Chương 7
CẤU TRÚC DỮ LIỆU NGĂN XẾP-STACK

TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

Bài 5 - 1

1. NGUYÊN LÝ HOẠT ĐỘNG

Cấu trúc dữ liệu ngăn xếp hoạt động theo nguyên lý **vào sau, ra trước** (LIFO- Last In - First Out)

TS. Nguyễn Tấn Trần Minh Khang

ThS. Cáp Phạm đình Thăng

2. CẤU TRÚC DỮ LIỆU CỦA NGĂN XẾP

```
1. struct stack
2. {
3.     int n;
4.     KDL a[100];
5. };
6. typedef struct stack STACK;
```

- KDL là kiểu dữ liệu của đối tượng được lưu trong stack.

3. KHỞI TẠO NGĂN XẾP

- Khái niệm: Khởi tạo ngăn xếp là tạo ra ngăn xếp rỗng không chứa đối tượng nào hết.
- Định nghĩa hàm

```
1. void Init(STACK &st)
2. {
3.     st.n = 0;
4. }
```

4. KIỂM TRA NGĂN XẾP RỖNG

- Khái niệm: Kiểm tra ngăn xếp rỗng là hàm trả về giá trị 1 khi ngăn xếp rỗng. Trong tình huống ngăn xếp chưa rỗng thì hàm sẽ trả về giá trị 0.
- Định nghĩa hàm

```
1. int IsEmpty(STACK st)
2. {
3.     if(st.n==0)
4.         return 1;
5.     return 0;
6. }
```

5. KIỂM TRA NGĂN XẾP ĐẦY

- Khái niệm: Kiểm tra ngăn xếp đầy là hàm trả về giá trị 1 khi ngăn xếp đã đầy. Trong trường hợp ngăn xếp chưa đầy thì hàm trả về giá trị 0.
- Định nghĩa hàm:

```
1. int IsFull(STACK st)
2. {
3.     if(st.n==100)
4.         return 1;
5.     return 0;
6. }
```

6. THÊM MỘT ĐỐI TƯỢNG VÀO TRONG NGĂN XẾP

- Khái niệm: Thêm một đối tượng vào trong ngăn xếp xét về mặt kỹ thuật với CTDL đã được khai báo bên trên là việc thêm đối tượng đó vào cuối mảng a đang có n phần tử của stack mà thôi.
- Định nghĩa hàm

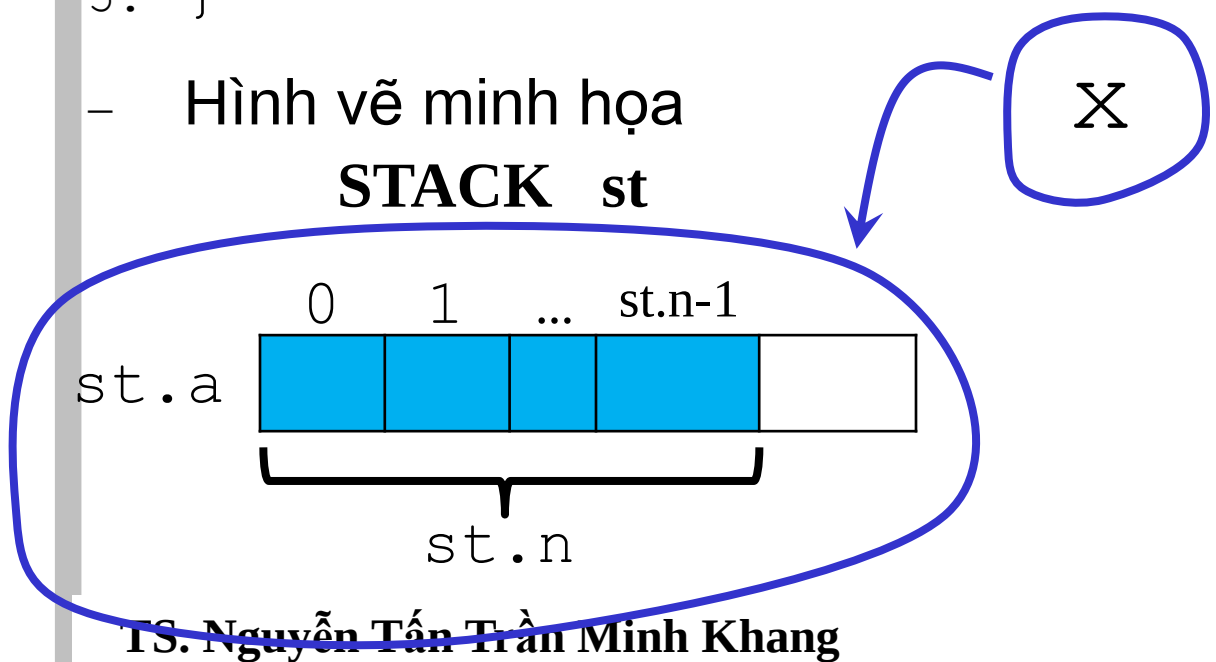
```
1. void Push(STACK &st, KDL x)
2. {
3.     st.a[st.n] = x;
4.     st.n++;
5. }
```

6. THÊM MỘT ĐỐI TƯỢNG VÀO TRONG NGĂN XẾP

- Định nghĩa hàm

```
1. void Push(STACK &st, KDL x)
2. {
3.     st.a[st.n] = x;
4.     st.n++;
5. }
```

- Hình vẽ minh họa



7. LẤY MỘT ĐỐI TƯỢNG RA KHỎI NGĂN XẾP

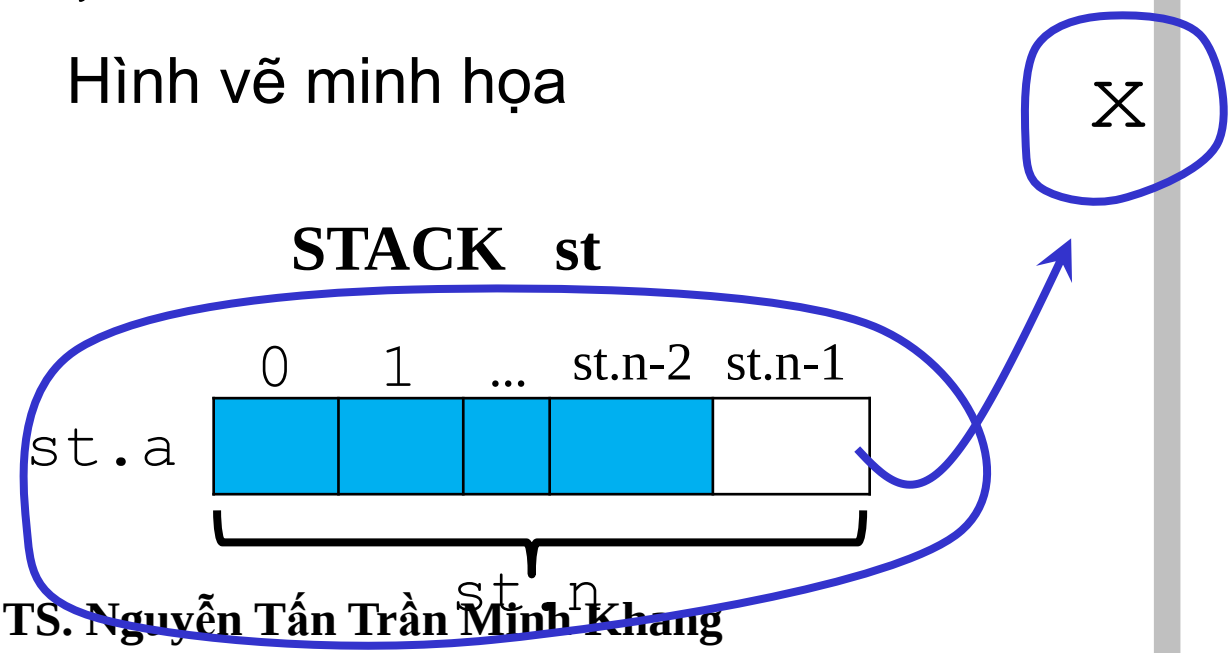
- Khái niệm: Lấy một đối tượng ra khỏi ngăn xếp xét về mặt kỹ thuật với CTDL đã được khai báo bên trên là việc lấy đối tượng cuối mảng a của stack ra khỏi mảng mà thôi.
- Định nghĩa hàm

```
1. KDL Pop(STACK &st)
2. {
3.     KDL x = st.a[st.n-1];
4.     st.n--;
5.     return x;
6. }
```

7. LẤY MỘT ĐỐI TƯỢNG RA KHỎI NGĂN XẾP

```
1. KDL Pop(STACK &st)
2. {
3.     KDL x = st.a[st.n-1];
4.     st.n--;
5.     return x;
6. }
```

- Hình vẽ minh họa



8. ỨNG DỤNG

- Bài toán: Định nghĩa hàm tính tổng các giá trị trong cây nhị phân các số thực bằng hai phương pháp
  - Đệ quy.
  - Khử đệ quy.

ỨNG DỤNG (tiếp)

- Cách 1: Đệ quy

```
1. struct node
2. {
3.     float info;
4.     struct node *pLeft;
5.     struct node *pRight;
6. };
7. typedef struct node NODE;
8. typedef NODE*TREE;

9. float Tong(TREE t)
10.{
11.    if(!t)
12.        return 0;
13.    float a=Tong(t->pLeft);
14.    float b=Tong(t->pRight);
15.    return a+b+t->info;
16.}
```

ỨNG DỤNG (tiếp)

- Cách 2: Khử đệ quy
  - **Cấu trúc dữ liệu:**

```
11. struct node
12. {
13.     float info;
14.     struct node *pLeft;
15.     struct node *pRight;
16. };
17. typedef struct node NODE;
18. typedef NODE*TREE;
9.  struct stack
10. {
11.     int n;
12.     NODE* a[100];
13. };
14. typedef struct stack STACK;
```

ỨNG DỤNG (tiếp)

- Định nghĩa hàm

```
11. void Init(STACK &st)
12. {
13.     st.n=0;
14. }
15. int IsEmpty(STACK st)
16. {
17.     if (st.n==0)
18.         return 1;
19.     return 0;
20. }
```

ỨNG DỤNG (tiếp)

```
11. int IsFull(STACK st)
12. {
13.     if(st.n==100)
14.         return 1;
15.     return 0;
16. }
17. void Push(STACK&st,NODE* x)
18. {
19.     st.a[st.n]=x;
20.     st.n++;
21. }
22. NODE* Pop(STACK &st)
23. {
24.    NODE* x = st.a[st.n-1];
25.    st.n--;
26.    return x;
27. }
```

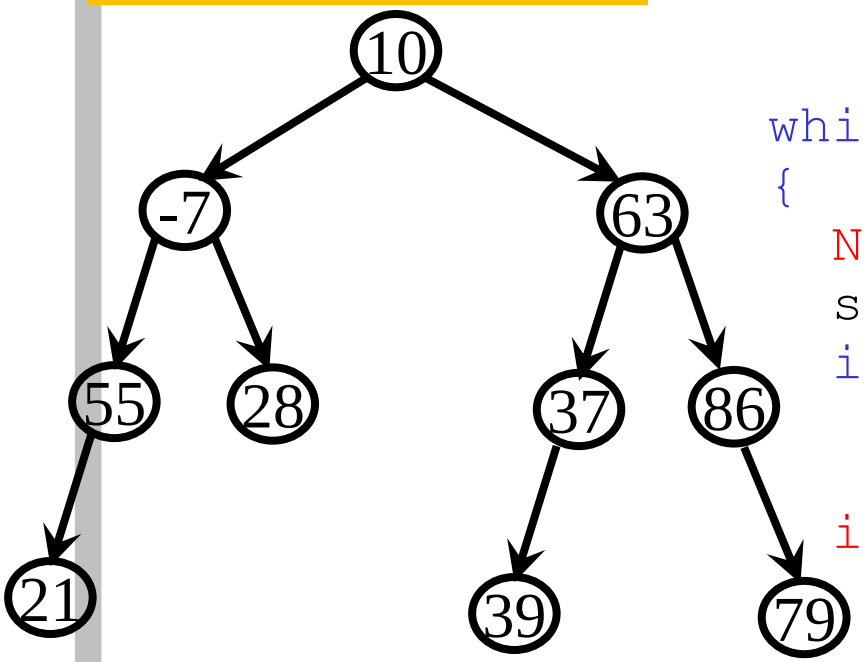
ỨNG DỤNG (tiếp)

```
11. float Tong(TREE t)
12. {
13.     float s=0;
14.     STACK stk;
15.     Init(stk);
16.     if (t!=NULL)
17.         Push(stk,t);
18.     while(IsEmty(stk)==0)
19.     {
20.         NODE*p = pop(stk);
21.         s=s+p->info;
22.         if (p->pLeft!=NULL)
23.             Push(stk,p->pLeft);
24.         if (p->pRight!=NULL)
25.             Push(stk,p->pRight);
26.     }
27.     return s;
28. }
```

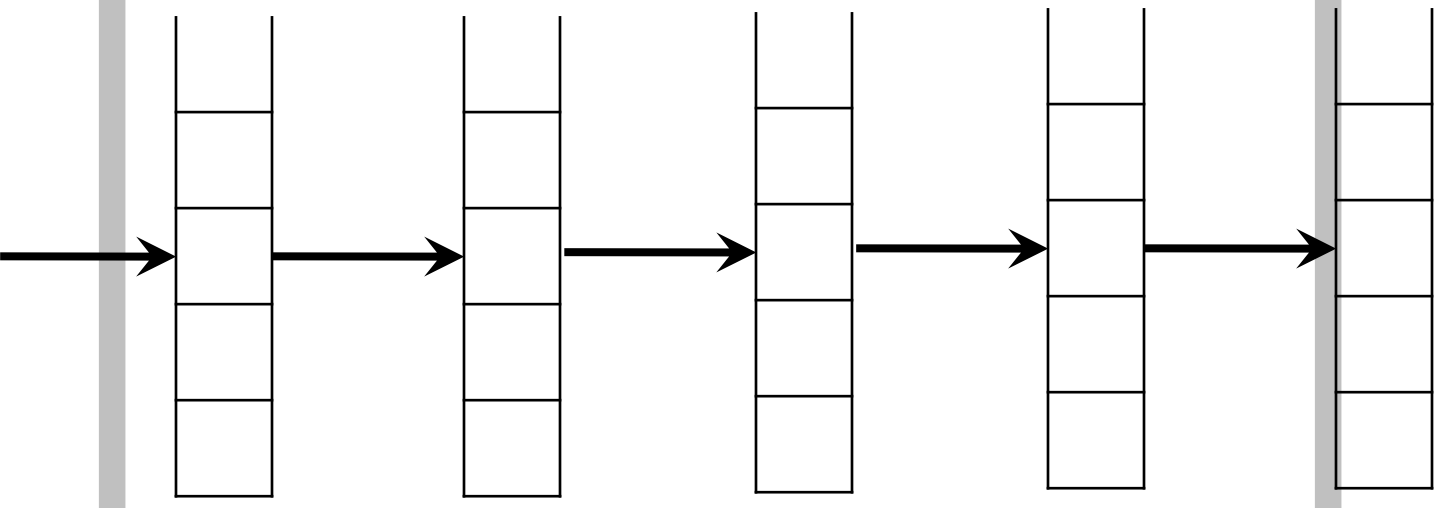
ỨNG DỤNG (tiếp)

```
1.  float Tong(TREE t)
2.  {
3.      float s=0;
4.      STACK stk;
5.      Init(stk);
6.      if (t!=NULL)
7.          Push(stk,t);
8.      while(IsEmty(stk)==0)
9.      {
10.         NODE*p = pop(stk);
11.         s=s+p->info;
12.         if (p->pLeft)
13.             Push(stk,p->pLeft);
14.         if (p->pRight)
15.             Push(stk,p->pRight);
16.     }
17.     return s;
18. }
```

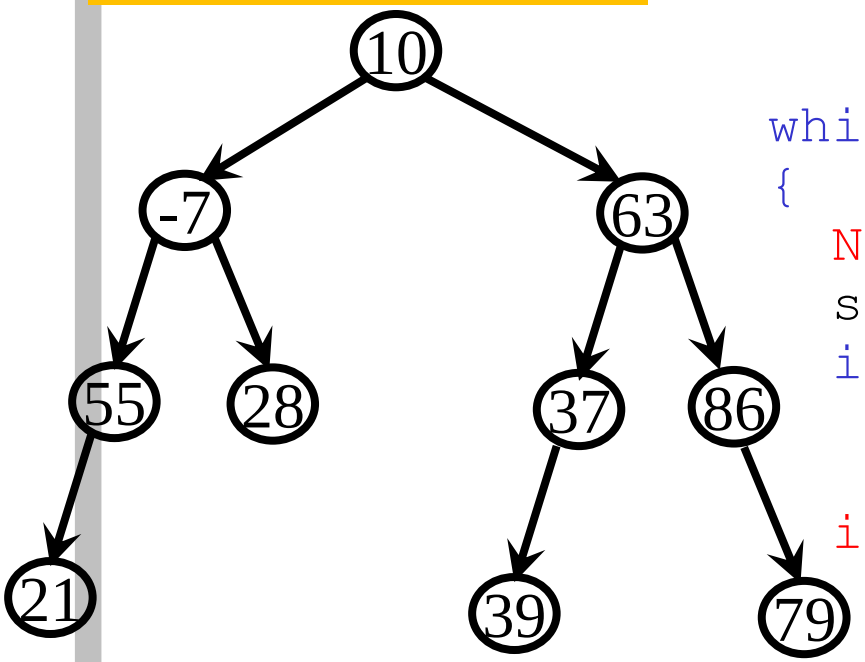
Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:



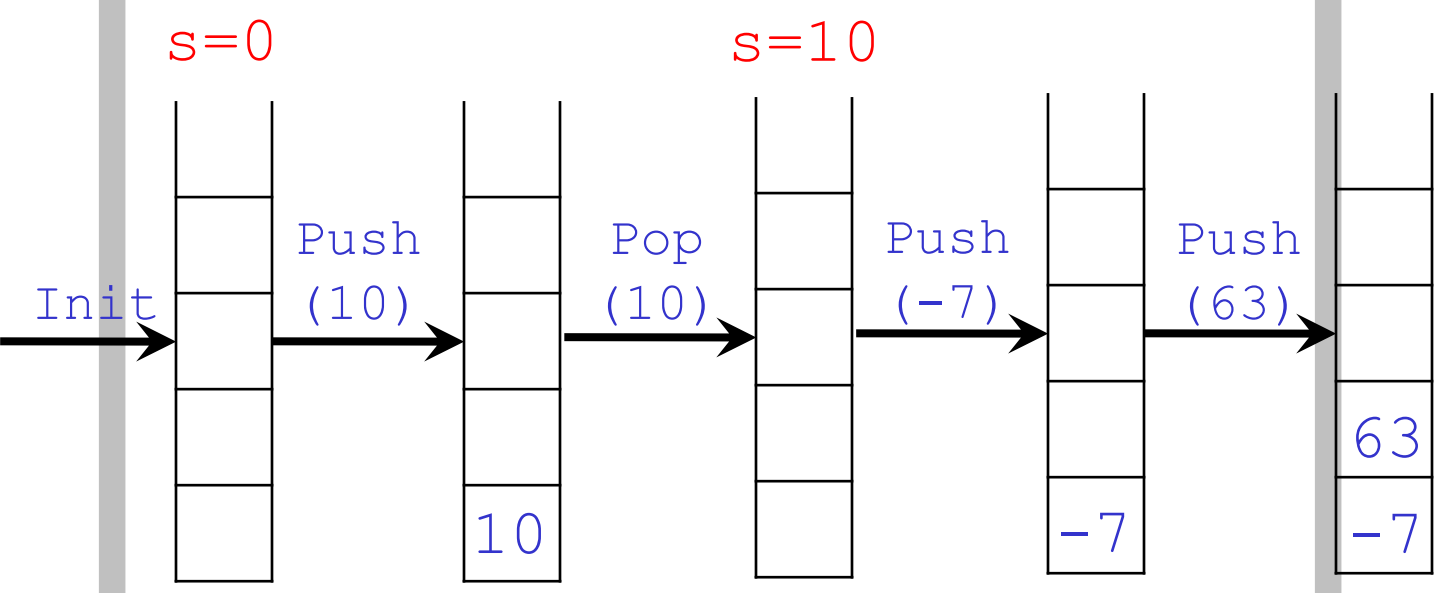

```
float s=0;
STACK stk;
Init(stk);
if (t!=NULL)
    Push(stk,t);
while(IsEmty(stk)==0)
{
  NODE*p = pop(stk);
  s=s+p->info;
  if (p->pLeft)
    Push(stk,
         p->pLeft);
  if (p->pRight)
    Push(stk,
         p->pRight);
}
```



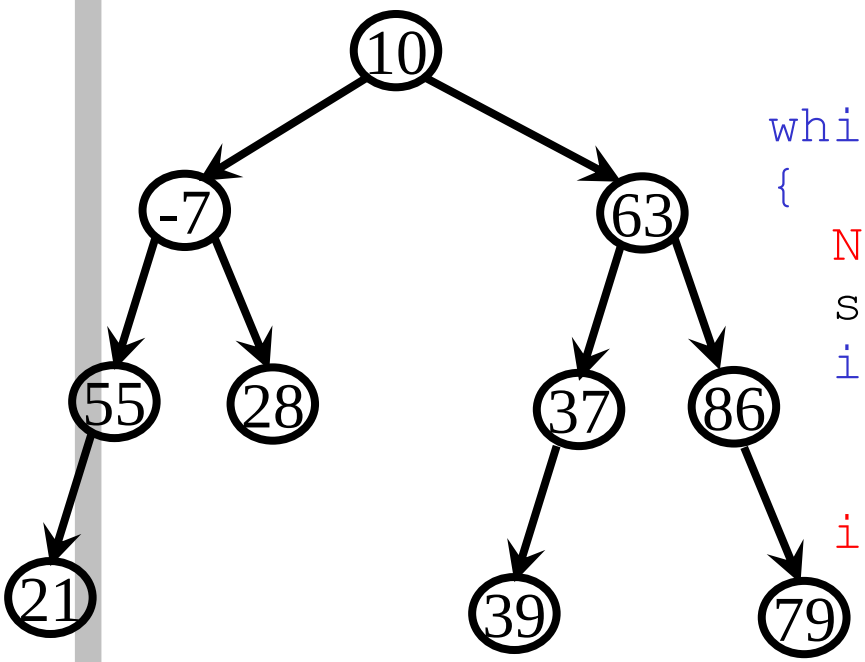
Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:



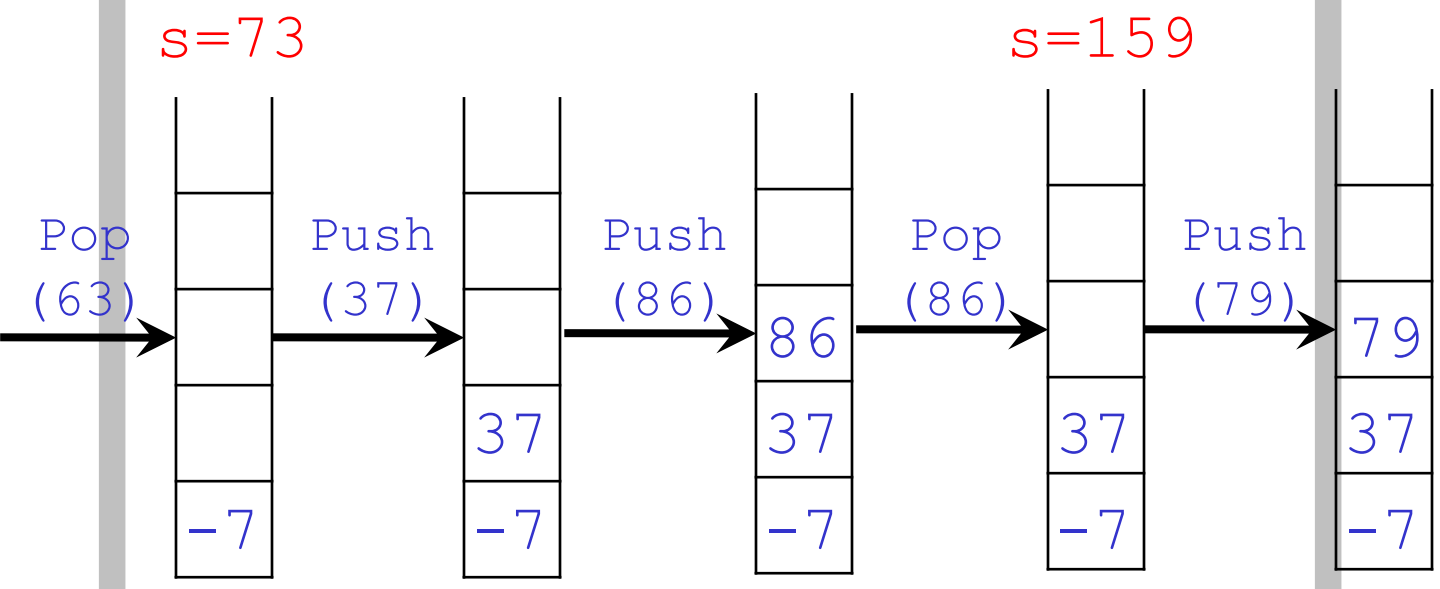

```
float s=0;
STACK stk;
Init(stk);
if (t!=NULL)
    Push(stk,t);
while(IsEmty(stk)==0)
{
  NODE*p = pop(stk);
  s=s+p->info;
  if (p->pLeft)
    Push(stk,
         p->pLeft);
  if (p->pRight)
    Push(stk,
         p->pRight);
}
```



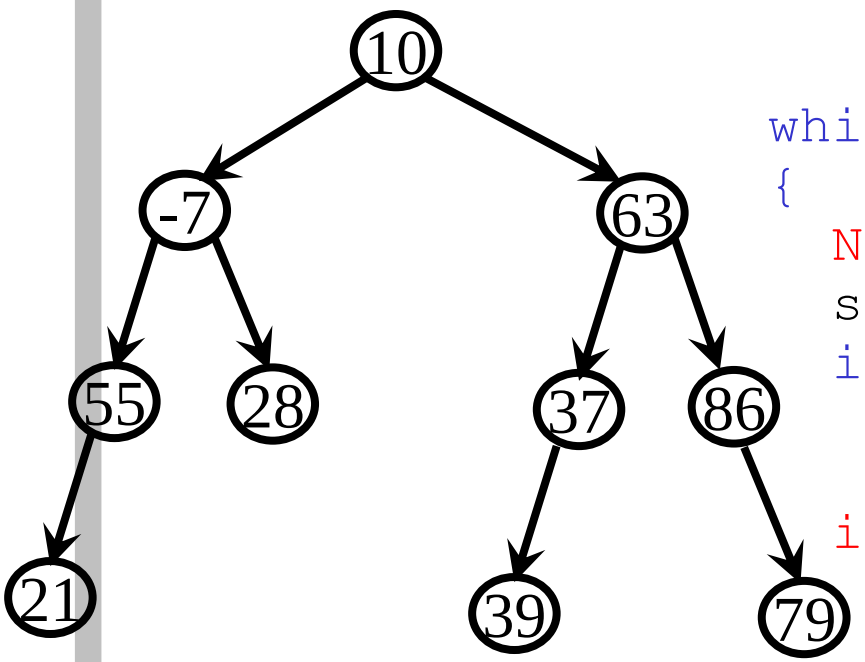
Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:



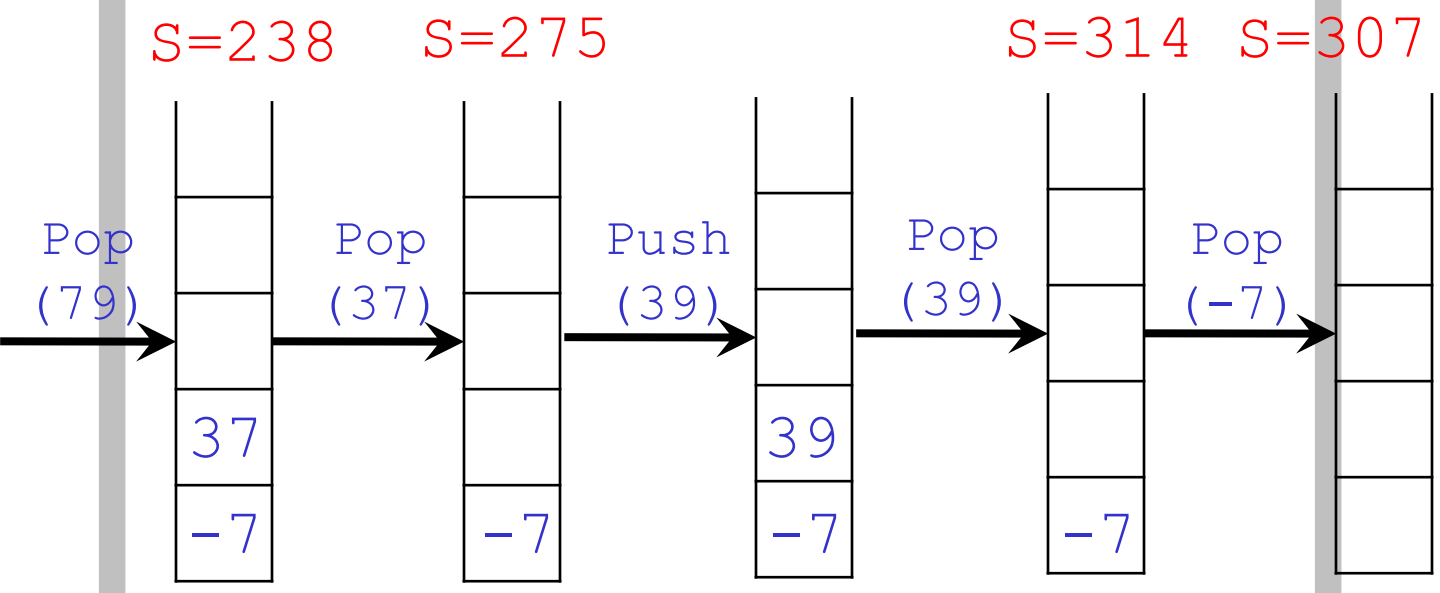

```
while(IsEmty(stk)==0)
{
   NODE*p = pop(stk);
   s=s+p->info;
   if (p->pLeft)
      Push(stk,
           p->pLeft);
   if (p->pRight)
      Push(stk,
           p->pRight);
}
```



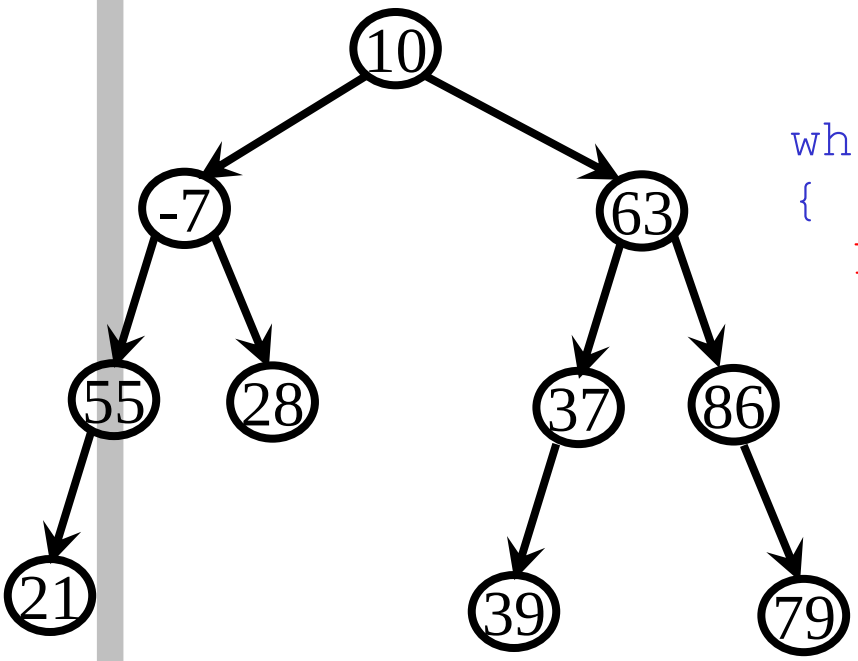
Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:




```
while(IsEmty(stk)==0)
{
   NODE*p = pop(stk);
   s=s+p->info;
   if (p->pLeft)
      Push(stk,
           p->pLeft);
   if (p->pRight)
      Push(stk,
           p->pRight);
}
```

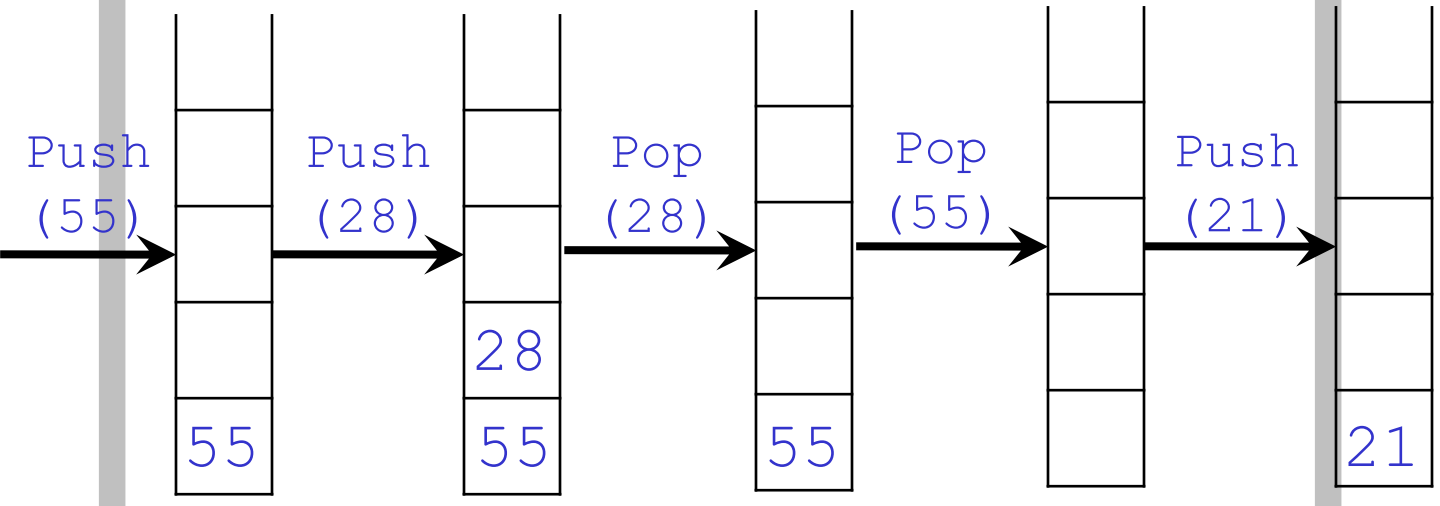


Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:

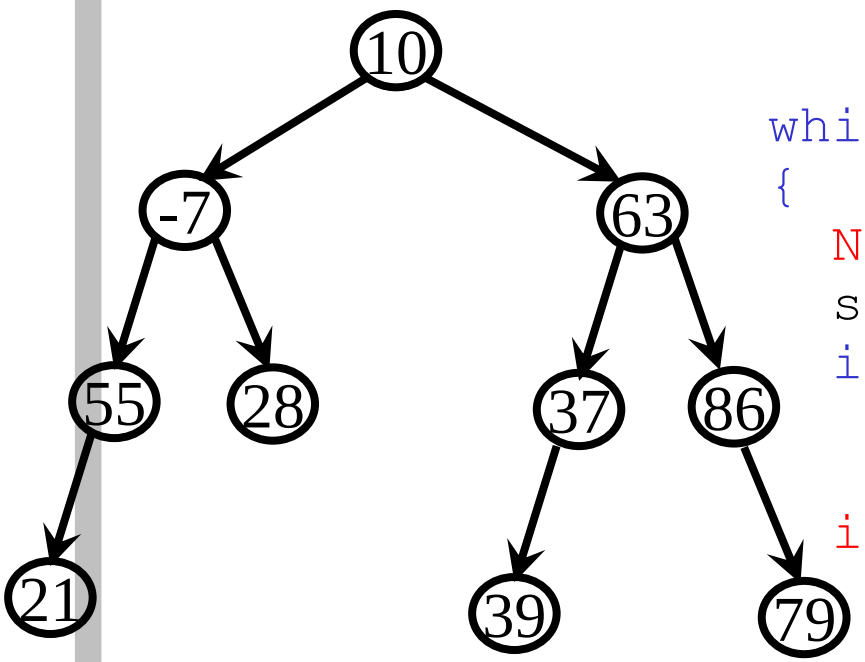



```
while(IsEmty(stk)==0)
{
   NODE*p = pop(stk);
   s=s+p->info;
   if (p->pLeft)
      Push(stk,
           p->pLeft);
   if (p->pRight)
      Push(stk,
           p->pRight);
}
```

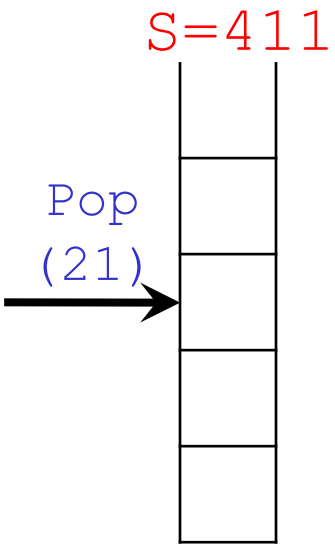
S=335 S=390



Minh họa: Chạy từng bước tính tổng các giá trị trong cây nhị phân sau:




```
while(IsEmty(stk)==0)
{
   NODE*p = pop(stk);
   s=s+p->info;
   if (p->pLeft)
     Push(stk,
          p->pLeft);
   if (p->pRight)
     Push(stk,
          p->pRight);
}
```



BÀI TẬP

- **Cho một ngăn xếp s và một đoạn chương trình sau:**

```
11. struct STACK s;
12. int x, y = 5;
13. Push(s, 8);
14. Push(s, y);
15. Push(s, 9);
16. Pop(s, x);
17. Push(s, 18);
18. Pop(s, x);
19. Push(s, 22);
20. while (IsEmpty(s) == 0)
21. {
22.     Pop(s, x);
23.     printf("%d ", y);
24. }
```

- **Hãy cho biết kết quả in ra màn hình khi thi hành đoạn chương trình trên là gì?**

BÀI TẬP

- **Bài làm**
- `struct STACK s;`
- Ngăn xếp rỗng
- `int x, y = 5;`
- Giá trị biến y=5, x ko xác định
- `Push(s, 8);`
- Ngăn xếp chứa (8)
- `Push(s, y);`
- Ngăn xếp chứa (5,8)
- `Push(s, 9);`
- Ngăn xếp chứa (9,5,8)
- `Pop(s, x);`
- Ngăn xếp chứa (5,8), x=9
- `Push(s, 18);`
- Ngăn xếp chứa (18,5,8)
- `Pop(s, x);`
- Ngăn xếp chứa (5,8), x=18
- `Push(s, 22);`
- Ngăn xếp chứa (22,5,8)

BÀI TẬP

```
1. while (IsEmpty(s) == 0)
2. {
3.     Pop(s, x);
4.     printf("%d ", y);
5. }
```

- **Lần lặp 1**
  1. Pop(s, x) x=22
  2. Ngăn xếp chứa (5,8)
  3. Xuất 5
- **Lần lặp 2**
  1. Pop(s, x) x=5
  2. Ngăn xếp chứa (8)
  3. Xuất 5
- **Lần lặp 3**
  1. Pop(s, x) x=8
  2. Ngăn xếp chứa ()
  3. Xuất 5
- **Kết luận: Đoạn chương trên xuất 5 5 5.**