Unit #: 2

Unit Name: Counting, Sorting and Searching

Topic: Arrays, Initialization and Traversal

Course objectives:

The objective(s) of this course is to make students

CObj1: Acquire knowledge on how to solve relevant and logical problems using computing machine

CObj2: Map algorithmic solutions to relevant features of C programming language constructs

CObj3: Gain knowledge about C constructs and it's associated eco-system

CObj4: Appreciate and gain knowledge about the issues with C Standards and it's respective behaviors

CObj5: Get insights about testing and debugging C Programs

Course outcomes:

At the end of the course, the student will be able to

CO1: Understand and apply algorithmic solutions to counting problems using appropriate C Constructs

CO2: Understand, analyze and apply text processing and string manipulation methods using C Arrays, Pointers and functions

CO3: Understand prioritized scheduling and implement the same using C structures

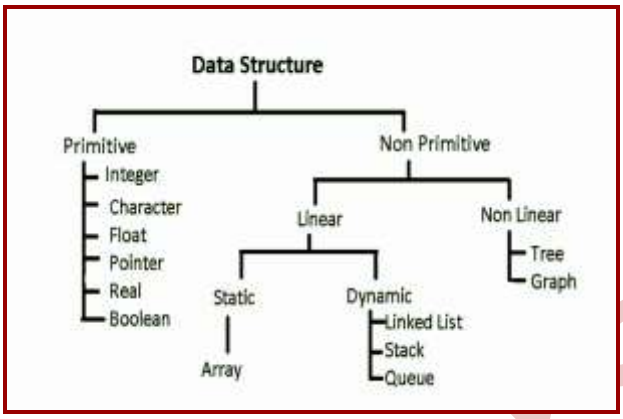
CO4: Understand and apply sorting techniques using advanced C constructs

CO5: Understand and evaluate portable programming techniques using preprocessor directives and conditional compilation of C Programs

Team – PSWC,
Jan - May, 2022
Dept. of CSE,
PES University

Data Structures

Data structures are **used** to **store data** in an **organized** and **efficient** manner so that the **retrieval is efficient.** Classification of data structures in C is as below.



When solving problems, it is important to visualize the data related to the problem. Sometimes the data consist of just a single number (as we discussed in unit-1, primitive types). At other times, the data may be a coordinate in a plane that can be represented as a pair of numbers, with one number representing the x-coordinate and the other number representing the y-coordinate. There are also times when we want to work with a set of similar data values, but we do not want to give each value a separate name. For example, we want to read marks of 1000 students and perform several computations. To store marks of thousand students, we do not want to use 1000 locations with 1000 names. To store group of values using a single identifier, we use a data structure called an Array.

An array is a linear data structure, which is a finite collection of similar data items stored in successive or consecutive memory locations. Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. It can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

To create an array, define the data type and specify the name of the array followed by square brackets [] with size mentioned in it. **Example: int s[100];**

Characteristics/Properties of arrays:

- Non-primary data or secondary data type
- Memory allocation is contiguous in nature
- Elements need not be unique.
- Demands same /homogenous types of elements
- Random access of elements in array is possible
- Elements are accessed using index/subscript
- Index or subscript starts from 0
- Memory is allocated at compile time.
- Size of the array is fixed at compile time. Returns the number of bytes occupied by the array.
- Cannot change the size at runtime.
- Arrays are assignment incompatible.
- Accessing elements of the array outside the bound will have undefined behaviour at runtime.

Types of Array

Broadly classified into two categories as

Category 1:

1. Fixed Length Array**:** Size of the array is fixed at compile time
2. Variable Length Array - Not supported by older few C standards. Better to use dynamic memory allocation functions than variable length array.

Category 2:

1. One Dimensional Array
2. Multi-Dimensional Array

One dimensional Array

- One dimensional array is also called as linear array(also called as 1-D array).
- The one dimensional array stores the data elements in a single row or column.

Array Declaration:

Syntax: Data_type Array_name[Size]; // Declaration syntax

- Data_type : Specifies the type of the element that will be contained in the array
- Size: Indicates the maximum number of elements that can be stored inside the array
- Array_name: Identifier to identify a variable.
- Subscripts in array can be integer constant or integer variable or expression that yields integer
- C performs no bound checking – care should be taken to ensure that the array indices are within the declared limits

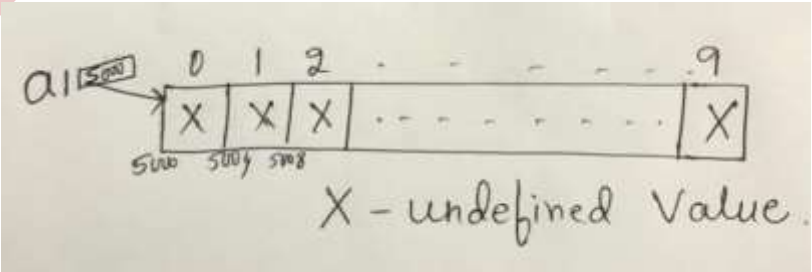
Examples:

- float average [6]: // Can contain 6 floating point elements, 0 to 5 are valid array indices
- char name[20]; // Can contain 20 char elements, 0 to 19 are valid array indices
- double x[15]; // Can contain 15 elements of type double, 0 to 14 are valid array indices.

Consider int a1[10];

Declaration allocates number of bytes in a contiguous manner based on the size specified. All these memory locations are filled with undefined values. If the starting address is 0x5000 and if the size of integer is 4 bytes, refer the diagrams below to understand this declaration clearly. Number of bytes allocated = size specified*size of integer i.e, 10*4.

printf("size of array is %d\n", sizeof(a1)); // 40 bytes



Address of the first element is called the Base address of the array. Address of i^{th} element of the array can be found using formula: **Address of i^{th} element = Base address + (size of each element * i)**

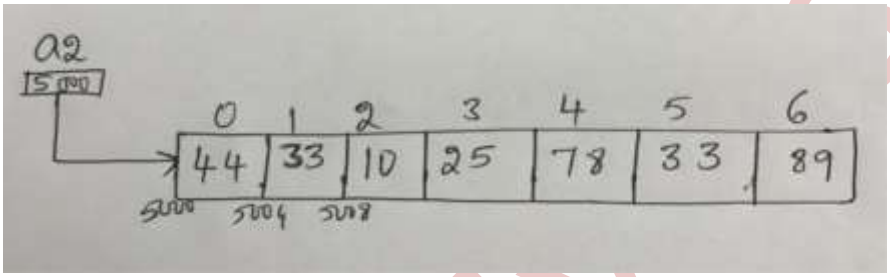
Array Initialization:

- After an array is declared it must be initialized.
- An Uninitialized array will contain undefined values/junk values.
- An array can be initialized at either compile time or at runtime

Consider int a2[ ] = {44,33,10,25,78,33,89};

Above initialization does not specify the size of the array. Size of the array is based on the number of elements stored in it and the size of type of elements stored. So, the above array occupies 7*4 = 28 bytes of memory in a contiguous manner.

printf("size of array is %d\n", sizeof(a2)); //28 bytes



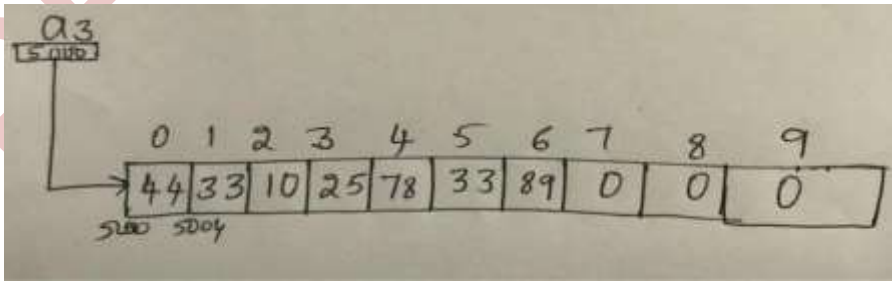
Partial Initialization:

Consider int a3[10] = {44,33,10,25,78,33,89};

Size of the array is specified. But only few elements are stored. Now, the size of the array is 10*4 = 40 bytes of memory in a contiguous manner. **All uninitialized memory locations in the array are filled with default value 0.**

printf("size of array is %d\n", sizeof(a3)); //40 bytes

printf("%p %p\n",a3, &a3); // Must be different. But shows same address



Compile time Array initialization:

Compile time initialization of array elements is same as ordinary variable initialization. The general form of initialization of array is,

Syntax: data-type array-name[size] = { list of values };

```
int marks[4]={ 67, 87, 56, 77 }; // integer array initialization
float area[5]={ 23.4, 6.8, 5.5 }; // float array initialization
intarr[] = {2, 3, 4};
int marks[4]={67,87,56,77,5} //undefined behavior
```

Variable length array:

Variable length arrays are also known as runtime sized or variable sized arrays. The size of such arrays is defined at run-time.

The variable length arrays are always unsafe. This dynamically creates stack memory based on data size, which can cause overflow of stack frame.

Designated initializers (C99):

- It is often the case that relatively few elements of an array need to be initialized explicitly; the other elements can be given default values:

  ```
  int a[15] = {0,0,29,0,0,0,0,0,0,7,0,0,0,0,48};
  ```

- So, we want element 2 to be 29, 9 to be 7 and 14 to be 48 and the other values to be zeros. For large arrays, writing an initializer in this fashion is tedious.
- C99's designated initializers

  ```
  int a[15] = {[2] = 29, [9] = 7, [14] = 48};
  ```

- Each number in the brackets is said to be a designator.
- Besides being shorter and easier to read, designated initializers have another advantage: the order in which the elements are listed no longer matters. The previous expression can be written as:

  ```
  int a[15] = {[14] = 48, [9] = 7, [2] = 29};
  ```

Designators must be constant expressions:

- If the array being initialized has length n, each designator must be between 0 and n-1.
- if the length of the array is omitted, a designator can be any non-negative integer.
- In that case, the compiler will deduce the length of the array by looking at the largest designator.

```
int b[] = {[2] = 6, [23]=87};
```

- Because 23 has appeared as a designator, the compiler will decide the length of this array to be 24.
- An initializer can use both the older technique and the later technique.

```
int c[10] = {5, 1, 9, [4] = 3, 7,2,[8]=6};
```

Runtime initialization: Using a loop and input function in c

Example:

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

Traversal of the Array:

Accessing each element of the array is known as traversing the array.

Consider int a1[10];

- How do you access the 5th element of the array? **a1[4]**
- How do you display each element of the array?

```
int i;
for(i = 0; i<10;i++)
  printf("%d\t",a1[i]);//Using the index operator [ ].
//Above code prints undefined values as a1 is just declared.
```

Consider int a2[ ] = {12,22,44,14,77,911};

```
int i;
for(i = 0; i<10;i++)
   printf("%d\t",a2[i]);//Using the index operator [ ].
```

Above code prints initialized values when i is between 0 to 5. When i becomes 6, a2[6] is outside bound as the size of a2 is fixed at compile time and it is 6*4(size of int is implementation specific) = 24 bytes

Anytime accessing elements outside the array bound is an undefined behavior.

Coding Example_1: Let us consider the program to read 10 elements from the user and display those 10 elements.

```
#include<stdio.h>
int main()
{       int arr[100];
        int n;
        printf("enter the number of elements\n");
        scanf("%d",&n);
        printf("enter %d elements\n",n);
        int i = 0;
        while(i<n)
        {       scanf("%d",&arr[i]);
                i++;
        }
        printf("entered elements are\n");
        i = 0;
        while(i<n)
        {       printf("%d\n",arr[i]);
                i++;
        }
```

```
    return 0;
    }
```

Output:

```
enter the number of elements
3
enter 3 elements
34
56
89
entered elements are
34
56
89


...Program finished with exit code 0
Press ENTER to exit console.
```

Coding Example_2: Let us consider the program to find the sum of the elements of an array.

```
int arr[] = {-1,4,3,1,7};
int i;
int sum = 0;
int n = sizeof(arr)/sizeof(arr[0]);
for(i = 0; i<n; i++)
{       sum += arr[i];
}
printf("sum of elements of the array is %d\n", sum);
```

Think about writing a function to find the sum of elements of the array. Few questions to think about above codes?

- Should I use while only? for and while are same in C except for syntax.
- Can I use arr[n] while declaration? Variable length array
- Can we write read and display user defined functions to do the same? Yes. To do this, we should understand functions and how to pass array to a function.

Happy Coding using Arrays!!