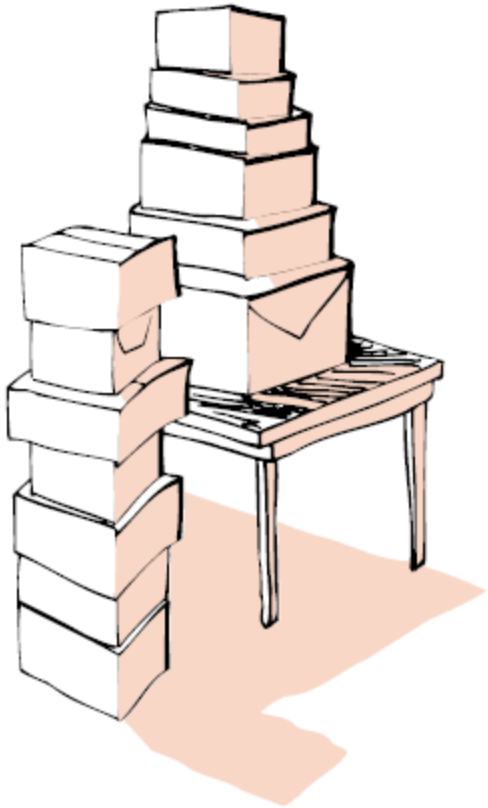
Basic C Programming
Bài 4+5
(Lớp học lần 2)

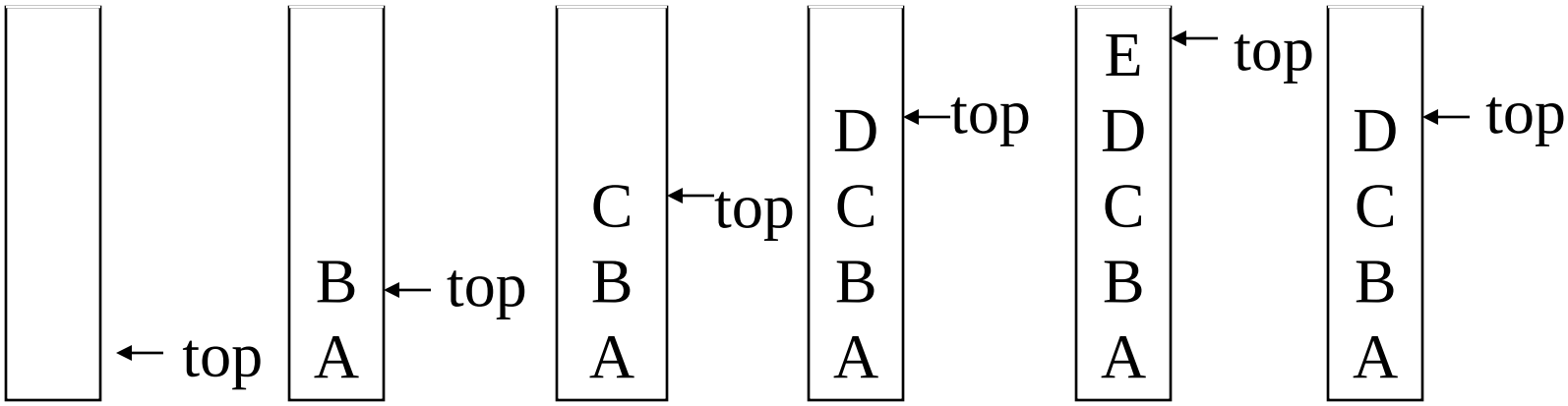
Nội dung

- Cấu trúc dữ liệu: Stack
  - Cài đặt stack sử dụng mảng
  - Cài đặt stack sử dụng danh sách liên kết
- Cấu trúc dữ liệu Queue
  - Cài đặt queue sử dụng mảng
  - Cài đặt queue sử dụng danh sách liên kết
- Bài tập về Stack & Queue



Stack

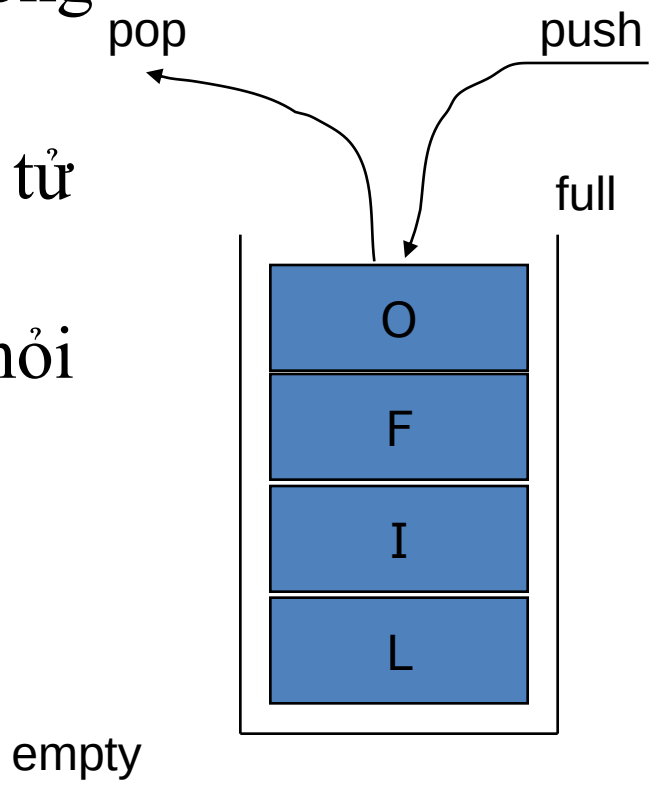
- Stack là một cấu trúc dữ liệu trong đó chỉ có thể thực hiện truy cập (đọc/ghi) tại một đầu của nó
- Là cấu trúc LIFO (Last In First Out) (Vào sau Ra trước)



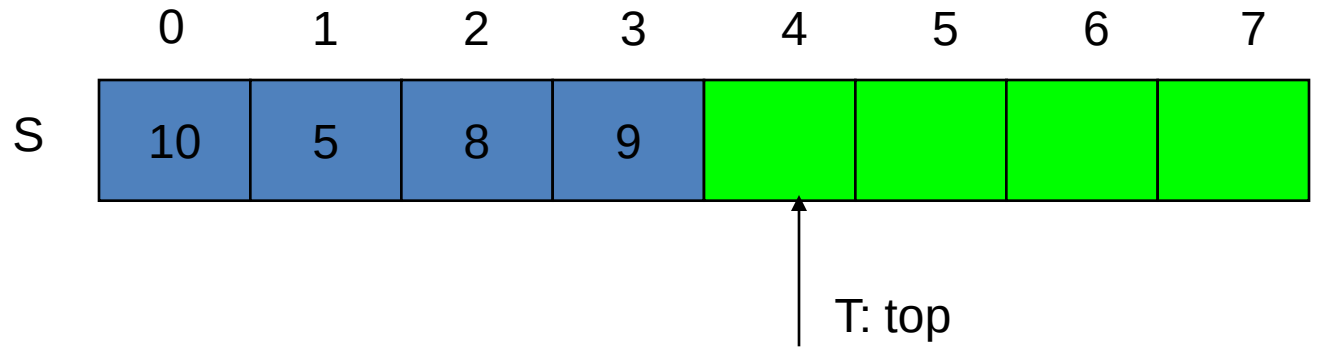
Ví dụ về chèn & xóa phần tử

Các thao tác trên stack

- *initialize(stack)* --- xóa các phần tử
- *empty(stack)* --- kiểm tra stack rỗng
- *full(stack)* --- kiểm tra stack đầy
- *push(el,stack)* --- chèn một phần tử vào đỉnh stack
- *pop(stack)* --- lấy một phần tử khỏi đỉnh stack



Cài đặt stack sử dụng mảng

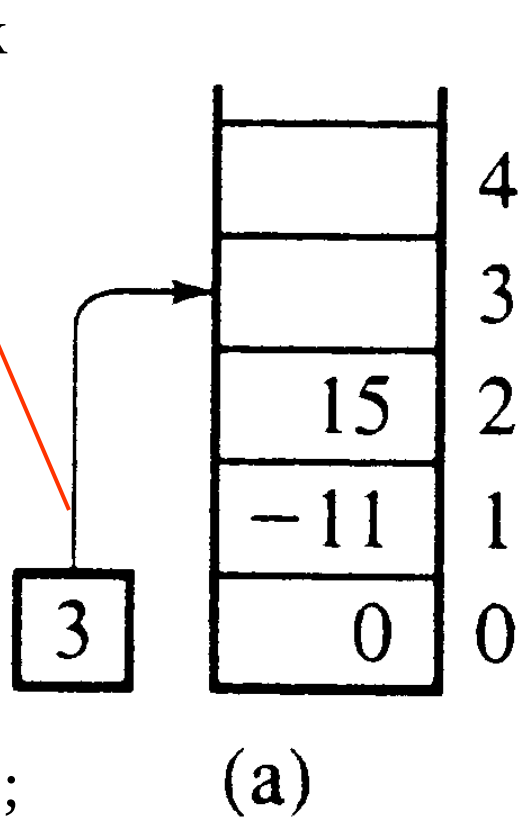


- Mỗi phần tử của stack được lưu trong một phần tử của mảng
- Push: chèn các phần tử vào mảng từ trái sang phải
- Pop: lấy phần tử cuối cùng (chỉ số lớn nhất) trong mảng
- Top: chỉ số phần tử sau phần tử cuối cùng
- stack khi empty: top= 0
- stack khi full: top = Max_Element

Các khai báo cần có (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 các hàm (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 stack sử dụng cấu trúc

- Cấu trúc gồm 2 trường
  - Mảng lưu trữ phần tử
  - Chỉ số top

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

Cài đặt stack sử dụng cấu trúc

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

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

}
```

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

Biên dịch

Chúng ta có: stack.h, stack.c và test.c
Trong stack.c và test.c thêm khai báo
#include "stack.h"

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

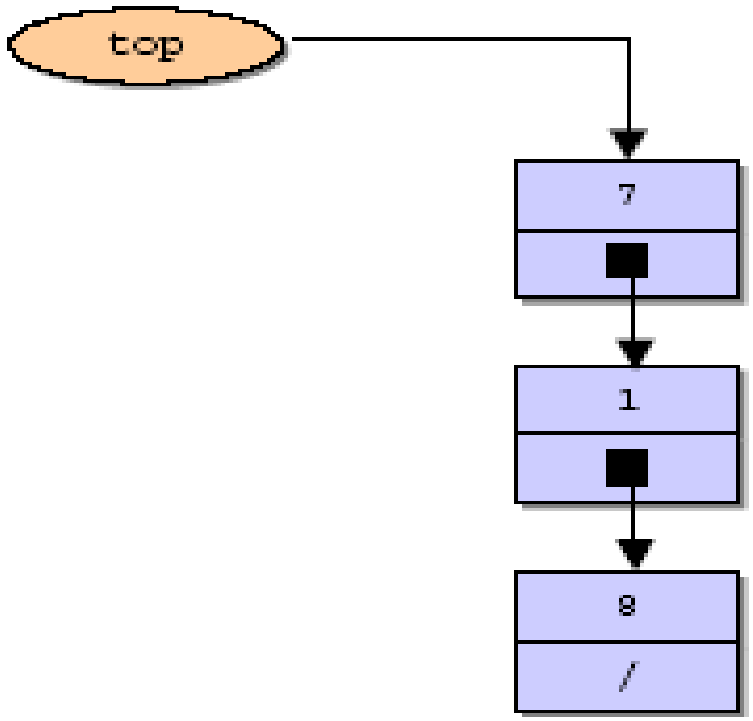
Bài 1

- Cài đặt stack bằng mảng để lưu trữ các số nguyên
- Sử dụng cấu trúc lặp, cho phép nhập các số nguyên vào từ bàn phím và chèn vào stack
- Sử dụng cấu trúc lặp, cho phép lấy ra lần lượt các phần tử khỏi stack

Chương trình minh họa

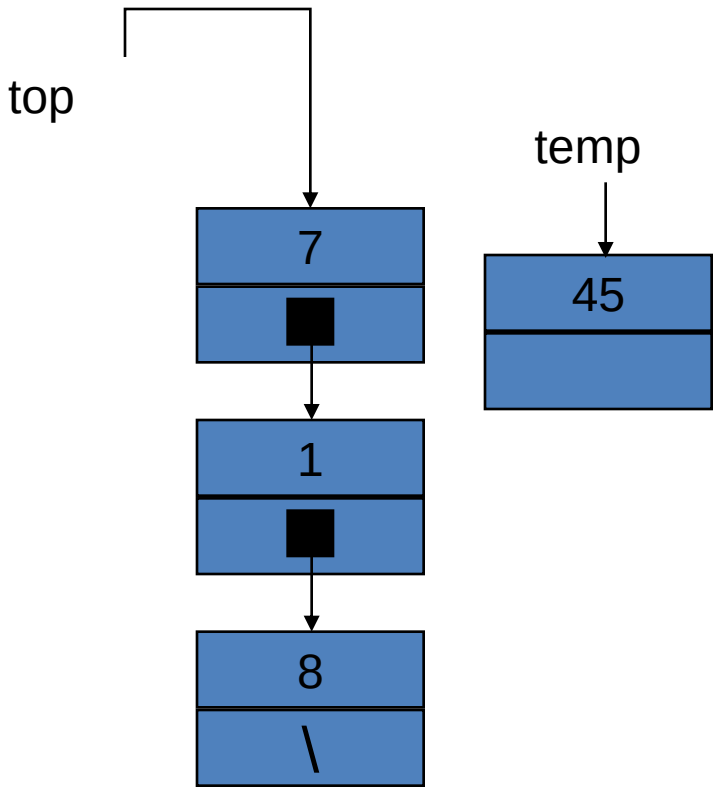
```
# include <stdio.h>
# include <stdlib.h>
#include « stack.h »
….
void main()
{
   StackType stack;
   int n,value;
   do
   {
      do
      {
         printf("Enter the element
          to be pushed\n");
         scanf("%d",&value);
         push(value, stack);
         printf("Enter 1 to
                continue\n");
         scanf("%d",&n);
      } while(n == 1);
      printf("Enter 1 to pop an element\n");
         scanf("%d",&n);
         while( n == 1)
         {
              value = pop(stack);
               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);
}
```

Cài đặt stack sử dụng danh sách liên kết



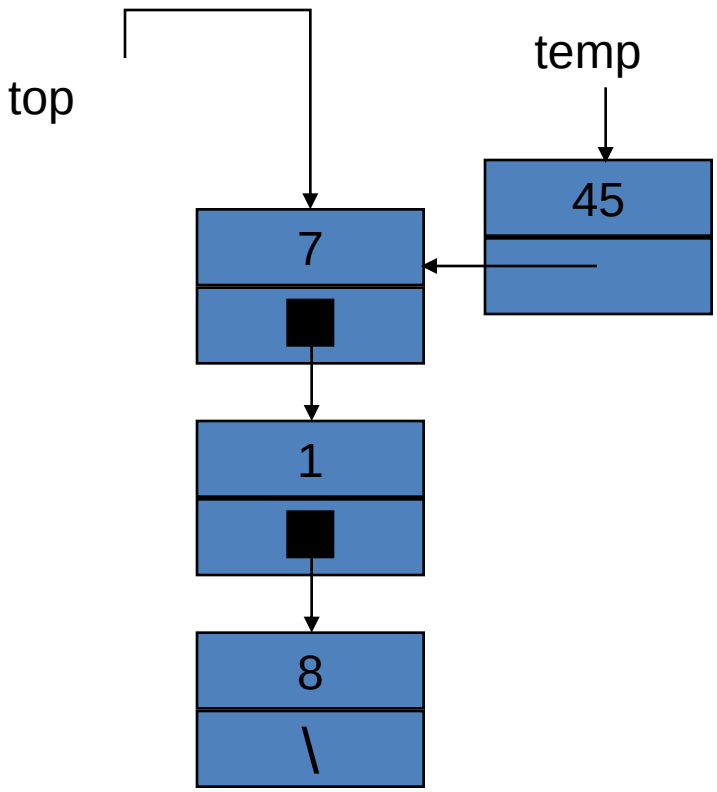

```
struct node {
int data;
struct node *link;
};
```

Push



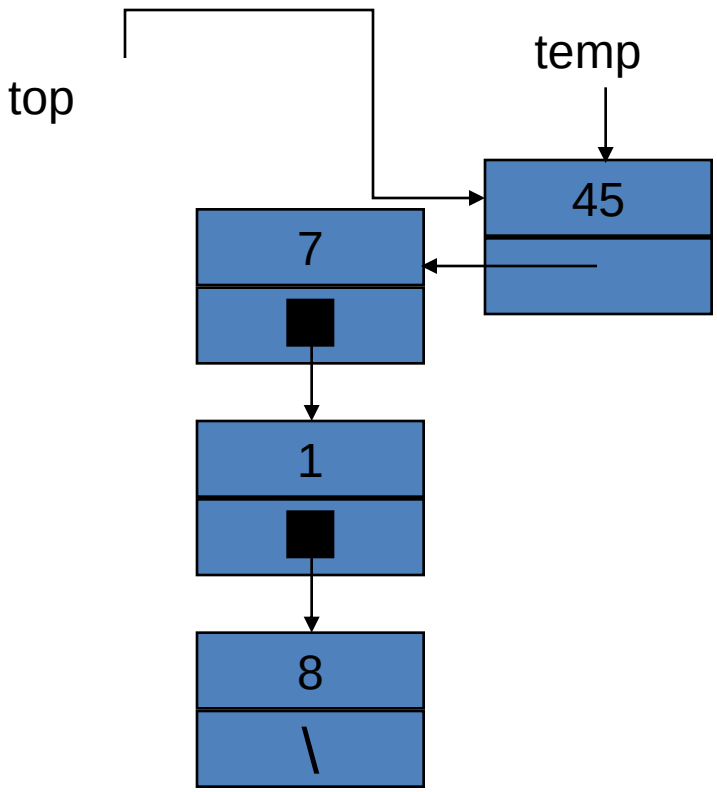

```
struct node *push(struct node *top, int value)
{
   struct node *temp;
   temp=(struct node *)malloc(sizeof(struct node));
   if(temp==NULL)  {
      printf("No Memory available Error\n");
      exit(0);
   }
   temp->data = value;
   temp->link = top;
   top = temp;
   return(top);
}
```

Push



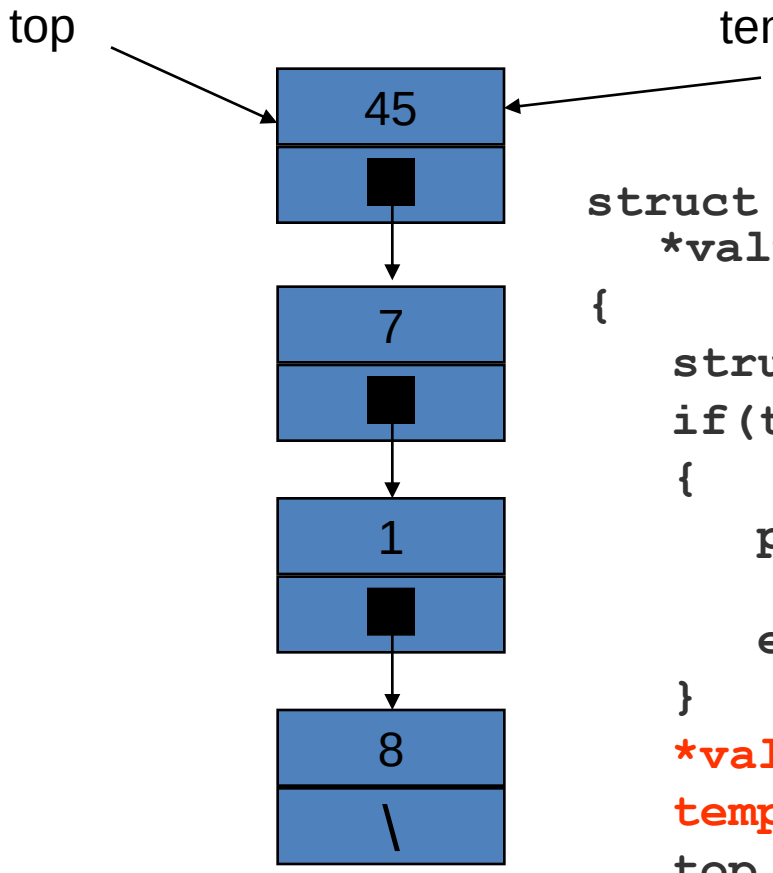

```
struct node *push(struct node *top, int value)
{
   struct node *temp;
   temp=(struct node *)malloc(sizeof(struct node));
      if(temp==NULL)  {
      printf("No Memory available Error\n");
      exit(0);
      }
   temp->data = value;
   temp->link = top;
   top = temp;
   return(top);
}
```

Push




```
struct node *push(struct node *top, 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 = top;
   top = temp;
   return(top);
}
```

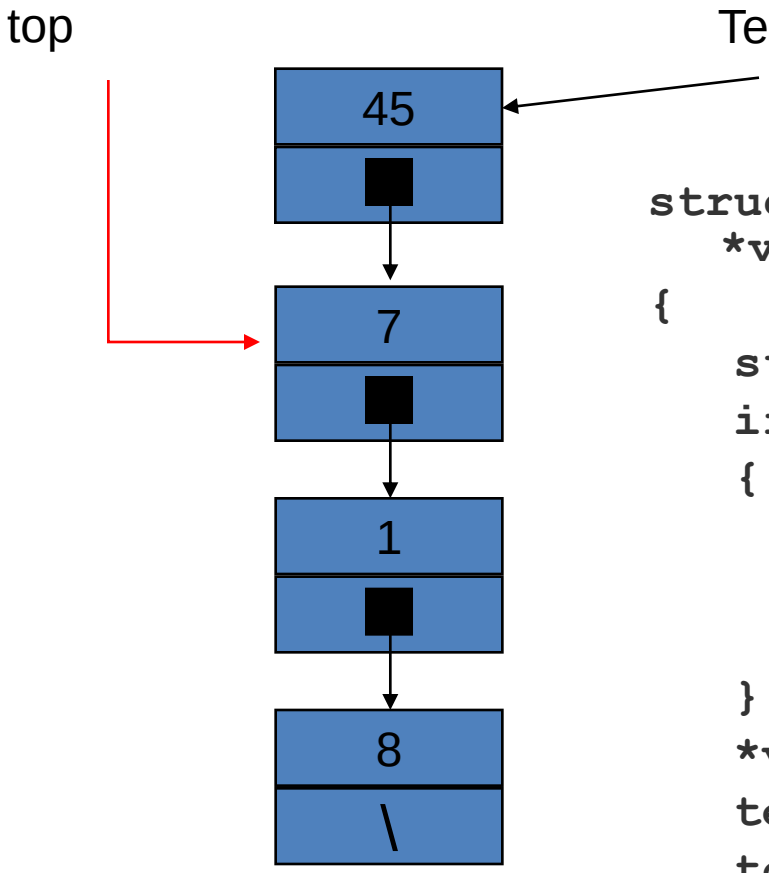
Pop (linked list)




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

Cần lưu lại giá trị ở đỉnh trước khi free nút đó

Pop (linked list)

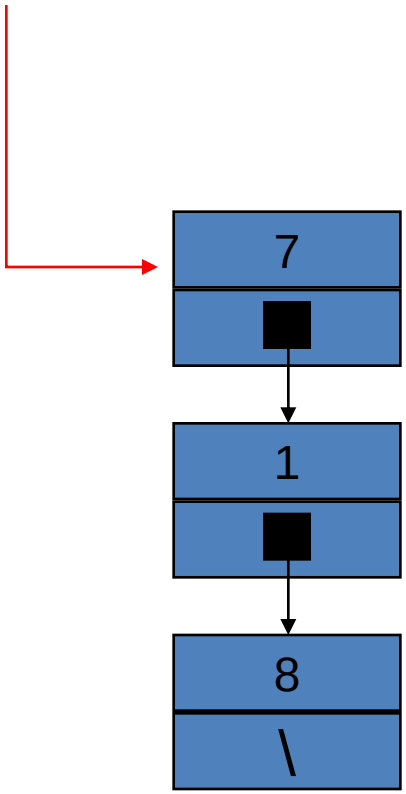



```
struct node *pop(struct node *top, int
   *value)
{
    struct node *temp;
    if(top==NULL)
    {
       printf(" The stack is empty can
        not pop Error\n");
       exit(0);
    }
    *value = top->data;
    temp = top;
    top = top->link;
    free(temp);
    return(top);
}
```

Pop (linked list)

top

Temp




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

Bài 2

- Cài đặt stack bằng danh sách liên kết để lưu trữ các số nguyên
- Sử dụng cấu trúc lặp, cho phép nhập các số nguyên vào từ bàn phím và chèn vào stack
- Sử dụng cấu trúc lặp, cho phép lấy ra lần lượt các phần tử khỏi stack

Bài tập

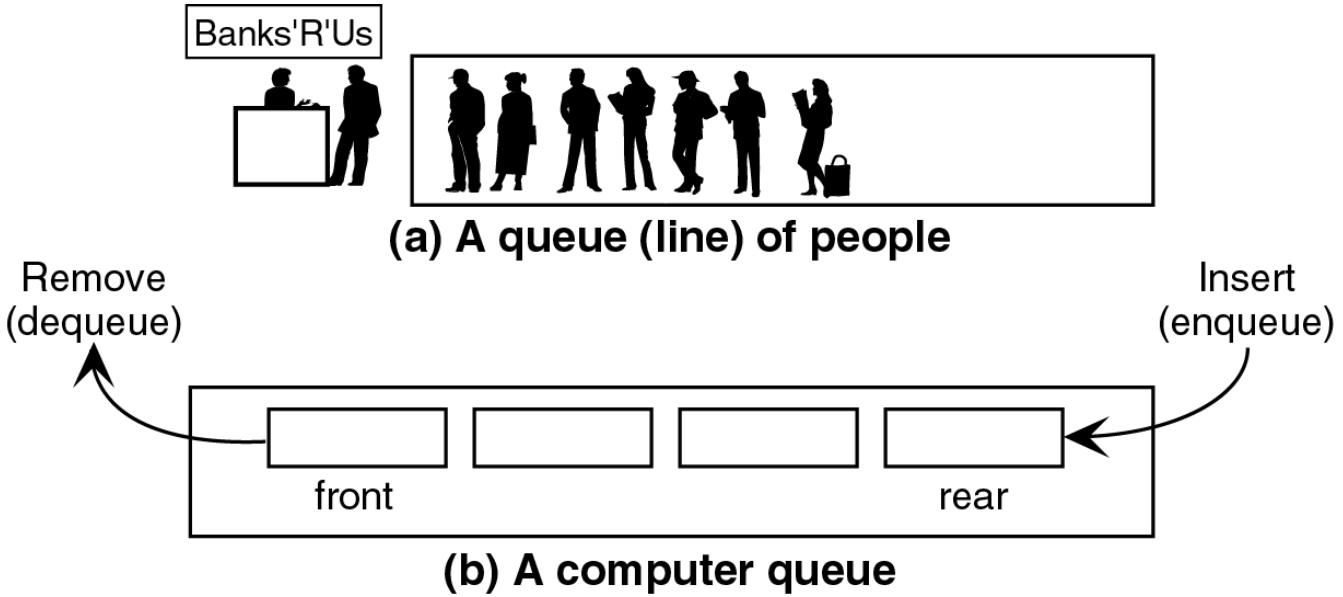
- Bài 23, 24 trong file bài tập

Chương trình minh họa

```
# include <stdio.h>
# include <stdlib.h>
….//định nghĩa stack, các hàm push, pop
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);
}
```

Queue

- Queue (hàng đợi)
- Cả 2 phía đều được sử dụng
  - Một phía: chèn phần tử - thao tác enqueue ở đuôi (rear)
  - Một phía: lấy ra phần tử - thao tác dequeue ở đầu (front)

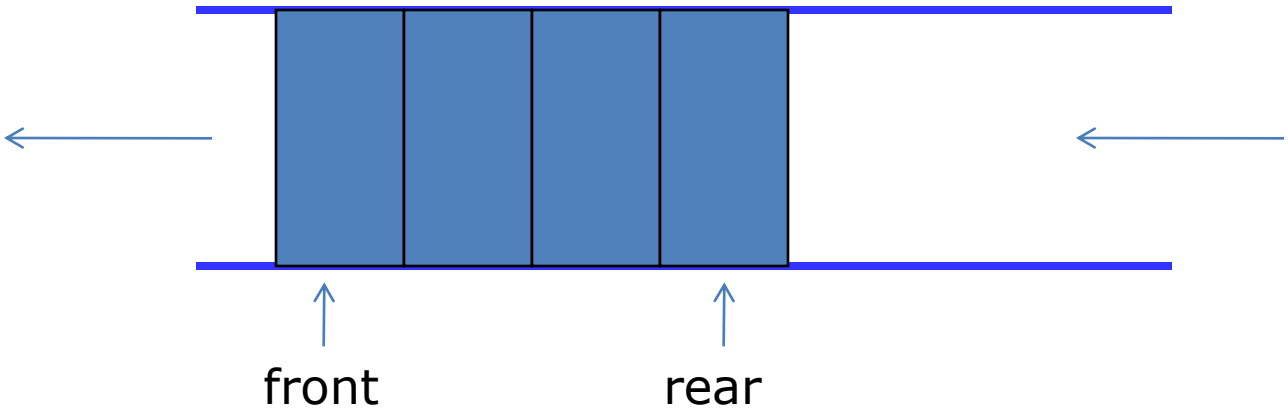


Cấu trúc FIFO

- Các phần tử trong Queue được lấy ra theo đúng thứ tự chúng được thêm vào

  FIFO structure: First in, First out

  (Vào trước, Ra trước)



Cài đặt queue sử dụng cấu trúc

```
#define MaxLength 100
typedef ... ElementType;
typedef struct {
   ElementType Elements[MaxLength];
   //Store the elements
   int Front, Rear;
} Queue;
```

Định nghĩa các hàm

```
void MakeNull_Queue(Queue *Q){
  Q->Front=-1;
  Q->Rear=-1;
}
int Empty_Queue(Queue Q){
  return Q.Front==-1;
}
int Full_Queue(Queue Q){
  return Q.Rear == MaxLength-1;
}
```

Enqueue

```
void EnQueue(ElementType X,Queue *Q){
   if (!Full_Queue(*Q)){
     if (Empty_Queue(*Q)) Q->Front=0;
     Q->Rear=Q->Rear+1;
     Q->Element[Q->Rear]=X;
   }
   else printf("Queue is full!");
}
```

Dequeue

```
void DeQueue(ElementType *X, Queue *Q){
   if (!Empty_Queue(*Q)){
      *X = Q->element[Q->Front];
     Q->Front=Q->Front+1;
     if (Q->Front > Q->Rear)
          MakeNull_Queue(Q);
   } else
      printf("Queue is empty!");
}
```

Cài đặt queue sử dụng danh sách liên kết

```
typedef ... ElementType;
typedef struct Node{
  ElementType Element;
  struct Node* Next;
}Node;
typedef Node* Position;
typedef struct{
  Position Front, Rear;
} Queue;
```

Khởi tạo Queue

```
void MakeNullQueue(Queue *Q){
  Q->Front = NULL;
  Q->Rear  = NULL;
}
```

Is-Empty

```
int EmptyQueue(Queue Q){
  return (Q.Front==NULL);
}
```

EnQueue

```
void EnQueue(ElementType X, Queue *Q){
   Node* new_node = (Node *)
                    malloc(sizeof(Node));
   new_node->Element = X;
   new_node->Next = NULL;
   if(EmptyQueue(Q))
      Q->Rear= Q->Front = new_node;
   else{ Q->Rear->Next=new_node;
         Q->Rear = Q->Rear->Next;  }
}
```

DeqQueue

```
void DeQueue(ElementType *X, Queue *Q){
   if (!Empty_Queue(Q)){
     Position T;
     T=Q->Front;
    *X = T->Element;
    Q->Front=Q->Front->Next;
    if(Q->Front==NULL) MakeNullQueue(Q);
    free(T);
   }
   else printf("Error: Queue is empty.");
}
```

Bài 3

- Cài đặt queue bằng mảng để lưu trữ các số nguyên
- Sử dụng cấu trúc lặp, cho phép nhập các số nguyên vào từ bàn phím và chèn vào queue
- Sử dụng cấu trúc lặp, cho phép lấy ra lần lượt các phần tử khỏi queue

Bài 4

- Tương tự bài 3, chỉ thay là sử dụng danh sách liên kết để lưu trữ queue

Bài tập

- Bài 25, 26, 27, 28 trong file Bài tập