ALX LESSON 0x05 C Pointers, arrays and strings

C - Programming

TABLE OF CONTENTS

01 Overview topics

02 Learning Objectives

03 Quiz questions

04 hands on lab practice

01

OVERVIEW topics

Topics

What are pointers and how to use them

What are arrays and how to use them

What are the differences between pointers and arrays

How to use strings and how to manipulate them

Scope of variables

Slides On Telegram

https://t.me/alx_2023

C Programming

Topics

02

Learning Objectives

Data Types

Data Type	Size (bytes)	Range of Values
`char`	1	-128 to 127 or 0 to 255 (unsigned)
`short`	2	-32,768 to 32,767 or 0 to 65,535 (unsigned)
`int`	4	-2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 (unsigned)
`long`	4	-2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 (unsigned)
`long long`	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 or 0 to 18,446,744,073,709,551,615 (unsigned)
`float`	4	approximately 1.2E-38 to 3.4E+38
`double`	8	approximately 2.2E-308 to 1.8E+308
`long double`	12 or more	depends on implementation

Data Types

Data Type	Size (bytes)	Range of Values	Format Specifier	Signed/Unsigned
`char`	1	-128 to 127 or 0 to 255 (unsigned)	`%c`	Signed/Unsigned
`short`	2	-32,768 to 32,767 or 0 to 65,535 (unsigned)	`%hd`	Signed
`int`	4	-2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 (unsigned)	`%d` or `%i`	Signed
`long`	4	-2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 (unsigned)	`%ld`	Signed
`long long`	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 or 0 to 18,446,744,073,709,551,615 (unsigned)	`%lld`	Signed
`float`	4	approximately 1.2E-38 to 3.4E+38	`%f`	Signed
`double`	8	approximately 2.2E-308 to 1.8E+308	`%lf`	Signed
`long double`	12 or more	depends on implementation	`%Lf`	Signed

Data Types (byte = 8 bit)

Data Type	Format Specifier	Minimal Range	Typical Bit Size
unsigned char	%c	0 to 255	8
char	%c	-127 to 127	8
signed char	%c	-127 to 127	8
int	%d, %i	-32,767 to 32,767	16 or 32
unsigned int	%u	0 to 65,535	16 or 32
signed int	%d, %i	Same as int	Same as int 16 or 32
short int	%hd	-32,767 to 32,767	16
unsigned short int	%hu	0 to 65,535	16
signed short int	%hd	Same as short int	16

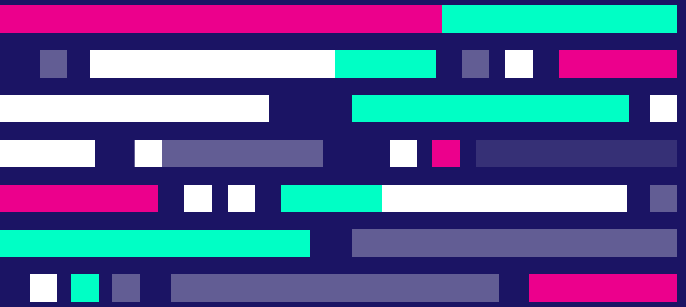
Data Types (byte = 8 bit)

long int	%ld, %li	-2,147,483,647 to 2,147,483,647	32
long long int	%lld, %lli	-(263 – 1) to 263 – 1 (It will be added by the C99 standard)	64
signed long int	%ld, %li	Same as long int	32
unsigned long int	%lu	0 to 4,294,967,295	32
unsigned long long int	%llu	264 – 1 (It will be added by the C99 standard)	64
float	%f	1E-37 to 1E+37 along with six digits of the precisions here	32
double	%lf	1E-37 to 1E+37 along with six digits of the precisions here	64
long double	%Lf	1E-37 to 1E+37 along with six digits of the precisions here	80

Specifiers

SPECIFIER	USED FOR
%c	a single character
%s	a string
%hi	short (signed)
%hu	short (unsigned)
%Lf	long double
%n	prints nothing
%d	a decimal integer (assumes base 10)
%i	a decimal integer (detects the base automatically)
%o	an octal (base 8) integer
%x	a hexadecimal (base 16) integer
%p	an address (or pointer)
%f	a floating point number for floats
%u	int unsigned decimal
%e	a floating point number in scientific notation
%E	a floating point number in scientific notation
%%	the % symbol

What is a Pointer in C?



Pointers in C are used to store the address of variables or a memory location. This variable can be of any data type i.e, int, char, function, array, or any other pointer. Pointers are one of the core concepts of C programming language that provides low-level memory access and facilitates dynamic memory allocation.

What is a Pointer in C?

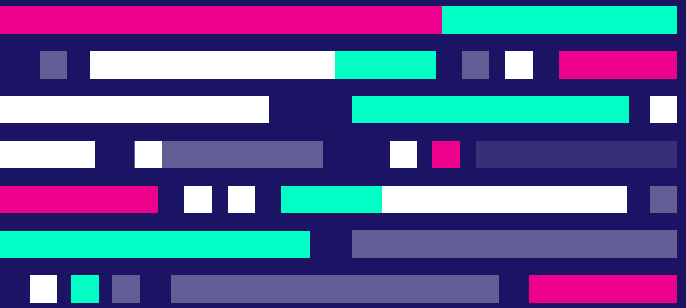
A pointer is a derived data type in C that can store the address of other variables or a memory. We can access and manipulate the data stored in that memory location using pointers.

Syntax of C Pointers

```
datatype * pointer_name;
```

The above syntax is the generic syntax of C pointers. The actual syntax depends on the type of data the pointer is pointing to.

How to Use Pointers?



1. Addressof Operator

The addressof operator (&) is a unary operator that returns the address of its operand. Its operand can be a variable, function, array, structure, etc.

Syntax of Address of Operator

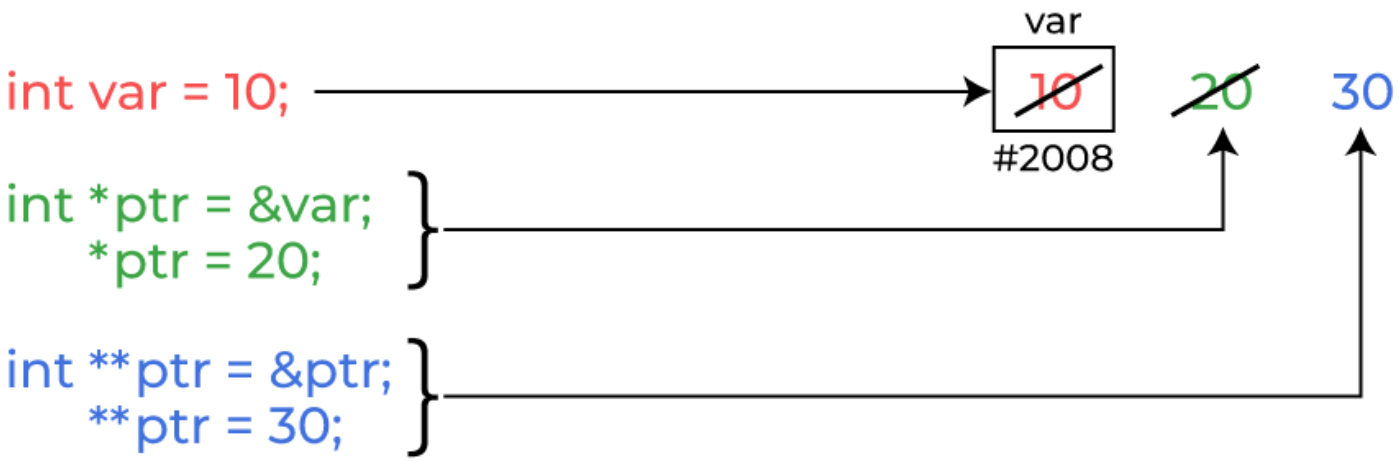
```
&variable_name;
```

2. Dereferencing Operator

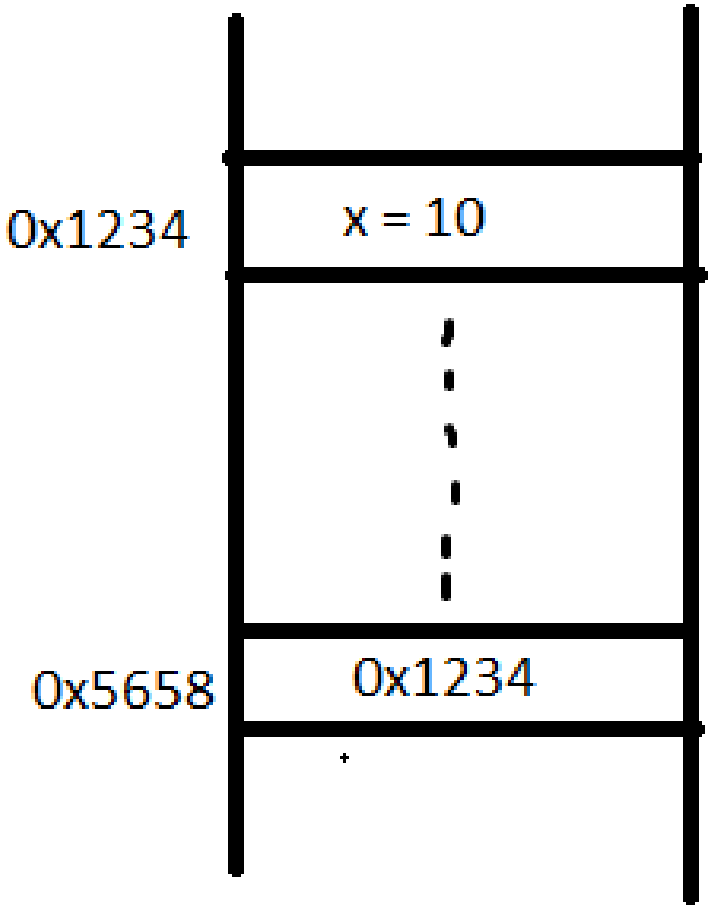
The dereference operator (*), also known as the indirection operator is a unary operator. It is used in pointer declaration and dereferencing.

How pointers works in C?

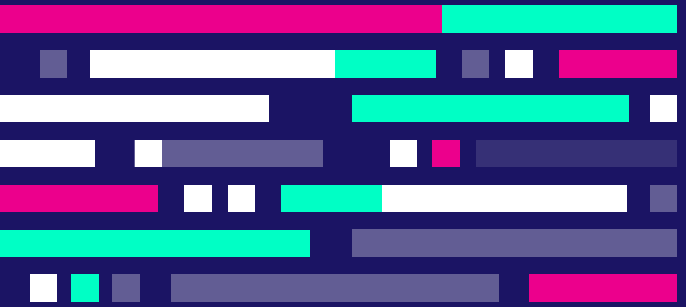
How pointer works in C



How pointers works in C?



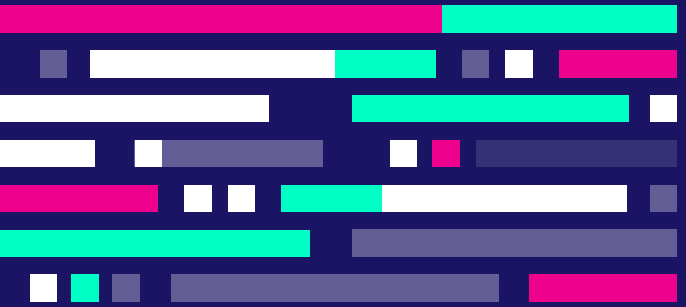
C Pointer Declaration



In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the * dereference operator before its name.

```
data_type * pointer_name;
```

C Pointer Initialization



Method 1: C Pointer Definition

```
datatype * pointer_name = address;
```

The above method is called Pointer Definition as the pointer is declared and initialized at the same time.

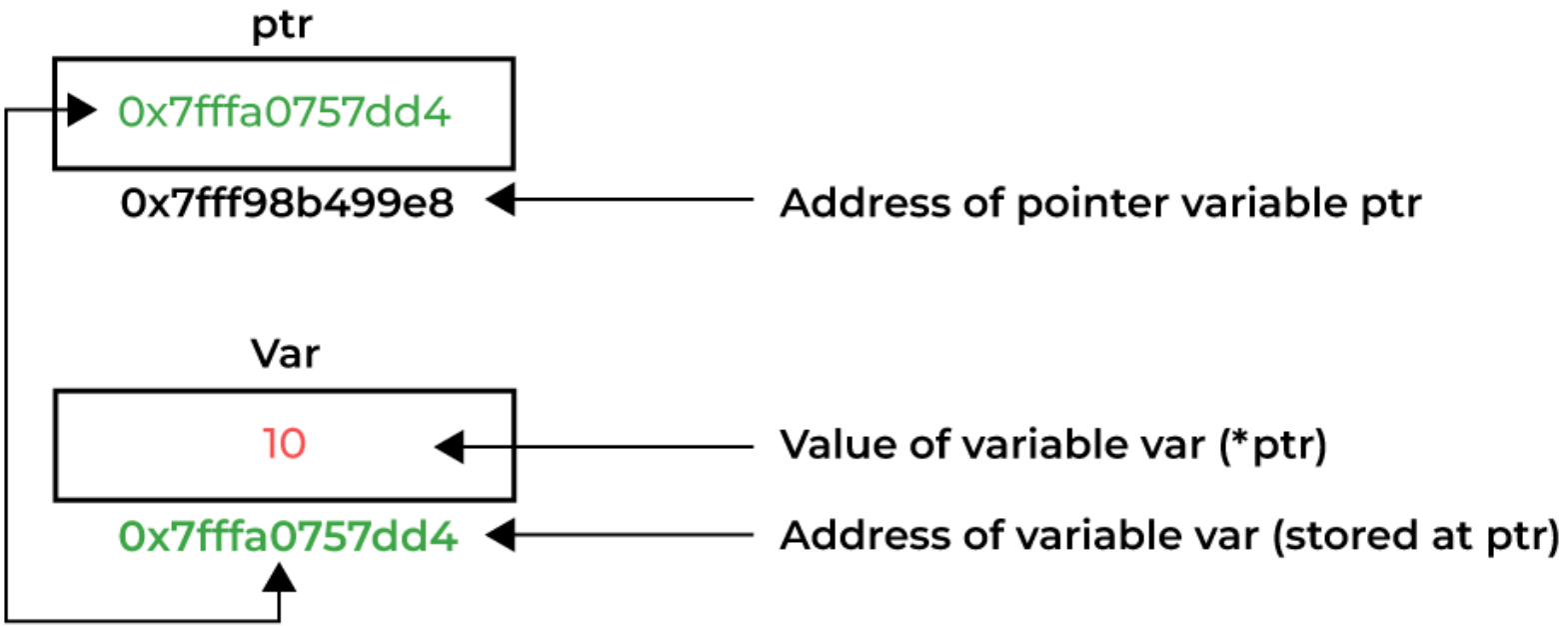
Method 2: Initialization After Declaration

The second method of pointer initialization in C the assigning some address after the declaration.

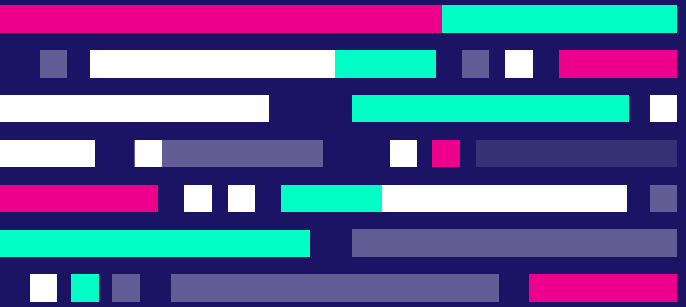
```
datatype * pointer_name;
pointer_name = addresss;
```

Dereferencing a C Pointer

Dereferencing is the process of accessing the value stored in the memory address specified in the pointer. We use dereferencing operator for that purpose.



C program to illustrate Pointers



```c
#include <stdio.h>

void func1()
{
    int var = 20;

    // declare pointer variable
    int* ptr;

    // note that data type of ptr and var must be same
    ptr = &var;

    // assign the address of a variable to a pointer
    printf("Value at ptr = %p \n", ptr);
    printf("Value at var = %d \n", var);
    printf("Value at *ptr = %d \n", *ptr);
}
int main()
{
    func1();
    return 0;
}
```

Types of Pointers

1. Integer Pointers

As the name suggests, these are the pointers that point to the integer values.

Syntax of Integer Pointers

```
int *pointer_name;
```

These pointers are pronounced as Pointer to Integer.

Types of Pointers

2. Array Pointer

Pointers and Array are closely related to each other. Even the array name is the pointer to its first element. They are also known as Pointer to Arrays. We can create a pointer to an array using the given syntax.

Syntax of Array Pointers

```
char *pointer_name = &array_name;
```

Types of Pointers

4. Function Pointers

Function pointers point to the functions. They are different from the rest of the pointers in the sense that instead of pointing to the data, they point to the code. Let's consider a function prototype - int func (int, char), the function pointer for this function will be

Syntax of Function Pointer

```
int (*pointer_name)(int, int);
```

Keep in mind that the syntax of the function pointers changes according to the function prototype.

Types of Pointers

5. Double Pointers

In C language, we can define a pointer that stores the memory address of another pointer. Such pointers are called double-pointers or pointers-to-pointer. Instead of pointing to a data value, they point to another pointer.

Syntax of Double Pointer in C

```
datatype ** pointer_name;
```

Dereferencing in Double Pointer

```
*pointer_name; // get the address stored in the inner level pointer
**pointer_name; // get the value pointed by inner level pointer
```

Note: In C, we can create multi-level pointers with any number of levels such as – ***ptr3, ****ptr4, ******ptr5 and so on.

Types of Pointers

6. NULL Pointer

The Null Pointers are those pointers that do not point to any memory location. They can be created by assigning a NULL value to the pointer. A pointer of any type can be assigned the NULL value.

Syntax of NULL Pointer in C

```
data_type *pointer_name = NULL;
```

or

```
pointer_name = NULL
```

It is said to be good practice to assign NULL to the pointers currently not in use.

Types of Pointers

7. Void Pointer

The Void pointers in C are the pointers of type void. It means that they do not have any associated data type. They are also called generic pointers as they can point to any type and can be typecasted to any type.

Syntax of Void Pointer

```
void * pointer_name;
```

One of the main properties of void pointers is that they cannot be dereferenced.

Types of Pointers

8. Wild Pointers

The Wild Pointers are pointers that have not been initialized with something yet. These types of C-pointers can cause problems in our programs and can eventually cause them to crash.

Example of Wild Pointers

```
int *ptr;
char *str;
```

Types of Pointers

9. Constant Pointers

In constant pointers, the memory address stored inside the pointer is constant and cannot be modified once it is defined. It will always point to the same memory address.

Syntax of Constant Pointer

```
const data_type * pointer_name;
```

Types of Pointers

10. Pointer to Constant

The pointers pointing to a constant value that cannot be modified are called pointers to a constant. Here we can only access the data pointed by the pointer, but cannot modify it. Although, we can change the address stored in the pointer to constant.

Syntax to Pointer to Constant

```
data_type * const pointer_name;
```

Size of Pointers in C

The size of the pointers in C is equal for every pointer type. The size of the pointer does not depend on the type it is pointing to. It only depends on the operating system and CPU architecture. The size of pointers in C is

8 bytes for a 64-bit System
4 bytes for a 32-bit System

C Program to find the size of different pointers types

```c
#include <stdio.h>

// dummy structure
struct str {
};

// dummy function
void func(int a, int b){};

int main()
{
    // dummy variables definitions
    int a = 10;
    char c = 'G';
    struct str x;

    // pointer definitions of different types
    int* ptr_int = &a;
    char* ptr_char = &c;
    struct str* ptr_str = &x;
    void (*ptr_func)(int, int) = &func;
    void* ptr_vn = NULL;

    // printing sizes
    printf("Size of Integer Pointer  \t:\t%ld bytes\n",
           sizeof(ptr_int));
    printf("Size of Character Pointer\t:\t%ld bytes\n",
           sizeof(ptr_char));
    printf("Size of Structure Pointer\t:\t%ld bytes\n",
           sizeof(ptr_str));
    printf("Size of Function Pointer\t:\t%ld bytes\n",
           sizeof(ptr_func));
    printf("Size of NULL Void Pointer\t:\t%ld bytes",
           sizeof(ptr_vn));

    return 0;
}
```

Pointer Arithmetic

Only a limited set of operations can be performed on pointers. The Pointer Arithmetic refers to the legal or valid operations that can be performed on a pointer. It is slightly different from the ones that we generally use for mathematical calculations. The operations are:

- Increment in a Pointer
- Decrement in a Pointer
- Addition of integer to a pointer
- Subtraction of integer to a pointer
- Subtracting two pointers of the same type
- Comparison of pointers of the same type.
- Assignment of pointers of the same type.

Pointer Arithmetic

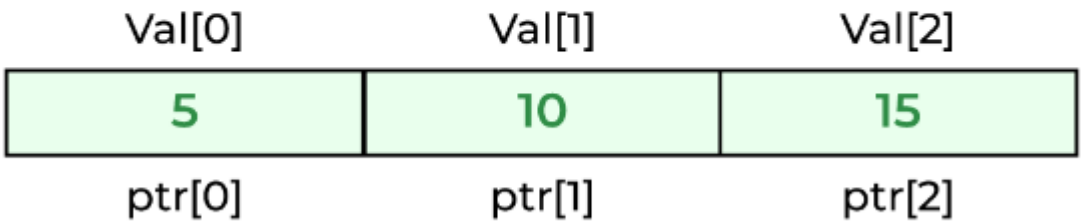
```c
#include <stdio.h>

int main()
{
    // Declare an array
    int v[3] = { 10, 100, 200 };

    // Declare pointer variable
    int* ptr;

    // Assign the address of v[0] to ptr
    ptr = v;
    //or
    //ptr = &v[0];
    for (int i = 0; i < 3; i++) {

        // print value at address which is stored in ptr
        printf("Value of *ptr = %d\n", *ptr);

        // print value of ptr
        printf("Value of ptr = %p\n\n", ptr);

        // Increment pointer ptr by 1
        ptr++;
    }
    return 0;
}
```

C Pointers and Arrays Relation

In C programming language, pointers and arrays are closely related. An array name acts like a pointer constant. The value of this pointer constant is the address of the first element. For example, if we have an array named val then `val` and `&val[0]` can be used interchangeably.

If we assign this value to a non-constant pointer to the array, then we can access the elements of the array using this pointer.

Example 1: Accessing Array Elements using Pointer with Array Subscript

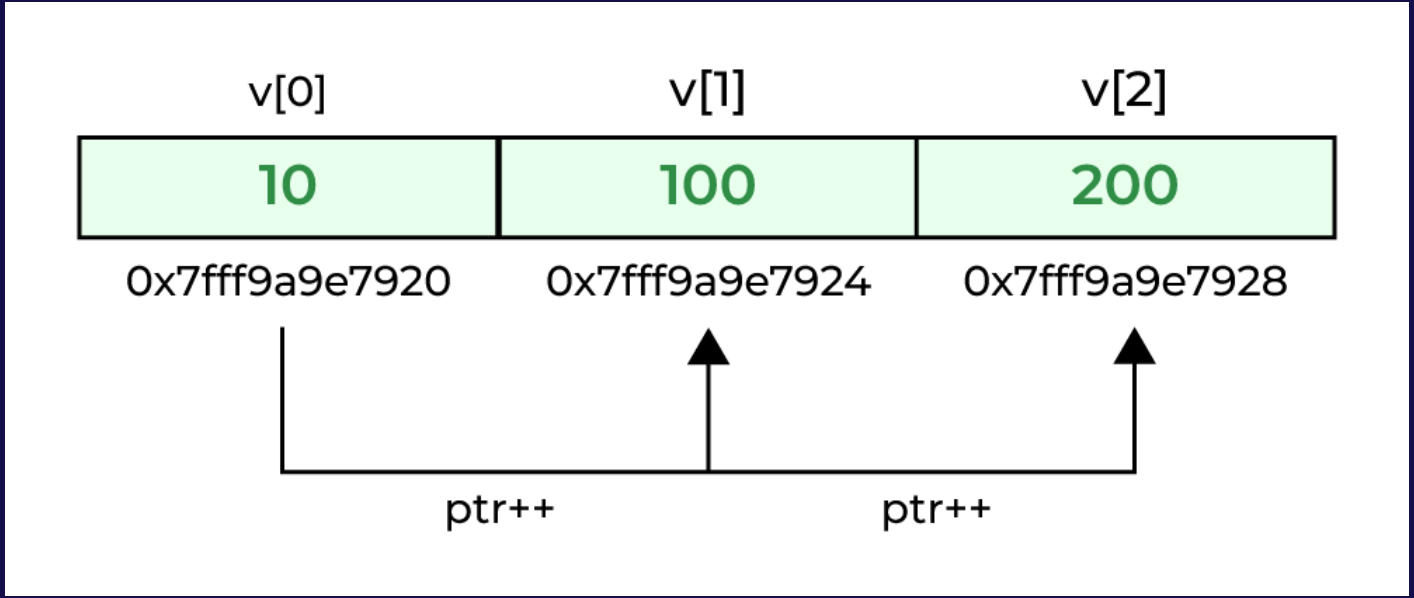


C Pointers and Arrays Relation

```c
#include <stdio.h>

void func2()
{
    // Declare an array
    int val[3] = { 5, 10, 15 };

    // Declare pointer variable
    int* ptr;

    // Assign address of val[0] to ptr.
    // We can use ptr=&val[0];(both are same)
    ptr = val;

    printf("Elements of the array are: ");

    printf("%d, %d, %d", ptr[0], ptr[1], ptr[2]);

    return;
}
int main()
{
    func2();
    return 0;
}
```

C Pointers and Arrays Relation

Not only that, as the array elements are stored continuously, we can pointer arithmetic operations such as increment, decrement, addition, and subtraction of integers on pointer to move between array elements.



C Pointers and Arrays Relation

```c
#include <stdio.h>

int main()
{

    // defining array
    int arr[5] = { 1, 2, 3, 4, 5 };

    // defining the pointer to array
    int* ptr_arr = &arr[0];

    // traversing array using pointer arithmetic
    for (int i = 0; i < 5; i++) {
        printf("%d ", *ptr_arr++);
    }
    return 0;
}
```

What are the differences between an array and a pointer?

Pointer	Array
A pointer is a derived data type that can store the address of other variables.	An array is a homogeneous collection of items of any type such as int, char, etc.
Pointers are allocated at run time.	Arrays are allocated at runtime.
The pointer is a single variable.	An array is a collection of variables of the same type.
Dynamic in Nature	Static in Nature.

What is the ASCII character set

```
cook@pop-os:~$ ascii -d
   0 NUL    16 DLE    32      48 0    64 @    80 P    96 `   112 p
   1 SOH    17 DC1    33 !    49 1    65 A    81 Q    97 a   113 q
   2 STX    18 DC2    34 "    50 2    66 B    82 R    98 b   114 r
   3 ETX    19 DC3    35 #    51 3    67 C    83 S    99 c   115 s
   4 EOT    20 DC4    36 $    52 4    68 D    84 T   100 d   116 t
   5 ENQ    21 NAK    37 %    53 5    69 E    85 U   101 e   117 u
   6 ACK    22 SYN    38 &    54 6    70 F    86 V   102 f   118 v
   7 BEL    23 ETB    39 '    55 7    71 G    87 W   103 g   119 w
   8 BS     24 CAN    40 (    56 8    72 H    88 X   104 h   120 x
   9 HT     25 EM     41 )    57 9    73 I    89 Y   105 i   121 y
  10 LF     26 SUB    42 *    58 :    74 J    90 Z   106 j   122 z
  11 VT     27 ESC    43 +    59 ;    75 K    91 [   107 k   123 {
  12 FF     28 FS     44 ,    60 <    76 L    92 \   108 l   124 |
  13 CR     29 GS     45 -    61 =    77 M    93 ]   109 m   125 }
  14 SO     30 RS     46 .    62 >    78 N    94 ^   110 n   126 ~
  15 SI     31 US     47 /    63 ?    79 O    95 _   111 o   127 DEL
```

Hexadecimal Numbering System

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

04

Hands on lab Practice


Have a Question
Leave a Comment!

Share

To let the others know more

Subscribe

To stay updated with latest videos

Thanks