ALX LESSON
0x0C C - More malloc, free

C - Programming

TABLE OF CONTENTS

01

OVERVIEW topics

Topics

C Programming

Topics

How to use the exit function

difference between exit and return

Better way to cast malloc or calloc or realloc

What are the functions calloc and realloc from the standard library and how to use them

Slides On Telegram

https://t.me/alx_2023

C Programming

Topics

02

Learning Objectives

How to use the exit function

the exit() function is used to terminate the program immediately. The exit() function is part of the stdlib.h library and takes a single integer argument, which is the exit status of the program.

The exit() function can be called from any function in the program, and it will terminate the program immediately, closing any open files and freeing any memory allocated by the program. The exit status value can be used by other programs that call the terminated program, for example, to determine whether the program terminated normally or encountered an error.

exit() method

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

int main() {
    int a, b, c;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    if (b == 0) {
        printf("Error: division by zero.\n");
        exit(1);  // terminate program with exit status 1
    }

    c = a / b;
    printf("%d / %d = %d\n", a, b, c);

    exit(0);  // terminate program with exit status 0
}
```

difference between exit and return

The exit() function is used to terminate the program immediately, without executing any further instructions in the current function or in the program. The exit() function takes an integer argument that represents the exit status of the program, which can be used by other programs that call the terminated program to determine whether the program terminated normally or encountered an error. The exit() function also performs some cleanup tasks, such as closing any open files and freeing any memory allocated by the program.

On the other hand, return is used to exit a function and return a value to the calling function or to the operating system, if the function is the main function. When a return statement is encountered, the program control returns to the calling function, and the returned value is used for further processing. If the return statement is used in the main function, the returned value is used as the exit status of the program.

realloc() method

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

int main() {
    int a, b, c;
    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);

    if (b == 0) {
        printf("Error: division by zero.\n");
        exit(1);  // terminate program with exit status 1
    }

    c = a / b;
    printf("%d / %d = %d\n", a, b, c);

    return 0;  // return from main function with exit status 0
}
```

Better way to cast malloc or calloc or realloc

```
int *sieve = malloc(sizeof(*sieve) * length);
```

better than:

```
int *sieve = (int *) malloc(sizeof(*sieve) * length);
```

Why would this be the case?

you don't cast the result, since:

- It is unnecessary, as void * is automatically and safely promoted (casted) to any other pointer type in this case.
- It adds clutter to the code, casts are not very easy to read (especially if the pointer type is long).
- It makes you repeat yourself, which is generally bad.

Better way to cast malloc or calloc or realloc

```
int *sieve = malloc(sizeof(*sieve) * length);
```

better than:

```
int *sieve = (int *) malloc(sizeof(*sieve) * length);
```

Why would this be the case?

It can hide an error if you forgot to include <stdlib.h>. This can cause crashes (or, worse, not cause a crash until way later in some totally different part of the code). Consider what happens if pointers and integers are differently sized; then you're hiding a warning by casting and might lose bits of your returned address.

Better way to cast malloc or calloc or realloc

Compare:

malloc(sizeof *sieve * length * width)

vs.

malloc(length * width * sizeof *sieve)

First one is better malloc(sizeof *sieve * length * width)

This expression multiplies sizeof(*sieve) with length and width, which guarantees that the multiplication is safe and the total size of the allocated memory is correct. The sizeof operator ensures that the size of the memory allocated is correctly determined based on the type of *sieve.

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