Bharath. S

Arrays.

I. Declaration of Arrays

Arrays ⇒ An array is a collection of elements of the same type that are reference by the common name.

Syntax: Data type Array name[size]; // Declaration of the array.

Ex: Int a[10].

Above Example is of 1-D array. 1-D array means 1 row to multiple columns.

Ex
Char c[10];
Float b[10];

1, 2, 3, 4, 5 ✓

10, 20, 0, -1 30 ✓

10, 20, 1.2, 2.2 ✗ // Datatypes are not same.

Array is also called Collection of homogeneous Elements in which are stored in Contigeous memory Location.

Points to Remember.

→ Int a[ ] : This would result with error. Always. Size should be specified.

→ Size should be of Integer data type. (Positive)

int a[5] ✗ int a[]; ✗.
int a[5] ✓ int a[11/2] ✗
int a[2+2] ✓ int a[n] ✗ // we can use macros. before main() write #define n 5;
int a[7*2] ✓ Now int a[n]; ✓

Shavath. S

Arrays.

I. Declaration of Arrays

Arrays ⇒ An array is a collection of elements of the same type that are reference by the common name.

Syntax: Data type Array name[size]; // Declaration of the array.

Ex: Int a[10].

Above Example is of 1-D array. 1-D array means 1 row to multiple columns.

Ex
Char c[10];
Float b[10];

1, 2, 3, 4, 5 ✓

10, 20, 0, -1 30 ✓

10, 20, 1.2, 2.2 ✗ // Datatypes are not same.

Array is also called Collection of homogeneous Elements in which are stored in contigeous memory Location.

Points to Remember.

→ Int a[ ] : This would result with error. Always. Size should be specified.

→ Size should be of Integer data type. (Positive)

int a[5] ✗
int a[5] ✓
int a[2+2] ✓
int a[7*2] ✓

Int a[]; ✗.
int a[1/2] ✗
int a[n] ✗ // we can use macros. before main() write #define n 5;
Now int a[n]; ✓

II Array Initialization. ⇒ Arrays may be initialised at time of declaration.

Arrays can be initialised in two ways.

1) Compile time.
2) Run time.

a) Compile time :- At the time of declaration itself. We specify data elements

OR

Specifying data items at the time of declaration is known as Compile-time initialisation.

Ex : ✓ int a[5] = {0, -1, 2, 3, 5} //Initialisation of array at the Compile time.

✓ int a[ ] = {0, 1, 2, 3, -1, 5}
Automatically size is calculated.

✓ int a[5] = {0, 1, 2};
We have left with space for 2 data items by default its filled with zero in memory allocation.

✓ int a[5];
Now array will be filled with garbage value.

int a[5] = {1, 2, 3, 4, 5, 6, 7}
Elements exceeds array size. This results with Error as we can store only 5 numbers.

✓ int a[5];

a[0] = 1
a[1] = 2
a[2] = 3
a[4] = 5
a[5] = 7

// Individual Items can be Initialised like this. This is not Preferable, ofcourse its correct.

✓ int a[5] = {0} // All memory locations would be filled with zero.

✗ int a[5] = { } // This gives error. Always need to specify size atleast one data item should not leave blank.

✓ Char b[ ] = { 's', 'k', I, T }

Here in Compile time we have fixed the size of the array. we can't change it. also we have fixed Elements & cannot be changed at run time.

To avoid above disadvantage to ask the users to Enter the data. This is knowas <u>run-time Initialisation</u>

b)

<u>At run-time</u> : Initialising array make use of Iterative Statements with Scanf().

```
int a[5]; // Ask user to enter data.
    Printf("Enter elements for array");
    for(i=0; i<5; i++)
    {  Scanf("%d", &a[i]);
    }
```

to print we can use print data.

Shaiath.S Arrays

Note: when size of array is Large, its always better to go with Runtime Initialisation.

Ex:
```
Int a[100];
For (i=0; i<100; i++)
{
    if (i<30)
      a[i] = 1;
    if (i>30 && i<=100) //OR simply use else.
      a[i] = 0;.
}
```

iii]

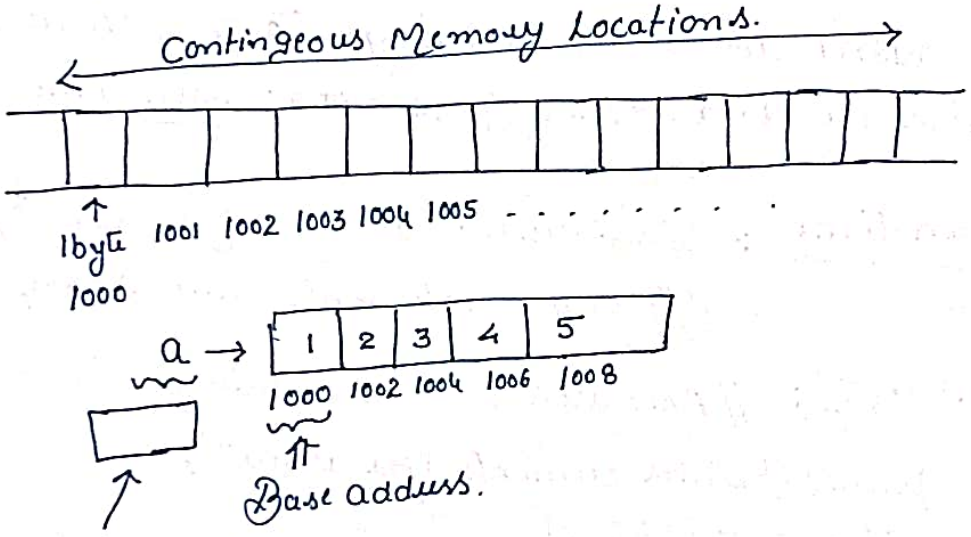
Memory Representation & accessing of arrays

int a[5] = {1, 2, 3, 4, 5}.

1 Integer takes 2-byte. Some m/c takes 4-bytes.

∴ 2 * 5 Elements ⇒ 10 bytes.

Note: always memory is allocated in contigeous memory Location.



Any variable that stores address of other variable is known as pointer.

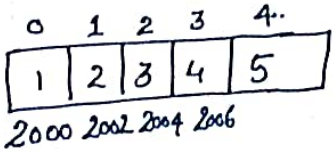
Hence array name is Pointer variable or Internal Pointer.

ie array name is Pointer variable.

Accessing Elements of array (1D)

Syntax ⇒ arrayname[index]

Ex: a[0] = 1
a[1] = 2
a[2] = 3.



Now Question is how address is Calculated.

Formula ⇒ Base address + Index * Size of int

Ex: To access 4th Element ⇒ 2000 + 3 * 2.
⇒ 2000 + 6.
⇒ 2006

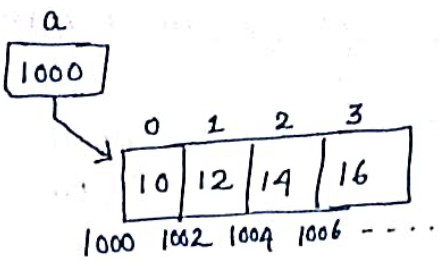
Points to remember.

1) Array is a collection of of more than one data item of same type.
2) All data items are stored in contigeous memory Location.
3) No of items array holds is size of array.
4) Once size has declared, cannot be changed at run time [fixed size]
5) Index starts from zero.
6) Known as derived data type.
7) Accessing an element is faster in arrays (Make use of Index)
8) Allow to store data in multidimensional form.
9) Inserting & deleting elements from array is costly.
10) No bound checking in C.

IV Operations on arrays.

a) Inserting an Element.

Ex:
```
void main()
{
  int a[5], i;
  for(i=0; i<5; i++)
  { scanf("%d", &a[i]);
  }
}
```



b) Printing an Element.

Ex:
```
void main()
{
  int a[5], i;
  for(i=0; i<5; i++)
  { printf("%d", a[i]);
  }
}
```

a[0]

Base add^{n} + i * Size of int.

a[2] ⇒ 1000 + 2 * 2.
1004

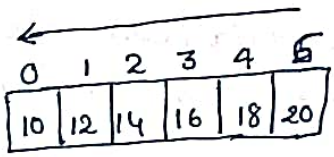
c) Traversing an array

Accessing arrays can be done in 3 ways.

→ From right to Left. (as shown above).

→ From Left to right

Ex:
```
for(i=5; i>=0; i--)
{ printf("%d", a[i]);
}
```



if i = -1 then stop traversing.

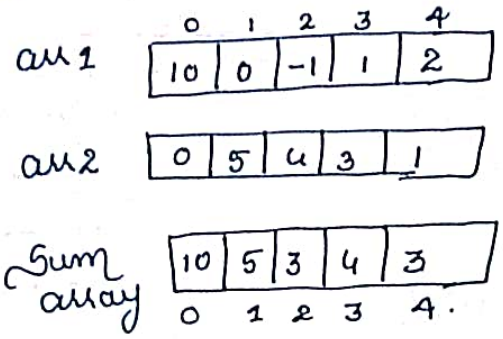
Ex: To calculate sum & average of marks.

```
#include <stdio.h>
void main()
{
    int marks[5], i;
    float sum=0, avg;
    for(i=0; i<5; i++)
    {
        scanf("%d", &marks[i]);
    }
    for(i=0; i<5; i++)
    {
        sum = sum + marks[i];
    }
    avg = sum/5;
    printf("sum = %d", sum);
    printf("\n average = %f", avg);
}
```

Ex: To read array of 10 Integers & count total number of even & odd elements.

```
void main()
{
    int a[10], even, odd=0;
    for(i=0; i<10; i++)
    {
        scanf("%d", &a[i]);
    }
    for(i=0; i<10; i++)
    {
        if(a[i]%2==0)
            even = even++;
        else
            odd = odd++;
    }
    printf("Even elements are
            %d", even);
    printf("Odd elements are
            %d", odd);
}
```

Prog to read two arrays of size 5 & store sum of these array in third array.

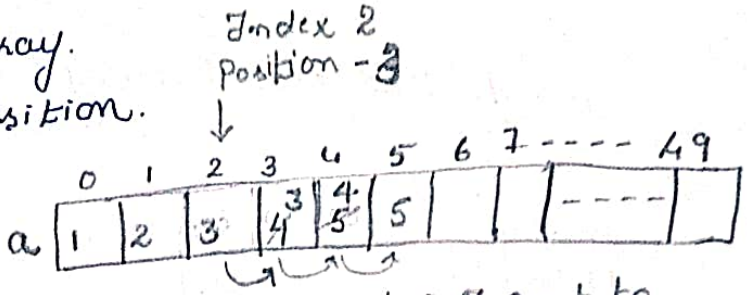



```
#include<stdio.h>
void main()
{
    int arr1[5], arr2[5], sumarr[5], i;
    for(i=0; i<5; i++)
        scanf("%d", &arr1[i]);
    for(i=0; i<5; i++)
        scanf("%d", &arr2[i]);
    for(i=0; i<5; i++)
    { sumarr[i] =
        arr1[i] + arr2[i];
      Printf("%d", sumarr[i]);
          OR
      Printf("Sum array element at index %d is = %d\n",
                                  i, sumarr[i]);
    }
}
```

Arrays Shaikh

IV] Inserting Element in an array.

a] Inserting at specific position.

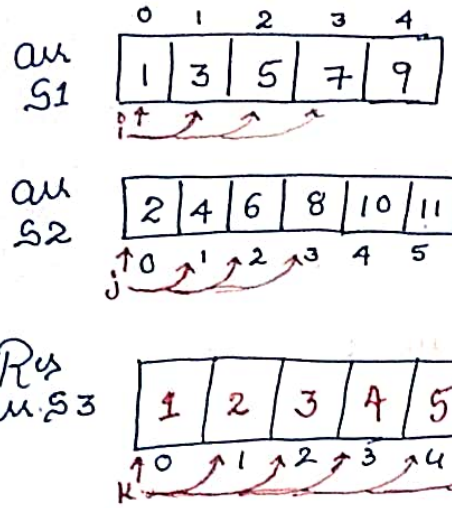


Now I want to Insert an Element at position 2 then we need to shift array Elements towards right.

```
void main()
{
    Int a[50], size, i, num, pos;
    Printf("Enter size of array");
    Scanf("%d", &size);
    Printf("Enter elements of array");
    for(i=0; i<size; i++)
    {
        Scanf("%d", &a[i]);
    }
    Printf("Enter data U want to Insert");
    Scanf("%d", &num);
    Printf("Enter position");
    Scanf("%d", &pos);
    for(i = size-1; i>=pos-1; i--)
    {
        a[i+1] = a[i];
    }
    a[pos-1] = num;
    Size++;
```

```
if(pos<0 || pos>size+1)
{ Printf("Invalid position");
}
```

Merge two sorted arrays.



1) a[0] < a2[0]
1 < 2.
True → Move 1 to arr3.
i++, k++.

2) a[1] < a2[0]
3 < 2
False → move 2 to arr3.

3) a[1] < a2[1]
3 < 4.
True → Move 3 to arr3.

4) a[2] < a2[1], 5 < 4
False → Move 4 to arr3

```
while(i < S1 && j < S2)
{
    if[arr1[i] < arr2[j]).
    {
        res[k] = arr1[i];
        i++;
        k++;
    }
    else.
    {
        res[k] = arr2[j];
        j++;
        k++;
    }
}
while( i < S1)
    res[k++] = arr1(i++];
while (j < S2)
    res[k++] = arr2 [j++];
```

Thumb rule.
Lower element will be moved to resultant array

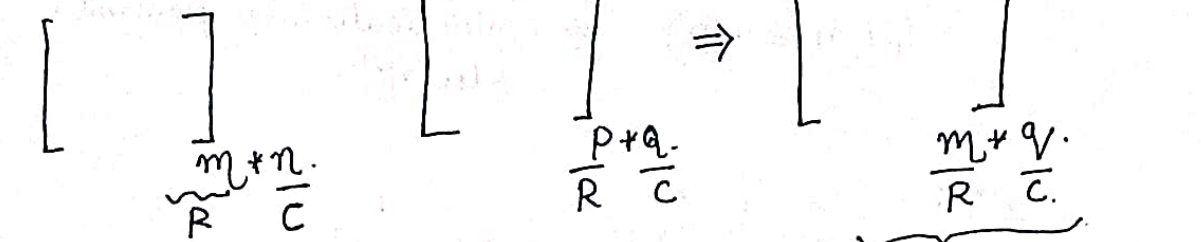
5) a[2] < a1[2]
5 < 6
True → Move 5 to arr3

6) a[3] < a1[2]
7 < 6
False → Move 6 to arr3

Call-by-Value & Call by Reference.

Matrix Multiplication.

① First Matrix. Second Matrix. Resultant matrix.



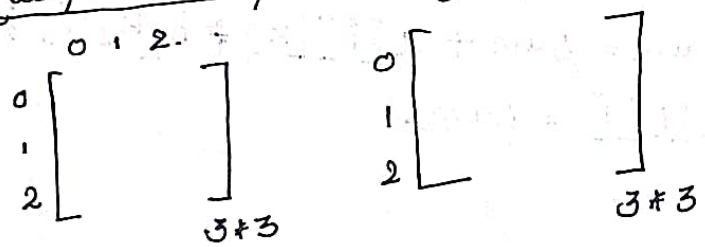
R ⇒ Rows
C ⇒ Columns.

Rows of 1st Matrix * Column of 2nd matrix.

② if $n = p$ ⇒ Matrix multiplication is possible. } Columns of first matrix = Rows of second matrix.

$n! = P$ X

③ Multiplication Proces.



Res = { 1st Rows of 1st matrix * Columns of first matrix. Columns of second matrix Columns of Third ..... .

2nd Rows of 1st Matrix * Colums of first. Colums of Second Columns of Third .....

Note: This illustration is only for understanding purpose.

Generalised form.

$A_{m*n} * B_{p*q}$

if ($n == p$) ⇒ Multiplication is possible.
Else not.

Ex:

$$a = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 3 & 2 & 1 \end{bmatrix} \quad b = \begin{bmatrix} 1 & 2 \\ 0 & 3 \\ 3 & 1 \end{bmatrix}$$

$3*3$ $m*n$ $\quad$ $3*2$ $p*q$

Resultant Matrix = $m*q$.

```
for (i=0; i<3; i++)
{
  for (j=0; j<2; j++)
  {
    for(k=0; k<3; k++)
    {sum = sum + a[i][k] * b[k][j];
    }
    c[i][j] = sum;
  }
}
```

i	j	k	sum
0	0	0	0

To print

```
for (i=0; i<3; i++)
{ for (j=0; j<2; j++)
  {
    printf ("%d", c[i][j]);
  }
}
```

Matrix addition

Note: Both the matrix dimensions should be same.

a_{m*n} b_{p*q} $\Rightarrow$ $c_{m=p\ n=q}$

$$a\begin{bmatrix}1 & 2 & 3\\ 0 & 5 & 2\end{bmatrix}_{2*3}$$

$$b\begin{bmatrix}5 & -1 & 2\\ 0 & 2 & 3\end{bmatrix}_{2*3}$$

```
for (i=0; i<m; i++)
{
   for (j=0; j<n; j++)
   {
      scanf ("%d" &a[i][j]);
   }
}

for(i=0; i<m; i++)
{
   for(j=0; j<n; j++)
   {
      scanf("%d", &b[i][j]);
   }
}

for(i=0; i<m; i++)
{
   for(j=0; j<n; j++)
   {
      c[i][j] = a[i][j] + b[i][j];
      printf("%d", c[i][j]);
   }
   printf("\n");
}
```

Bharath.S

Arrays.

Sum of Individual rows & Columns.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 5 \\ 0 & 5 & 1 \end{bmatrix} \begin{matrix} = 4 \\ = 10 \\ = 6 \end{matrix}$$

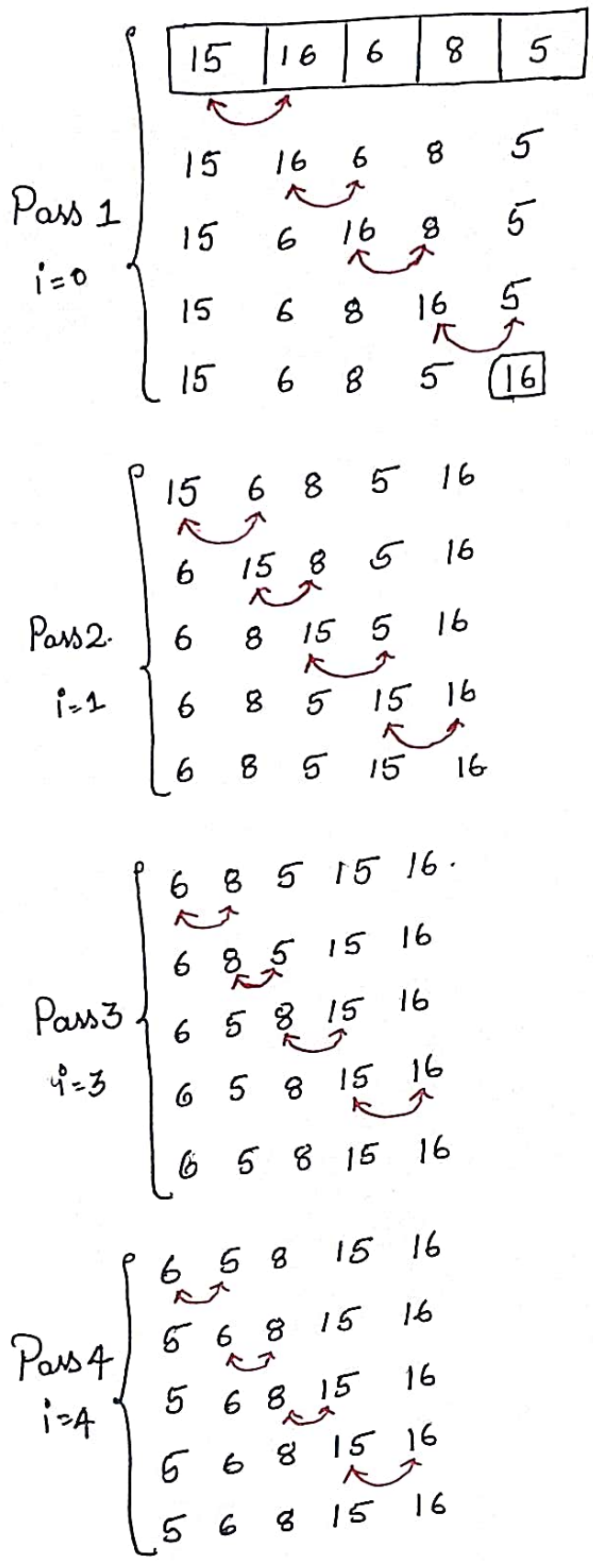
$$\begin{matrix} " & " & " \\ 3 & 10 & 6 \end{matrix}$$

```
int a[3][3], i, j, sumrow, sumcol;
for (i=0; i<3; j++)
{
   for(j=0; j<3; j++)
   {
      scanf("%d", &a[i][j]);
   }
}

for(i=0; i<3; i++)
{
   sumrow = sumcol = 0;
   for(j=0; j<3; j++)
   {
      sumrow = sumrow + a[i][j];
      sumcol = sumcol + a[j][i];
   }

  Printf ("sumrow = %d, sumcol = %d",
                        sumrow, sumcol);
}
```

Arrays

Bubble sort

$n = 5$



n = pass.
j = Comparison.

∴ No of pass always will be size-1.

We need two loops.
First loop ⇒ iteration
Second loop ⇒ for comparison.

```
for(i=0; i<n-1; i++)
{
    for(j=0; j<n-1; j++)
                j<n-1-i
    {
        if (a[j] > a[j+1])
        {
            temp = a[j];
            a[j] = a[j+1];
            a[j+1] = temp;
        }
    }
}
```

Application of arrays:

Storing Multiple Values: Arrays store a collection of data elements of the same type, e.g., storing student marks:
Ex: int marks [5] = {85, 90, 78, 92, 88};

String Handling: Arrays of characters are used for handling strings:
Ex: char name[] = "John";

Matrix Operations: 2D arrays are used for matrix operations like addition and multiplication:
int matrix[2][2] = {{1, 2}, {3, 4}};

Searching and Sorting: Arrays support algorithms like linear search and bubble sort:

```
for(int i = 0; i< n; i++)
{
if(arr[i] == key)
        {
        printf("Found");
        }
}
```

Implementing Stacks and Queues: Arrays can be used to implement stack and queue data structures.

Temporary Storage: Arrays act as temporary storage for data, such as storing intermediate results in computations.

Database Management: Arrays assist in storing records and tables for efficient data retrieval and manipulation.

Operations on arrays:

Traversing (Accessing Elements): Iterating through each element of the array.

```
int arr[5] = {1, 2, 3, 4, 5};
for(int i = 0; i< 5; i++)
{
printf("%d ", arr[i]);
}
```

Insertion (Adding Elements): Adding a new element at a specific position.

```
int arr[6] = {1, 2, 3, 4, 5};
arr[5] = 6;  // Insert 6 at index 5
```

Deletion (Removing Elements): Deleting an element from a specific position.

```
int arr[5] = {1, 2, 3, 4, 5};
int pos = 2;  // Remove element at index 2
for (int i = pos; i< 5-1; i++)
{
arr[i] = arr[i + 1];
}
```

Searching (Finding an Element): Finding an element using **linear search.**

```
int arr[5] = {1, 2, 3, 4, 5};
int key = 3, found = 0;
for(int i = 0; i< 5; i++)
{
  if(arr[i] == key)
 {
     found = 1;
printf("Element found at index %d", i);
     break;
  }
}
```

Sorting (Arranging Elements): Sorting elements using **bubble sort.**

```
int arr[5] = {5, 3, 1, 4, 2};
for(int i = 0; i< 5; i++)
{
for(int j = 0; j < 5-i-1; j++)
{
     if(arr[j] >arr[j+1]) {
        int temp = arr[j];
arr[j] = arr[j+1];
arr[j+1] = temp;
     }
  }
}
```

Merging (Combining Arrays): Combining two arrays into one.

```
int arr1[3] = {1, 2, 3}, arr2[3] = {4, 5, 6}, arr3[6];
for(int i = 0; i< 3; i++)
{
  arr3[i] = arr1[i];
}
for(int i = 0; i< 3; i++)
{
  arr3[i + 3] = arr2[i];
}
```

Copying (Cloning an Array): Copying elements from one array to another.

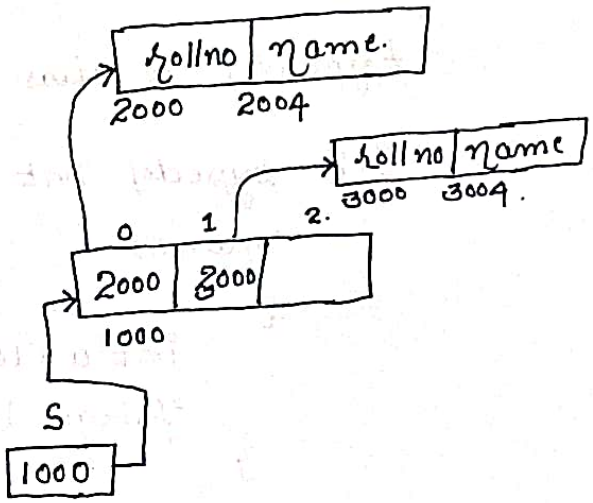
```
int arr1[5] = {1, 2, 3, 4, 5}, arr2[5];
for(int i = 0; i< 5; i++)
{
  arr2[i] = arr1[i];
}
```

Reversing an Array: Reversing the order of elements in an array.

```
int arr[5] = {1, 2, 3, 4, 5};
for(int i = 0; i< 5/2; i++)
{
  int temp = arr[i];
  arr[i] = arr[5-1-i];
 arr[5-1-i] = temp;
}
```

Array of structure.

```
Struct student
{
    Int rollno;
    Char name[10];
} s1[3];
```



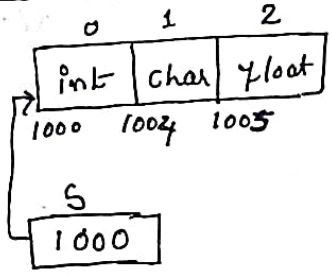
Unions in 'C'

→ User defined data type.
→ Almost similar to structure.
→ Difference is only with memory allocation.
→ Memory of max datatype is allocated.
→ Last value entered will be stored in the memory.

```
Struct student
{
  Int a;
  Char b;
  Float c;
};
Void main()
{
  Struct student s;
  Scanf("%d,%c,%f",
        &s.a, s.b, &s.c);
  Printf("%d %c %f",
    s.a, s.b, s.c);
}
```

```
Union student
{
  Int a; Char b;
  Float c;
};
Void main()
{
   union student us;
    u.a = 1
    u.b = S
    u.c = 10.5
   Pf("%d %c %f",
        u.a, u.b, u.c);
}
```

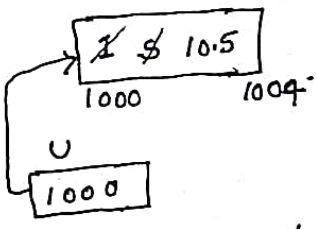
Structure.



Union.

Float, int = 4 byte
Char = 1

Max byte allocated is 4 bytes. So 4 bytes will be allocated.



Memory is shared among data types.

Structures in 'C'

Int a;

Int a[50]; ⇒ Array.
Collection of homogeneous data type.

Suppose If we want to store data of different data types.

Ex: Name ⇒ String
RollNO ⇒ Int.
Marks ⇒ float.
} This repeats for all the 50 students. In this case we go for structures with arrays.

Structure ⇒ user defined data type.

Declaration ⇒
```
struct structurename
{
    Declarations;
    Members of structures.
};
```

Ex:
```
struct student
{
    int rollno;
    char name[30];
    float marks;
};
```
struct — Keyword. student — Structure Name. struct student — Datatype. The three declarations — Members of a structure.

using the above we can define Info of a student.

Note: This is only study material. Plz go through text book also.

Structure is user defined datatypes that groups Elements/variables of different datatypes under a single name.

Points to Remember.

1) Always structure name should be followed by the keyword struct
2) At the End of the structure should place semicolon.
3) Individual declaration is not allowed.
4) Duplicate variables cannot be repeated with different data types.

```
Struct student
{
   int rollno;
   float marks;
   char name[20];
};
```

Point-1.

Point-2

```
struct student.
{
   int rollno = 1;
   float marks;
   char name[20];
};
```

Point-3

```
Struct student.
{
   int rollno;
   float rollno;
   char name[10];
   float marks;
}
```

Point-4.

Declaring variables/objects of structure.

Ex: Struct student
{
int rollno; — 4 byte
float marks; — 4 byte
char name[20]; — 20 bytes
} ; } Size of the structure is 28 bytes.

void main()
{
Struct student S1;
.....
.....
}

{ sizeof(S1);
sizeof(struct student); } → To find size of structure.

→ Object of the structure. OR Variable of the structure.

Ex 2: Struct student
{
int rollno;
float Marks;
char name[10];
} S1; ⇒ We can also declare more objects.
Like S1, S2, S3....

void main()
{
.....
.....
}

In Practical scenario if we have assume 60 students we can't give S1, S2, S3, ... S60.
This is not recomended.

Shanath Babu. S.

Initialising & Accessing of Structure Members

```
Struct Student
{
    int rollno;
    Char name[30];
    Float marks;
};
```

OR

```
Struct Student
{
    int rollno;
    Char name[30];
    Float mark;
} S={1,"SRI",90};
```

```
Void main()
{
    Struct Student S1={1,"SRI",25}; //Compile time Execution.
    Struct Student S1={1}; // will be assigned to rollno & next two values would be NULL.
```

Note: We can declare outside main also

```
//Note: while entering the data order should be matched.
Scanf("%d %s %f", S1.rollno, S1.name, S1.marks);
Printf("%d", S1.rollno);      // using . operator the members of Structure variables
Printf("%s", S1.name);        // can be accessed.
printf("%f", S1.marks);

// Copy of object variables is allowed.
S1 = S2 ✓
S1 > S2 X
S1 != S2 X
```

Shanath's Structures.

// Comparing of Individual members of the Structures is possible. but not objects.

```
if (S1.rollno < S2.rollno)
{
  Printf("Hi");
}
```

Ex:

```
#include <stdio.h>
Struct student
{
  int rollno;
  Char name[20];
  float marks;
};
Void main ()
{
  Struct student S1 = {1, "Sri" 90};
  Struct student S2 = {2, "Krishna", 95};
  Printf(" Information of S1");
  Printf(" %d %s %f", S1.rollno, S1.name,
                      S1.marks);
  Printf(" %d %s %f", S2.rollno, S2.name,
                      S2.marks);
}
```

*** → Compile time. Initialisation

** If declared as S1 = S2.
Then o/p will be same for S1 & S2.

//Run time Initialisation.
Scanf ("%d %s %f", &S1.rollno, S1.name, &S1.marks);

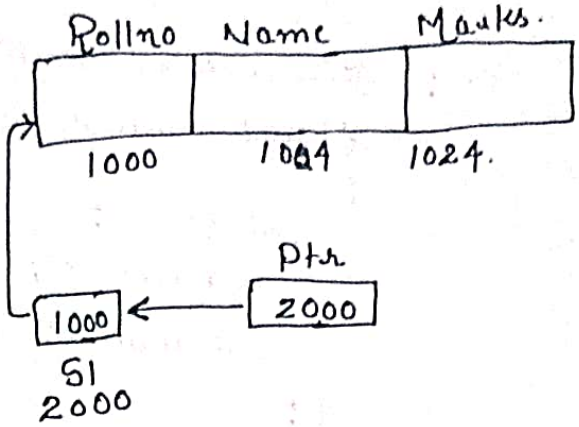
Shaath's

Structures.

To access members of more than one object then we can go for array of structures.

```
Struct student.
{
   Int rollno;
   Char name[20];
   Float marks;
};
```

```
int a;
int *ptr = &a
```



we can access members of structure in two ways.

1) using structure objects.

~~Printf~~ Ex: S1.rollno.
S1.Name.
S1.marks.

2) using pointer variable.

Struct student *ptr = &S1; → Structure Pointer.

Ex: ptr → rollno OR (*ptr).rollno
ptr → Name
ptr → Marks.

Structure Pointer ⇒ Pointer that points to the address of the memory block that stores the structure.

// Prog to demonstrate structure using arrays.

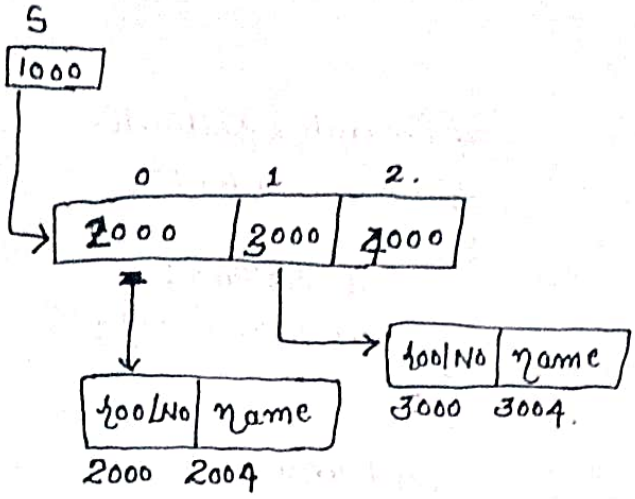
```
#include <stdio.h>
struct student
{
   int rollno;
   char name[30];
};
void main()
{
   struct student s[2];
   int i;
   printf("Enter the details");
   for(i=0; i<=2; i++)
   {
      scanf("%d", &s.rollno);
      scanf("%s", s.name);

      scanf("%d", &s[i].rollno);
      scanf("%d", s[i].name);
   }
   printf("Details after Entering the values");
   for(i=0; i<=2; i++)
   {
      printf("%d", s[i].rollno);
      printf("%s", s[i].name);
   }
}
```

} Results with Error. whose name & rollno has to be accessed.

Structures.

Array of structure.

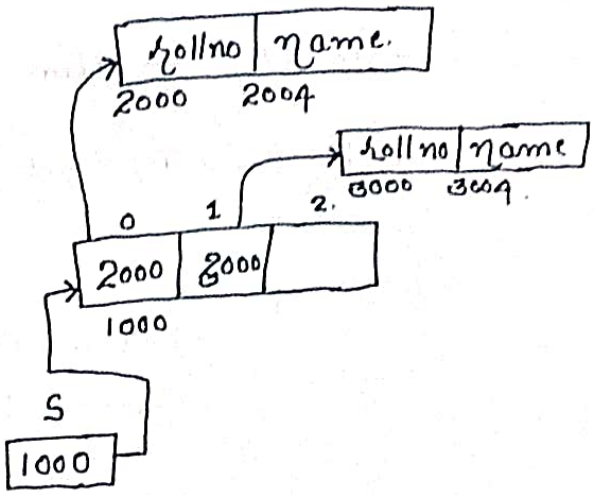
```
struct student
{
   int rollno;
   char name[10];
} s[3];
```




```
#include <stdio.h>
struct student
{
   int rollno;
   char name[10];
};
void main()
{
   struct student s[2];
   int i;
   printf("Enter the details:");
   for(i=0; i<2; i++)
   {
      scanf("%d", &s[i].rollno);
      scanf("%s", s[i].name);
   }
   for(i=0; i<2; i++)
   {
      printf("%d", s[i].rollno);
      printf("%d", s[i].name);
   }
}
```

Array of Structure.

```
Struct Student
{
    Int rollno;
    Char name[10];
} s1[3];
```



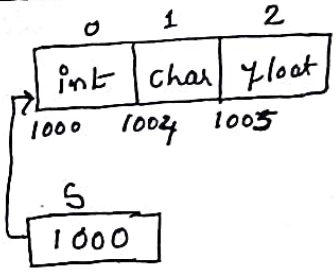
Unions in 'C'

→ User defined data type.
→ Almost similar to structure.
→ Difference is only with memory allocation.
→ Memory of max datatype is allocated.
→ Last value entered will be stored in the memory.

```
Struct student
{
  Int a;
  Char b;
  Float c;
};
Void main()
{
  Struct student s;
  Scanf("%d,%c,%f",
        &s.a, s.b, &s.c);
  Printf("%d %c %f",
     s.a, s.b, s.c);
}
```

```
Union student
{
  Int a; Char b;
  Float c;
};
Void main()
{
   union student us;
    u.a = 1
    u.b = s
    u.c = 10.5
   Pf("%d %c %f",
        u.a, u.b, u.c);
}
```

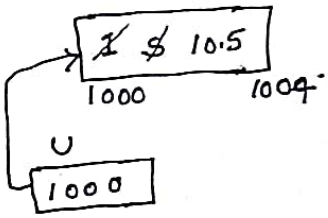
Structure.



Union.

Float, int = 4 byte
Char = 1

Max byte allocated is 4 bytes. So 4 bytes will be allocated.



Memory is shared among data types.

typedef ⇒ alias to the structure or datatype.

Ex:
```
typedef int integer;
main()
{
    int a = 10;
    Integer b = 20;
}
```

```
typedef struct student
{
    Int a;
    Char b;
    Float c;
} stu;
```
↳ alias to the structure.

```
Void main().
{
    stu s = {10, 'c', 10.5}
    Printf ("%d %c %s", s.a, s.b, s.c);
}.
```

Structures. Shaiath Babu. S.

To demonstrate pointer with structure.

```
#include<stdio.h>
Struct student
{
    int rollno;
    Char name[20];
};
Void main()
{
  Struct student s;
  Struct student *ptr = &s;
  Printf("Enter the details");
  Scanf("%d", &(ptr)→rollno);
  Scanf("%s", (ptr)→name);
  Printf("Entered details");
  Printf("%d", (ptr)→rollno);
  Printf("%s", (ptr)→name);
}
```

using arrays.

```
#include<stdio.h>
Struct student
{ int rollno;
  Char name[20];
}
Void main()
{
  Struct student s[2];
  Struct student *ptr = s;
  int i;
  Pf("Enter details");
  For(i=0; i<2; i++)
  {
     Sf("%d", &(ptr+i)→rollno);
     Sf("%s", (ptr+i)→name);
  }
Pf("Entered details");
For(i=0; i<2; i++).
 {
   Pf("%d", (ptr+i)→rollno);
   Pf("%d", (ptr+i)→name);
  }
}
```

Enum in 'C'

Enum ⇒ user defined datatype to assign names to integer constants.

→ uses keyword enum. (enumerate)

→ Creat new datatypes to contain values that are not limited to the values that fundamental datatypes may take.

→ Variables can have set of values.

→ At any point of time enum can have only one value.

Ex : Directions → East, west, North, South.

week days → Sun, mon, tue, ----, sat

Month → Jan, Feb, ------, Dec.

Gender → Male, female, others.

ie Directions ⇒ only 4 possibilities

Month ⇒ 12 possibilities

Week days ⇒ 07 possibilities.

Scope ⇒ It may have Local scope as well as Global values.

```
#include <stdio.h>
enum weekday {sun, mon, tue, wed, thu, fri, sat };

void main().
{
   enum weekday today;
   today = Monday;
   pf ("todays value = %d", today);
```

Enum.

```
//using enums in loop.
Printf ("Days of the week");.
for (int day = Sun ; day <= Sat ; day++)
{
   Printf ("%d", day);
}

//To print names of days
for (day = Sun ; day <= Sat ; day++)
{
   if (day == Sun)
      Printf ("SUNDAY");
   if (day == Mon)
      Printf ("MONDAY");
   if (day == Tue)
      Printf ("TUESDAY");
      :
   if (day == Sat):
      Printf ("SATURDAY");
}
}
```

OUTPUT Todays value = 1
Days of week = 0, 1, 2, 3, 4, 5, 6, ∓.
Days of week = SUNDAY - - - - -

Points to Remember.

1) only Integer values is assigned automatically by the Compiler.

Ex: Enum weekday { sun, mon, tue, wed, thu, fri, sat }
(a) 0 1 2 3 4 5 6

2) you can also assign your own values to the enumerations.

Ex: Enum weekday { sun = 5, mon, tue, wed, thu, fri, sat }.
(b)

In this case value of mon = 6, tue = 7 ....., sat = 11

NOTE: Values can be assigned individually or for all values of Enum.

3) Suppose if you have entered Integer values for only first 2~~a~~ days.

Ex: Enum weekday { sun = 10, mon = 1, tue, wed, thu, fri, sat }.
(c)

in this case Compiler reads.

sun = 10, mon = 1, tue = 2, wed = 3, thu = 4, fri = 5, sat = 6;.

Ex: Enum weekdays (sun = 10, mon, tue, wed = 20, thu fri, sat);
(d)

in this case Compiler reads.

sun = 10, mon = 11, tue = 12.
wed = 20, thu = 21, fri = 22, sat = 23.

Enum

//Program to display Color names using enum.

```
#include <stdio.h>
Enum Colors {"Red","Blue","green","Yellow","Black",
Enum Colors {Red, Blue, green, Yellow, Black, white}

Int main()
{
   Enum Colors C;
   Printf ("Colors name");
   For (C=Red ; C <= white ; C++)
   {
      if (C == Red)
         Printf ("Red");
      if (C == Blue)
         Printf ("Blue");
       :
   }
}
```