PAGE :
DATE : / /

int a[8]

RISHI

Memory allocated at the time of execution [Run Time]

(DMA) Dynamic Memory Allocation.

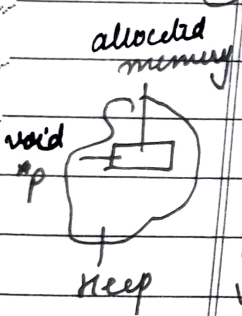
* **Heap** is the segment of memory where dynamic memory allocation takes place.
* memory is allocated without any order

POINTERS play an imp role in DMA

Functions:
- malloc ()
- calloc ()
- realloc ()
- free ()

<stdlib.h>

① Malloc ( ) → allocates a memory block acc. to size, on success returns a pointer pointing to the first byte [else NULL]



Syntax :

(void*) malloc (size_t size);

↑ pointer — size_t [unsigned int] — data type ke size e.g int ke liye 4

why?
void pointer
because Malloc ko
pta hi nhi hai
kisko memory
dena
Bs de rha hai

e.g

int *ptr = (int *) malloc(4)

iska address yeh wale pointer mei store hoga.

```
int size;
int *ptr;            (size of array)
cout<<"enter the no. of values";
cin >> size;
[ ptr = new int[size]; ]
  for ( int i =0; i< size; i++)
    {
      cin >> ptr[i];
    }
```

new
delete

* initialized with some garbage value

(2) Calloc () → multiple blocks of memory
→ needs two argument

Syntax : void * calloc (size_t n, size_t size);

↓ no. of Blocks — ↓ size of Blocks

e.g. int *ptr = (int *) calloc (10, size of (int));

equivalent Malloc ()

int *ptr = int * malloc (10 * size of(int));

✓ Memory allocated is initialized to zero.

Returns NULL when not enough memory.

(3) Realloc () → change the size of the memory block ~~without losing the old data~~.

syntax: void * realloc (void *ptr, size_t newsize);

↓ pointer to the previously allocated memory

e.g. int *ptr = (int *) malloc (size of (int))

yaha hum dobara memory dene → ptr = int * realloc (ptr, 2 * size of (int));

↓ previous wale — ↳ because ek pehle and ek abhi kr rhe.

PAGE :
DATE: / /

④ free() → Frees Release the dynamically allocated memory in heap

Syntax void free(ptr)

e.g

```
{ int *ptr = (int *) malloc(4 * sizeof(int));

  free(ptr);
}

int * input()
{
```

Array

(1) 1D address of A[i] = Base adress + no. of bytes (i − lower limit)

(2) 2D Row Major
address of A[i][j] = Base adress + no. of bytes [no. of columns (i − LBR) + (j − LBC)]

(3) Column Major
address of A[i][j] = B + No. of Bytes [(i − LBR) + No. of Rows (j − LBC)]

(4) A [LBR .. UBR, LBC, ... UBC]

$$\begin{cases} M = (UBR - LBR) + 1 \\ N = (UBC - LBC) + 1 \end{cases}$$

operations in Array (traverse, insert, delete, search.

```
int traverse(int arr[], int n)
{ for(int i=0; i<n; i++)
  { cout<<arr[i];
  }
}
```

```
int search(int arr[], int n, int num)
{ for(int i=0; i<n; i++)
  { if(a[i]==num)
      return i;
  }
```

```
void insert(int arr[], int num, int pos)
{ for(int i=n-1; i>=pos; i--)
  { a[i+1]=a[i]
    a[pos]=num
    n++;
  }
```

```
int deleteele(int arr[], int pos)
{ int temp=a[pos];
  for(int i=pos; i<n-2; i++)
  { a[i]=a[i+1]
    n--;
  } return temp;
}
```

Single Link list.

PAGE:
DATE: / /

```
#include<iostream>
using namespace std;
struct ll {
  int item;
  struct ll* nent;
  };
typedef struct ll node;
  node *head = NULL;
  int flag = 0;

* void create()
{
  if(head==NULL)
  head = new node;
  cout<<"Enter the Element:"
  cin >> head -> item;
  head -> nent = NULL;
  else
  { node *p = head; q*;
  while (p -> nent != NULL)
    { p = p -> nent; }

  q = new node;
  cout<<"enter the element";
  cin >> q -> item;
    q -> nent = NULL;
    p -> nent = q;

* void display()
    { node *p = head;
      if (p == NULL)
{ cout<<" list is empty";
  return;
}
while (p != NULL)
{
  cout<<" "<<p -> item;
    p = p -> nent;
}
}

* void insert_beg()
{
    node *p;
    p = new node;
    cin >> p -> item;
    p -> nent = head
      head = p;
}

* void insert_mid()
{  int num;
cout<<"Enter the item after
which number you want
to add the node";
cin >> num;
node *p = head;
while (p -> item != num)
{
    p = p -> nent;
}
node *q;
q = new node;
```

PAGE :
DATE: / /

```
cin>>q->item;
q->next=p->next
p->next=q;
}
```

```
* void del_beg()
{
  int ele;
  if(head==NULL)
  { cout<<"No element to be
      deleted";
    return;
  }
  node *p=head;
  ele=p->item;
  head=p->next;
  delete(p);
  cout<<"deleted element"<<ele;
}
```

```
* void del_end()
  {
    if(head==NULL)
    {
      cout<<"No element to be
          deleted";
      return;
    }
  node *p=head;
  if(p->next==NULL)
  cout<<"deleted element"<<p->item;
    delete(p);
  head=NULL;
```

```
else
{
while(p->next->next!=NULL)
  {
    p=p->next;
  }
int ele;
ele=p->next->item;
 delete(p->next);
 p->next=NULL;
cout<<"deleted elem"<<ele;
}
}
```

```
* void del_mid()
  {
    int num;
    if(head==NULL)
  {
  cout<<"No element to
    be deleted";
    return;
  }
  cout<<"Enter the ele
  to be deleted";
  cin>>num;
  node* p=head, *q;
  while(p->next->item!
        =num)
  {
    p=p->next;
  }
```

```
q=p->next->next;
delete(p->next);

p->next=q;
}
void search()
{
  int ele;
  node *p;
cout << "enter the element";
cin >> ele;
for(p=head; p!=NULL; p=p->next)
{
  if(p->item==ele)
  {
    cout << "found";
    flag=1;
    break; }
  if(flag==0)
  {
    cout << "not found";
  }
void reverse()
{
  node *p, *q;
  p=NULL;
  q=NULL;
while(head!=NULL)
  { q=head->next;
    head->next=p;
    p=head;
  } head=q;
}
```

```
void delete_whole()
{
  node *p=head;
while(p!=NULL)
  { p=p->next;
    delete(head);
    head=p;
  }
}
```

Double linked list

```
#include<iostream>
using namespace std;
struct ll
{
   int data;
  struct ll* prev;
  struct ll* next;
   };
typedef struct ll node;
  node * head = NULL;
void create()
{
  if(head == NULL)
  {
   head = new node;
   cin >> head->data;
   head->next = NULL;
   head->prev = NULL;
  }
  else
  {
   node* p, *q;
    p = head;
while (p->next != NULL)
  {
     p = p->next
  }
  q = new node;
cin >> q->data;
q->next = NULL;
q->prev = p;
p->next = q;
  }
}
```

```
void insert_beg()
{
  node* p;
  p = new node;
  p->prev = NULL;
  p->next = head;
  head->prev = p;
   head = p;
}

void insert_mid()
{
  int num;
cout<<"enter the no. after which you
want to add an element:";
cin >> num;
  node* p;
   p = head;
while (p->data != num)
  {
     p = p->next;
  }
  node* q;
  q = new node;
  cin >> q->data;
p->next->prev = q;
q->prev = p;
q->next = p->next;
p->next = q;
q->prev = p
}
```

Inf
sequ

```
void del_beg()
{
  int ele;
  if (head==NULL)
  {
    cout<<"no element to be deleted";
    return;
  }
  node * p;
  p = head
  ele = p→data;
  head = p→next,
  p→next→prev = NULL;
  delete(p);
  cout<<"deleted element"<<ele;
}
void del_end()
{
  int ele;
  if (head==NULL)
  { cout<<"No element to be
      deleted";
    return;
  }
  node* p=head;
  while(p→next!=NULL)
  { if (p→next==NULL)
    {
  cout<<"deleted element"<<p→data;
    delete(p);
    head=NULL;
  }
```

```
else
{
  while(p→next→next!=NULL)
  {
    p = p→next;
  }
  int ele;
  ele = p→next→data;
  delete(p→next);
  cout<<"deleted ele"<<ele;
}
}
void del_mid()
{
  int num;
  if (head==NULL)
  {
    cout<<"No element to be
        deleted";
    return;
  }
  cout<<"enter the no. to be
      deleted";
  cin>>num;
  node* p=head;
  while (p→next→data!=nu
  {
    p=p→next;
  }
  node* q;
  q=p→next→next;
  delete(p→next);
  p→next=q
  q→prev=p;
}
```

PAGE :
DATE: / /

```
void display()
{
  node *p = head;
  if (p == NULL)
  { cout << " list is empty";
  }
  while (p != NULL)
  {
    cout << " " << p->data << endl;
      p = p->next;
  }
}
```