C programming

Trainer : Nisha Dingare

Email : nisha.dingare@sunbeaminfo.com

Arrays Introduction :

- An array is a finite collection of similar data items stored in contiguous memory locations.
- All the elements in the array share the same name i.e. the name of array. However each element in an array is uniquely identified by its index.
- Array indexing starts from 0. max index is n-1.
- They can be used to store the collection of primitive data types i.e. int, float, char etc. as well as derived data types like structures, pointers etc.
- We can store only the fixed size of elements in the array. Its size cannot grow at runtime.
- Checking array bounds is the responsibility of the programmer.

1D Array Declaration :

Array Declaration :

data_type variable_name [array-size];

int arr[5];

arr[0] = 10;

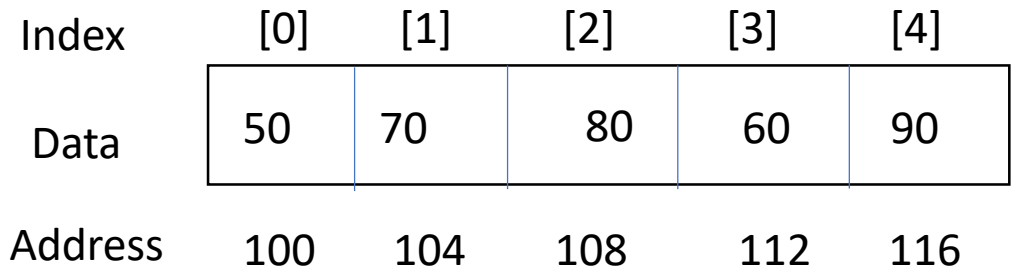
arr[1] = 20;

Array initialization :

int arr[5] = {50,70,80,60,90};

int arr[] = {10,20,30,40};

int arr[5] = {10,20,30};



Arrays : Points to remember :

- If array is initialized partially at its point of declaration rest of elements are initialized to zero.
- If array is initialized at its point of declaration, giving array size is optional. It will be inferred from number of elements in initializer list.
- The array name is treated as address of 0th element in any runtime expression.
- Arrays can be implemented statically or dynamically. If array is implemented statically then such array can not be shrink or grown at runtime.
- If programmer is aware of no. of elements to processed in advance then use static implementation.
- Pointer to array is pointer to 0th element of the array.
- Array is always passed by address to a function.

Passing array to function :

- Arrays are passed to function by address.
- The address of starting element of array(Base address) is passed to the function.
- The address is collected in a pointer.
- We can use pointer as well as array notation to access array elements in the function.
- In a function definition, a formal parameter that is declared as an array is actually a pointer.
- When an array is passed, its base address is passed call-by-value.
- The array elements themselves are not copied.
- As a notational convenience, the compiler allows array bracket notation to be used in declaring pointers as parameters

Pointer Arithmetic :

- We usually perform arithmetic operations on numeric values. A pointer is an address which is a numeric value so arithmetic ops on pointers are possible.
- The operators used for pointers are ++, -- ,+ , -.
- The most important term in pointer arithmetic is scale factor.
- The size of a data type of the pointer is the scale factor of that pointer.
- When pointer is incremented or decremented by 1, it changes by the size of data type to which it is pointing (scale factor)
- Example :

```
Int main()
{
   int arr[5] = {10,20,30,40,50};
   int *ptr = &arr[1];    // ptr pointing towards arr[1]
   ptr++;                 // ptr moves ahead 4 bytes and points to  next integer location at arr[2].
   }
```

At line ptr++, the pointer points at a next integer location without impacting the actual value at the location.

Points to Remember :

- Multiplication, division, and modulus of any integer with the pointer is not allowed.
- Addition, multiplication and division of two pointers is not allowed.
- Subtraction of two pointers can be meaningful if both are storing addresses of the elements of the same array. Here, it gives the number of elements between the two pointers.
- An array name represents the address of the beginning of the array in memory.
- When an array is declared, the compiler must allocate a base address and a sufficient amount of storage (RAM) to contain all the elements of the array.
- The base address of the array is the initial location in memory where the array is stored.
- It is the address of the first element (index 0) of the array.

Pointers and arrays :

- Accessing the array elements using pointer is possible using the '*' (value at) operator.
- Example :

Pointer notation :

*ptr = value at 0^{th} element.

*(ptr+1) = value at 1^{st} element.

Array notation :

arr[0] = value at 0^{th} element.

arr[1] = value at 1^{st} element.

arr[1] == *(arr+1) → subscript op internally follows pointer arithmetic

arr[1] == *(arr+1) == *(1+arr) == 1[arr] → same output.

Type Qualifier - const :

- const keyword informs compiler that the variable is not intended to be modified.
- Compiler do not allow using any operator on the variable which may modify it e.g. ++, --, =, +=, -=, etc.
- Note that const variables may be modified indirectly using pointers. Compiler only check source code (and do not monitor runtime execution).

- const float PI = 3.14; //float PI is constant here
- Points To Note: Need to initialize at the time of declaration. Utilizes memory depending on data type location to which it is applied.
- Can be modified using pointer.
- Const float *fptr = &PI;
- *fptr = 40.2; // chanigng value of variable with pointer is not allowed as fptr is constant.
- float fval = 12.34;
- fptr = &fval ; // this is allowed as changing pointer is not const.
- const float * const fptr=&PI; // here changing value of variable as well as changing address to which the fptr is pointing is not allowed

Thank You !