LINKED LIST

- Insert as a first node
- Insert as a last node
- Delete first node
- Delete last node
- Insert after a node
- Insert before a node
- Traverse

INSERT AS A FIRST NODE

```
void insertf()
 {
    struct studinfo  *newnode;
    newnode=(struct studinfo *) malloc(sizeof(struct studinfo));
    printf("\nEnter a new  record : marks and name ");
    scanf("%d%s",&newnode->marks,newnode->name);
    newnode->next=NULL;

    if(start==NULL)
                         start = newnode;
              else
      {
               newnode->next = start;
                start = newnode;
      }
}
```

INSERT AS A LAST NODE

```
void insertl()
{
struct studinfo *newnode, *ptr;
newnode=(struct studinfo *) malloc(sizeof(struct studinfo));
printf("\nEnter a new  record : marks and name ");
scanf("%d%s",&newnode->marks,newnode->name);
newnode->next=NULL;
if (start == NULL)
            start =newnode;
else
    {
    ptr =start;
    while (ptr->next != NULL)
                        ptr= ptr->next;
    ptr->next = newnode;
     }
}
```

Delete first node

```
void deletef()
{
   struct studinfo *ptr;
   if (start == NULL)
   {
    printf("\n List is empty");
    return;
   }
   ptr=start;
   start=start->next;
   free(ptr);
}
```

DELETE LAST NODE

```
void deletel()
 {
    struct studinfo *ptr,*prevptr;
    ptr=start;
    if (ptr->next==NULL)
         start = NULL;
    else
    {
    while(ptr->next!=NULL)
     {
              prevptr=ptr;
              ptr=ptr->next;
     }
     prevptr->next =NULL;
     }
     free(ptr);

}
```

INSERT AFTER A NODE

```
void inserta()
{
  int cnt=1, no;
    struct studinfo *ptr,*prevptr, *newnode;
     printf("\nEnter number ...");
     scanf("%d",&no);
     ptr=start;
     while (cnt != no)
     {
     ptr = ptr->next;
     cnt++;
     }
    newnode=(struct studinfo *) malloc(sizeof(struct studinfo));
    printf("\nEnter a new  record : marks and name ");
    scanf("%d%s",&newnode->marks,newnode->name);
    newnode->next =NULL;
     newnode->next = ptr->next;
     ptr->next = newnode;

}
```

INSERT BEFORE A NODE

```
void insertb()
 {

      int cnt=1, no;
      struct studinfo *ptr,*prevptr, *newnode;
       printf("\nEnter number ...");
       scanf("%d",&no);
       ptr=start;
      if(no==1)
       {
                  insertf();
       }
       else
       {
                  while(cnt !=no)
                   {
                            prevptr=ptr;
                            ptr = ptr->next;
                            cnt++;
                   }
```

Continue…

```
newnode=(struct studinfo *) malloc(sizeof(struct
   studinfo));
    printf("\nEnter a new  record : marks and name
   ");
    scanf("%d%s",&newnode->marks,newnode-
   >name);
    newnode->next=ptr;
    prevptr->next=newnode;
    }
}
```

TRAVERSE

```
void traverse()
{
struct studinfo *ptr;
if (start == NULL)
    {
    printf("\n List is empty");
    return;
    }
ptr= start;
while (ptr !=NULL)
{
printf("\nRecord: marks and name %d  %s\n",ptr->marks, ptr->name);
ptr = ptr->next;
}
getch();
}
```

STACK

- Stack using array

a. Push
b. Pop
c. display

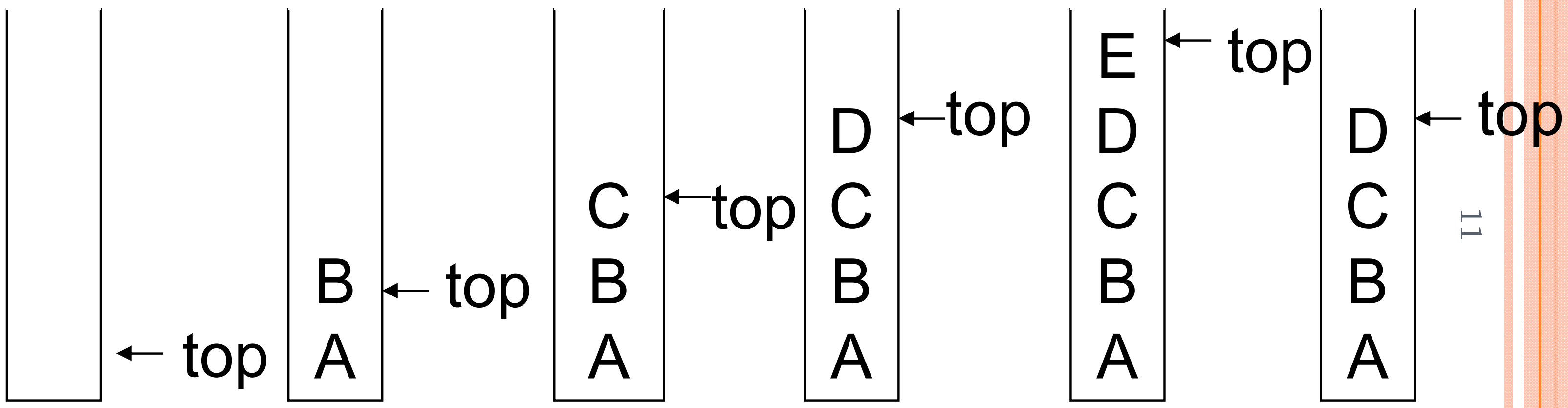
- Stack using linked list

a. Create
b. Push
c. Pop
d. display

- Stack Application

a) Towe of Hanoi

stack: a Last-In-First-Out (LIFO) list



Inserting and deleting elements in a stack

A

PUSH

```
void push()
{
    if(top==9)
    {
        printf("\nStack Full!!!!\n");
        return;
    }

    printf("\nEnter element:\n");
    scanf("%d",&num);

    top=top+1;
    arr[top]=num;
    printf("\n %d pushed,\n",num);
}
```

Pop

```
void pop()
{
    if(top==-1)
    {
        printf("\nStack Empty!!!!\n");
        return;
    }

    printf("\nThe pop value is %d.\n",arr[top]);
    top=top-1;
}
```

Display

```
void display()
{
    int i;

     if(top==-1)
    {
        printf("\nStack Empty!!!!\n");
        return;
    }

    printf("\nThe contents of the stack is:\n");
    printf("\n=============================================\n");
    for(i=top;i>=0;i--)
    {
        printf("%d\t",arr[i]);
    }
}
```

CREATE

```
struct stack{
int items;
struct stack *next;
}*top;
```

PUSH

```
void push()
 {
    struct stack  *newnode;
    newnode=(struct stack *) malloc(sizeof(struct stack));
    printf("Enter a new items: ");
    scanf("%d",&newnode->items);
    newnode->next=NULL;

    if(top==NULL)
        top = newnode;
    else
      {
      newnode->next = top;
      top = newnode;
      }
 }
```

POP

```
void pop()
 {
   struct stack *ptr;
   ptr=top;
   top=top->next;
   free(ptr);
 }
```

DISPLAY

```
void display()
{
struct stack *ptr;
ptr= top;
printf("\n Stack items: ");
while (ptr !=NULL)
{
printf("\n %d \n",ptr->items);
ptr = ptr->next;
}
getch();
}
```

Evaluation of Expressions

X = A / B - C + D * E - A * C

A = 4, B = C = 2, D = E = 3

INTERPRETATION 1:

((4/2)-2)+(3*3)-(4*2)=0 + 8+9=1

INTERPRETATION 2:

(4/(2-2+3))*(3-4)*2=(4/3)*(-1)*2=-2.66666…

HOW TO GENERATE THE MACHINE INSTRUCTIONS CORRESPONDING TO A GIVEN EXPRESSION?

PRECEDENCE RULE + ASSOCIATIVE RULE

Token	Operator	Precedence[1]	Associativity
( ) [ ] -> .	function call array element struct or union member	17	left-to-right
-- ++	increment, decrement[2]	16	left-to-right
-- ++ ! - - + & * sizeof	decrement, increment[3] logical not one’s complement unary minus or plus address or indirection size (in bytes)	15	right-to-left
(type)	type cast	14	right-to-left
* / %	mutiplicative	13	Left-to-right

+ -	binary add or subtract	12	left-to-right
<< >>	shift	11	left-to-right
> >= < <=	relational	10	left-to-right
== !=	equality	9	left-to-right
&	bitwise and	8	left-to-right
^	bitwise exclusive or	7	left-to-right
|	bitwise or	6	left-to-right
&&	logical and	5	left-to-right
⊠	logical or	4	left-to-right

?:	conditional	3	right-to-left
= += -= /= *= %= <<= >>= &= ^= ☒	assignment	2	right-to-left
,	comma	1	left-to-right

1.The precedence column is taken from Harbison and Steele.
2.Postfix form
3.prefix form

Precedence hierarchy for C

user

compiler

Infix	Postfix
2+3*4	234*+
a*b+5	ab*5+
(1+2)*7	12+7*
a*b/c	ab*c/
(a/(b-c+d))*(e-a)*c	abc-d+/ea-*c*
a/b-c+d*e-a*c	ab/c-de*ac*-

Infix and postfix notation

Postfix: no parentheses, no precedence

Token	Stack			Top
	[0]	[1]	[2]	
6	6			0
2	6	2		1
/	6/2			0
3	6/2	3		1
-	6/2-3			0
4	6/2-3	4		1
2	6/2-3	4	2	2
*	6/2-3	4*2		1
+	6/2-3+4*2			0

Postfix evaluation

TOWER OF HANOI

```
void towers(char needle1, char needle2, char  needle3, int n)
{
    if( n <= 0)
            printf("\n Illegal entry ");
    if(n == 1) /*If only one disk, make the move and return */
    {
            printf("\n Move Disk 1 from needle %c to needle %c", needle1, needle2);
            return;
    }
    towers(needle1, needle3, needle2, n-1); /* Move top n -1 disks from A to B,
    using C as auxiliary */
    printf("\n Move Disk %d from needle %c to needle %c",n, needle1, needle2);
/* Move remaining disk from A to C */
    towers(needle3, needle2, needle1, n-1); /* Move  n -1 disks from B to C, using A
    as auxiliary */
}
```

QUEUE

- Queue using array

a. Insert

b. Delete

c. display

Queue using linked list

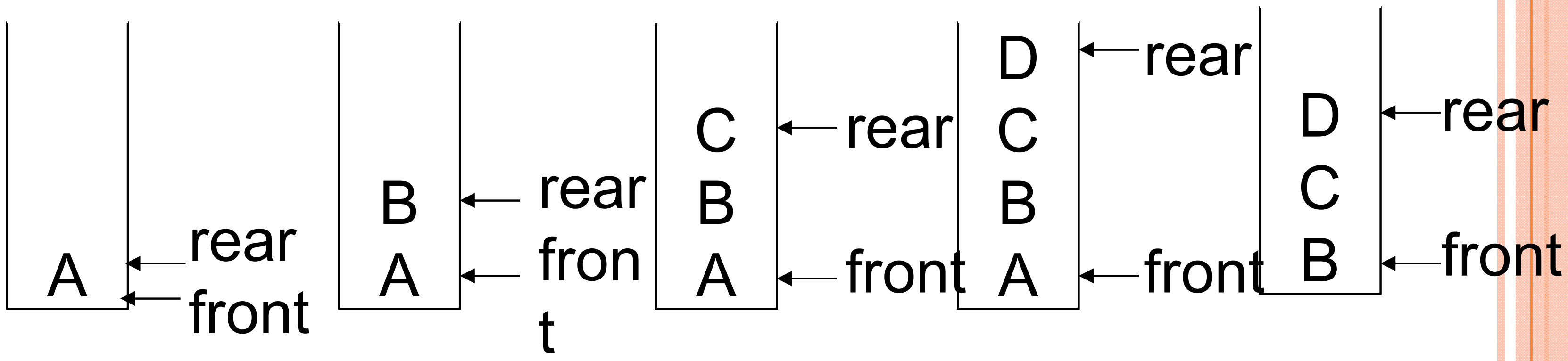
a. Create

b. Insert

c. Delete

d. display

Queue: a First-In-First-Out (FIFO) list



Inserting and deleting elements in a queue

Application: Job scheduling

front	rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
-1	-1					queue is empty
-1	0	J1				Job 1 is added
-1	1	J1	J2			Job 2 is added
-1	2	J1	J2	J3		Job 3 is added
0	2		J2	J3		Job 1 is deleted
1	2			J3		Job 2 is deleted

*Figure 3.5: Insertion and deletion from a sequential queue (p.108)

CREATE

```
struct queue{
   int items[MAXSIZE];
   int rear;
   int front;
}q;
```

INSERT

```
void queueins(struct queue *q ,int x)
{
if (q->rear==MAXSIZE -1)
    {
    printf("Queue full\n");
    return;
     }
else
    {
    if (q->front == -1)
             {
             q->front =0; q->rear = 0;
             }
    else
              q->rear = q->rear+1 ;
    q->items[q->rear] = x;
    }
}
```

Delete

```
int queuedel(struct queue *q)
 {
 int x;
 if (q->front== -1)
     printf(" Queue is empty\n");
 x = q->items[q->front];
 if (q->front == q->rear)
    {
    q->front = -1;
    q->rear =-1;
    }
 else
    q->front = q->front +1 ;
return x;

}
```

DISPLAY

```
void display(struct queue *q)
 {
 int i;
 if (q->front != -1)
  if (q->front <= q->rear)
   for( i=q->front;i<=q->rear;i++)
    printf("%d      ",q->items[i]);
}
```

Create

```
struct queue{
int items;
struct queue *next;
}*rear,*front;
```

INSERT

```
void insertion()
{
struct queue *newnode, *ptr;
newnode=(struct queue *) malloc(sizeof(struct queue));
printf(" Enter items ");
scanf("%d",&newnode->items);
newnode->next=NULL;
if (rear == NULL)
    {
    rear =newnode;
    front = newnode;
    }
else
    {
    rear->next = newnode;
    rear = newnode;
      }
}
```

Delete

```
void deletion()
 {
   struct queue *ptr;
   if (front == NULL)
   {
    printf("\n Queue is empty ");
    rear=NULL;
    return ;
   }
   ptr=front;
   front=front->next;
   free(ptr);
 }
```

DISPLAY

```
void display(struct queue *q)
 {
 int i;
 if (q->front != -1)
  if (q->front <= q->rear)
   for( i=q->front;i<=q->rear;i++)
   printf("%d      ",q->items[i]);
}
```