PROBLEM SOLVING WITH C
UE23CS151B

Prof. Sindhu R Pai
Department of Computer Science and Engineering

PROBLEM SOLVING WITH C

Arrays - Initialization and Traversal Pointers

Prof. Sindhu R Pai
Department of Computer Science and Engineering

PROBLEM SOLVING WITH C

Arrays: Initialization and Traversal, Pointers:

- Array
  1. What is an Array?
  2. Properties of Arrays.
  3. Classification of Arrays
  4. Declaration and Initialization
  5. Representation of the Array
  6. Array Traversal

- Pointer
  1. What is a Pointer?
  2. Declaration and Initialization
  3. Arithmetic operations on Pointer
  4. Array Traversal using pointers
  5. Array and Pointer

What is an Array?

- A linear data structure, which is a Finite collection of similar data items stored in successive or consecutive memory locations
- **Only homogenous types of data** is allowed in any array. May contain all integer or all character elements, but not both together.
- char c_array [10]; short s_array [20];

Properties of Arrays

- Non-primary data type 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 which starts from 0
- Memory is allocated at compile time.
- Size of the array is fixed at compile time and cannot be changed at runtime. Returns the number of bytes occupied by the array.
- Arrays are assignment incompatible.
- Accessing elements of the array outside the bound can have undefined behavior at runtime

Classification of Arrays

Category 1:

- Fixed Length Array
  - Size of the array is fixed at compile time
- Variable Length Array – Not discussed here

Category 2:

- One Dimensional (1-D) Array
  - Stores the data elements in a single row or column.
- Multi Dimensional Array
  - More than one row and column is used to store the data elements

Declaration and Initialization

Declaration: **Data_type Array_name[Size];**

Data_type: Specifies the type of the element that will be contained in the array
Array_name: Identifier to identify a variable as an array
Size: Indicates the max no. of elements that can be stored inside the array

Example: double x[15]; // Can contain 15 elements of type double, 0 to 14 are valid array indices or subscript

- 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

Declaration and Initialization

- After an array is declared, it must be initialized.
- An array can be initialized at either **compile time** or at **runtime.**

Compile time Initialization

- data-type array-name[size] = { list of values };
- float area[5]={ 23.4, 6.8, 5.5 }; // Partial initialization
- int a[15] = {[2] = 29, [9] = 7, [14] = 48}; //C99's designated initializers
- int a[15] = {0,0,29,0,0,0,0,0,0,7,0,0,0,0,48}; //C99's designated initializers
- int arr[] = {2, 3, 4}; // sizeof arr is decided
- int marks[4]={ 67, 87, 56, 77, 59 }; // undefined behavior
- Coding Examples

Runtime Initialization

- Using a loop and input function in C
- Coding examples

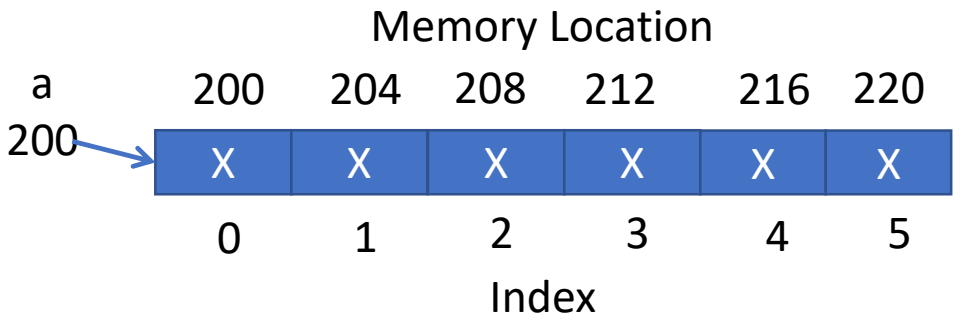
Representation of the Array

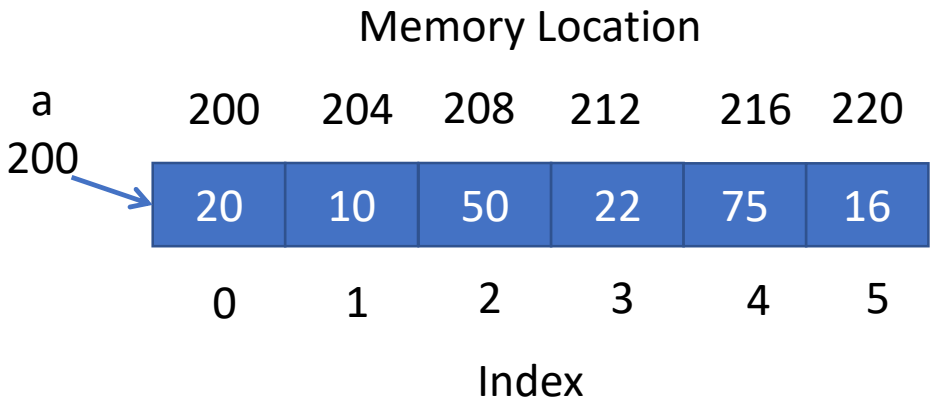
- int a[6];
- int a[6] = {20, 10, 50, 22, 75,16};

- Address of the first element is called the Base address of the array.

- Address of ith element of the array can be found using formula:

  Address of ith element = Base address + (size of each element * i)





Traversal

- Nothing but accessing each element of the array

- int a[10];
  - How do you access the 5th element of the array? // a[4]
  - How do you display each element of the array? // Using Loop
  - How much memory allocated for this?

    Number of bytes allocated = size specified*size of integer

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

- Coding examples

Pointers

What is a Pointer?

- A variable which contains the address. This address is the location of another object in the memory

- Used to access and manipulate data stored in memory.

- Pointer of particular type can point to address of any value in that particular type.

- Size of pointer of any type is same/constant in that system

- Not all pointers actually contain an address

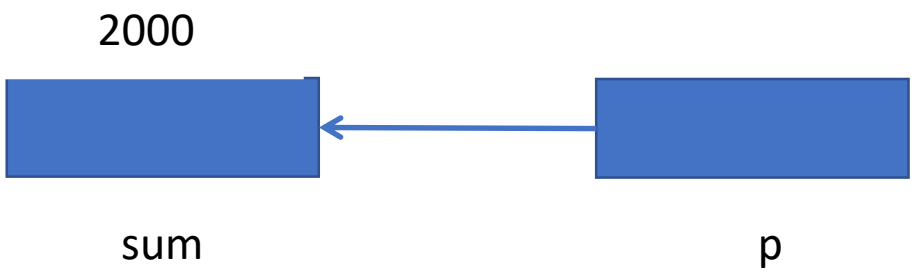
  Example: NULL pointer // Value of NULL pointer is 0.

- Pointer can have three kinds of contents in it
  - The address of an object, which can be dereferenced.
  - A NULL pointer
  - Undefined value // If p is a pointer to integer, then – int *p;

Declaration and Initialization

Declaration : Data-type *name;

- int *p;
  - Compiler assumes that any address that it holds points to an integer type.
- p= ∑
  - Memory address of sum variable is stored into p.



Example code:

```
int *p;      // p can point to anything where integer is
                stored. int* is the type. Not just int.
int a = 100;
p=&a;
printf("a is %d and *p is %d", a,*p);
```

Pointer Arithmetic Operations

1. Add an int to a pointer
2. Subtract an int from a pointer
3. Difference of two pointers when they point to the same array.

Note: Integer is not same as pointer.

- Coding examples

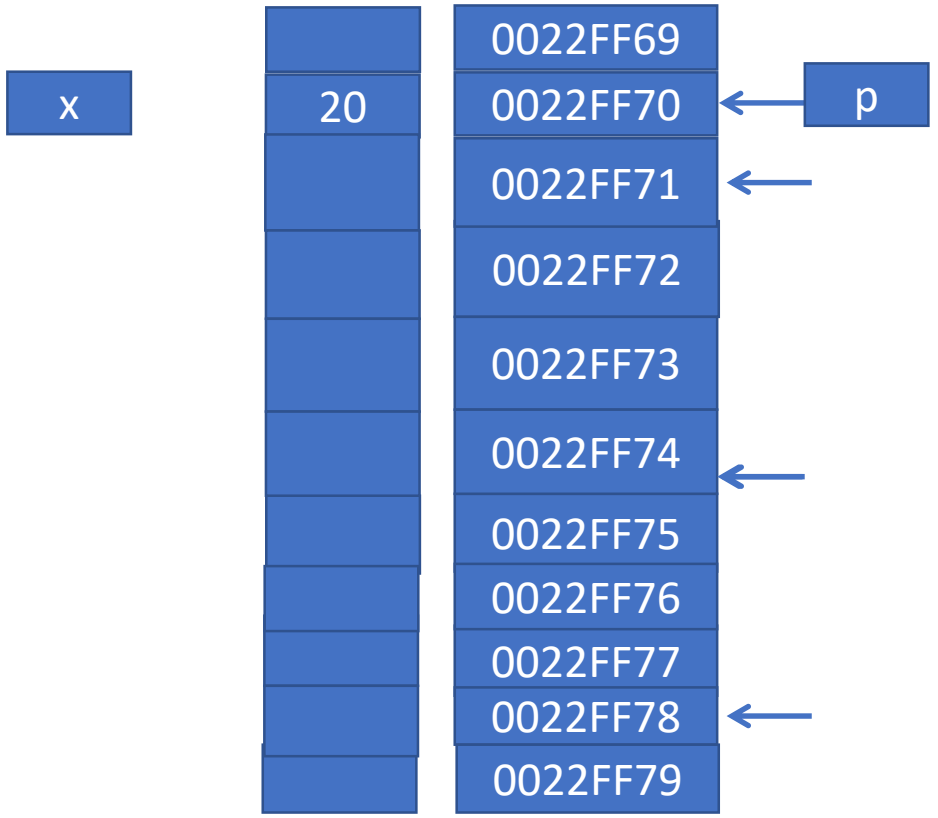
Pointer Arithmetic Operations continued..

Example Code:

```
int *p, x = 20;
p = &x;
printf("p     = %p\n", p);
printf("p+1 = %p\n", (int*)p+1);
printf("p+1 = %p\n", (char*)p+1);
printf("p+1 = %p\n", (float*)p+1);
printf("p+1 = %p\n", (double*)p+1);
```

Sample output:

p = 0022FF70
p+1 = 0022FF74
p+1 = 0022FF71
p+1 = 0022FF74
p+1 = 0022FF78



Array Traversal using Pointers

Consider int arr[] ={12,44,22,33,55}; int *p = arr; int i;

Coding examples to demo below points

- Array notation. Index operator can be applied on pointer.
- Pointer notation
- Using *p++
- Using *++p, Undefined behavior if you try to access outside bound
- Using (*p)++
- Using *p and then p++

Array and Pointer

- An array during compile time is an actual array but degenerates to a constant pointer during run time.

- Size of the array returns the number of bytes occupied by the array. But the size of pointer is always constant in that particular system.

```
int *p1; float *f1 ; char *c1;
printf("%d %d %d ",sizeof(p1),sizeof(f1),sizeof(c1)); // Same value for all
```

- An array is a constant pointer. It cannot point to anything in the world

Array and Pointer continued..

Example code:

```
int a[] = {22,11,44,5};
int *p = a;
a++;                    // Error : a is constant pointer
p++;                    // Fine
p[1] = 222;
a[1] = 222 ;            // Fine
```

- If variable i is used in loop for the traversal, a[i], *(a+i), p[i], *(p+i), i[a], i[p] are all same.

Array and Pointer continued..

Differences

1. the sizeof operator
   a. sizeof(array) returns the amount of memory used by all elements in array
   b. sizeof(pointer) only returns the amount of memory used by the pointer variable itself
2. the & operator
   a. &array is an alias for &array[0] and returns the address of the first element in array
   b. &pointer returns the address of pointer
3. string literal initialization of a character array
   a. char array[] = “abc” sets the first four elements in array to ‘a’, ‘b’, ‘c’, and ‘\0’
   b. char *pointer = “abc” sets pointer to the address of the “abc” string (which may be stored in read-only memory and thus unchangeable)
4. Pointer variable can be assigned a value whereas array variable cannot be.
5. Arithmetic operations on pointer variable is allowed. On array, not allowed.

THANK YOU

Prof. Sindhu R Pai

Department of Computer Science and Engineering

sindhurpai@pes.edu