3.5. Ngăn xếp (Stack)

3.5.1. Định nghĩa. Tập hợp các node thông tin được tổ chức liên tục hoặc rời rạc nhau trong bộ nhớ và thực hiện theo cơ chế FILO (First – In – Last – Out).

3.5.2. Biểu diễn

- Biểu diễn liên tục: sử dụng mảng.
- Biểu diễn rời rạc: sử dụng danh sách liên kết.

3.5.3. Thao tác

- Kiểm tra tính rỗng của stack: empty()
- Lấy số lượng phần tử trong stack: size()
- Đưa phần tử x vào ngăn xếp: push(x)
- Truy cập phần tử đầu ngăn xếp: top()
- Lấy phần tử ra khỏi ngăn xếp: pop().

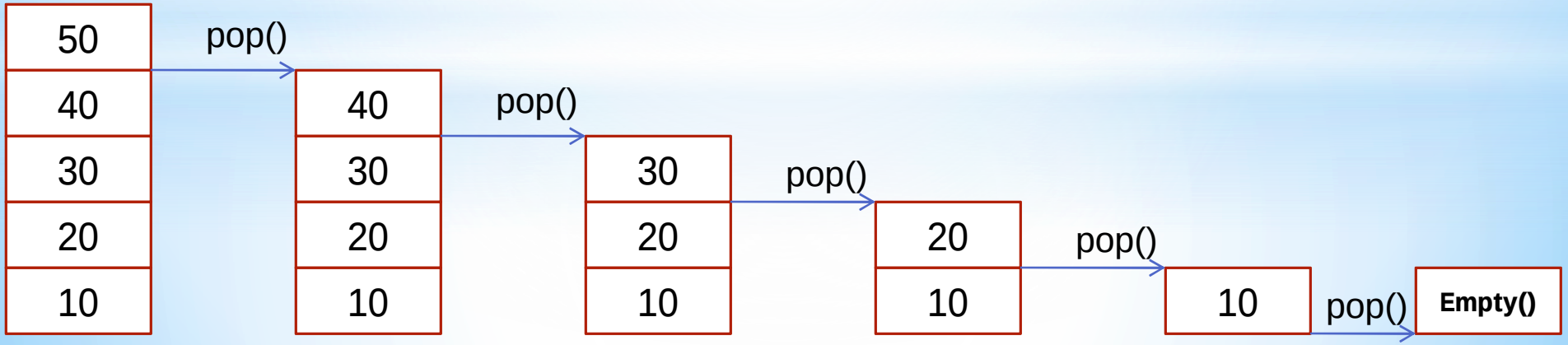
3.5.4. Ứng dụng

- Xây dựng các giải thuật đệ qui.
- Khử bỏ các giải thuật đệ qui.
- Biểu diễn tính toán
- Duyệt cây, duyệt đồ thị

Ví dụ:

```
#include <iostream>
#include <stack>
using namespace std;
int main(void){
        stack <int> s; //khai báo stack thích nghi với tùy biến int
        for(int i=1; i<=5; i++)  s.push(i*10); //đưa i*10 vào stack
        cout<<"Kích cỡ stack:"<<s.size()<<endl;// s.size() = 10
        while(!s.empty()){ //lặp đến khi stack rỗng
                int t = s.top();cout<<t<<" "; //lấy phần tử đầu stack
                s.pop(); //đưa phần tử đầu tiên ra khỏi stack
        }
}
```

Hình 1: push(1*10), push(2*10), push(3*10), push(4*10), push(5*10)



BÀI TẬP

1. Ta gọi NGE(i) của một mảng A[] là phần tử lớn hơn A[i] đầu tiên bên phải A[i]; NGE(i) = -1 nếu i là phần tử cuối cùng của mảng hoặc bên phải A[i] không có phần tử nào lớn hơn A[i]. Cho mảng A[] gồm n phần tử, hãy in ra NGE(i) của mỗi phần tử với độ phức tạp thời gian O(n).

Ví dụ:

Input:

```
2
4
13  7  6  12
5
8  12  9  7  5
```

Output:

```
-1  12  12  -1
12  -1  -1  -1  -1
```

Thuật toán NGE(A[], n):

Bước 1 (khởi tạo):

Stack <int> s; s.push(A[0]); *//đưa A[0] vào ngăn xếp*

Bước 2 (tìm NGE của các số có NGE):

```
For( int i=1; i<n; i++) { //lặp từ phần tử 1 đến cuối mảng
    X = A[i]; //X là phần tử A[i];
    If (!s.empty()) { //nếu stack không rỗng
        T = s.top(); s.pop(); //đưa T ra khỏi ngăn xếp
        While( T<X ) { //lặp đến khi T>X
            <đưa ra NGE(T) là X>;
            If (s.empty() ) Break; // nếu stack rỗng  rời khỏi lặp while
            T = s.top(); s.pop(); //đưa phần tử tiếp theo ra khỏi ngăn xếp
        }//endwhile
        If ( T>X) //nếu T lớn hơn X
            s.push(T); //đưa T vào ngăn xếp
    }//endif
    s.push(X) ; //đưa X vào ngăn xếp
} //endfor
```

Bước 3 (tìm NGE của các số không có NGE):

```
While(!s.empty()) { //lặp đến khi stack rỗng
    T = s.top(); s.pop();//đưa T ra khỏi ngăn xếp
    < đưa ra NGE(T) là -1>;
}
// A[] = { 13, 7, 6, 12 }
```

BÀI TẬP

Bracket Numbers (Flipkart). Cho biểu thức exp độ dài n chứa đựng một số ký tự '(', ')'. Hãy in ra số thứ tự của các cặp '(', ')' khi phân tích biểu thức.

Input:	**Output:**
(a + (b *c)) + (d/e)	1 2 2 1 3 3
((()) (()))	1 2 3 3 2 4 5 5 4 1

Thuật toán Index-bracket(exp, n) {//n là độ dài biểu thức exp

Bước 1(khởi tạo):

Left = 1; stack <int> right; //chỉ số đầu tiên bên trái là 1

Bước 2 (lặp):

```
For(i=0; i<n; i++) { //duyệt từ trái qua phải exp
    If ( exp[i] ==‘(‘) {//nếu exp[i] là ‘(‘
        <đưa ra chỉ số left>; right.push(left); //đưa chỉ số left vào stack
        Left++; //chỉ số tiếp theo được tăng lên 1
    }
    Else if (exp[i] =‘)’ ) { //nếu exp[i] là ‘)’
        <đưa ra chỉ số tương ứng là right.top()>;
        right.pop();// đưa chỉ số ra khỏi ngăn xếp
    }
}//endfor
}
```

BÀI TẬP

Prefix to Infix Conversion. Có ba dạng biểu diễn cho các biểu thức số học và logic:

Infix (trung tố): Biểu diễn biểu thức dưới dạng trung tố là phép biểu diễn biểu thức trong đó phép toán được đặt giữa hai toán hạng. Ví dụ (A+B) * (C-D).

Prefix (tiền tố): Biểu diễn biểu thức dưới dạng tiền tố là phép biểu diễn biểu thức trong đó phép toán được đặt trước hai toán hạng. Ví dụ *+AB-CD (tương ứng với biểu thức trung tố (A+B)*(C-D).

Postfix (hậu tố): Biểu diễn biểu thức dưới dạng hậu tố là phép biểu diễn biểu thức trong đó phép toán được đặt sau hai toán hạng. Ví dụ AB+CD-* (tương ứng với biểu thức trung tố (A+B)*(C-D).

Hãy viết chương trình chuyển đổi biểu thức biểu diễn dưới dạng tiền tố về dạng trung tố.

Input:	**Output:**
*+AB-CD	((A+B)*(C-D))
*-A/BC-/AKL	((A-(B/C))*((A/K)-L)

Thuat toan PreToInfix (Pre_exp) {// *Pre-exp là một string tiền tố*

Bước 1(khởi tạo):

Stack <string> s; *//tạo stack s thích nghi với tùy biến string*

n = Pre_exp.zise(); *//lấy n là độ dài Pre_exp*

Bước 2 (lặp):

For(i=n-1; i>=0; i--) { *//duyệt từ phải qua trái Pre_exp*

X = Pre_exp[i]; // l*ấy X là Pre_exp[i]*

If (isOperator(X)) { *//nếu X là phép toán*

String Op1 = s.top(); s.pop(); *//đưa toán hạng 1 ra khỏi stack*

String Op2 = s.top(); s.pop(); *//đưa toán hạng 2 ra khỏi stack*

String temp =“(“ +Op1 + X + OP2 + “)”; *//thành lập string temp*

s.push(temp); *//đưa temp trở lại stack*

}

Else { *//nếu X là toán hạng*

s.push(string(1), X); *//đưa X vào stack với kiểu string độ dài 1*

}

}

Return (s.top()); *//phần tử cuối cùng chính là biểu thức trung tố*

}

//Chú ý:

// isOperator(X) = true nếu x là phép toán ngược lại isOperator(X) = false

BÀI TẬP

Prefix to Infix Conversion. Có ba dạng biểu diễn cho các biểu thức số học và logic:

Infix (trung tố): Biểu diễn biểu thức dưới dạng trung tố là phép biểu diễn biểu thức trong đó phép toán được đặt giữa hai toán hạng. Ví dụ (A+B) * (C-D).

Prefix (tiền tố): Biểu diễn biểu thức dưới dạng tiền tố là phép biểu diễn biểu thức trong đó phép toán được đặt trước hai toán hạng. Ví dụ *+AB-CD (tương ứng với biểu thức trung tố (A+B)*(C-D).

Postfix (hậu tố): Biểu diễn biểu thức dưới dạng hậu tố là phép biểu diễn biểu thức trong đó phép toán được đặt sau hai toán hạng. Ví dụ AB+CD-* (tương ứng với biểu thức trung tố (A+B)*(C-D).

Hãy viết chương trình chuyển đổi biểu thức biểu diễn dưới dạng tiền tố về dạng trung tố.

Input:	**Output:**
*+AB-CD	((A+B)*(C-D))
*-A/BC-/AKL	((A-(B/C))*((A/K)-L)