**Pointers**

# Pointers store address of variables or a memory location.

**int \*ptr;**

#include <stdio.h>

int main**()**

**{**

int x**;**

// Prints address of x - 0x7ffff3014144

printf**(**"%p"**,** **&**x**);**

**return** 0**;**

**}**

#include <stdio.h>

int main**()**

**{**

int x **=** 10**;**

// 1) Since there is \* in declaration, ptr

// becomes a pointer varaible (a variable

// that stores address of another variable)

// 2) Since there is int before \*, ptr is

// pointer to an integer type variable

int **\***ptr**;**

// & operator before x is used to get address

// of x. The address of x is assigned to ptr.

ptr **=** **&**x**;**

**return** 0**;**

**}**

**// C program to demonstrate use of \* for pointers in C**

#include <stdio.h>

int main**()**

**{**

// A normal integer variable

int Var **=** 10**;**

// A pointer variable that holds address of var.

int **\***ptr **=** **&**Var**;**

// This line prints value at address stored in ptr.

// Value stored is value of variable "var"

printf**(**"Value of Var = %d\n"**,** **\***ptr**);**

// The output of this line may be different in different

// runs even on same machine.

printf**(**"Address of Var = %p\n"**,** ptr**);**

// We can also use ptr as lvalue (Left hand

// side of assignment)

**\***ptr **=** 20**;** // Value at address is now 20

// This prints 20

printf**(**"After doing \*ptr = 20, \*ptr is %d\n"**,** **\***ptr**);**

**return** 0**;**

**}**

Value of Var **=** 10

Address of Var **=** 0x7ffd61ae1ffc

After doing **\***ptr **=** 20**,** **\***ptr is 20

# Pointers arithmetic

<https://www.codeproject.com/Articles/82880/C-Pointer-Tricks>

void main**()**

**{**

int arr**[]** **=** **{**1**,** 2**,** 3**};**

int **\***ptr**;**

ptr **=** arr**;**

printf**(**"arr[0] = %d, arr[1] = %d, arr[2] = %d, ptr = %p, \*ptr = %d\n"**,**

arr**[**0**],** arr**[**1**],** arr**[**2**],** ptr**,** **\***ptr**);**

**\***ptr**++** **=** **-**1**;**

printf**(**"arr[0] = %d, arr[1] = %d, arr[2] = %d, ptr = %p, \*ptr = %d\n"**,**

arr**[**0**],** arr**[**1**],** arr**[**2**],** ptr**,** **\***ptr**);**

**\*++**ptr **=** **-**2**;**

printf**(**"arr[0] = %d, arr[1] = %d, arr[2] = %d, ptr = %p, \*ptr = %d\n"**,**

arr**[**0**],** arr**[**1**],** arr**[**2**],** ptr**,** **\***ptr**);**

**(\***ptr**)++;**

printf**(**"arr[0] = %d, arr[1] = %d, arr[2] = %d, ptr = %p, \*ptr = %d\n"**,**

arr**[**0**],** arr**[**1**],** arr**[**2**],** ptr**,** **\***ptr**);**

**}**

arr[0] = 1, arr[1] = 2, arr[2] = 3, ptr = 0043FE44, \*ptr = 1

arr[0] = -1, arr[1] = 2, arr[2] = 3, ptr = 0043FE48, \*ptr = 2

arr[0] = -1, arr[1] = 2, arr[2] = -2, ptr = 0043FE4C, \*ptr = -2

arr[0] = -1, arr[1] = 2, arr[2] = -1, ptr = 0043FE4C, \*ptr = -1

void main**()**

**{**

int arr**[]** **=** **{**1**,** 2**,** 3**};**

**\***arr **=** 5**;**

printf**(**"arr[0] = %d, arr[1] = %d, arr[2] = %d\n"**,**

arr**[**0**],** arr**[**1**],** arr**[**2**]);**

**\*(**arr **+** 1**)** **=** 10**;**

printf**(**"arr[0] = %d, arr[1] = %d, arr[2] = %d\n"**,**

arr**[**0**],** arr**[**1**],** arr**[**2**]);**

2**[**arr**]** **=** 15**;**

printf**(**"0[arr] = %d, 1[arr] = %d, 2[arr] = %d\n"**,**

0**[**arr**],** 1**[**arr**],** 2**[**arr**]);**

**}**

arr[0] = 5, arr[1] = 2, arr[2] = 3

arr[0] = 5, arr[1] = 10, arr[2] = 3

0[arr] = 5, 1[arr] = 10, 2[arr] = 15

void main**()**

**{**

char str1**[]** **=** **{** 'A'**,** 'B'**,** 'C'**,** 'D'**,** 'E' **};**

char str2**[]** **=** "ABCDE"**;**

char **\***str3 **=** "ABCDE"**;**

char **\***cPtr **=** str1**;**

short sArr**[]** **=** **{**1**,** 2**,** 3**,** 4**,** 5**};**

short sArr2D**[][**5**]** **=** **{** **{**1**,** 2**,** 3**,** 4**,** 5**},**

**{**6**,** 7**,** 8**,** 9**,** 10**}** **};**

short **\***sPtr1 **=** sArr**;**

short **(\***sPtr2**)[**5**]** **=** sArr2D**;**

short **\***sPtr3**[**5**];**

printf**(**"sizeof(str1) = %u\n"**,** **sizeof(**str1**));**

printf**(**"sizeof(str2) = %u\n"**,** **sizeof(**str2**));**

printf**(**"sizeof(str3) = %u\n"**,** **sizeof(**str3**));**

printf**(**"sizeof(cPtr) = %u\n"**,** **sizeof(**cPtr**));**

printf**(**"\n"**);**

printf**(**"sizeof(sArr) = %u\n"**,** **sizeof(**sArr**));**

printf**(**"sizeof(sPtr1) = %u\n"**,** **sizeof(**sPtr1**));**

printf**(**"sizeof(sArr2D) = %u\n"**,** **sizeof(**sArr2D**));**

printf**(**"sizeof(sPtr2) = %u\n"**,** **sizeof(**sPtr2**));**

printf**(**"sizeof(\*sPtr2) = %u\n"**,** **sizeof(\***sPtr2**));**

printf**(**"sizeof(sPtr3) = %u\n"**,** **sizeof(**sPtr3**));**

printf**(**"\n"**);**

printf**(**"sArr2D[1][2] = %d\n"**,** sArr2D**[**1**][**2**]);**

printf**(**"sPtr2[0][0] = %d, sPtr2[1][2] = %d\n"**,** sPtr2**[**0**][**0**],** sPtr2**[**1**][**2**]);**

printf**(**"\*(\* (sArr2D + 1) + 2) = %d\n"**,** **\*(\*** **(**sArr2D **+** 1**)** **+** 2**));**

printf**(**"\*(\*(sArr2D) + 1\*5 + 2) = %d\n\n"**,** **\*(\*(**sArr2D**)** **+** 1**\***5 **+** 2**));**

**}**

sizeof(str1) = 5

sizeof(str2) = 6

sizeof(str3) = 4

sizeof(cPtr) = 4

sizeof(sArr) = 10

sizeof(sPtr1) = 4

sizeof(sArr2D) = 20

sizeof(sPtr2) = 4

sizeof(\*sPtr2) = 10

sizeof(sPtr3) = 20

sArr2D[1][2] = 8

sPtr2[0][0] = 1, sPtr2[1][2] = 8

\*(\* (sArr2D + 1) + 2) = 8

\*(\*(sArr2D) + 1\*5 + 2) = 8

* sArr2D[1][2] simply tries to access the third element of the second array in the 2-dimensional array sArr2D. Think of sArr2D as an array of 2 arrays of 5 int values. So sArr2D[1][2] = 8.
* sPtr2 is a pointer to the 2-dimensional array sArr2D and thus can be used in the same way as variable sArr2D. So sPtr2[1][2] is the same as sArr2D[1][2].
* \* (sArr2D + 1) is like sArr2D[1] which points us to the first element in the 2nd array sArr2D[1][0]. \*(\* (sArr2D + 1) + 2) is like \*(sArr2D[1] + 2), which is like \*(&sArr2D[1][0] + 2), which is like \*(&sArr2D[1][1]), which is sArr2D[1][2].
* \*(sArr2D) + 1 \* (column count) is the same as sArr2D[1]. So \*(\*(sArr2D) + 1\*5 + 2) is \*(&sArr2D[1][0] + 2), which is sArr2D[1][2].

# # Some Facts

void main**()**

**{**

int **\***ptr **=** 10**;** /\*\* Pointer to a const (pointing to address location 10) \*/

// \*ptr = 19 ; /\*\* Can not change the value pointing to i.e 10 ; Will cause error\*/

printf**(**"Value of ptr is %d\n"**,** ptr**);** /\*\* Pointer can hold address of a variable(data) as well as value; \*/

printf**(**"Value of ptr is %d\n"**,** **++**ptr**);** /\*\* 14 \*/

// printf("Value of \*ptr %d\n", \*ptr); /\*\* This will throw error, base address 10 is not accesible\*/

**return** 0**;**

**}**

# Difference between ++\*p, \*p++ and \*++p.

// PROGRAM 1

#include <stdio.h>

int main**(**void**)**

**{**

int arr**[]** **=** **{**10**,** 20**};**

int **\***p **=** arr**;**

**++\***p**;**

printf**(**"arr[0] = %d, arr[1] = %d, \*p = %d"**,** arr**[**0**],** arr**[**1**],** **\***p**);**

**return** 0**;**

**}**

// arr[0] = 11, arr[1] = 20, \*p = 11

// PROGRAM 2

#include <stdio.h>

int main**(**void**)**

**{**

int arr**[]** **=** **{**10**,** 20**};**

int **\***p **=** arr**;**

**\***p**++;**

printf**(**"arr[0] = %d, arr[1] = %d, \*p = %d"**,** arr**[**0**],** arr**[**1**],** **\***p**);**

**return** 0**;**

**}**

// arr[0] = 10, arr[1] = 20, \*p = 20

// PROGRAM 3

#include <stdio.h>

int main**(**void**)**

**{**

int arr**[]** **=** **{**10**,** 20**};**

int **\***p **=** arr**;**

**\*++**p**;**

printf**(**"arr[0] = %d, arr[1] = %d, \*p = %d"**,** arr**[**0**],** arr**[**1**],** **\***p**);**

**return** 0**;**

**}**

// arr[0] = 10, arr[1] = 20, \*p = 20

**NOTE:**

**++\***p **=** **++(\***p**)**

**\***p**++** **=** **\*(**p**++)**

**\*++**p **=** **\*(++**p**)**

***1) Precedence of prefix ++ and \* is same. Associativity of both is right to left.***

***2) Precedence of postfix ++ is higher than both \* and prefix ++. Associativity of postfix ++ is left to right.***

# void \*

void \* type pointers cannot be de-referenced. We must type cast them before de-referencing.

# Pointers to Strings 1

#include <stdio.h>

int main**()**

**{**

char **\***cities**[]** **=** **{**"Iran"**,** "Iraq"**};**

int i**;**

**for(**i **=** 0**;** i **<** 2**;** i**++)**

printf**(**"%s\n"**,** cities**[**i**]);**

**return** 0**;**

**}**

Iran

Iraq

Note:

In the above pointer to string program, we declared a pointer array of character datatypes and then few strings like "Iran", "Iraq" where initialized to the pointer array (\*cities[]). *Note that we have not declared the size of the array as it is of character pointer type.* Coming to the explanation, cities[] is an array which has its own address and it holds the address of first element (I (Iran) ) in it as a value. This address is then executed by the pointer, i.e pointer start reading the value from the address stored in the array cities[0] and ends with '\0' by default. Next cities[1] holds the address of (I (Iraq).This address is then executed by the pointer, i.e pointer start reading the value from the address stored in the array cities[1] and ends with '\0' by default. As a result Iran and Iraq is outputted.

#include <stdio.h>

#include <string.h>

void function**(**char**\*\*);**

int main**()**

**{**

char **\***str **=** "Pointer-to-string"**;**

int i**,** j **=** strlen**(**str**);**

**for(**i **=** 0**;** i **<** j**;** i**++)**

printf**(**"%c"**,** **\***str**++);**

**return** 0**;**

**}**

**Pointer-to-string**

\*str is a char pointer variable which is initialized by a string "Pointer-to-String". Then strlen() is used to find the length of the string to do iteration using for loop is done to print the complete characters store with the variable name \*str.

char ch\_arr**[**3**][**10**]** **=** **{**

**{**'s'**,** 'p'**,** 'i'**,** 'k'**,** 'e'**,** '\0'**},**

**{**'t'**,** 'o'**,** 'm'**,**'\0'**},**

**{**'j'**,** 'e'**,** 'r'**,** 'r'**,** 'y'**,**'\0'**}**

**};**

It is important to end each 1-D array by the null character, otherwise, it will be just an array of characters. We can’t use them as strings.

char ch\_arr**[**3**][**10**]** **=** **{**

"spike"**,**

"tom"**,**

"jerry"

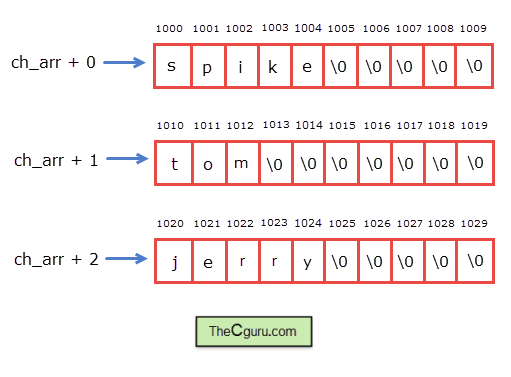
**};**

above statement it allocates 30 bytes (3\*10) of memory

The ch\_arr is a pointer to an array of 10 characters

Therefore, if ch\_arr points to address 1000 then ch\_arr + 1 will point to address 1010.

ch\_arr + 0 points to the 0th string or 0th 1-D array.  
ch\_arr + 1 points to the 1st string or 1st 1-D array.  
ch\_arr + 2 points to the 2nd string or 2nd 1-D array.



\*(ch\_arr + 0) + 0 points to the 0th character of 0th 1-D array (i.e s)  
\*(ch\_arr + 0) + 1 points to the 1st character of 0th 1-D array (i.e p)  
\*(ch\_arr + 1) + 2 points to the 2nd character of 1st 1-D array  (i.e m)

Note that the base type of \*(ch\_arr + i) + j is a pointer to char or (char\*), while the base type of ch\_arr + i is array of 10 characters or int(\*)[10].

To get the element at jth position of ith 1-D array just dereference the whole expression\*(ch\_arr + i) + j.

# FUNCTION POINTER (pointers to function)

In C**,** like normal data pointers **(**int **\*,** char **\*,** etc**),** we can have pointers to functions**.** Following is a simple example that shows declaration and function call using function pointer**.**

#include <stdio.h>

// A normal function with an int parameter

// and void return type

void fun**(**int a**)**

**{**

printf**(**"Value of a is %d\n"**,** a**);**

**}**

int main**()**

**{**

// fun\_ptr is a pointer to function fun()

void **(\***fun\_ptr**)(**int**)** **=** **&**fun**;**

/\* The above line is equivalent of following two

void (\*fun\_ptr)(int);

fun\_ptr = &fun;

\*/

// Invoking fun() using fun\_ptr

**(\***fun\_ptr**)(**10**);**

**return** 0**;**

**}**

Output**:**

Value of a is 10

Why **do** we need an extra bracket around function pointers like fun\_ptr in above example**?**

If we remove bracket**,** then the expression “void **(\***fun\_ptr**)(**int**)**” becomes “void **\***fun\_ptr**(**int**)**” which is declaration of a function that returns void pointer**.** See following post **for** details**.**

How to declare a pointer to a function**?**

Following are some interesting facts about function pointers**.**

1**)** Unlike normal pointers**,** a function pointer points to code**,** not data**.** Typically a function pointer stores the start of executable code**.**

2**)** Unlike normal pointers**,** we **do** not allocate de**-**allocate memory using function pointers**.**

3**)** A function’s name can also be used to get functions’ address**.** For example**,** in the below program**,** we have removed address operator ‘**&**’ in assignment**.** We have also changed function call by removing **\*,** the program still works**.**

#include <stdio.h>

// A normal function with an int parameter

// and void return type

void fun**(**int a**)**

**{**

printf**(**"Value of a is %d\n"**,** a**);**

**}**

int main**()**

**{**

void **(\***fun\_ptr**)(**int**)** **=** fun**;** // & removed

fun\_ptr**(**10**);** // \* removed

**return** 0**;**

**}**

Output**:**

Value of a is 10

4**)** Like normal pointers**,** we can have an array of function pointers**.** Below example in point 5 shows syntax **for** array of pointers**.**

5**)** Function pointer can be used in place of **switch** **case.** For example**,** in below program**,** user is asked **for** a choice between 0 and 2 to **do** different tasks**.**

#include <stdio.h>

void add**(**int a**,** int b**)**

**{**

printf**(**"Addition is %d\n"**,** a**+**b**);**

**}**

void subtract**(**int a**,** int b**)**

**{**

printf**(**"Subtraction is %d\n"**,** a**-**b**);**

**}**

void multiply**(**int a**,** int b**)**

**{**

printf**(**"Multiplication is %d\n"**,** a**\***b**);**

**}**

int main**()**

**{**

// fun\_ptr\_arr is an array of function pointers

void **(\***fun\_ptr\_arr**[])(**int**,** int**)** **=** **{**add**,** subtract**,** multiply**};**

unsigned int ch**,** a **=** 15**,** b **=** 10**;**

printf**(**"Enter Choice: 0 for add, 1 for subtract and 2 "

"for multiply\n"**);**

scanf**(**"%d"**,** **&**ch**);**

**if** **(**ch **>** 2**)** **return** 0**;**

**(\***fun\_ptr\_arr**[**ch**])(**a**,** b**);**

**return** 0**;**

**}**

Enter Choice**:** 0 **for** add**,** 1 **for** subtract and 2 **for** multiply

2

Multiplication is 150

6**)** Like normal data pointers**,** a function pointer can be passed as an argument and can also be returned from a function**.**

For example**,** consider the following C program where wrapper**()** receives a void fun**()** as parameter and calls the passed function**.**

// A simple C program to show function pointers as parameter

#include <stdio.h>

// Two simple functions

void fun1**()** **{** printf**(**"Fun1\n"**);** **}**

void fun2**()** **{** printf**(**"Fun2\n"**);** **}**

// A function that receives a simple function

// as parameter and calls the function

void wrapper**(**void **(\***fun**)())**

**{**

fun**();**

**}**

int main**()**

**{**

wrapper**(**fun1**);**

wrapper**(**fun2**);**

**return** 0**;**

**}**

This point in particular is very useful in C**.** In C**,** we can use function pointers to avoid code redundancy**.** For example a simple qsort**()** function can be used to sort arrays in ascending order or descending or by any other order in **case** of array of structures**.** Not only this**,** with function pointers and void pointers**,** it is possible to use qsort **for** any data type**.**

// An example for qsort and comparator

#include <stdio.h>

#include <stdlib.h>

// A sample comparator function that is used

// for sorting an integer array in ascending order.

// To sort any array for any other data type and/or

// criteria, all we need to do is write more compare

// functions. And we can use the same qsort()

int compare **(**const void **\*** a**,** const void **\*** b**)**

**{**

**return** **(** **\*(**int**\*)**a **-** **\*(**int**\*)**b **);**

**}**

int main **()**

**{**

int arr**[]** **=** **{**10**,** 5**,** 15**,** 12**,** 90**,** 80**};**

int n **=** **sizeof(**arr**)/sizeof(**arr**[**0**]),** i**;**

qsort **(**arr**,** n**,** **sizeof(**int**),** compare**);**

**for** **(**i**=**0**;** i**<**n**;** i**++)**

printf **(**"%d "**,** arr**[**i**]);**

**return** 0**;**

**}**

Output**:**

5 10 12 15 80 90

Similar to qsort**(),** we can write our own functions that can be used **for** any data type and can **do** different tasks without code redundancy**.** Below is an example search function that can be used **for** any data type**.** In fact we can use this search function to find close elements **(**below a threshold**)** by writing a customized compare function**.**

#include <stdio.h>

#include <stdbool.h>

// A compare function that is used for searching an integer

// array

bool compare **(**const void **\*** a**,** const void **\*** b**)**

**{**

**return** **(** **\*(**int**\*)**a **==** **\*(**int**\*)**b **);**

**}**

// General purpose search() function that can be used

// for searching an element \*x in an array arr[] of

// arr\_size. Note that void pointers are used so that

// the function can be called by passing a pointer of

// any type. ele\_size is size of an array element

int search**(**void **\***arr**,** int arr\_size**,** int ele\_size**,** void **\***x**,**

bool compare **(**const void **\*** **,** const void **\*))**

**{**

// Since char takes one byte, we can use char pointer

// for any type/ To get pointer arithmetic correct,

// we need to multiply index with size of an array

// element ele\_size

char **\***ptr **=** **(**char **\*)**arr**;**

int i**;**

**for** **(**i**=**0**;** i**<**arr\_size**;** i**++)**

**if** **(**compare**(**ptr **+** i**\***ele\_size**,** x**))**

**return** i**;**

// If element not found

**return** **-**1**;**

**}**

int main**()**

**{**

int arr**[]** **=** **{**2**,** 5**,** 7**,** 90**,** 70**};**

int n **=** **sizeof(**arr**)/sizeof(**arr**[**0**]);**

int x **=** 7**;**

printf **(**"Returned index is %d "**,** search**(**arr**,** n**,**

**sizeof(**int**),** **&**x**,** compare**));**

**return** 0**;**

**}**

Output**:**

Returned index is 2

The above search function can be used **for** any data type by writing a separate customized compare**().**

7**)** Many object oriented features in C**++** are implemented using function pointers in C**.** For example virtual functions**.** Class methods are another example implemented using function pointers**.** Refer this book **for** more details**.**

Related Article**:**Pointers in C and C**++** **|** Set 1 **(**Introduction**,** Arithmetic and Array**)**

References**:**

http://www.cs.cmu.edu/~ab/15-123S11/AnnotatedNotes/Lecture14.pdf

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-087-practical-programming-in-c-january-iap-2010/lecture-notes/MIT6\_087IAP10\_lec08.pdf

http://www.cs.cmu.edu/~guna/15-123S11/Lectures/Lecture14.pdf

# Return pointer from functions in C

1. Return variable should be local static variable

#include <stdio.h>

#include <time.h>

/\* function to generate and return random numbers. \*/

int **\*** getRandom**(** **)**

**{**

static int r**[**10**];**

int i**;**

/\* set the seed \*/

srand**(** **(**unsigned**)**time**(** **NULL** **)** **);**

**for** **(** i **=** 0**;** i **<** 10**;** **++**i**)**

**{**

r**[**i**]** **=** rand**();**

printf**(**"%d\n"**,** r**[**i**]** **);**

**}**

**return** r**;**

**}**

/\* main function to call above defined function \*/

int main **()**

**{**

/\* a pointer to an int \*/

int **\***p**;**

int i**;**

p **=** getRandom**();**

**for** **(** i **=** 0**;** i **<** 10**;** i**++** **)**

**{**

printf**(**"\*(p + [%d]) : %d\n"**,** i**,** **\*(**p **+** i**)** **);**

**}**

**return** 0**;**

**}**

OUTPUT**:**

1572137106

239294930

2043050450

1209150391

1463724724

685239805

1470579469

622798986

1557074008

560728029

**\*(**p **+** **[**0**])** **:** 1572137106

**\*(**p **+** **[**1**])** **:** 239294930

**\*(**p **+** **[**2**])** **:** 2043050450

**\*(**p **+** **[**3**])** **:** 1209150391

**\*(**p **+** **[**4**])** **:** 1463724724

**\*(**p **+** **[**5**])** **:** 685239805

**\*(**p **+** **[**6**])** **:** 1470579469

**\*(**p **+** **[**7**])** **:** 622798986

**\*(**p **+** **[**8**])** **:** 1557074008

**\*(**p **+** **[**9**])** **:** 560728029