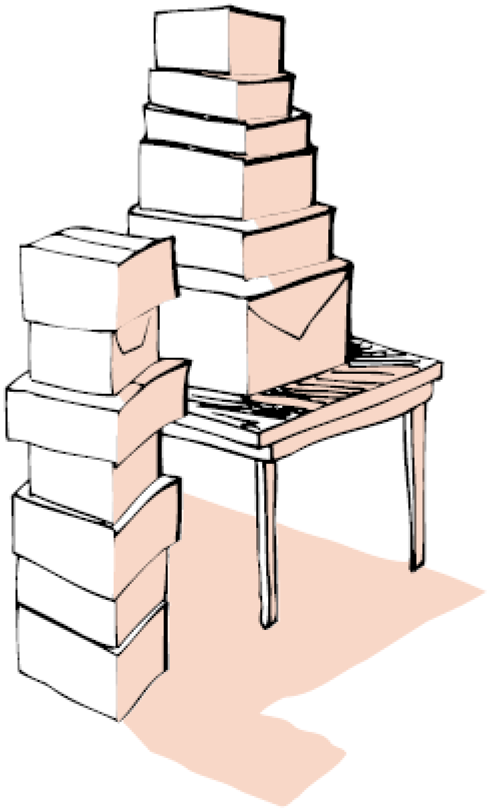
C Programming Basic – week 4

Topics

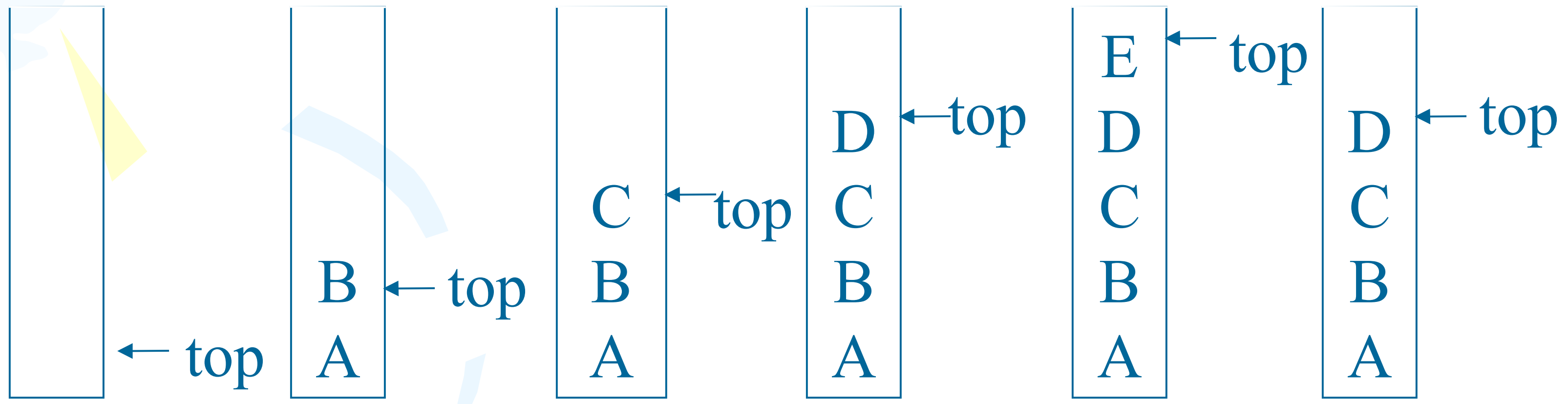
- Stack
  - Implementation using array
  - Implementation using linked list
- Exercises



Stack

- A stack is a **linear data structure** which can be **accessed only at one of its ends** for storing and retrieving data.

- A LIFO (Last In First Out) structure

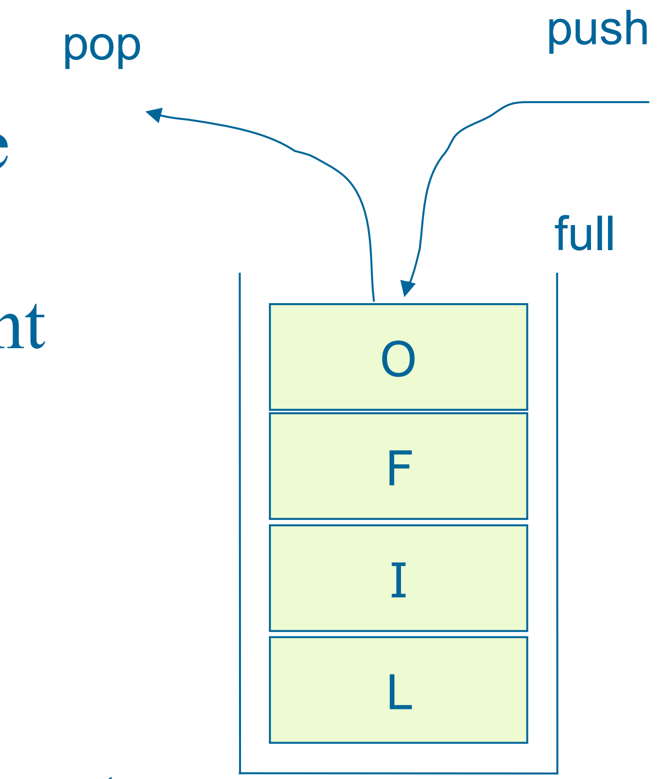


Inserting and deleting elements in a stack

Operations on a stack

- *initialize(stack)* --- clear the stack
- *empty(stack)* --- check to see if the stack is empty
- *full(stack)* --- check to see if the stack is full
- *push(el,stack)* --- put the element *el* on the top of the stack
- *pop(stack)* --- take the topmost element from the stack

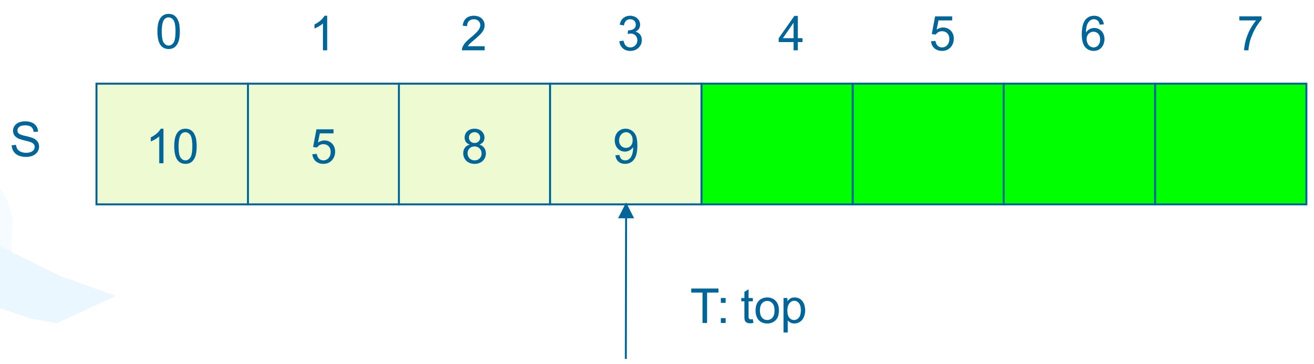
- How to implement a stack?



Separate implementation from specification

- INTERFACE: specify the allowed operations
- IMPLEMENTATION: provide code for operations
- CLIENT: code that uses them.
- Could use either array or linked list to implement stack
- Client can work at higher level of abstraction

Implementation using array

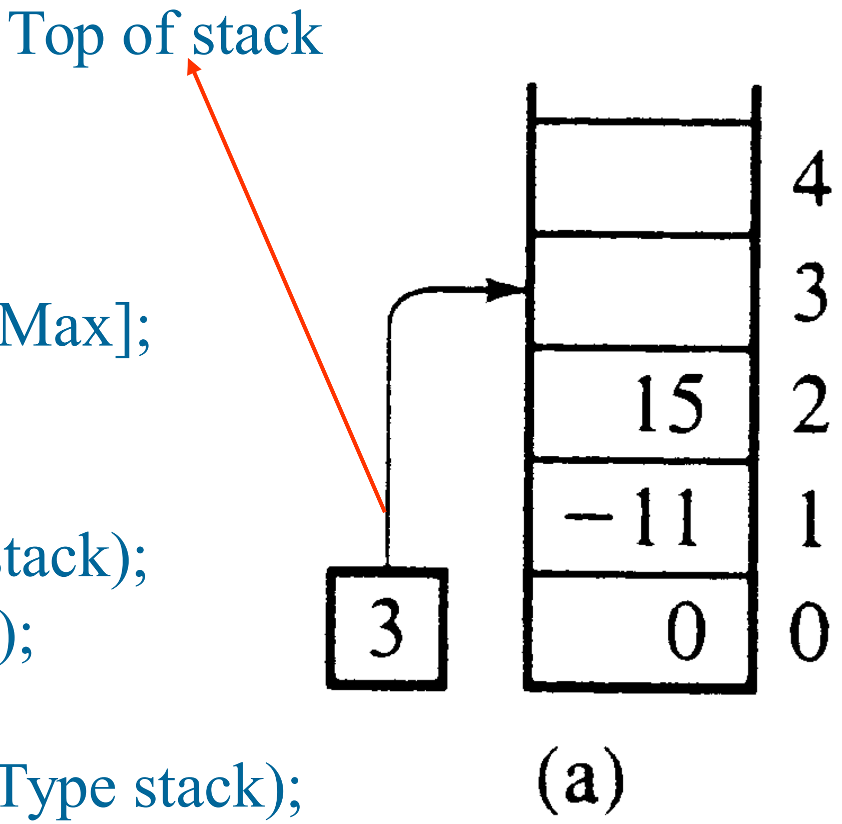


- Each element is stored as an array's element.
- stack is empty: top= 0
- stack is full: top = Max_Element

Stack specification (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);
```



array implementation of stack (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];
  }
```

stack implementation using structure

- Implementation (c): stack is declared as *a structure* with two fields: one for storage, one for keeping track of the topmost position

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

stack implementation using structure

```
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[--

   }
```

Compile file with library

You'got stack.h, stack.c and test.c

You need to insert this line:

```
#include "stack.h"
```

into stack.c and test.c

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

Simple program using stack

- Write a program that convert a positive integer in decimal form to binary form using library stack you have developpped.

- Extend this program to transform from decimal to hexadecimal base.

Implementation using linked list

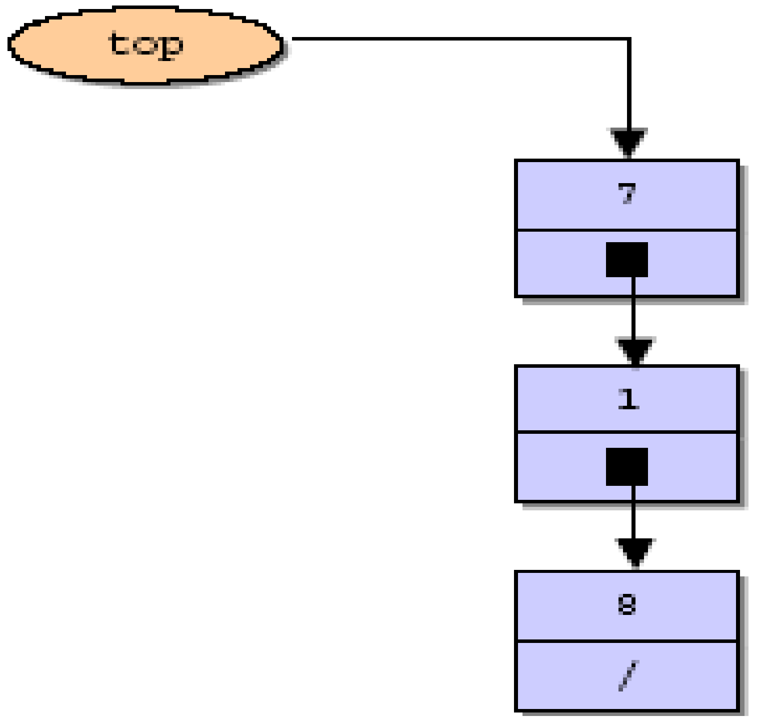
- Implementation of stacks using linked lists are very simple

- The difference between a normal linked list and a stack using a linked list is that some of the linked list operations are not available for stacks

- Being a stack we have only one insert operation called push().
  - In many ways push is the same as insert in the front

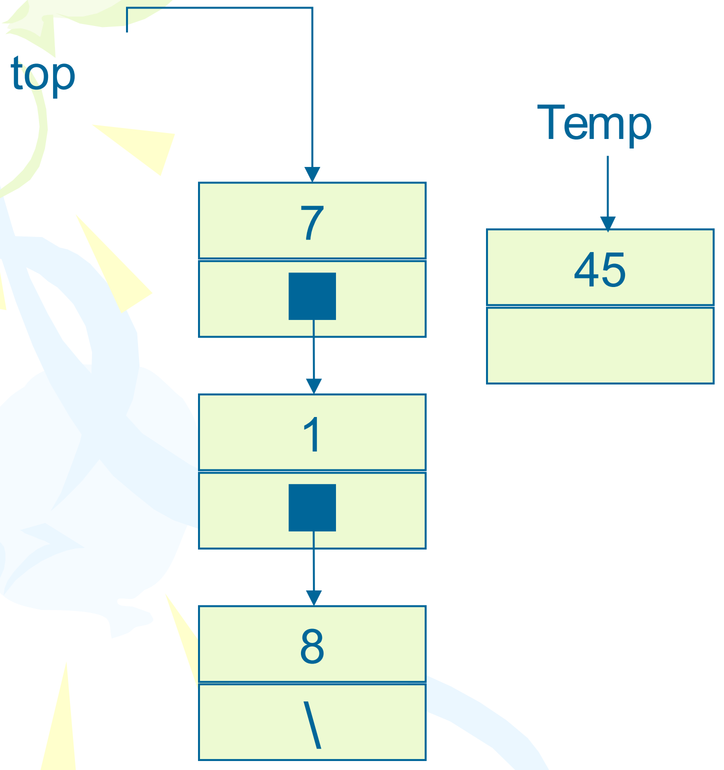
- We have also one delete operation called pop()
  - This operation is the same as the operation delete from the front

Pictorial view of stack



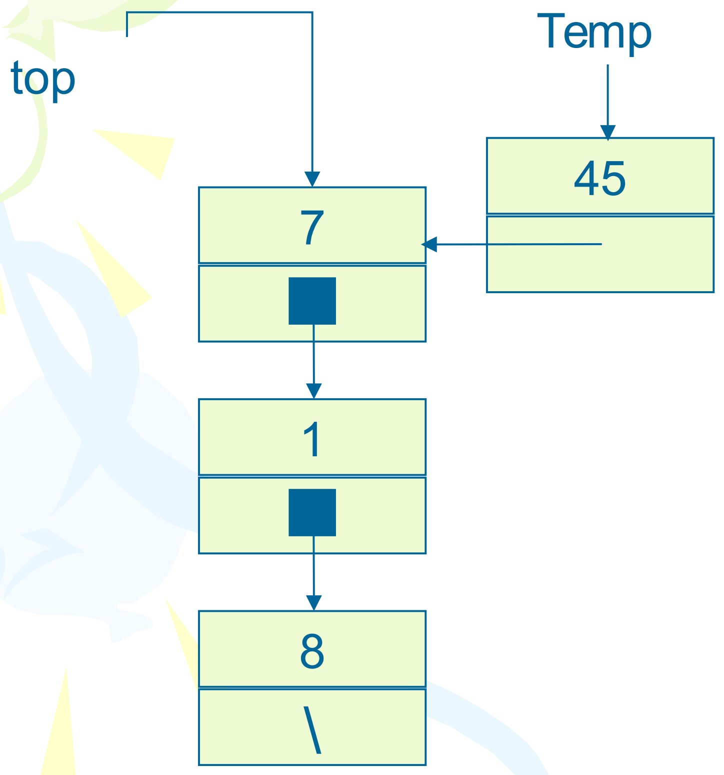
```
struct node {
int data;
struct node *link;
};
```

Push



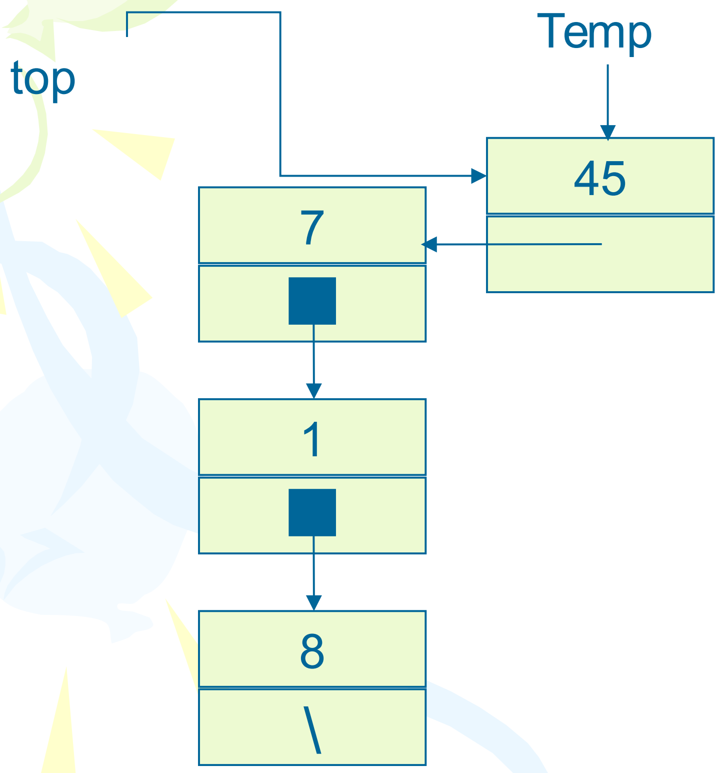
```
struct node *push(struct node *p, int value)
{
   struct node *temp;
   temp=(struct node *)malloc(sizeof(struct
     node));
     if(temp==NULL)  {
       printf("No Memory available Error\n");
       exit(0);
     }
   temp->data = value;
   temp->link = p;
   p = temp;
   return(p);
}
```

Push



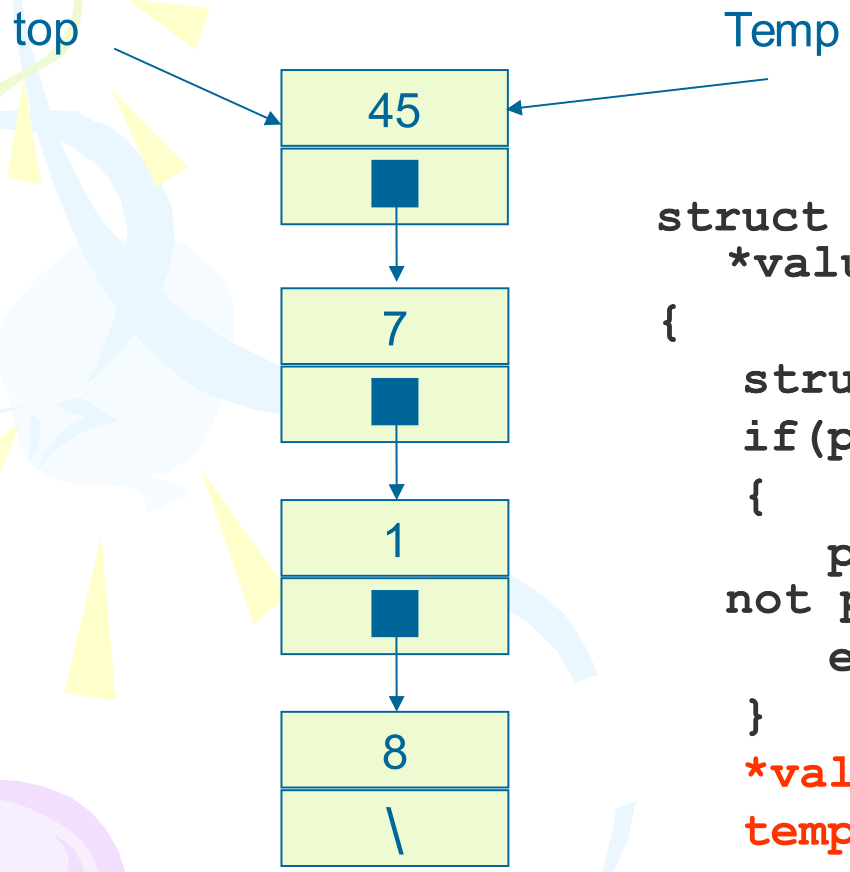
```
struct node *push(struct node *p, int value)
{
   struct node *temp;
   temp=(struct node *)malloc(sizeof(struct
     node));
     if(temp==NULL)  {
       printf("No Memory available Error\n");
       exit(0);
     }
   temp->data = value;
   temp->link = p;
   p = temp;
   return(p);
}
```

Push



```
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);
}
```

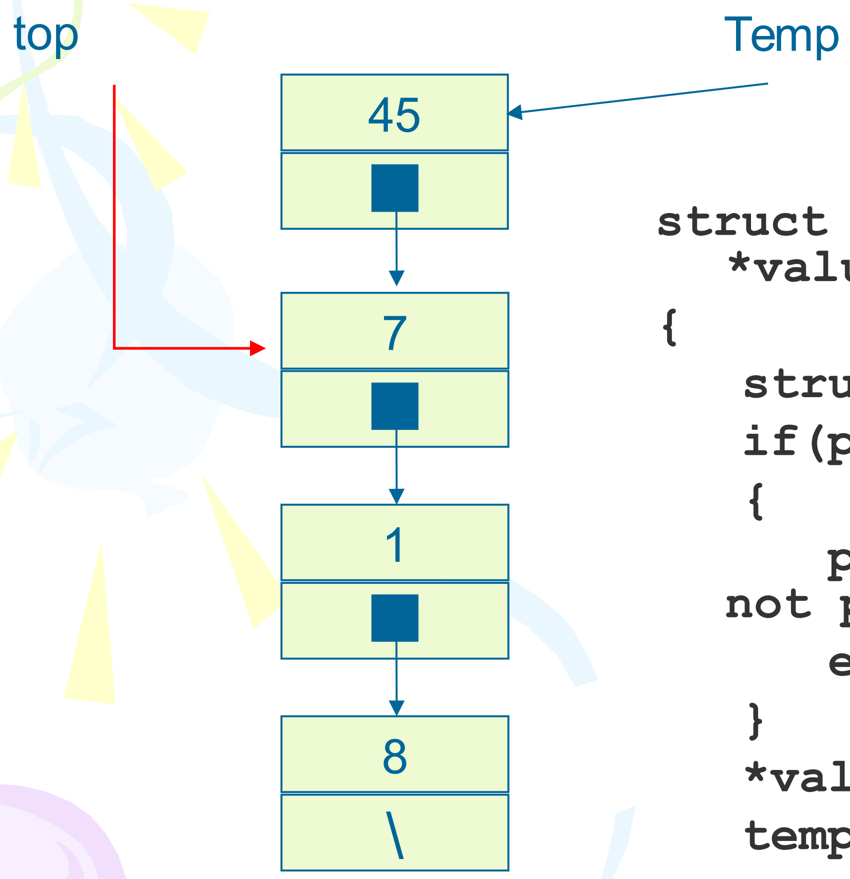
Pop (linked list)



```
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);
}
```

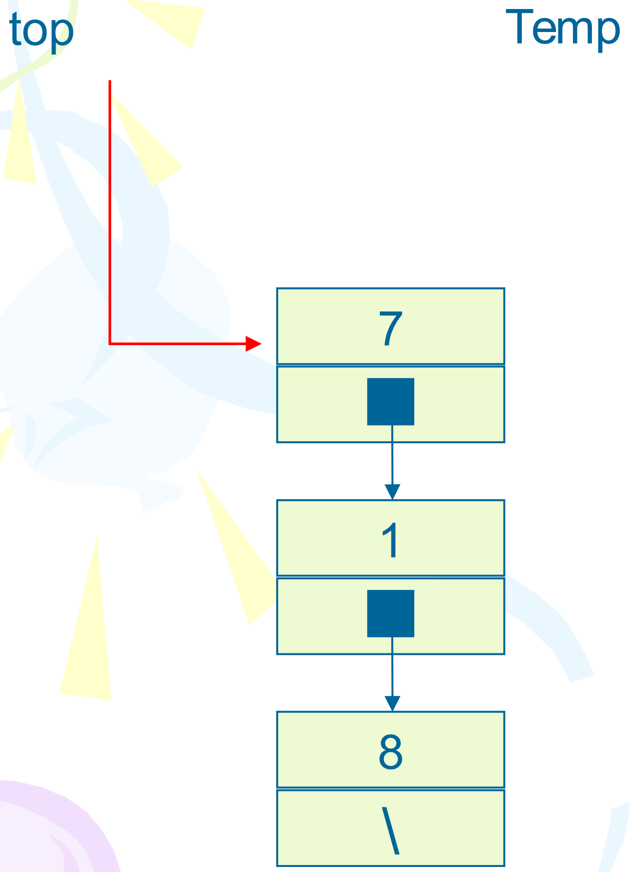
Value at top element need to be save before pop operation

Pop (linked list)



```
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 (linked list)



```
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);
}
```

Attention

- Remeber to implement FreeStack Function in your lib.

Using stack in program

```
# 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);
}
```

Using stack in program

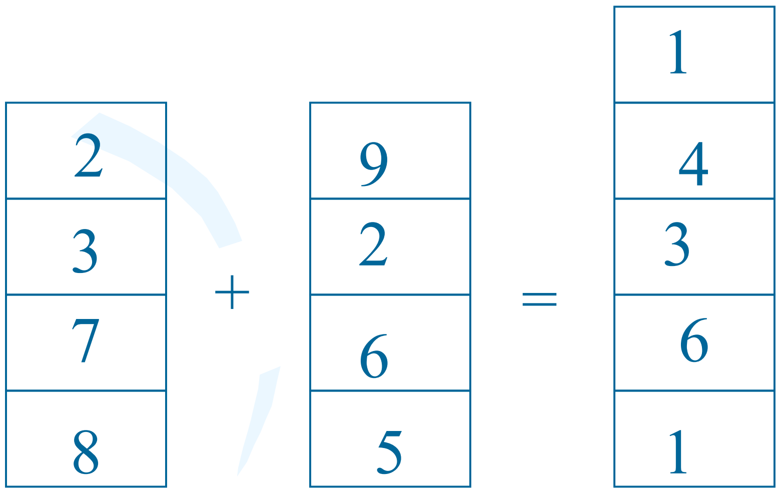
```
printf("Enter 1 to pop an element\n");
    scanf("%d",&n);
    while( n == 1)
    {
        top = pop(top,&value);
      printf("The value poped is %d\n",value);
        printf("Enter 1 to pop an element\n");
        scanf("%d",&n);
    }
    printf("Enter 1 to continue\n");
    scanf("%d",&n);
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

Exercise 4.1

- Test the "stack" type that you've defined in a program that read from user a string, then reverse it.

Homework 1

- Treat these numbers as strings of numerals, store the numbers corresponding to these numerals on two stacks, and then perform addition by popping numbers from the stacks



8732 + 5629 = 14361

Hint

```
Read the numerals of the first number and store
   the numbers corresponding to them on one
   stack;
Read the numerals of the second number and store
   the numbers corresponding to them on another
   stack;
result=0;
while at least one stack is not empty
    pop a number from each non-empty stack and
   add them;
    push the sum (minus 10 if necessary) on the
   result stack;
    store carry in result;
push carry on the result stack if it is not zero;
pop numbers from the result stack and display
   them;
```

Exercise 4.2

- We assume that you make a mobile phone’s address book.
- Declare a structure "Address" that can hold at least "name", "telephone number" and "e-mail address".
- Write a program that copies data of an address book from a file to another file using a stack. First, read data of the address book from the file and push them on a stack. Then pop data from the stack and write them to the file in the order of popped. In other words, data read first should be read out last and data read last should be read out first.

Exercise 4.3

- Write a program that converts an expression in the infix notation to an expression in the reverse polish notation. An expression consists of single-digit positive numbers (from 1 to 9) and four operators (+, -, *, /). Read an expression in the infix notation from the standard input, convert it to the reverse polish notation, and output an expression to the standard output. Refer to the textbook for more details about the Reverse Polish Notation.
- For example,

  3+5*4

  is input, the following will be output.

  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

- Write a program that reads any postfix expression involving multiplication and addition of interger.
- For example
- ./posteval 5 4 + 6 * => 54

STACK.H

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

Homework 2

- Study algorithms for multiplication of large numbers

- Solution: Chunk the larger number into segments of length three (or length-*k*). Apply multiplication as normal and add the results.