ALX LESSON

0x02. C - Functions, nested loops

C - Programming

TABLE OF CONTENTS

01

OVERVIEW topics

Topics

What are nested loops and to use them

What is a function and how do you use fuctions

What is the difference between a declaration and a definition of a function

What is a prototype

Scope of variables

What are the gcc flags -Wall -Werror -pedantic -Wextra -std=gnu89

What are header files and how to use them with #include

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

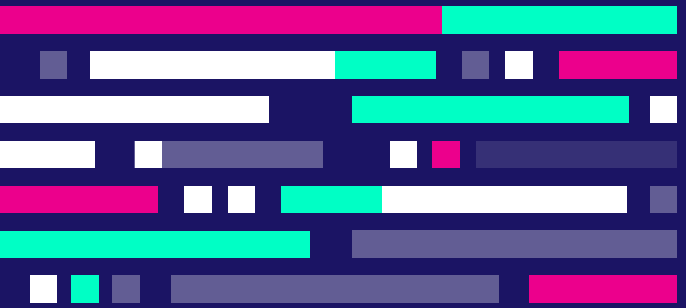
What are nested loops and to use them

Nested loops are a programming construct in which one loop is placed inside another loop. The inner loop is executed multiple times for each iteration of the outer loop. The number of iterations of the inner loop depends on the current iteration of the outer loop.

Nested loops are useful when we need to perform a task multiple times, and each time we perform the task, we need to do another set of tasks. For example, we might need to print out a multiplication table, where we iterate over the rows and columns of the table using nested loops.

```
#include <stdio.h>
int main() {
    int i, j;

    for(i=1; i<=5; i++) {
        for(j=1; j<=i; j++) {
            printf("* ");
        }
        printf("\n");
    }
    return 0;
}
```

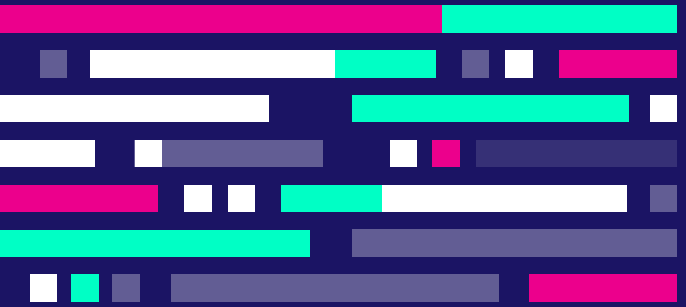
What is a function and how do you use fuctions



In programming, a function is a reusable block of code that performs a specific task. Functions allow programmers to break down a large program into smaller, modular pieces of code that can be easily tested, reused, and maintained.

Functions typically have a name, a set of input parameters, a return type, and a body of code that performs a specific operation. When a function is called, the input parameters are passed into the function, the body of code is executed, and a value is returned.

What is a function and how do you use fuctions



```c
#include <stdio.h>

int add(int a, int b) {
    int result = a + b;
    return result;
}

int main() {
    int x = 5;
    int y = 7;
    int z = add(x, y);
    printf("The sum of %d and %d is %d\n", x, y, z);
    return 0;
}
```

What is the difference between a declaration and a definition of a function

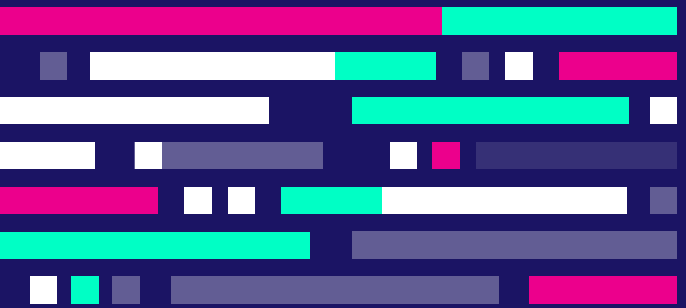
A function declaration provides information to the compiler about a function's name, return type, and parameters. This information is used by the compiler to verify that a function is being called correctly in the program. A function declaration does not provide the implementation of the function's code.

A function definition, on the other hand, provides both the declaration and the implementation of the function's code. A function definition specifies the function's name, return type, parameters, and the code that will be executed when the function is called.

it is useful to differentiate between function declarations and function definitions in programming because it allows for modular code design and improved code reusability.

By separating the declaration and definition of a function, you can create header files that contain the function declarations, which can then be included in multiple source files. This allows the same function to be used in different parts of your program without having to redefine the function each time.

What is the difference between a declaration and a definition of a function



```
// Function declaration
int add(int a, int b);

int main() {
    int x = 5;
    int y = 7;
    int z = add(x, y);  // Function call
    return 0;
}

// Function definition
int add(int a, int b) {
    int result = a + b;
    return result;
}
```

What is a prototype

a prototype is a declaration of a function that specifies the function's name, return type, and the type and number of its parameters. The prototype is typically placed at the beginning of a program or in a header file, and it allows the compiler to check the correctness of function calls and to ensure that the function is used correctly.

For example, consider the following function prototype:

```
int add(int x, int y);
```

This prototype declares a function named add that takes two integer parameters and returns an integer. The actual implementation of the function can be defined later in the program.

Prototypes are especially useful in large programs where functions may be defined in separate source files or libraries.

What is the difference between a declaration and a definition of a function

What are header files and how to use them with #include

myHeaders.h

```
#ifndef MYHEADERS_H
#define MYHEADERS_H

int add(int a, int b);

#endif
```

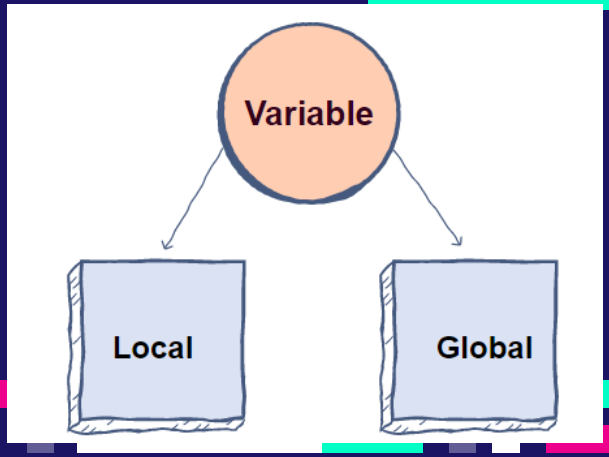
addition.c

```
#include "myHeaders.h"

int add(int a, int b) {
    return a + b;
}
```

main.c

```
#include <stdio.h>
#include "myHeaders.h"

int main() {
    int x = 5;
    int y = 7;
    int z = add(x, y);
    printf("The sum of %d and %d is %d\n", x, y, z);
    return 0;
}
```

Scope of variables



1. Local variables are defined within a function body. They are not accessible outside the function.

2. Global variables are defined outside the function body. They are accessible to the entire program.

The scope of a variable in C refers to the region of the program in which the variable can be accessed. In C, there are three main types of scope:

1. Block Scope
2. Function Scope
3. File Scope

Scope of variables

1. Block Scope

Block Scope:
Variables with block scope are defined within a block of code, such as within a function or a loop. They are only accessible within that block and are destroyed when the block is exited. For example:

```
int main() {
   int x = 10;
   {
      int y = 5;
      printf("%d\n", x + y);
   }
   printf("%d\n", x); // OK
   printf("%d\n", y); // Error: y is not in scope
   return 0;
}
```

Scope of variables

1. Function Scope:

Variables with function scope are defined at the beginning of a function, and are accessible throughout the function, but not outside of it. For example:

```
int x = 5; // global variable
void foo() {
   int y = 10; // local variable
   printf("%d\n", x + y); // OK
}
int main() {
   foo();
   printf("%d\n", y); // Error: y is not in scope
   return 0;
}
```

The four standard storage classes in C

1. auto: This is the default storage class for local variables defined within a block or function. The variable is allocated memory when the block or function is entered and is destroyed when the block or function is exited.

2. register: This storage class is used to define variables that should be stored in the CPU register for faster access. However, the use of the register keyword is only a suggestion to the compiler, and the compiler may choose to ignore it.

3. static: Variables declared with the static keyword have a lifetime that extends beyond the block or function in which they are declared. They retain their values between function calls and are initialized only once, at program startup.

4. extern: This storage class is used to declare variables that are defined in another file. The extern keyword tells the compiler that the variable is defined elsewhere and is available for use in the current file.

Scope of variables

1. File Scope:

Variables with file scope are declared outside of any function, and can be accessed by any function in the same file. These variables are visible throughout the entire file, but not outside of it. For example:

```
// file1.c
int x = 10; // file-scope variable

// file2.c
extern int x; // declare x from file1.c
void foo() {
    printf("%d\n", x); // OK
}
int main() {
    foo();
    return 0;
}
```

auto, register

```c
#include <stdio.h>

void myFunction() {
  auto int x = 5; // auto variable
  printf("x = %d\n", x);
}

int main() {
  myFunction();
  // x is not accessible here because it is
a local auto variable to myFunction
  return 0;
}
```

```c
register int i; // register variable
for (i = 0; i < 10000000; i++) {}
printf("i = %d\n", i);
```

What are the gcc flags -Wall -Werror -pedantic -Wextra -std=gnu89

-Wall: This flag enables all warning messages that are supported by the compiler. It helps to catch potential issues and bugs in the code.

-Werror: This flag makes all warnings into errors, meaning that the compiler will stop with an error message if any warning is generated during compilation.

-pedantic: This flag enforces strict adherence to the ANSI C standard, which helps to ensure portability of code across different compilers and platforms.

-Wextra: This flag enables additional warning messages that are not enabled by -Wall. These warnings provide more information about potential issues in the code.

-std=gnu89: This flag specifies the C standard to use during compilation. In this case, gnu89 refers to the C language standard defined by the GNU project, which is an extension of the ANSI C standard.

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