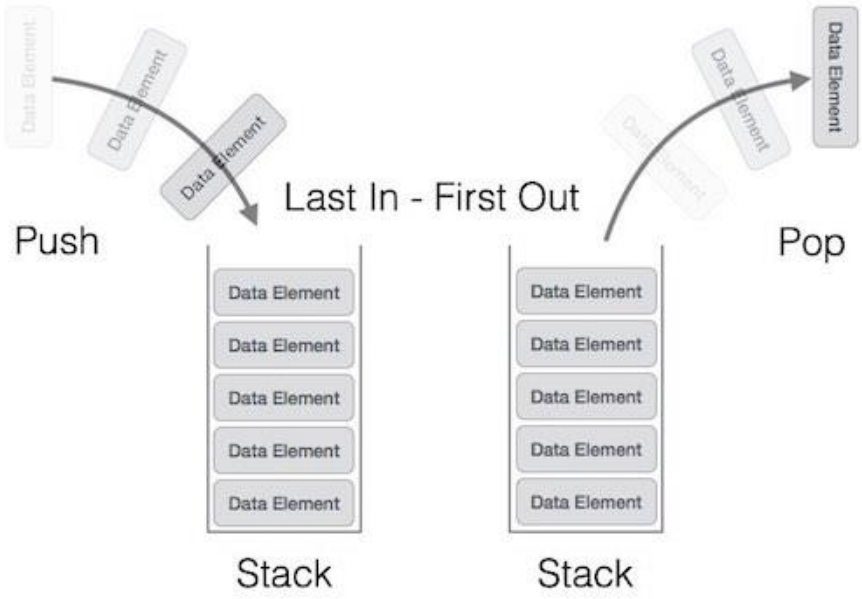
Cấu trúc dữ liệu ngăn xếp (stack) là một cấu trúc dữ liệu trừu tượng (Abstract Data Type – viết tắt là ADT), tuyến tính, hoạt động theo nguyên tắc vào sau ra trước (LIFO – Last in first out).

Giải thích:

Stack là cách tổ chức, lưu trữ các đối tượng dưới dạng một danh sách tuyến tính mà việc bổ sung đối tượng và lấy các đối tượng ra được thể hiện ở cùng một đầu của danh sách. Stack được gọi là danh sách kiểu LIFO. Ở đây, phần tử được đặt vào (được chèn, được thêm vào) cuối cùng sẽ được truy cập đầu tiên. Trong thuật ngữ ngăn xếp, hoạt động chèn được gọi là hoạt động PUSH và hoạt động xóa được gọi là hoạt động POP.

2.Biểu diễn cấu trúc dữ liệu ngăn xếp (stack)

Dưới đây là sơ đồ minh họa một ngăn xếp và các hoạt động diễn ra trên ngăn xếp



Một ngăn xếp có thể được triển khai theo phương thức của Mảng (Array), Cấu trúc (Struct), Con trỏ (Pointer) và Danh sách liên kết (Linked List). Ngăn xếp có thể là ở dạng kích cỡ cố định hoặc ngăn xếp có thể thay đổi kích cỡ. Phần dưới chúng ta sẽ triển khai ngăn xếp bởi sử dụng các mảng với việc triển khai các ngăn xếp cố định.

3.Các hoạt động cơ bản trên cấu trúc dữ liệu ngăn xếp

a. Initialize

- Chức năng: Khởi động Stack
- Dữ liệ nhập: Không
- Dữ liệu xuất: Stack top về vị trí khởi đầu

b. Empty

- Chức năng: Kiểm tra Stack có rỗng hay không
- Dữ liệu nhập: Không
- Dữ liệu xuất: True or False (True: khi Stack rỗng, False: Khi Stack không bị rỗng)

c. Size

- Chức năng: Xác định số nút hiện có trong Stack
- Dữ liệu nhâp: Không
- Dữ liệu xuất: Số nút hiện có trong Stack

d. Push

- Chức năng: Thêm một phần tử vào Stack
- Dữ liệu nhập: Phần tử mới
- Dữ liệu xuất: Không

e. Pop

- Chức năng: Lấy một phần tử ra tại đỉnh Stack
- Dữ liệu nhập: Không
- Điều kiện: Stack không bị rỗng
- Dữ liệu xuất: Phần tử bị lấy ra

f. Top

- Chức năng: Lấy giá trị của phần tử đỉnh
- Dữ liệu nhập: Không
- Điều kiện: Stack không bị rỗng
- Dữ liệu xuất: Phần tử tại đỉnh của Stack

g. Clear

- Chức năng: Xóa tất cả các phần tử có trong stack
- Dữ liệu nhập: Không
- Dữ liệu xuất: Stack top về vị trí khởi đầu

h. Copy

- Chức năng: Copy Stack thành Stack mới
- Dữ liệu nhập: Stack nguồn
- Dữ liệu xuất: Stack mới giống Stack cũ

4. Phương pháp cài đặt Stack

Có rất nhiều cách cài đặt một ngăn xếp, nhưng để đơn giản cũng như dễ hình dung nhất là cài đặt bằng mảng (hay còn gọi là cài đặt theo kiểu kê tiếp). Ngoài ra còn có nhiều cách cài đặt khác như cài đặt bàng danh sách liên kết (Danh sách liên kết đơn, Danh sách liên kết vòng, Danh sách liên kết kép, Danh sách liên kết vòng + kép).

Phần 2:
Phân tích bài toán

Bài toán:

1.Xây dựng lớp Stack

2.Viết chương trình cho phép thực hiện các chức năng sau:

- Nhập vào một biểu thức dạng trung tố
- Chuyển biểu thức đó sang dạng hậu tố
- Tính giá trị của biểu thức

1.Xác định yêu cầu:

- Xây dựng 1 lớp stack để thực hiện ứng dụng định giá biểu thức số học: chuyển biểu thức trung tố sang biểu thứ hậu tố.

- Để định giá biểu thức số học thì phải xác định thứ tự ưu tiên của các toán tử :

‘$’ < ‘(’ < ‘+’ = ‘-’ < ‘*’ = ‘/’ < ‘^’

- Các lớp xây dựng Stack :

- Hàm mẫu Stack gồm: num(số lượng phần tử),cap(sức chứa của phần tử),buff(mảng lưu).

  Bao gồm một lớp Stack gồm 6 phương thức

  - Size : trả về số lượng phần tử trong 1 Stack vừa tạo
  - Empty: trả về số lượng phần tử bằng 0 nếu không có phần tử trong Stack
  - Top: trả về địa chỉ của phần tử đầu tiên trong mảng
  - Pop: trả về số lượng của mảng sau khi xóa 1 phần tử

- Push : trả về số lượng của mang sau khi thêm vào 1 phần tử và nếu số lượng và sức chứa thì sẽ tạo ra 1 stack với chiều dài gấp 1,6 nhân với sức chứa cộng thêm 5.
- Peek: Nếu Stack khác rỗng thì trả về giá trị của phần tử đầu nếu Stack rỗng thì trả về mảng rỗng.

- Các lớp,các thuộc tính,các phương thức của bài toán
  - Lớp kttoantu: nếu đúng là những toán tử +,-,/^,* trả về đúng còn lại trả về giá trị sai.
  - Lớp uutientt: đưa ra thứ tự trả về của các toán tử
  - Lớp infixtopostfix :chuyển giá trị trung tố về về giá trị hậu tố dựa theo thứ tự ưu tiên của toán tử và lớp stack đã xây dựng
  - Lớp taostack : Tạo ra một stack mới rỗng
  - Lớp giatri: lấy biểu thức hậu tố bên trên ở lớp infixtopostfix gán vào lớp để tính ra kết quả của biểu thức, áp dụng các trường hợp để pop các phần tử trong stack và cho ra kết quả cuối cùng.
- Mô tả bài toán

Thuật toán 1: Chuyển biểu thức từ trung tố sang hậu tố sang hậu tố: thì biểu thức sẽ mất dấu ngoặc (chỉ còn lại số và phép toán)

Đọc từng kí tự của chuỗi (từ đầu đến cuối). Mỗi kí tự ta xử lí:

- Nếu là dấu thì thêm vào ngăn xếp
- Nếu là số thì cho ra chuỗi kết quả
- Nếu gặp dấu đóng ngoặc thì lấy tất cả những gì trong ngăn xếp ra cho vào chuỗi kết quả cho đến khi gặp dấu mở ngoặc thì thôi (Dấu mở ngoặc cũng bị lấy ra)
- Nếu là toán tử (phép tính) thì: Lấy trong ngăn xếp các toán tử có độ ưu tiên cao hơn ra chuỗi kết quả. Cuối cùng cho toán tử hiện tại vào ngăn xếp rồi lấy tất cả những gì còn lại trong ngăn xếp ra.

Thuật toán 2: Tính giá trị biểu thức dạng hậu tố.

Đọc từng kí tự của chuỗi hậu tố, mỗi kí tự ta xử lí:

- Nếu là số thì cho vào ngăn xếp ( )
- Ngược lại (phép tính): Lấy 2 số trong ngăn xếp ra,tính toán với nhau (với phép tính này), rồi lại cho kết quả vào ngăn xếp. Kết quả cuối cùng còn lại trong ngăn xếp chính là giá trị cuối cùng của biểu thức.

Phần 3:
Cài đặt các lớp bằng hàm main bằng C++

```
#include<bits/stdc++.h>
#include"STACK.cpp"
using namespace std;
class Stack{
	public:
	unsigned cap;
	int top;
	int *buff;
};
bool kttoantu(char c)
{
	if(c=='+' || c == '-' || c=='/' || c=='^' || c=='*' || c=='$')
	{
		return true;
	}
	else
	{
		return false;
	}
}
```

```
int uutientt(char c)
{
	if(c == '^')
	return 3;
	else if(c == '*' || c=='/' || c=='%')
	return 2;
	else if(c == '+' || c=='-')
	return 1;
	else
	return -1;
}
string infixtopostfix(STACK<char> s,string infix) //bt trung to sang hau to
{
	string postfix;
	for(int i=0;i<infix.length();i++){
		if(infix[i] >= 'a' &&infix[i]<='z' || infix[i] >='A' && infix[i] <='Z' ||
infix[i] >='0' && infix[i] <='9')
		{
			postfix+=infix[i];
		}
		else if(infix[i] == '(')
		{
			s.push(infix[i]);
		}
		else if(infix[i] == ')')
		{
			while((s.top()!= '(') && (!s.empty()))
			{
				char temp=s.top();
				postfix+=temp;
				s.pop();
			}
			if(s.top()=='(')
			{
				s.pop();
```

```
                }
        }
        else if(kttoantu(infix[i]))
        {
                if(s.empty())
                {
                        s.push(infix[i]);
                }
                else
                {
                        if(uutientt(infix[i])>uutientt(s.top()))
                        {
                                s.push(infix[i]);
                        }
                        else
if((uutientt(infix[i])==uutientt(s.top()))&&(infix[i]=='^'))
                        {
                                s.push(infix[i]);
                        }
                        else
                        {

        while((!s.empty())&&(uutientt(infix[i])<=uutientt(s.top()))){
                                        char temp=s.top();
                                        postfix+=temp;
                                        s.pop();
                                }
                                s.push(infix[i]);
                        }
                }
        }

        }
        while(!s.empty())
        {
```

```
            postfix+=s.top();
            s.pop();
        }
        return postfix;
}
Stack *taostack(unsigned cap)
{
        Stack *s= new Stack ();
        if(!s) return NULL;
        s->top=-1;
        s->cap=cap;
        s->buff=new int[(s->cap * sizeof(int))];
        if(!s->buff) return NULL;
        return s;
}
int empty(Stack *s)
{
        return s->top==-1;
}
int peek(Stack *s)
{
        return s->buff[s->top];
}
int pop(Stack *s)
{
        if(!empty(s))
            return s->buff[s->top--];
        return '$';
}
void push(Stack *s,int op)
{
        s->buff[++s->top] =op;
}
int giatri(char *exp)
{
```

```
	Stack *stack=taostack(strlen(exp));
	int i;
	if(!stack) return -1;
	for(i=0;exp[i];i++)
	{
		if(exp[i] == ' ') continue;
		else if(isdigit(exp[i]))
		{
			int num=0;
			while(isdigit(exp[i]))
			{
				num = num*10+ (int)(exp[i]-'0');
				i++;
			}
			i--;
			push(stack,num);
		}
		else {
			int val1=pop(stack);
			int val2=pop(stack);
			switch(exp[i])
			{
				case '+': push(stack,val2 +val1); break;
				case '-': push(stack,val2 -val1); break;
				case '*': push(stack,val2 *val1); break;
				case '/': push(stack,val2 /val1); break;
			}
		}
	}
	return pop(stack);
}
int main (){
	string infix_exp,postfix_exp;
	cout<<"Nhap vao bieu thuc trung to:"<<endl;
	cin>>infix_exp;
```

```
    STACK<char> s;
    cout<<"Bieu thuc trung to:"<<infix_exp<<endl;
    postfix_exp = infixtopostfix(s,infix_exp);
    cout<<endl<<"Bieu thuc hau to :"<<postfix_exp;
    cout<<endl<<"Gia tri bieu thuc hau to: "<<giatri("100 30 + 5 *");
}
```

Phần 4:

Phân tích thời gian chạy của từng phương thức tronglớp

-Lớp giatri có thời gian chạy là O(n).

-Lớp infixtopostfix có thời gian chạy là O(n).

-Các lớp còn lại có thời gian chạy là O(n).

V. Tài liệu tham khảo

-GeeksforGeeks

-Simple Snippets