ALX LESSON

0x01. C Variables, if, else, while

C - Programming

TABLE OF CONTENTS

01

Overview topics

02

Learning Objectives

03

Quiz questions

04

hands on lab practice

01

OVERVIEW topics

C Programming

Topics

What are the arithmetic operators and how to use them

What are the logical operators (sometimes called boolean operators) and how to use them

What the relational operators and how to use them

What values are considered TRUE and FALCE in C

What are the boolean operators and how to use them

How to use if , if .... else satements

How to use comments

C Programming

Topics

How to declare variables of types char, int, unsigned int

How to assign values to variables

How to print the values of variables of type char, int, unsigned int, with printf

How to use the while loop

How to use variables with the while loop

How to print variables using printf

What is the ASCII character set

What are the purpose of the gcc flags -m32 and -m64

C Programming

Topics

- How to use the for loop
- How to use variables with the for loop
- How to use the do while loop
- How to use variables with the do while loop

Slides On Telegram

https://t.me/alx_2023

C Programming

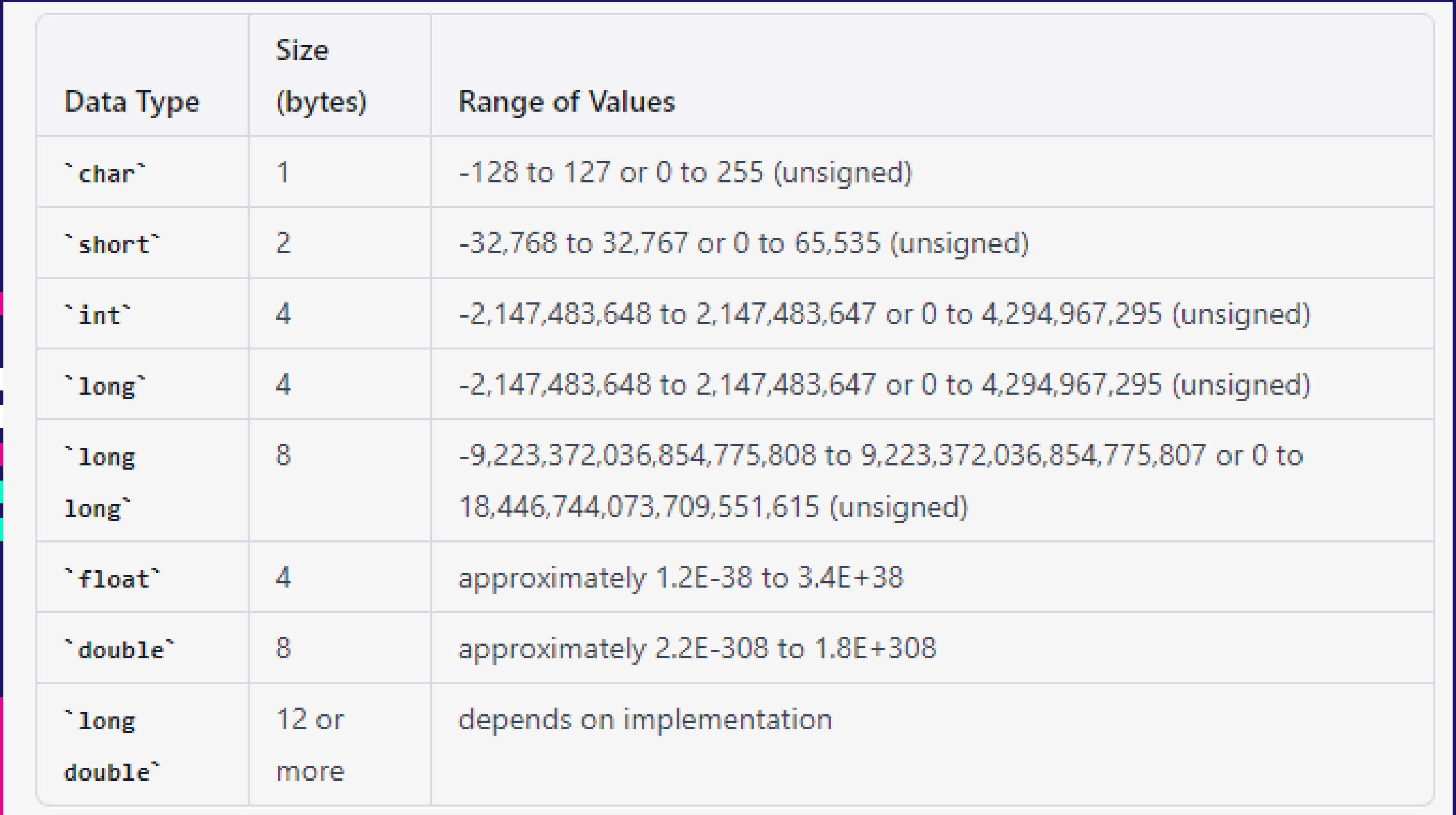
Topics

@ALX_2023

02

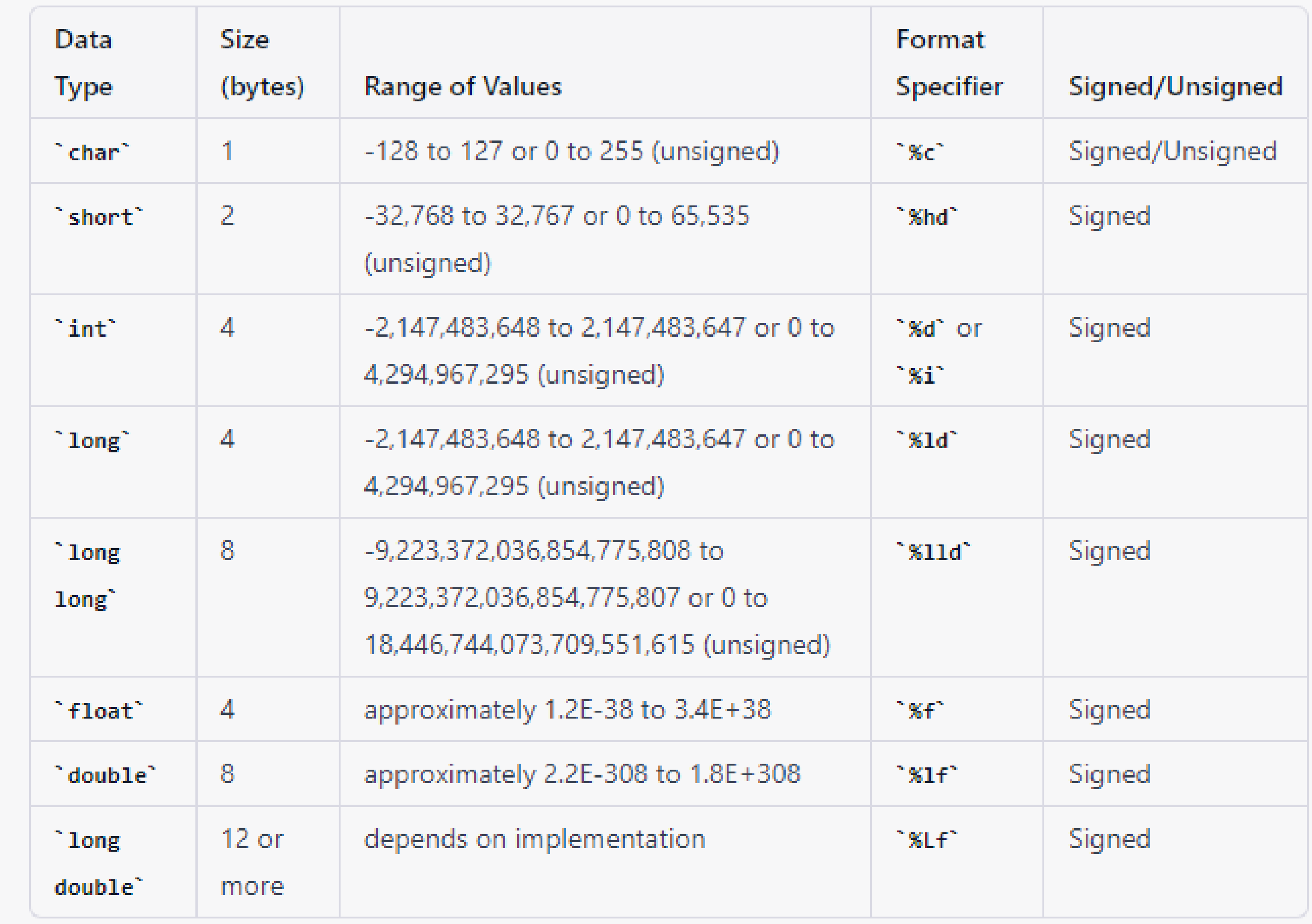
Learning Objectives

Data Types



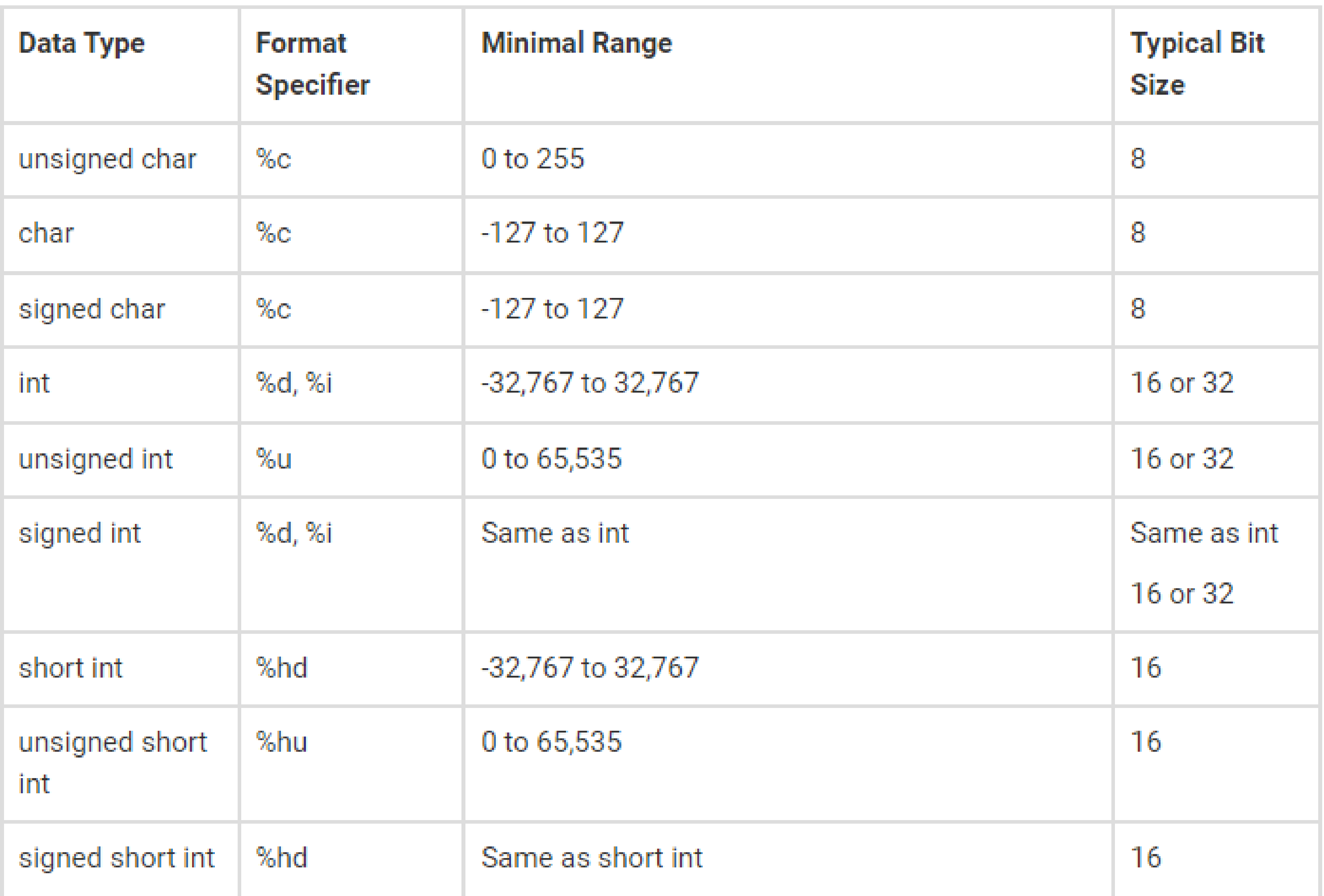
Data Type	Size (bytes)	Range of Values
`char`	1	-128 to 127 or 0 to 255 (unsigned)
`short`	2	-32,768 to 32,767 or 0 to 65,535 (unsigned)
`int`	4	-2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 (unsigned)
`long`	4	-2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 (unsigned)
`long long`	8	-9,223,372,036,854,775,808 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



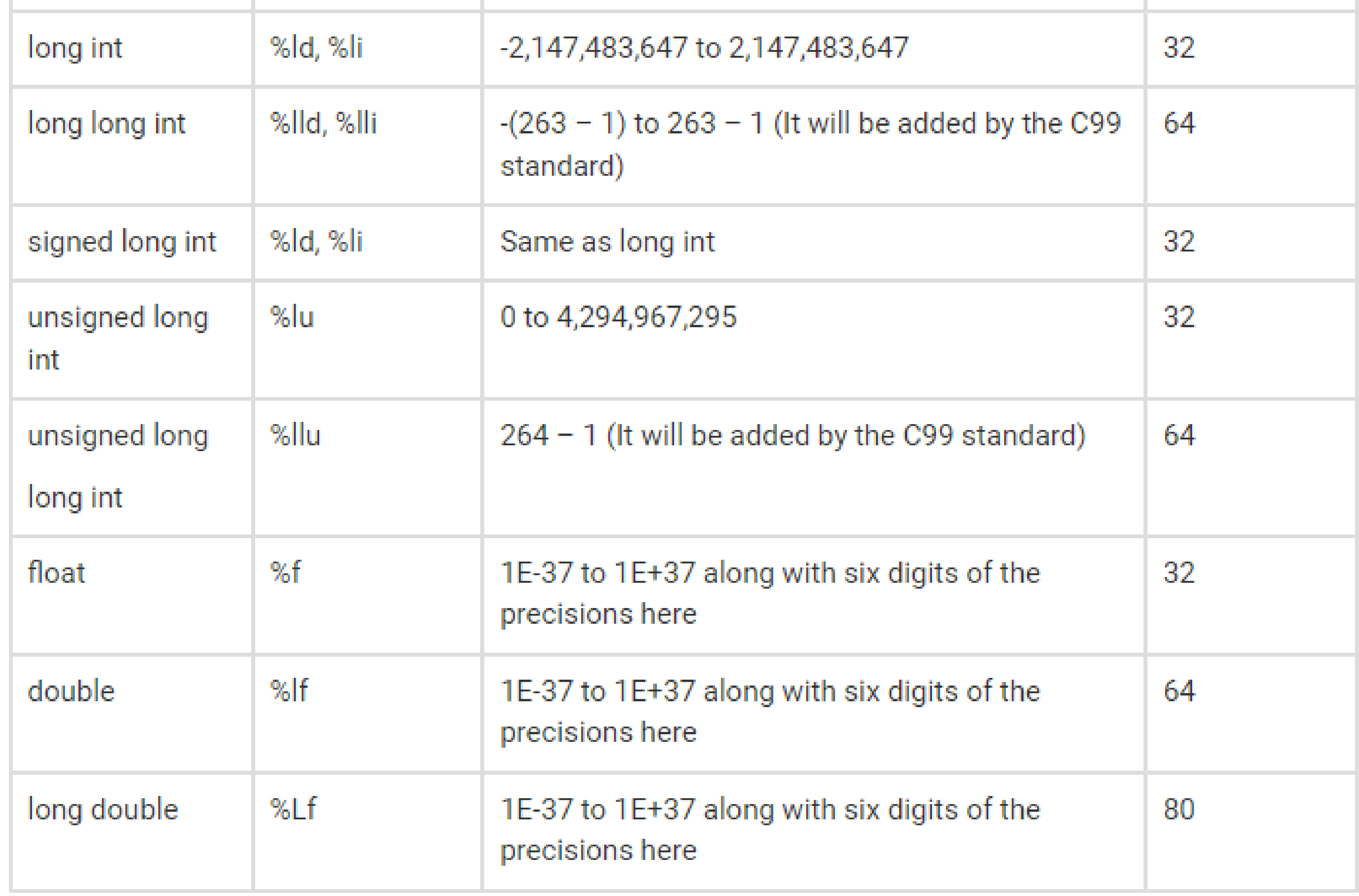
Data Type	Size (bytes)	Range of Values	Format Specifier	Signed/Unsigned
`char`	1	-128 to 127 or 0 to 255 (unsigned)	`%c`	Signed/Unsigned
`short`	2	-32,768 to 32,767 or 0 to 65,535 (unsigned)	`%hd`	Signed
`int`	4	-2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 (unsigned)	`%d` or `%i`	Signed
`long`	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 the arithmetic operators and how to use them

In C, there are several arithmetic operators that can be used to perform mathematical operations on variables and values. These operators include:

+ (addition) - adds two operands together
- (subtraction) - subtracts one operand from another
* (multiplication) - multiplies two operands together
/ (division) - divides one operand by another
% (modulus) - returns the remainder of a division operation

Here's an example program that demonstrates the use of these arithmetic operators:

What are the arithmetic operators and how to use them

```
#include <stdio.h>

int main() {
    int x = 10;
    int y = 3;

    printf("x + y = %d\n", x + y);
    printf("x - y = %d\n", x - y);
    printf("x * y = %d\n", x * y);
    printf("x / y = %d\n", x / y);
    printf("x %% y = %d\n", x % y);

    return 0;
}
```

OUTPUT:

```
x + y = 13
x - y = 7
x * y = 30
x / y = 3
x % y = 1
```

Note that the modulus operator % returns the remainder of the division operation. In the example above, x % y returns 1 because 10 divided by 3 leaves a remainder of 1.

Arithmetic operators can also be used with variables of other data types, such as float or double, as well as with literal values. It's important to keep in mind that division by zero is undefined in C and can lead to runtime errors.

What are the logical operators (sometimes called boolean operators) and how to use them

In C, there are three logical operators (also called boolean operators) that can be used to perform logical operations on boolean values (true or false). These operators are:

&& (logical AND) - returns true if both operands are true, otherwise returns false

|| (logical OR) - returns true if either operand is true, otherwise returns false

! (logical NOT) - returns true if the operand is false, otherwise returns false

Here's an example program that demonstrates the use of these logical operators:

What are the logical operators (sometimes called boolean operators) and how to use them

```
#include <stdio.h>
#include <stdbool.h>

int main() {
    bool a = true;
    bool b = false;

    printf("a && b = %d\n", a && b);
    printf("a || b = %d\n", a || b);
    printf("!a = %d\n", !a);
    printf("!b = %d\n", !b);

    return 0;
}
```

OUTPUT:

```
a && b = 0
a || b = 1
!a = 0
!b = 1
```

Logical operators can also be used in conjunction with comparison operators (e.g. ==, !=, <, >, <=, >=) to create complex logical expressions. It's important to keep in mind the order of operations when using logical operators in conjunction with comparison operators, and to use parentheses to group expressions when necessary to avoid ambiguity.

What values are considered TRUE and FALCE in C

In C, the values 0 and false are considered false, while any other value (including negative numbers and non-zero values) is considered true. This applies to all data types in C, including integers, floating point numbers, and pointers.

However, in C99 and later versions, a new header file called stdbool.h was introduced that defines a new data type called bool, which can have two possible values: true and false. These values are defined as macros that evaluate to integer constants (true is defined as 1 and false is defined as 0). When using the bool data type, it's recommended to include the stdbool.h header and use the true and false macros instead of the integer constants 1 and 0.

What values are considered TRUE and FALCE in C

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

int main() {
    bool a = true;
    bool b = false;

    if (a) {
        printf("a is true\n");
    } else {
        printf("a is false\n");
    }

    if (b) {
        printf("b is true\n");
    } else {
        printf("b is false\n");
    }

    return 0;
}
```

OUTPUT:

```
a is true
b is false
```

How to use if , if .... else satements

In C, the if statement is used to conditionally execute a block of code. The syntax of the if statement is as follows:

```
if (condition) {
    // code to be executed if the condition is true
}
```

The condition in the parentheses can be any expression that evaluates to a boolean value (true or false). If the condition is true, then the block of code inside the curly braces is executed. If the condition is false, then the block of code is skipped and execution continues with the next statement after the if block.

How to use if , if .... else satements

```
OUTPUT:
num is positive
num is odd
num is less than or equal to 5
```

```c
#include <stdio.h>

int main() {
    int num = 5;

    if (num > 0) {
        printf("num is positive\n");
    }

    if (num % 2 == 0) {
        printf("num is even\n");
    } else {
        printf("num is odd\n");
    }

    if (num > 10) {
        printf("num is greater than 10\n");
    } else if (num > 5) {
        printf("num is greater than 5\n");
    } else {
        printf("num is less than or equal to 5\n");
    }

    return 0;
}
```

How to use comments

```
// comment

// this is main function

/* multi line comment */

/* This program takes age input from the user
It stores it in the age variable
And, print the value using printf() */



 documenting

/**
 * Main Entry Point
*/
```

How to declare variables of types char, int, unsigned int

In C, you can declare variables of types char, int, and unsigned int using the following syntax:

```
// Declare a variable of type char
char my_char;

// Declare a variable of type int
int my_int;

// Declare a variable of type unsigned int
unsigned int my_unsigned_int;
```

You can also initialize the variables at the time of declaration by assigning a value to them:

```
// Declare and initialize a variable of type char
char my_char = 'a';

// Declare and initialize a variable of type int
int my_int = -10;

// Declare and initialize a variable of type unsigned int
unsigned int my_unsigned_int = 20;
```

How to print the values of variables of type char, int, unsigned int, with printf

```c
char my_char = 'a';
int my_int = 10;
unsigned int my_unsigned_int = 20;

// Print the value of a char variable
printf("The value of my_char is: %c\n", my_char);

// Print the value of an int variable
printf("The value of my_int is: %d\n", my_int);

// Print the value of an unsigned int variable
printf("The value of my_unsigned_int is: %u\n", my_unsigned_int);
```

How to use the while loop

In C programming, the while loop is used to execute a block of code repeatedly as long as a condition is true. The syntax of the while loop is as follows:

```
while (condition) {
    // code to be executed
}
```

Here, condition is an expression that is evaluated at the beginning of each iteration of the loop. If the condition is true, the code inside the loop is executed. This continues until the condition becomes false.

How to use the while loop and variables with it

Here's an example of how to use the while loop to print the numbers 1 to 5:

```
int i = 1;

while (i <= 5) {
    printf("%d ", i);
    i++;
}
```

In this example, the loop will continue to execute as long as the value of i is less than or equal to 5. Inside the loop, the value of i is printed using the printf() function, and then i is incremented by 1 using the i++ statement.

The output of this code will be:

```
1 2 3 4 5
```

Note that it's important to make sure that the condition in the while loop will eventually become false, otherwise the loop will execute indefinitely and the program will hang.

How to use variables with the while loop

Here's an example of how to use the while loop to print the numbers 1 to 5:

```
int i = 1;

while (i <= 5) {
    printf("%d ", i);
    i++;
}
```

In this example, the loop will continue to execute as long as the value of i is less than or equal to 5. Inside the loop, the value of i is printed using the printf() function, and then i is incremented by 1 using the i++ statement.

The output of this code will be:

```
1 2 3 4 5
```

Note that it's important to make sure that the condition in the while loop will eventually become false, otherwise the loop will execute indefinitely and the program will hang.

for loop

The for loop is a control flow statement in C programming that allows you to execute a block of code repeatedly for a fixed number of times.

The syntax of the for loop is as follows:

```
for (initialization; condition; increment/decrement) {
   // code to be executed
}
```

Initialization: The first expression is executed only once, before the loop starts. It is typically used to initialize the loop variable.

Condition: The second expression is a condition that is checked before each iteration of the loop. If the condition evaluates to true, the loop body is executed. If it evaluates to false, the loop terminates.

Increment/decrement: The third expression is executed after each iteration of the loop. It is typically used to update the loop variable.

How to use variables with the for loop

```
#include <stdio.h>

int main() {
   // print numbers from 1 to 10
   for (int i = 1; i <= 10; i++) {
      printf("%d ", i);
   }

   return 0;
}
```

```
#include <stdio.h>

int main() {
   int i;

   // print numbers from 1 to 10
   for (i = 1; i <= 10; i++) {
      printf("%d ", i);
   }

   return 0;
}
```

How to use variables with the for loop

```
#include <stdio.h>

int main() {
   // print numbers from 1 to 10
   int i = 1;
   for (;i <= 10;) {
      printf("%d ", i);
      i++;
   }

   return 0;
}
```

do while

The do-while loop is another type of loop in C programming that is similar to the while loop. The difference is that the do-while loop executes the loop body at least once before checking the loop condition. The basic syntax of the do-while loop is as follows:

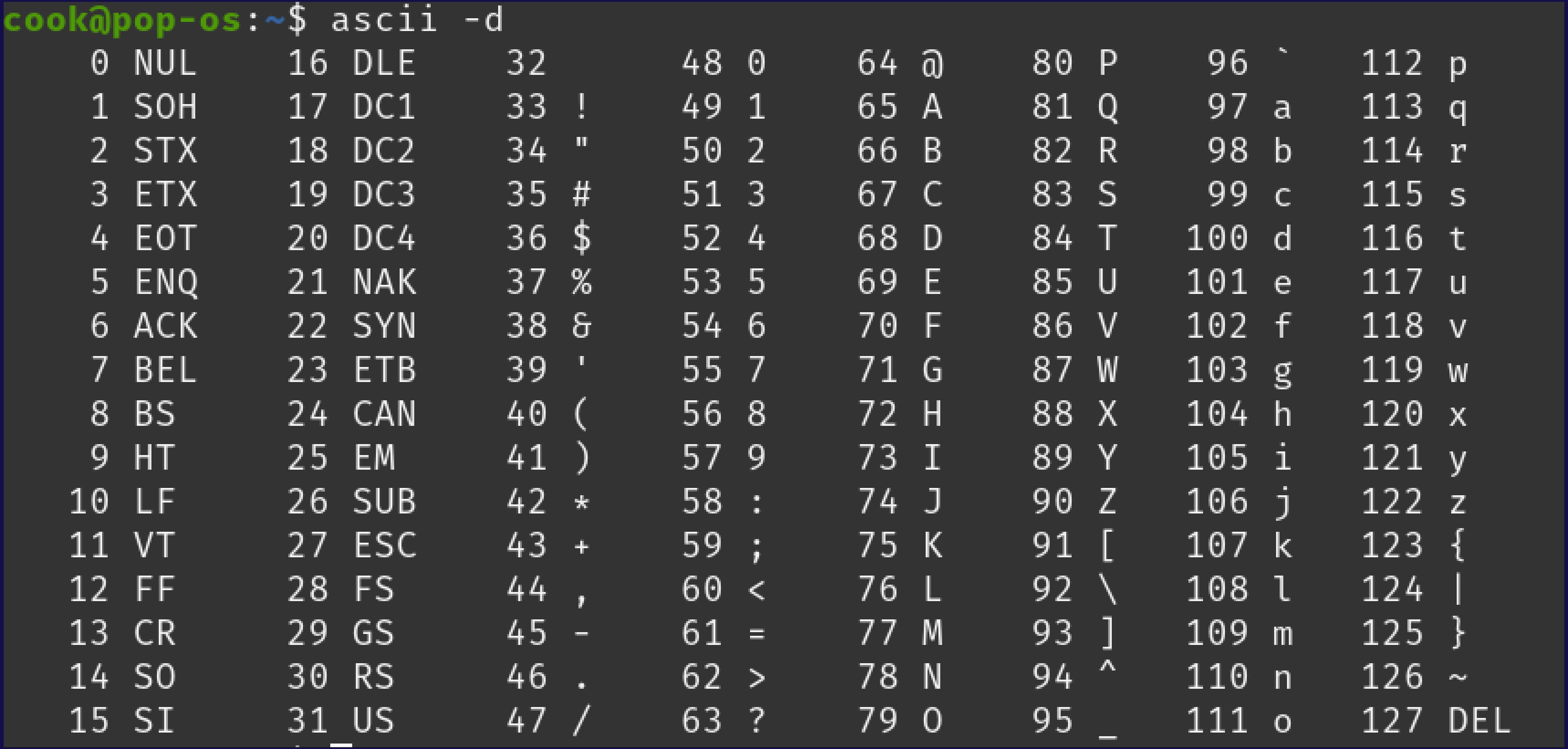
```
do {
    // loop body
} while (condition);
```

In this syntax, the loop body is executed first, and then the condition is checked. If the condition is true, the loop body is executed again, and the process repeats until the condition becomes false. Note that the loop body is guaranteed to be executed at least once, regardless of the condition.

How to use variables with the do while loop

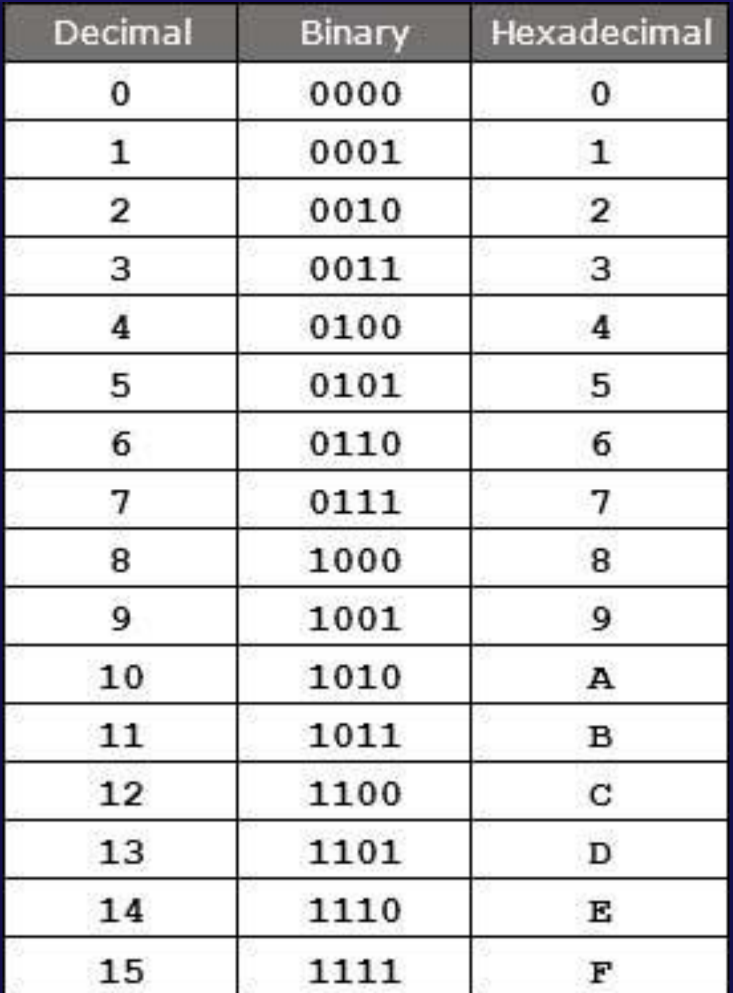
```c
#include <stdio.h>

int main() {
    int num;
    do {
        printf("Enter a positive integer (or a negative integer to exit): ");
        scanf("%d", &num);
    } while (num >= 0);

    printf("You entered a negative integer. Goodbye!\n");
    return 0;
}
```

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

what are the purpose of the gcc flags -m32 and -m64

The GCC flags -m32 and -m64 are used to specify the target architecture for the compiled binary.

The -m32 flag tells the compiler to generate code for a 32-bit architecture, which is capable of addressing up to 4GB of memory. This flag is typically used on older systems or when compatibility with 32-bit libraries or operating systems is required.

The -m64 flag, on the other hand, tells the compiler to generate code for a 64-bit architecture, which is capable of addressing much larger amounts of memory than a 32-bit architecture. This flag is typically used on newer systems or when the application needs to access more than 4GB of memory.

It's important to note that using these flags alone will not necessarily make the program faster or more efficient. The choice of architecture depends on the specific requirements of the application and the system it will be running on. Additionally, not all processors support both 32-bit and 64-bit architectures, so it's important to check the system specifications before choosing which flag to use.

04

Hands on lab Practice

Have a Question Leave a Comment!

Subscribe

To stay updated with latest videos

Share

To let the others know more

Thanks