ALX LESSON

0x07 C Even More pointers, arrays and strings

C - Programming

TABLE OF CONTENTS

01

OVERVIEW topics

Topics

C Programming

Topics

what are the pointers to pointers and how to use them

what are multidimensional arrays and how to use them

what are the most common c standard library function to mainpulate strings

memset
memcpy
strchr
strspn
strpbrk
strstr

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

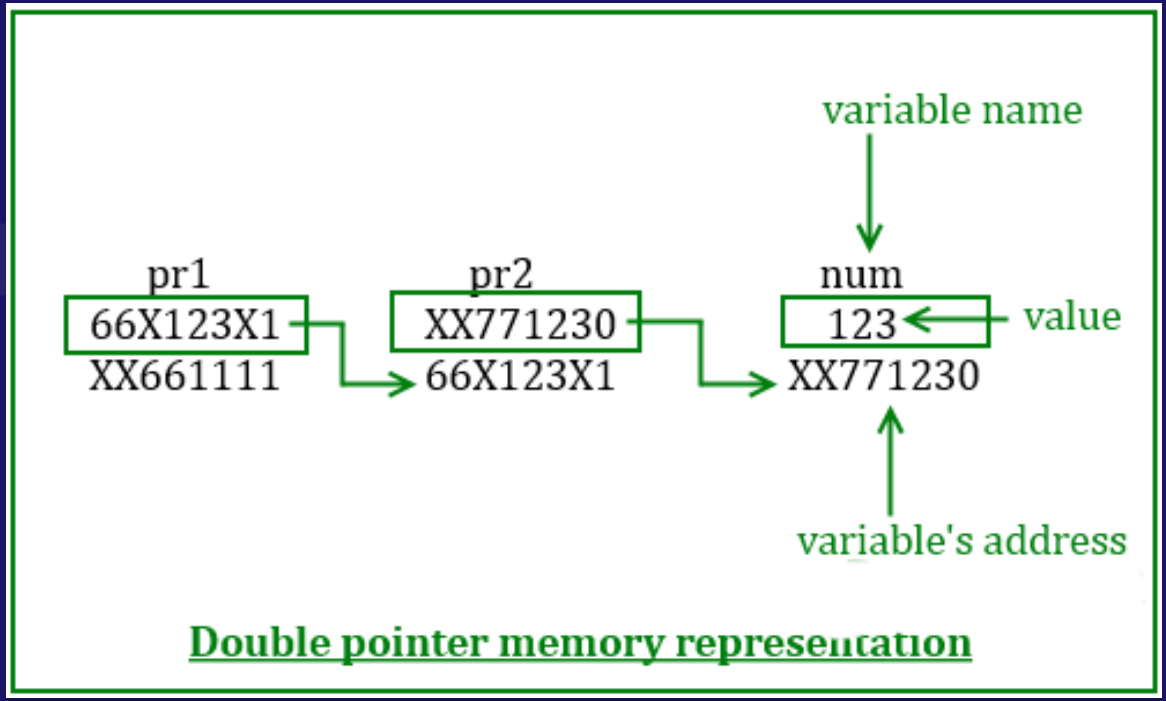
what are the pointers to pointers and how to use them

A pointer to a pointer is a form of multiple indirection, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.

A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, the following declaration declares a pointer to a pointer of type int

```
int **var;
```

what are the pointers to pointers and how to use them




```
Variable num has address: XX771230
Address of Pointer pr1 is: XX661111
Address of Pointer pr2 is: 66X123X1
```

what are the pointers to pointers and how to use them

```c
#include <stdio.h>

int main () {

   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;
   /* take the address of var */
   ptr = &var;

   /* take the address of ptr using address of operator & */
   pptr = &ptr;

   /* take the value using pptr */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n", **pptr);

   return 0;
}
```

what are multidimensional arrays and how to use them

Multidimensional arrays are arrays that have more than one dimension.

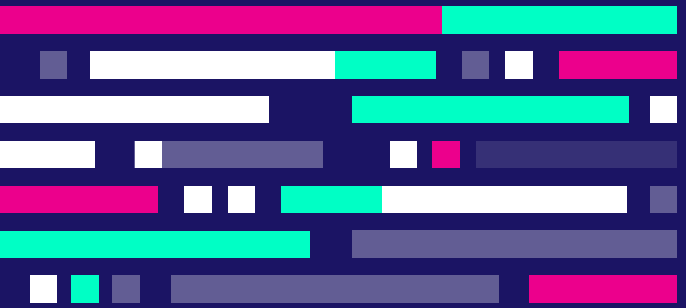
C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration

```
type name[size1][size2]...[sizeN];
```

For example, the following declaration creates a three dimensional integer array -

```
int threedim[5][10][4];
```

Two-dimensional Arrays



The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows

```
type arrayName [ x ][ y ];
```

Where type can be any valid C data type and arrayName will be a valid C identifier.

Two-dimensional Arrays

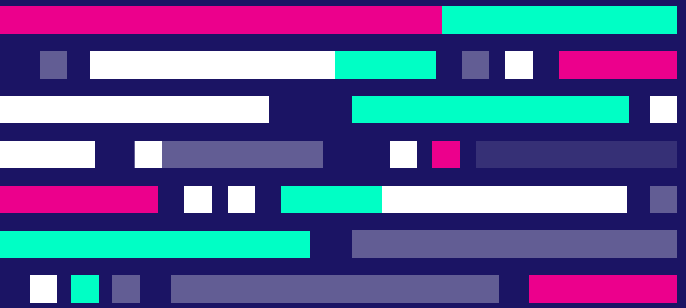
A two-dimensional array can be considered as a table which will have x number of rows and y number of columns.

A two-dimensional array a, which contains three rows and four columns can be shown as follows:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Thus, every element in the array a is identified by an element name of the form a[i][j], where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

Initializing Two-Dimensional Arrays



Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example -

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Accessing Two-Dimensional Array Elements

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example -

```
int val = a[2][3];
```

The above statement will take the 4th element from the 3rd row of the array.

Answer : 11

Example 1

```c
#include <stdio.h>

int main () {

   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;

   /* output each array element's value */
   for ( i = 0; i < 5; i++ ) {

      for ( j = 0; j < 2; j++ ) {
         printf("a[%d][%d] = %d\n", i,j, a[i][j] );
      }
   }

   return 0;
}
```

Example 2

```c
#include<stdio.h>
int main(){
   /* 2D array declaration*/
   int disp[2][3];
   /*Counter variables for the loop*/
   int i, j;
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("Enter value for disp[%d][%d]:", i, j);
         scanf("%d", &disp[i][j]);
      }
   }
   //Displaying array elements
   printf("Two Dimensional array elements:\n");
   for(i=0; i<2; i++) {
      for(j=0;j<3;j++) {
         printf("%d ", disp[i][j]);
         if(j==2){
            printf("\n");
         }
      }
   }
   return 0;
}
```

memset

The C library function

```
void *memset(void *str, int c, size_t n)
```

copies the character c (an unsigned char) to the first n characters of the string pointed to, by the argument str.

str - This is a pointer to the block of memory to fill.

c - This is the value to be set. The value is passed as an int, but the function fills the block of memory using the unsigned char conversion of this value.

n - This is the number of bytes to be set to the value.

memset

```c
#include <stdio.h>
#include <string.h>

int main () {
   char str[50];

   strcpy(str,"This is string.h library function");
   puts(str);

   memset(str,'$',7);
   puts(str);

   return(0);
}
```

```
This is string.h library function
$$$$$$$ string.h library function
```

memcpy

The C library function

```
void *memcpy(void *dest, const void *src, size_t n)
```

copies n characters from memory area src to memory area dest.

dest - This is pointer to the destination array where the content is to be copied, type-casted to a pointer of type void*.

src - This is pointer to the source of data to be copied, type-casted to a pointer of type void*.

n - This is the number of bytes to be copied.

memcpy

```c
#include <stdio.h>
#include <string.h>
int main () {
const char src[50] = "Alx_2023";
char dest[50];

strcpy(dest,"Heloooo!!");
printf("Before memcpy dest = %s\n", dest);
memcpy(dest, src, strlen(src)+1);
printf("After memcpy dest = %s\n", dest);

return(0);
}
```

```
Before memcpy dest = Heloooo!!
After memcpy dest = Alx_2023
```

strchr

The C library function

```
char *strchr(const char *str, int c)
```

searches for the first occurrence of the character c (an unsigned char) in the string pointed to by the argument str.

str - This is the C string to be scanned.

c - This is the character to be searched in str.

strchr

```c
#include <stdio.h>
#include <string.h>

int main() {
  char string[] = "Hello, world!";
  char *result;

  // Search for the first occurrence of the letter 'o'
  result = strchr(string, 'o');

  // Print out the result
  printf("The first occurrence of 'o' is at position %ld.\n", result - string);

  return 0;
}
```

```
The first occurrence of 'o' is at position 4.
```

strspn

The C library function

```
size_t strspn(const char *str1, const char *str2)
```

calculates the length of the initial segment of str1 which consists entirely of characters in str2.

str1 - This is the main C string to be scanned.

str2 - This is the string containing the list of characters to match in str1.

strspn

```c
#include <stdio.h>
#include <string.h>

int main () {
   int len;
   const char str1[] = "ABCDEFG019874";
   const char str2[] = "ABCD";

   len = strspn(str1, str2);

   printf("Length of initial segment matching %d\n", len );

   return(0);
}
```

```
Length of initial segment matching 4
```

strpbrk

The C library function

```
char *strpbrk(const char *str1, const char *str2)
```

finds the first character in the string str1 that matches any character specified in str2. This does not include the terminating null-characters.

str1 - This is the C string to be scanned.

str2 - This is the C string containing the characters to match.

strpbrk

```c
#include <stdio.h>
#include <string.h>

int main() {
  char string[] = "Hello, world!";
  char chars[] = "ow";
  char *result;

  // Search for the first occurrence of any character in chars
  result = strpbrk(string, chars);

  // Print out the result
  printf("The first occurrence of any character in chars is at position %ld.\n", result - string);

  return 0;
}
```

```
The first occurrence of any character in chars is at position 4.
```

strstr

The C library function

```
char *strstr(const char *haystack, const char *needle)
```

function finds the first occurrence of the substring needle in the string haystack. The terminating '\0' characters are not compared.

haystack - This is the main C string to be scanned.

needle - This is the small string to be searched with-in haystack string.

strstr

```c
#include <stdio.h>
#include <string.h>


int main () {
   const char haystack[30] = "ALX_2023 Helps You And Me";
   const char needle[10] = "Helps";
   char *ret;

   ret = strstr(haystack, needle);

   printf("The substring is: %s\n", ret);

   return(0);
}
```

```
The substring is: Helps You And Me
```

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