

Cohort 18

Pointers, Arrays and Strings

@C18mentor



Prerequisite

- Write a basic C program
- Declare and initialize a variable
- Perform basic arithmetic operations using variables.

- To understand pointers, we need to understand how various data types and variables are stored in a computer memory

```
int main() {  
  
    int x, int y;  
    int sum;  
    sum = x +y;  
    return sum;  
}
```

Simple Illustration of Computer Memory Architecture

- When we talk about computer memory here in context of program execution e.g. 2 GB
- Let's say that these partitions/segments is each 1 byte of memory.
- In a typical memory architecture, each byte of the memory has an address. Let's say the first byte in this drawing is address 0 and address keeps on increasing as we go towards the top, i.e. go like 0,1,2,3...
- Let's say the first address here is 101, the next one would be 102, 103, 104 etc.

- When we declare a variable in our program, say 'x' of type integer, then when this program executes, the computer allocates some amount of memory corresponding to this particular variable.
- How much memory it allocates depends on the data type and on the compiler. eg, in modern compilers like the GCC, an integer is allocated 4 bytes of memory, char – 1 byte and float – 4 bytes. So as soon as the computer sees a declaration like these during program execution, it recognises that this is an int variable and needs to allocate 4 bytes of memory to it. In our diagram, it would start from, say, 102 to 105.
- A computer has an internal structure like a lookup table where it stores variable x of type integer and it is allocated at address 102 which is the starting address of the variable.

```
int x = 10;
```

Starting address
102

Address hexadecimal		
110		
109		
108	P = &102	1 byte char y
107		
106		
105	int x	4 bytes int x
104		
103		
102		
101	Char z;	1 byte
...		
0		

int x; 4 bytes
char y; 1 byte
float d; 4 bytes

Memory address x
=> 102

Lookup table Sample

Variable	Address
Int x	102
char y	108
Char z	

- If we declare another variable, 'y' of char type, it knows that it needs 1 byte of memory. So it looks for some free space, say in this case, it allocates the address 108, the byte 108 for 'y' and once again it keeps an entry for it in an internal structure, referred to as the lookup table. So if we say that `x = 5`; we change the value in x to be 5; In reality it is written in binary but for the sake of understanding we write it as 5. We can then have another statement which increments 'x' `x++`; When the computer sees this it goes to the address for x and changes the value to 6;

```
int x;
```

```
char y;
```

```
x = 5;
```

then if we modify the value for x, then the value of x changes to the new value

```
x++; // x = 6;
```


Introducing Pointers

- All this is great but can we know the value of address go a variable in our program? Or can we operate upon these memory addresses in our program?
- **We can do so in C using the concept of pointers.**
- **Pointers.** => Is a variable that store address of another variable.
- Let's say the we have a block of 4 bytes, at address 102 that stores an integer variable 'x'. We can have another variable, the type of which is pointer to integer, let's name the variable 'p'. This variable p can store the address of 'x', $p = 102$, and using the properties of 'p' or using some operators upon 'p', we can reach 'x'. 'p' also takes some memory, let's say, it is stored at location address 48 and it also takes 4 bytes of memory. We can also modify p to point to another integer, let's say, if we have another integer at an address 109 named y and has a value 10 and if we change the address in 'p' from 102 to 109, then p now points to y.

address		
110		
109		
108	num[1]	4 bytes int y
107		
106		
105		4 bytes int x
104	num[0]	
103		
...
48	p = &108	
...		
0		

int num[] = {5, 10, 23, 89, 90}

num[4] //

}
4 bytes int x
}

Hexadecimal Numbers


0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f

Array


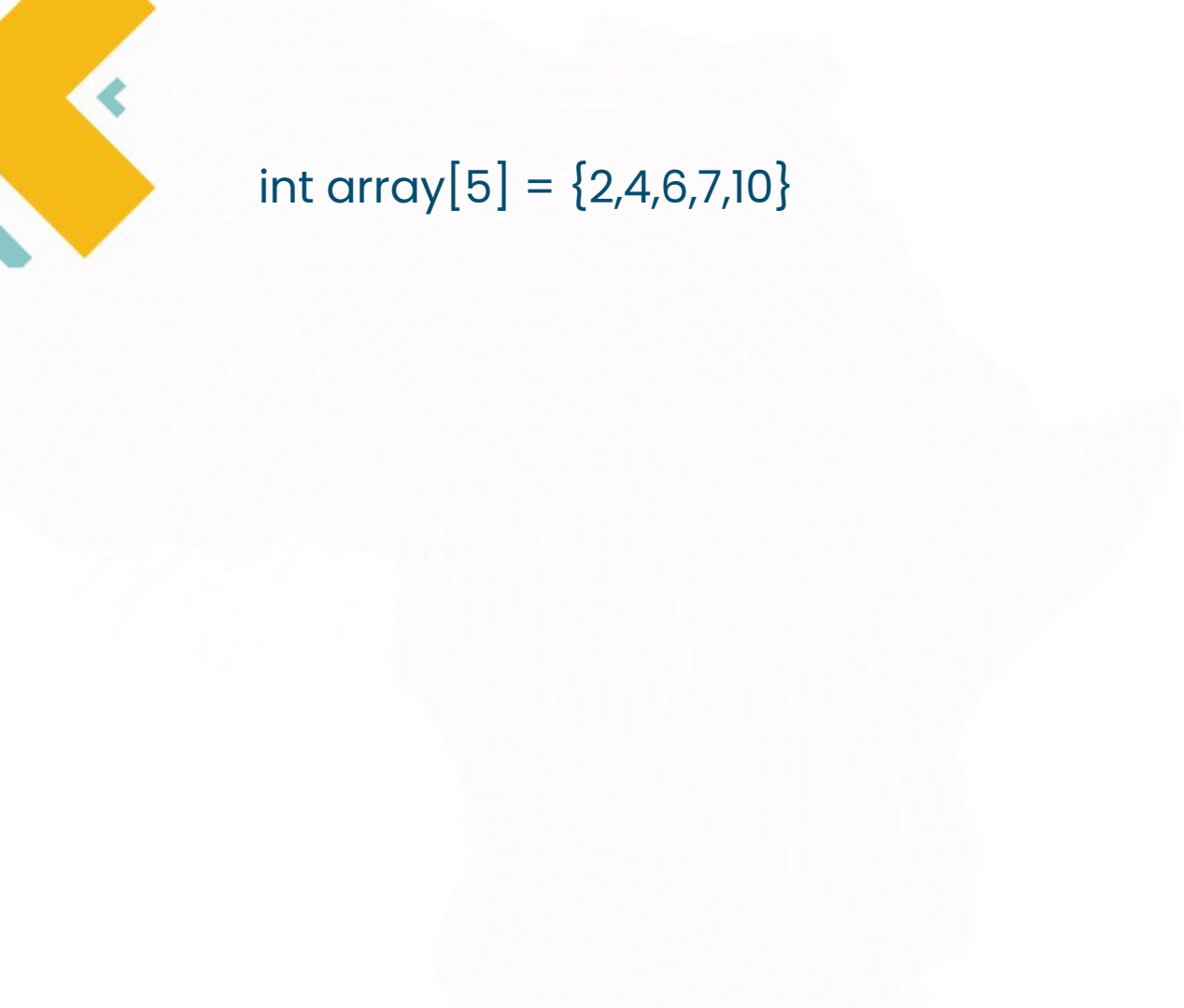
- Arrays in C are contiguous memory areas that hold a number of values of the same type.
- The computer will reserve a continuous space for 5 integers in memory.

[int , float, str]

In C => 1 data type int



```
int array[5] = {2,4,6,7,10}
```



address		
...		
119	arr[4] = 10	
115	arr[3] = 7	
111	arr[2] = 6	
107	arr[1] = 4	
103	arr[0] = 2	4-bytes
...
48		
...		
0		

```
int arr[5] =
{2,4,6,7,10}
```

```
&(arr[0]) =>
location => &arr
```

```
Strings => '\0'
```

Strings

- Since every string in C ends with a `'\0'` we do not need to know their size to use them. By knowing the address of the first character of strings (with a pointer to a char), C functions can easily print them using a loop, one character at a time, until they hit the character `'\0'`

```
char name[] = "Firdaus Hassan"
```

```
char two = "a"
```

```
char three = "t"
```

```
char cat_string[] = "cat";
```

**See you at
the next
session!**

