Sumedh

• Topics

1. Structures
2. DMA
3. Matrix mul
4. DSA
5. Notations

• Structures

• Basic Structures

Program : To take Student Info

```
#include <stdio.h>

struct Student {          // Structure name //
    char name [40] ;
    int roll ;
    float cg ;
} stud ;          // Structure variable declration //
                      └ (To call variables, just
                                 like object)

int main () {

printf (" Enter Student Name : ");
scanf (" %s" , stud.name );

print ( " Enter Roll no. : ");                    // input //
scanf ( " %d " , stud.roll );

print ( " Enter Cg : ");
scanf ( " %f " , stud. cg) ;

print ( " Student details ");
printf ( stud.name );
printf ( stud. roll );          // display //
printf ( stud. cg);

}
```

→ '.' or '->'

Output :

```
Enter Student Name : Sumedh
Enter roll no : 691
Enter cg : 9.6

Student details
Sumedh 691 9.6
```

- Structure with arrays

Prog : To n Student info

```
# include <stdio.h>

Struct Student {
     Char name [40];
     int roll ;
     float cg
} stud [n] ;

int main () {
   int n;
   printf (" Enter no. of students : ');
   scanf (" %d ", &n);

  for (int i=1 ; i <=n ; i++) {
     printf (" Enter Details for Student : " ,i);
     print f (" Enter name);
     scanf (" % s" , stud [i]. name) ;
     printf (" Enter Roll no) ;
     scanf (" % d " ,&stud [i] . roll);
     printf (" Enter cg : ");
     scanf ( " %f " , &stud [i]. cg );
 }

 printf ("\n\t Name \t RollNo \t Marks\t\n");
  for ( int i=0 ; i<n ; i++) {
      printf (" \t %s \t %d \t %.2f \t \n" , stud [i]. name , stud[i].roll,
              stud [i] . cg );
  }
  return 0;
}
```

Output :

```
Enter no. of Students :2
Enter details for Student : 1
Enter name : Sumedh
Enter Roll no : 691
Enter cg :9.6
Enter details for Student : 2
Enter name : Aditya
Enter Roll no : 700
Enter cg : 10

 Name      Rollno     Cg
Sumedh      691      9.6
Aditya      700      10
```

- Nested Structures

Syntax :-

```
struct  one  {
        int a;
        int b;
 };
struct  two {
      int c;
      int d;
     Struct  one  obj
};

int  main () {
    struct  two  s;

     s.c;        // to  access  struct 2//
     s.d ;

     s.obj.a ;    // to  access  struct  one//
     s.obj.b ;

}
```

- Structure using pointers

```
  # include <stdio.h>
  struct student  {
       Char  name [40];
       int  roll;
       float  cg;
  };
 int  main () {

   struct  Student *studPtr , stud 1;
   studPtr = &stud 1;
   printf (" Enter Nam : ");
   scanf (" %s ", studPtr -> name);
       // Similary  roll & cg //
  printf (" Name : %s ", StudPtr->name);
return 0;
}
```

- ## Dyanamic Memory Allocation

1. malloc ()

- Memory allocation , gives a block of memory
- malloc returns void pointer i.e all data types
- Then typecast void * into resp data type ptr

```
ptr = (int*) malloc (n * sizeof (int))
```

Program :-

```
#include <stdio.h>
# include < stdlib.h >

 int main () {
     int i , n , *ptr;
     printf (" enter no of vals");
     scanf ("%d" , &n);

     if (ptr==NULL) {
       printf (" Mmry not allocated ");
  }
     ptr = (int *) malloc (n* sizeof (int));

     printf (" Enter vals :");
     for (i= 0 ; i<n; i++){
         scanf ( "%d", (ptr+i) );
     }

     for (i= 0 ; i<n; i++){
         printf ("%d" , *(ptr+i));
     }

  free (ptr);

  return 0;

 }
```

2. Calloc ()

- Contiguous alloc
- used to dyanamicly alloc block of mmry & each block is of same size
- Initially prints '0' value while malloc gives garbage values

Syntax : ptr = (int*) calloc (n* sizeof (int));

3. realloc ()

- reallocation i.e size inc / dec

Syntax : void * realloc (void* ptr , size);

└ ptr name

eg:

```
        point = (int *) malloc ( 5 * sizeof (int));
int * point1 = (int * ) realloc ( point , 7 * sizeof (int) );
```

↳ i.e size inc by2

4. free ()

- deallocate the allocated mmry

- Matrix Multiplication

1. Basic Multiplication

Program :-

```
#include < stdio.h>

int main () {
      int  A[2][2] , B[2][2] , C[2][2];
      int  i , j , k ;

printf (" Enter elements of matrix A : ");
      for (i=0 ; i<2 ; i++) {
          for (j=0 ; j< 2 ; j++) {
               scanf (" %d", &A[i][j] );
        }
    }

 printf (" Enter elements of matrix B : ");
       for (i=0 ; i<2 ; i++) {
           for (j=0 ; j< 2 ; j++) {
                scanf (" %d", &B[i][j] );
         }
     }

    for (i=0 ; i<2 ; i++) {
        for (j=0 ; j< 2 ; j++) {
                c [i] [j]=0
         }
  }

     for (i=0 ; i<2 ; i++) {
         for (j=0 ; j< 2 ; j++) {
             for (k=0 ; k < 2 ; k++) {

                C [i] [j] += A[i] [j] * B[i] [j] ;
      }
  }
}
```

```
  printf (" Result  c: ");
  for (i=0 ; i<2 ; i++) {
     for (j=0 ; j< 2 ; j++) {
        printf ("%d" , c[i][j]);
    }
    printf(" \n") ;
}

  return 0;
}
```

2. Using dyanamic mmry

```
#include <stdio.h>
# include <stdlib.h>

int main () {
   int **a , **b , **c ;
   int ar , ac , br, bc;
   int i, j, k;

  again :
  printf (" Enter rows & cols of m1: ");
  scanf ("%d %d ", &ar, &ac);

  printf (" Enter rows & cols of m2: ");
  scanf ("%d %d ", &br, &bc);

   if ( ac != br) {
      printf (" cannot multiply mats ");
      goto again ;
  }

  a = (int **) malloc ( ar * sizeof (int*) );            // alloc mmry
  for (i=0 ; i<ar ; i++ ) {                                    to m1 //
       a[i] = (int *) malloc (ac * sizeof (int*) );
  }

 b = (int **) malloc ( br * sizeof (int*) );
 for (i=0 ; i<br ; i++ ) {                                  // to m2 //
     b[i] = (int *) malloc (bc * sizeof (int*) );
 }
```

```
c = (int **) malloc ( ar * sizeof (int*) );                    // to res
for (i=0 ; i<ar; i++) {                                             m //
    c[i] = (int *) malloc (bc * sizeof (int*) );
}

 printf (" Enter m1 elem (%d x %d) : , ar,ac);
    for (i=0;i<ar ; i++){                                          // input m1 //
        for ( j=0 ; j < ac ;j++) {
            scanf ("%d " , & a[i][j] );
      }
   }

printf (" Enter m2 elem (%d x %d) : , br, bc );                    // m2 //
   for (i=0;i< br ; i++){
       for ( j=0 ; j < bc ;j++) {
           scanf ("%d " , & b [i][j] );
     }
  }

   for (i=0;i< ar ; i++){                                 // initialise res m //
       for ( j=0 ; j < bc ;j++) {
           c[i][j] =0 ;
     }
  }

  for (i=0;i< ar ; i++){
      for ( j=0 ; j < bc ;j++) {                              // Mul //
         for ( k=0 ; k < ac ; k++) {

         c [i] [j] += a[i][j] * b[i][j] ;
     }
  }
}

  printf (" Product : ");
     for ( i= 0 ; i<ar ; i++) {                             // print res //
         for (j=0 ; j < ac ; j++ ) {

              printf ( "%d " , c[i] [j] );
      }
     printf ("\n");
  }
```

```
for ( i=0 ; i <ar ; i++) {
        free(a[i]);
        free(c [i] );                              // free mmry//
 }
 for (i=0 ;  i <br ; i++){
        free ( b [i] );
 }
 free (a);
 free (b);
 free (c);

 return 0;
}
```

- DSA

1. Traversing

Program :-

```
 # include <stdio.h>
# include <stdlib>

 void traverse (int arr [] , int size) {
        printf (" Traversing");
          for (int i=0;  i < size ; i++){
               printf ("%d" , arr [i]);
     }
       printf ("\n");
 }
int main () {
   int arr [] = { 1, 2, 3 ,4}
   int size =  sizeof (arr) / sizeof (arr [0]);

   traverse (arr , size);
   return 0;
 }
```

2. Searching

```
#include <stdio.h>

int search ( int arr [] , int size , int pt ){
          for  (int i=0 ;  i < size ;  i++) {
              if (arr [i] == pt ){
                  return i  ;
              }
            else {
              return -1  }
    }
    int main () {
      int arr [] = { 10, 20, 30, 40 }
      int Size  =  sizeof (arr) / sizeof (arr [0]);
      int pt = 30
      int result  =  search ( arr , size, Pt);

   if ( result!= -1) {
      printf( " Element %d at index %d ", pt , result );
  }
  else  {
     printf  (" Element %d not found "  ,target);
  }
 return 0;
}
```

3. Insertion

```
#include <stdio.h>

void insert ( arr [] , int *size,  int elem,  int pos) {
        for ( int i= * size ; i>pos ; i--){
            arr [i] = arr [i-1];
        }
      arr [pos] = elem ;
      (* Size ) ++ ;
  }
int main () {
    int arr [6] = { 10, 20, 30, 40, 50}
insert ( arr, 5 , 25, 2 );
printf (" Array : ");
   for (int i=0 ; i< size; i++){
        printf ("%d" , arr [i]);
    }
   return 0;
 }
```

4. Update

```
#include < stdio.h>

void upd (int arr []  , int indx , int val){
          arr [indx ]= val;
 }

int main () {
  int arr [] = {10 ,20, 30, 40}
  update (arr, 2, 100);

printf (" Array " );
   for (int i=0 ; i< 4; i++){
         printf ("%d", arr[i] );
    }
   return 0;
}
```

6. Deleting

```
#include < stdio. h >

void del (int arr []  ,int *size  ,int pos ){
       for  (int i= pos;  i < *size - 1 ; i++ ){

             arr [i]= arr[i+1];
      }
     (*size)-- ;
 }
int main () {
   int arr  [] = { 10, 20 , 30, 40, 50}
  del ( arr , 5 , 2);
  printf
 retur 0 ;
 }
```

● Mathematical Notation & time Complexity

1. Big (O)

- Worst Case
- lowest upper bound

- $f(n) \leq c.gn$

  where $f(n) = O\ g(n)$

  $c > 0$

  $n \geq 0$

2. Big omega (Ω)

- Best Case
- Greatest lower bound

- $f(n) \geq c.g(n)$

3. Theta (θ)

- avg case
- exact time

- $c_1 g(n) \leq f(n) \leq c_2 .g(n)$

eg $2n^2 + n$

→ i) lower bound (big o)

$2n^2 + n \leq c\,g(n)$

$2n^2 + n \leq c\,(n^2)$

$2n^2 + n \leq 3n^2$

$n \leq n^2$

Ltrue

ii) upper bound (big Ω)

$2n^2 + n \geq c.g(n)$

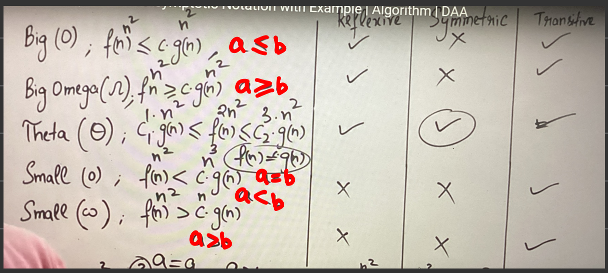
$2n^2 + n \geq c.n^2$

$2n^2 + n \geq 2n^2$

iii) Theta (Θ)

$$c_1\, g(n) \le 2n^2 + n \le c_2\, g(n)$$
$$c_1\, n^2 \le 2n^2 + n \le c_2\, n^2$$
$$2n^2 \le 2n^2 + n \le 3n^2$$

- Comparing



Notation with Example | Algorithm | DAA

	Reflexive	Symmetric	Transitive
Big (O) ; $\overset{n^2}{f(n)} \le c \cdot \overset{n^2}{g(n)}$ a≤b	✓	✗	✓
Big Omega (Ω); $\overset{n^2}{fn} \ge c \cdot \overset{n^2}{g(n)}$ a≥b	✓	✗	✓
Theta (Θ) ; $c_1 \cdot \overset{1 \cdot n^2}{g(n)} \le \overset{2n^2}{f(n)} \le \overset{3 \cdot n^2}{c_2 \cdot g(n)}$	✓	✓	✓
Small (o) ; $\overset{n^2}{f(n)} < c \cdot \overset{n^3}{g(n)}$ $f(n) \ne g(n)$ a=b a<b	✗	✗	✓
Small (ω) ; $\overset{n^2}{f(n)} > c \cdot \overset{n}{g(n)}$ a>b	✗	✗	✓

- Comparing time complexities

$$c < \log(\log n) < \log n < n^{1/2} < n < n.\log n < n^2 < n^3 < n^k < 2^n$$
$$< n^n < 2^{2^n}$$