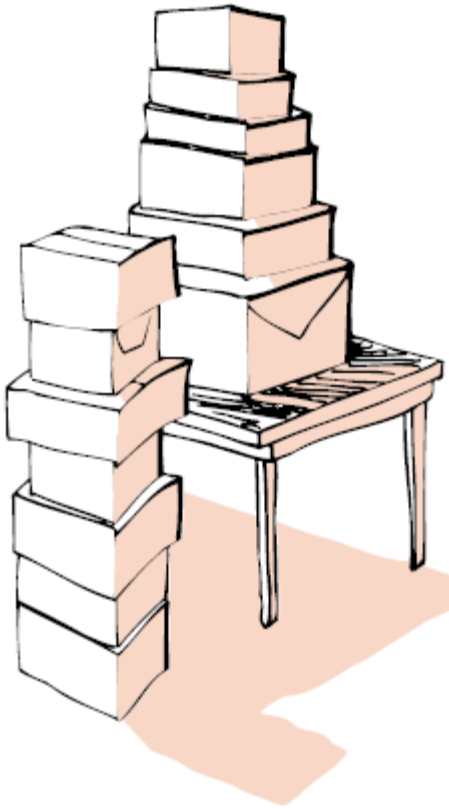
C Programming Basic – week 4

Chủ đề

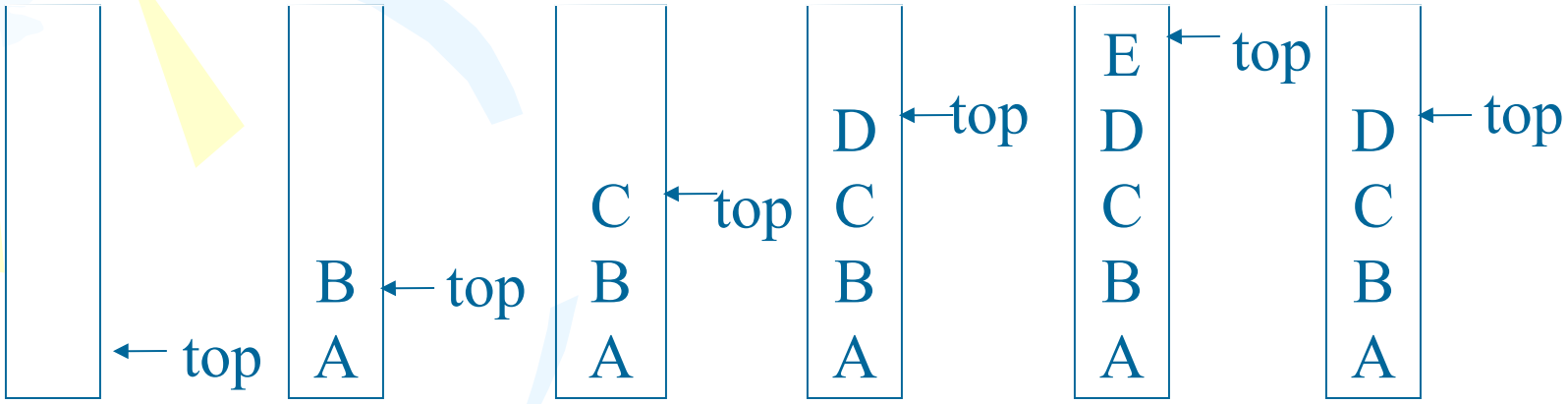
- Ngăn xếp
  - Cài đặt dựa trên mảng
  - Cài đặt dựa trên danh sách liên kết
- Bài tập



Ngăn xếp

- Ngăn xếp là **một cấu trúc dữ liệu tuyến tính** mà **chỉ có thể truy cập** ở một đầu để lưu trữ và truy vấn dữ liệu

- Cấu trúc LIFO (Last In First Out)

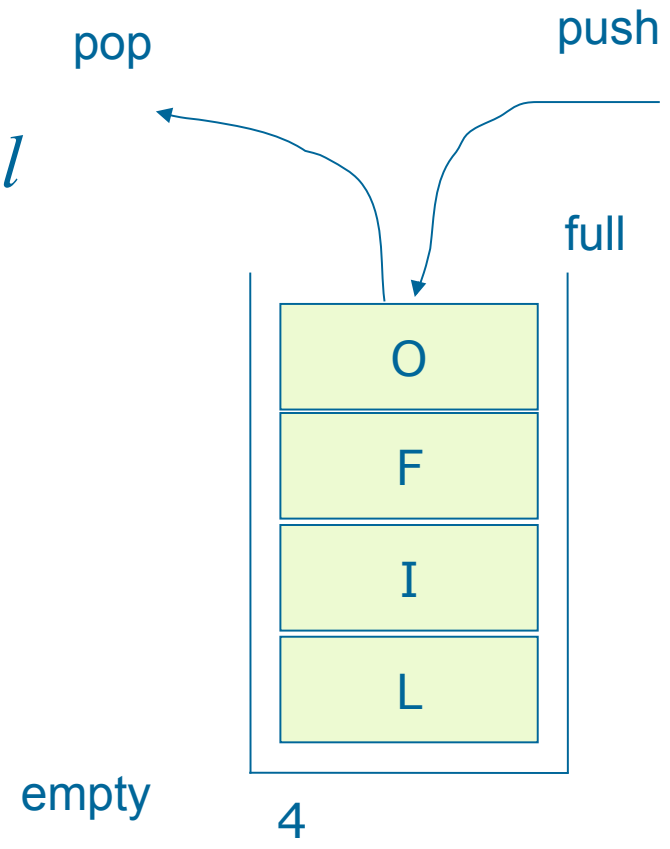


Chèn và xóa phần tử trong ngăn xếp

Các thao tác trên ngăn xếp

- *initialize(stack)* --- xóa ngăn xếp
- *empty(stack)* --- kiểm tra rỗng
- *full(stack)* --- kiểm tra đầy
- *push(el,stack)* --- đưa phần tử *el* vào đầu ngăn xếp
- *pop(stack)* --- lấy phần tử trên cùng của ngăn xếp

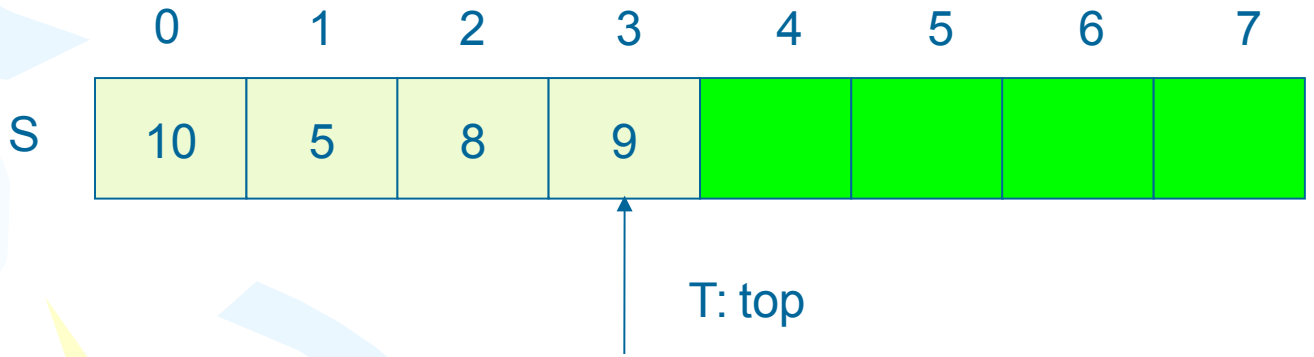
- Cài đặt ngăn xếp ntn?



Phân tách cài đặt vs đặc tả

- INTERFACE: (giao diện) cung cấp các đặc tả của các thao tác
- IMPLEMENTATION: (cài đặt) cung cấp mã nguồn của thao tác
- CLIENT: chương trình sử dụng các thao tác
- Cài đặt dựa trên mảng hoặc danh sách liên kết
- Client có thể làm việc ở mức trừu tượng cao

Cài đặt sử dụng mảng

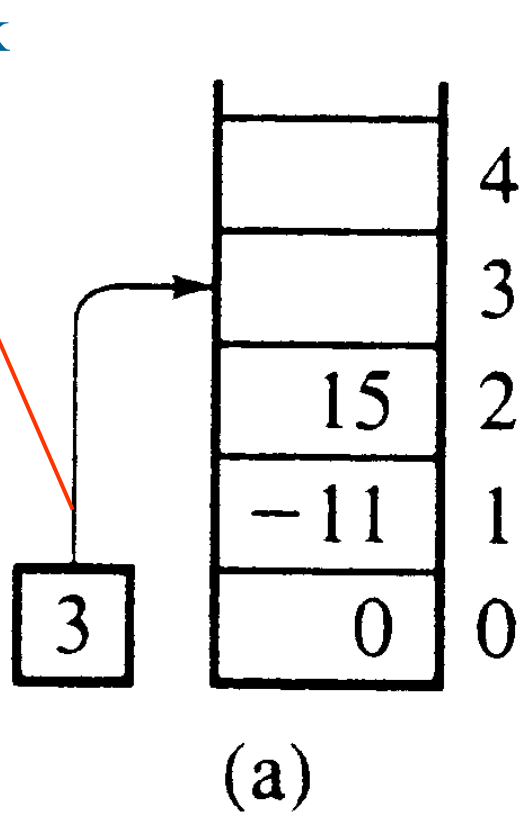


- Các phần tử được lưu trữ trên mảng
- Ngăn xếp rỗng: top= 0
- Ngăn xếp đầy: top = Max_Element

Đặc tả ngăn xếp (stack.h)

```
#define Max 50
typedef int Eltype;
typedef Eltype StackType[Max];
int top;

void Initialize(StackType stack);
int empty(StackType stack);
int full(StackType stack);
void push(Eltype el, StackType stack);
Eltype pop(StackType stack);
```



Cài đặt ngăn xếp (stack.c)

```
Initialize(StackType stack)
{
    top = 0;
}
empty(StackType stack)
{
    return top == 0;
}
full(StackType stack)
{
   return top == Max;
}
```

```
push(Eltype el, StackType stack)
{
   if (full(*stack))
        printf("stack overflow");
   else stack[top++] = el;
 }
Eltype pop(StackType stack)
{
    if (empty(stack))
        printf("stack underflow");
    else return stack[--top];
 }
```

Cài đặt dựa trên cấu trúc

- Ngăn xếp được khai báo như *một cấu trúc* với hai trường: một để lưu trữ, một để quản lý vị trí trên cùng

```
#define Max 50
typedef int Eltype;
typedef struct StackRec    {
   Eltype storage[Max];
   int top;
};
typedef struct StackRec StackType;
```

Cài đặt dựa trên cấu trúc (2)

```
Initialize(StackType *stack)
{
    (*stack).top=0;
}
empty(StackType stack)

{
     return stack.top ==0;
}
full(StackType stack)
{
    return stack.top == Max;

    (*stack).top];;
}
```

```
push(Eltype el, StackType *stack)
 {
      if (full(*stack))
           printf("stack overflow");
      else (*stack).storage[
           (*stack).top++]=el;
  }
 Eltype pop(StackType *stack)
  {
  if (empty(*stack))
       printf("stack underflow");
       else return
       (*stack).storage[--

   }
```

Biên dịch với thư viện

Ta có các tệp stack.h, stack.c và test.c

Chèn dòng sau
#include "stack.h"
vào stack.c và test.c

gcc – c stack.c
gcc –c test.c
gcc – o test.out test.o stack.o

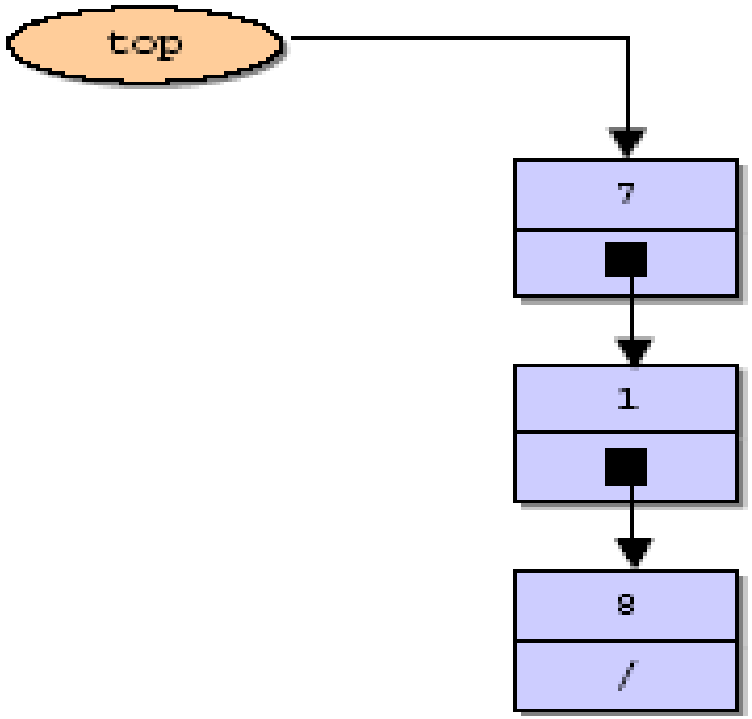
Chương trình sử dụng ngăn xếp

- Viết chương trình chuyển đổi số thập phân sang dạng nhị phân tương ứng sử dụng thư viện ngăn xếp
- Mở rộng để chuyển đổi sang dạng thập lục phân

Cài đặt dựa trên danh sách liên kết

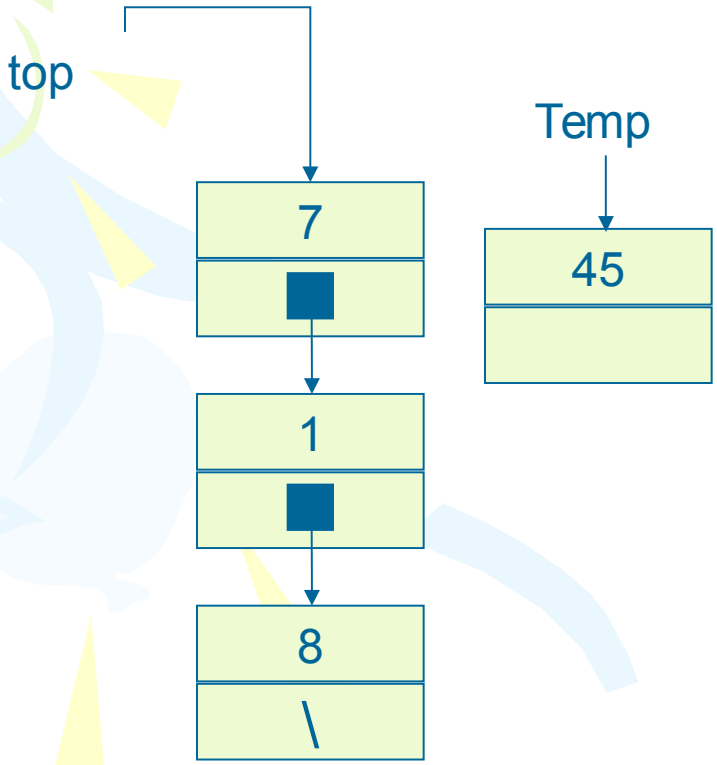
- Điểm khác biệt giữa danh sách liên kết thông thường và ngăn xếp dựa trên danh sách liên kết là một số thao tác không thực hiện được
- Thao tác chèn: push().
  - Tương tự chèn vào đầu danh sách
- Thao tác xóa: pop()
  - Tương tự xóa ở đầu danh sách

Khai báo



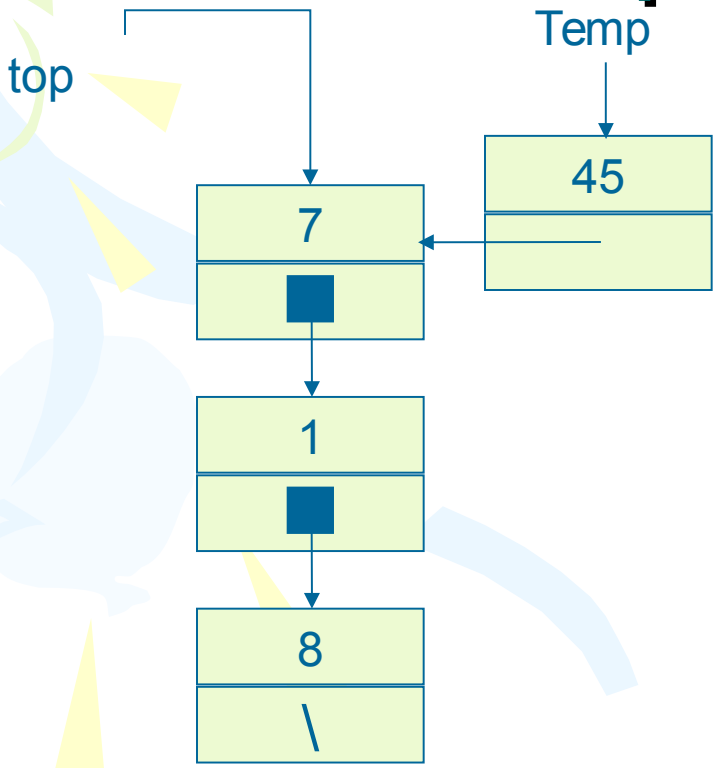

```
struct node {
int data;
struct node *link;
};
```

Push



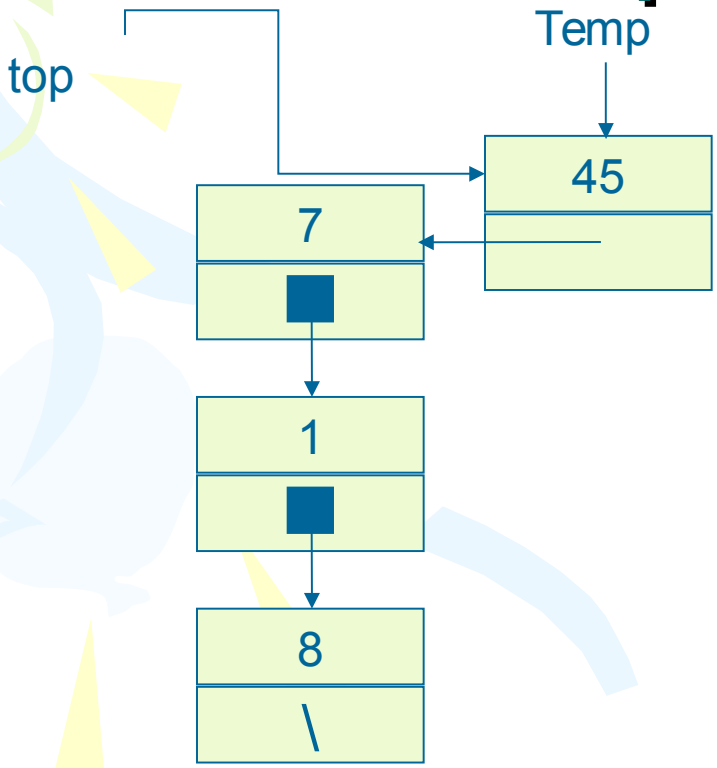

```
struct node *push(struct node *p, int value)
{
   struct node *temp;
   temp=(struct node *)malloc(sizeof(struct
    node));
    if(temp==NULL)  {
      printf("No Memory available Error\n");
      exit(0);
    }
   temp->data = value;
   temp->link = p;
   p = temp;
   return(p);
}
```

Push (2)



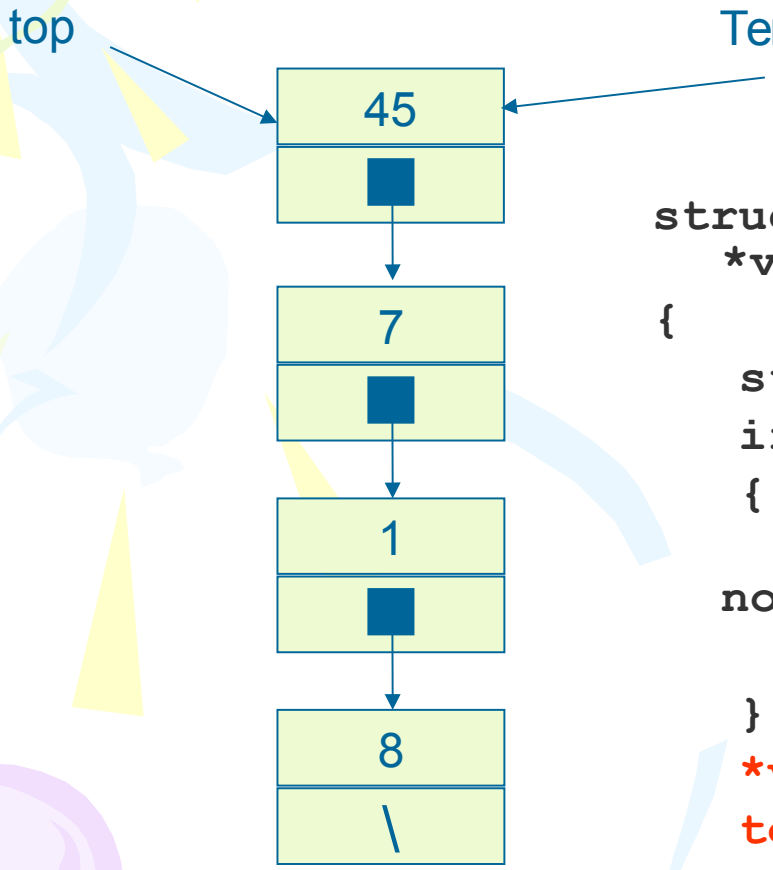

```
struct node *push(struct node *p, int value)
{
   struct node *temp;
   temp=(struct node *)malloc(sizeof(struct
    node));
    if(temp==NULL)  {
      printf("No Memory available Error\n");
      exit(0);
    }
   temp->data = value;
   temp->link = p;
   p = temp;
   return(p);
}
```

Push (3)




```
struct node *push(struct node *p, int value)
{
   struct node *temp;
   temp=(struct node *)malloc(sizeof(struct
    node));
    if(temp==NULL)  {
      printf("No Memory available Error\n");
      exit(0);
    }
   temp->data = value;
   temp->link = p;
   p = temp;
   return(p);
}
```

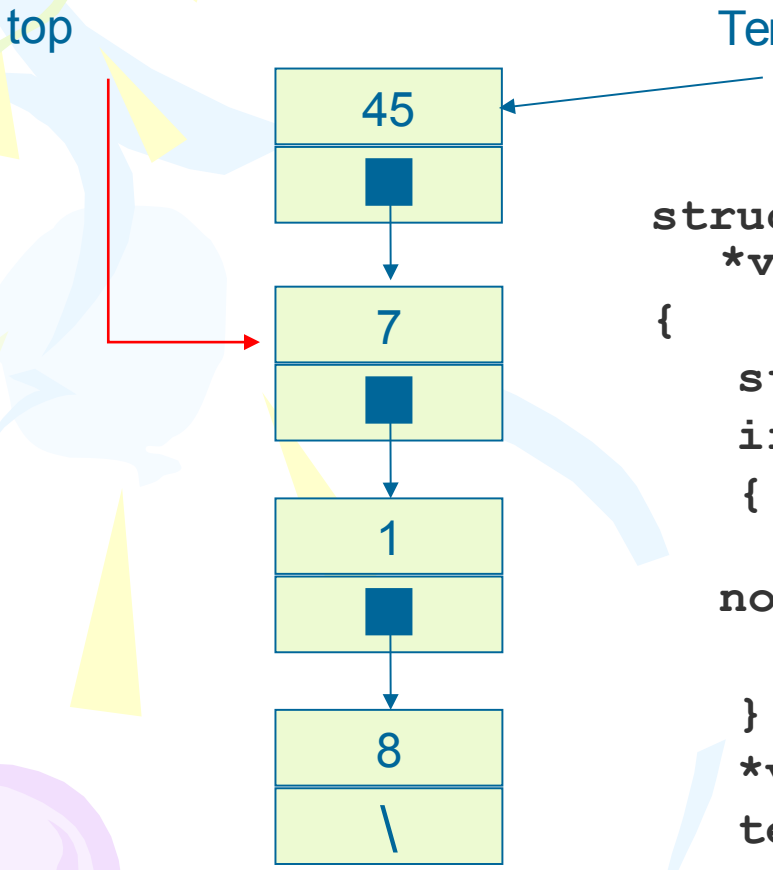
Pop




```
struct node *pop(struct node *p, int
   *value)
{
    struct node *temp;
    if(p==NULL)
    {
       printf(" The stack is empty can
   not pop Error\n");
       exit(0);
    }
    *value = p->data;
    temp = p;
    p = p->link;
    free(temp);
    return(p);
}
```

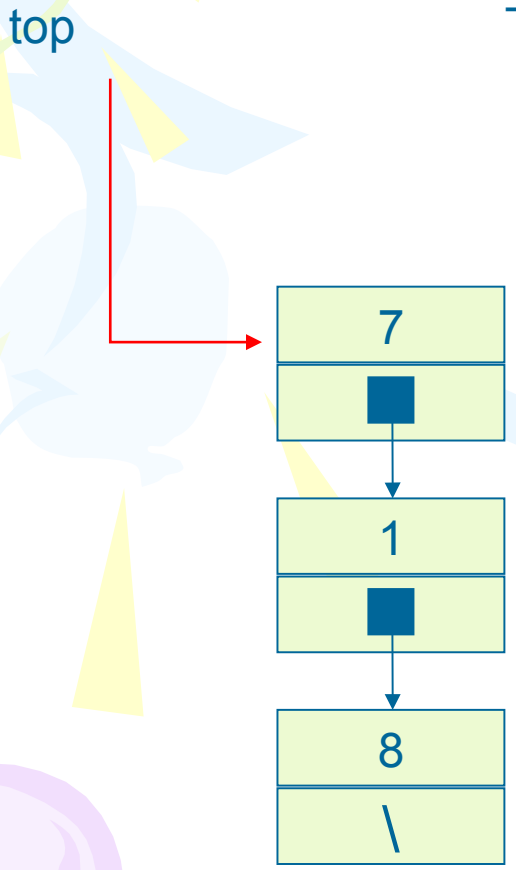
Value at top element need to be save before pop operation

Pop (2)




```
struct node *pop(struct node *p, int
   *value)
{
    struct node *temp;
    if(p==NULL)
    {
       printf(" The stack is empty can
   not pop Error\n");
       exit(0);
    }
    *value = p->data;
    temp = p;
    p = p->link;
    free(temp);
    return(p);
}
```

Pop (3)




```
struct node *pop(struct node *p, int
   *value)
{
    struct node *temp;
    if(p==NULL)
    {
       printf(" The stack is empty can
   not pop Error\n");
       exit(0);
    }
    *value = p->data;
    temp = p;
    p = p->link;
    free(temp);
    return(p);
}
```

Chú ý

- Cài đặt hàm FreeStack trong thư viện để xóa ngăn xếp

Sử dụng ngăn xếp

```
# include <stdio.h>
# include <stdlib.h>
void main()
{
   struct node *top = NULL;
   int n,value;
   do
   {
      do
      {
         printf("Enter the element
        to be pushed\n");
         scanf("%d",&value);
         top = push(top,value);
         printf("Enter 1 to
    continue\n");
         scanf("%d",&n);
      } while(n == 1);
```

```
printf("Enter 1 to pop an element\n");
      scanf("%d",&n);
      while( n == 1)
      {
            top = pop(top,&value);
            printf("The value poped is
     %d\n",value);
            printf("Enter 1 to pop an element\n");
            scanf("%d",&n);
      }
      printf("Enter 1 to continue\n");
      scanf("%d",&n);
   } while(n == 1);
}
```

Sử dụng ngăn xếp (2)

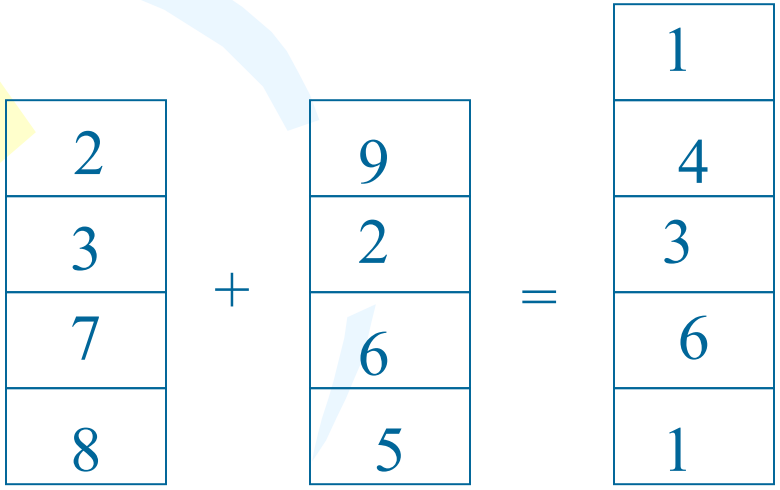
```
printf("Enter 1 to pop an element\n");
     scanf("%d",&n);
     while( n == 1)
     {
          top = pop(top,&value);
        printf("The value poped is %d\n",value);
          printf("Enter 1 to pop an element\n");
          scanf("%d",&n);
     }
     printf("Enter 1 to continue\n");
     scanf("%d",&n);
   } while(n == 1);
}
```

Exercise 4.1

- Sử dụng ngăn xếp để viết chương trình nhận vào một xâu và đảo ngược xâu đó

Homework 1

- Viết chương trình yêu cầu người dùng nhập vào hai số nguyên dương và tính tổng hai số đó. Lưu số nhập vào dưới dạng chuỗi các chữ số. Sử dụng ngăn xếp để thực hiện phép cộng.



8732 + 5629 = 14361

Gợi ý

Đọc các chữ số của số đầu tiên và lưu vào một ngăn xếp

Đọc các chữ số của số thứ hai và lưu vào một ngăn xếp khác

result=*0;*

while ít nhất một ngăn xếp còn chưa rỗng

lấy mỗi chữ số từ ngăn xếp và tính tổng

đẩy tổng (-10 nếu dư) vào ngăn xếp kết quả;

lưu số nhớ vào **result***;*

đưa số nhớ vào ngăn xếp kết quả nếu khác 0

lấy các chữ số từ ngăn xếp kết quả và hiển thị

Exercise 4.2

- Xây dựng danh bạ điện thoại
- Khai báo cấu trúc "Address" chứa ít nhất "name", "telephone number" và "e-mail address".
- Viết chương trình sao chép danh bạ từ một tệp sang tệp khác, sử dụng ngăn xếp. Đầu tiên, đọc danh bạ từ tệp vào đưa vào một ngăn xếp. Sau đó lấy các bản ghi ra khỏi ngăn xếp và ghi vào tệp.

Exercise 4.3

- Viết chương trình chuyển một biểu thức từ dạng infix sang dạng reverse polish. Biểu thức bao gồm các toán hạng là các số nguyên dương (từ 1 tới 9) và bốn toán tử (+, -, *, /). Đọc một biểu thức ở dạng infix từ bàn phím, chuyển sang dạng reverse polish và in ra màn hình.
- Ví dụ,

  3+5*4
  là đầu vào, đầu ra có dạng

  3 5 4 * +

STACK.h

```
void STACKinit(int);
int STACKempty();
void STACKpush(Item);
Item STACKpop();
```

STACK.c

```
#include <stdlib.h>
#include "Item.h"
#include "STACK.h"
static Item *s;
static int N;
void STACKinit(int maxN)
  { s = malloc(maxN*sizeof(Item)); N = 0; }
int STACKempty()
  { return N == 0; }
void STACKpush(Item item)
  { s[N++] = item; }
Item STACKpop()
  { return s[--N]; }
```

Exercise 4.4

- Viết chương trình tính một biểu thức dạng postfix bao gồm phép nhân và phép cộng các số nguyên.
- Ví dụ:

- ./posteval 5 4 + 6 * => 54

Homework 2

- Viết chương trình cho phép nhân số lớn

- Gợi ý: Chia số lớn thành các đoạn độ dài 3 (hoặc length-*k*). Thực hiện phép nhân như bình thường và cộng kết quả