ARRAYS

VALID OR INVALID?

```
int n;
n=10;
int a[n];
```

- Invalid
- The compiler allocates memory for the array during compile time(also called as static memory allocation).
- However, the statement (n=10) executes during runtime i.e after compilation.
- The compiler does not know the exact value of “n” as “n” is garbage.
- Thus, the size of an array during declaration in C must ALWAYS be a constant.

ACCESSING ALL ELEMENTS OF AN ARRAY

- Syntax:

```
for(i=0;i<=MAX_SIZE-1;i++)
{
// Access elements using a[i]
}
```

- Here i denotes the index starting from 0 and goes upto MAX_SIZE-1.
- As "i" i.e the position changes, a[i] i.e the value in the array at that position also changes.

ACCESSING ALL ELEMENTS OF AN ARRAY

- Initialization of array elements through user input(Method 2):

Syntax:

```
for(i=0;i<=MAX_SIZE-1;i++)
{
scanf(“%d”,&a[i]);
}
```

Example:

```
int a[10];
printf(“Enter the array elements”);
for(i=0;i<=9;i++)
{
scanf(“%d”,&a[i]);
}
```

USER INPUT OF ARRAY ELEMENTS

- If we consider the previous example, we are forcibly making the user enter 10 elements.
- In order to make it more user friendly, we can also ask the user, “how many elements” are actually required for the operation. Let us call that “n”.
- Note that this “n” must be <=MAX_SIZE.
- <u>Points To Note:</u>
- MAX_SIZE must always be chosen such that it is sufficient to do our operations.
- Even though we ask the user to enter “n”, we still cannot use this “n” for declaration i.e int a[n]; is invalid.
- Through this method, some space in array will be wasted but as C is a primitive language, this is a small flaw in the design of the language.

USER INPUT OF ARRAY ELEMENTS

Example:

```
int a[10],n;
printf("Enter how many elements");
scanf("%d",&n);
printf("Enter the array elements");
for(i=0;i<=n-1;i++)
{
scanf("%d",&a[i]);
}
```

RELATIONSHIP BETWEEN ARRAYS AND POINTERS

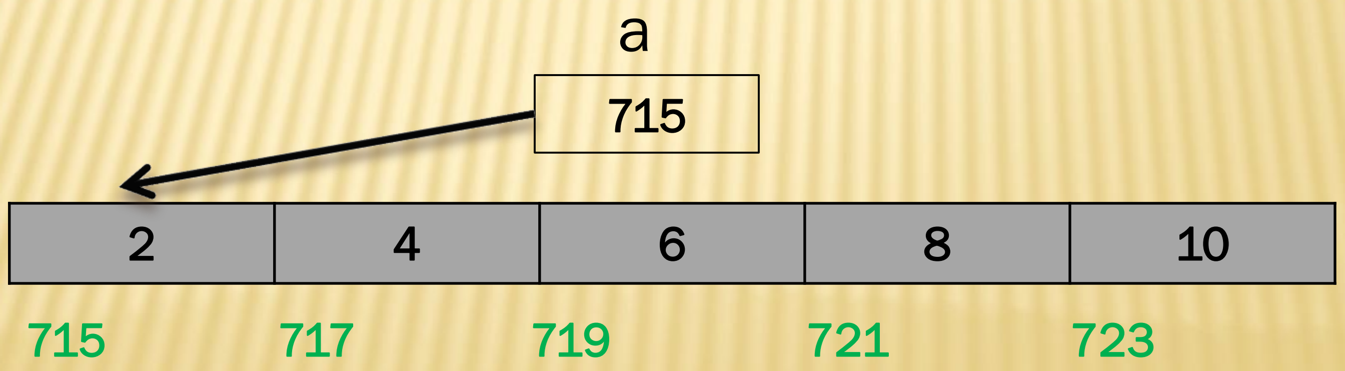
int a[10]={2,4,6,8,10};

printf("%d", a[0]); $\longrightarrow$ 2

printf("%d", a[2]); $\longrightarrow$ 6

printf("%u , a); $\longrightarrow$ Address of the first element of the array

- Name of an array is a pointer variable pointing to the first element of the array.



RELATIONSHIP BETWEEN ARRAYS AND POINTERS

```
int a[10]={2,4,6,8,10};
printf(“%__”, a[0]);
printf(“%__”, a[2]);
printf(“%__, a);
printf(“%__, a+1);
printf(“%__, &a[1]);
printf(“%__, *(a+1));
```

RELATIONSHIP BETWEEN ARRAYS AND POINTERS (ASSUME STARTING ADDRESS IS 715)

int a[10]={2,4,6,8,10};

printf(“%d”, a[0]); $\longrightarrow$ 2

printf(“%d”, a[2]); $\longrightarrow$ 6

printf(“%u, a); $\longrightarrow$ 715

printf(“%u, a+1); $\longrightarrow$ 717(Using pointer arithmetic)

printf(“%u, &a[1]); $\longrightarrow$ 717

printf(“%d, *(a+1)); $\longrightarrow$ 4 (*(715 +1)= *(717))

RELATIONSHIP BETWEEN ARRAYS AND POINTERS

- Using the idea of pointer arithmetic,

  a+i ≡ &a[i]

  *(a+i) ≡a[i]

- Point to note:

  As the array_name is a pointer variable, pointing to the FIRST element ,

  a ≡ a+0 ≡ &a[0]

  *a ≡ *(a+0) ≡ a[0]

- Hence, array index starts with "0".

Relationship between Arrays and Pointers (Assume starting address is 715)

int a[10]= {2,4,6,8,10};

int *p;

p=a; $\longrightarrow$ ✓(Both are int pointers)

printf("%d", *p); $\longrightarrow$ ✓2

printf("%u", p); $\longrightarrow$ ✓715

p++; $\longrightarrow$ ✓(Pointer arithmetic)

printf("%u", p); $\longrightarrow$ ✓717

printf("%d, *p); $\longrightarrow$ ✓4

printf("%d", p[0]); $\longrightarrow$ ✓4 (p[0] = *(p+0)

a++; $\longrightarrow$ X (Array_name is a CONSTANT pointer)

Point to note:

p++ is valid as p is a general pointer

a++ is invalid as array name is a CONSTANT pointer variable.

PROGRAMS

- Access all elements of an integer array using a pointer variable.
- Input “n” elements of an array. (“n” is also input by the user). Count and display the number of odd elements in the array and number of even elements in the array.
- Program to copy the contents of one array into another