ALX LESSON 0x00. C - Hello, World

gcc - printf - puts - putchar - sizeof

C - Programming

TABLE OF CONTENTS

01

OVERVIEW topics

C Programming

Topics

Why C programming is awesome

Who invented C

Who are Dennis Ritchie, Brain Kernighan and Linus Torvalds

What happens when you type gcc main.c

What is main

How to print text using printf, puts and putchar

How to get the size of a specific type using the unary operator sizeof

C Programming

Topics

How to compile using gcc

What is the default program name when compiling with gcc

What is the official C coding style and how to check your code with betty-style

How to find the right header to include in your source code when using a standard library function

How does the main function influence the return value of the program

C Programming

Topics

- gcc
- printf
- puts
- putchar
- sizeof

Slides On Telegram

https://t.me/alx_2023

C Programming

Topics

ALX

@ALX_2023

02

Learning Objectives

Why C programming is awesome

C programmming is awesome for several reasons:

Efficiency: C is known for its efficiency and speed. It is a compiled language that provides low-level access to computer memory and hardware, making it ideal for systems programming. C code can be optimized for specific hardware platforms to achieve maximum performance.

Portability: C code is highly portable and can be compiled on different platforms with minor modifications. C programs can be written once and compiled on different operating systems, making it a popular choice for cross-platform development.

Why C programming is awesome

Flexibility: C is a flexible language that allows programmers to write code at a low level, making it ideal for system-level programming. At the same time, it also supports high-level programming constructs such as functions, arrays, and pointers, making it possible to write complex applications.

Popular: C is a widely used programming language, and there is a large community of developers who use and contribute to it. As a result, there are many resources available for learning C, including online tutorials, books, and forums.

Foundation for other languages: Many other programming languages, including C++, Java, and Python, are based on C or have a syntax similar to C. Learning C can provide a solid foundation for learning these other languages.

Who invented C ?

C programming language was developed by Dennis Ritchie at Bell Labs in the early 1970s. Ritchie created C as an evolution of an earlier language called B, which he had also developed at Bell Labs. The development of C was motivated by a need for a more efficient language that could be used to write operating systems and system software.

Dennis Ritchie was a computer scientist and a key figure in the development of Unix operating system, which was also created at Bell Labs. Ritchie's work on C and Unix had a profound impact on the field of computer science and software engineering, and C has since become one of the most widely used programming languages in the world.

Dennis Ritchie

Who are Brain Kernighan?

Brian Kernighan is a computer scientist who has worked at Bell Labs, Princeton University, and other institutions. He is the co-author of the book "The C Programming Language," which he wrote with Dennis Ritchie. Kernighan has also made significant contributions to other areas of computer science, including software

Brain Kernighan

Who are Linus Torvalds?

Linus Torvalds is a software engineer who is best known for creating the Linux operating system. Torvalds developed Linux while he was a student at the University of Helsinki, and he has continued to work on the project ever since. Linux is now one of the most widely used operating systems in the world, and Torvalds is considered to be one of the most influential figures in the open-source software movement.

Linus Torvalds

What happens when you type gcc main.c

1. The command "gcc" invokes the GNU Compiler Collection, which is a set of programming tools that includes compilers, linkers, and other utilities.

2. The compiler reads the source code file "main.c" and performs lexical analysis, parsing, and semantic analysis to generate an object file that contains machine code instructions for the program.

3. If the source code contains any errors, the compiler will report them and the compilation process will be terminated.

4. If the compilation is successful, the linker is invoked to combine the object file generated by the compiler with any necessary libraries and generate an executable file.

5. The resulting executable file can be run by typing "./a.out" in the terminal, assuming the default output file name "a.out" was not changed using the "-o" option.

What is main

In the C programming language, "main" is a special function that serves as the entry point for a C program. When a C program is run, the operating system loads the program into memory and starts executing instructions at the beginning of the "main" function.

The "main" function has the following signature:

```
int main(void)
```

This means that it takes no arguments and returns an integer value. The integer value returned by the "main" function is used as the exit status of the program. A return value of 0 indicates success, while any other value indicates an error or abnormal termination.

What is main

The "main" function typically contains the top-level logic for the program, including any initialization code, input/output operations, and other functionality. It can also call other functions or external libraries as needed to perform more complex operations.

In short, the "main" function is the starting point for a C program and contains the primary logic for the program's execution.

How to print text using printf, puts and putchar

1- printf

The printf() function is used to print formatted text to the console. It takes a string as the first argument, which can contain placeholders for variables that will be replaced with their corresponding values. Here's an example:

```
#include <stdio.h>
int main(){
    int age = 25;
    printf("My age is %d\n", age);
    return 0;
}
```

This will output the following text to the console:

```
My age is 25
```

How to print text using printf, puts and putchar

2- puts

The puts() function is used to print a string to the console, followed by a newline character. It takes a single argument, which is the string to be printed. Here's an example:

```
#include <stdio.h>
int main(){
    puts("Hello, world!");
    return 0;
}
```

This will output the following text to the console:

```
Hello, world!
```

How to print text using printf, puts and putchar

3- putchar

The putchar() function is used to print a single character to the console. It takes a single argument, which is the character to be printed. Here's an example:

```
#include <stdio.h>
int main(){
    putchar('H');
    putchar('i');
    return 0;
}
```

This will output the following text to the console:

```
Hi
```

Note that the putchar() function needs to be called for each character individually. If you want to print a string with putchar(), you'll need to use a loop to iterate over each character in the string.

How to get the size of a specific type using the unary operator sizeof

you can use the sizeof operator to determine the size (in bytes) of a specific data type or variable. The syntax for using the sizeof operator is as follows:

sizeof(type)

where type is the name of the data type you want to get the size of. Here are some examples:

sizeof(int)   // returns the size of an integer (usually 4 bytes)

sizeof(float)   // returns the size of a floating-point number (usually 4 bytes)

sizeof(double)   // returns the size of a double-precision floating-point number (usually 8 bytes)

sizeof(char)   // returns the size of a character (usually 1 byte)

How to get the size of a specific type using the unary operator sizeof

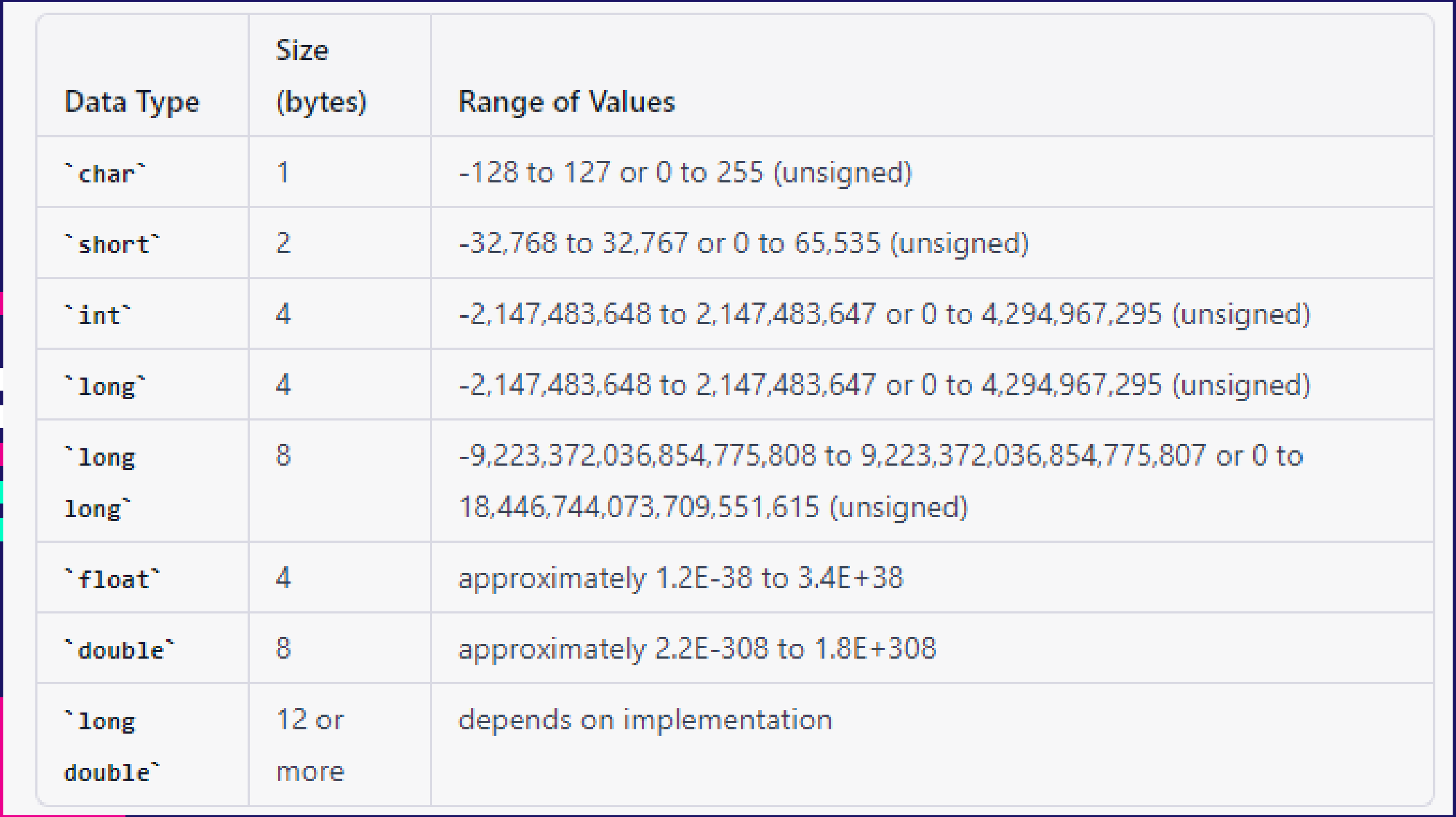
You can also use the sizeof operator to get the size of a variable, like this:

```
int my_var = 42;
printf("The size of my_var is %lu bytes\n", sizeof(my_var));
```

This will output the following text to the console:

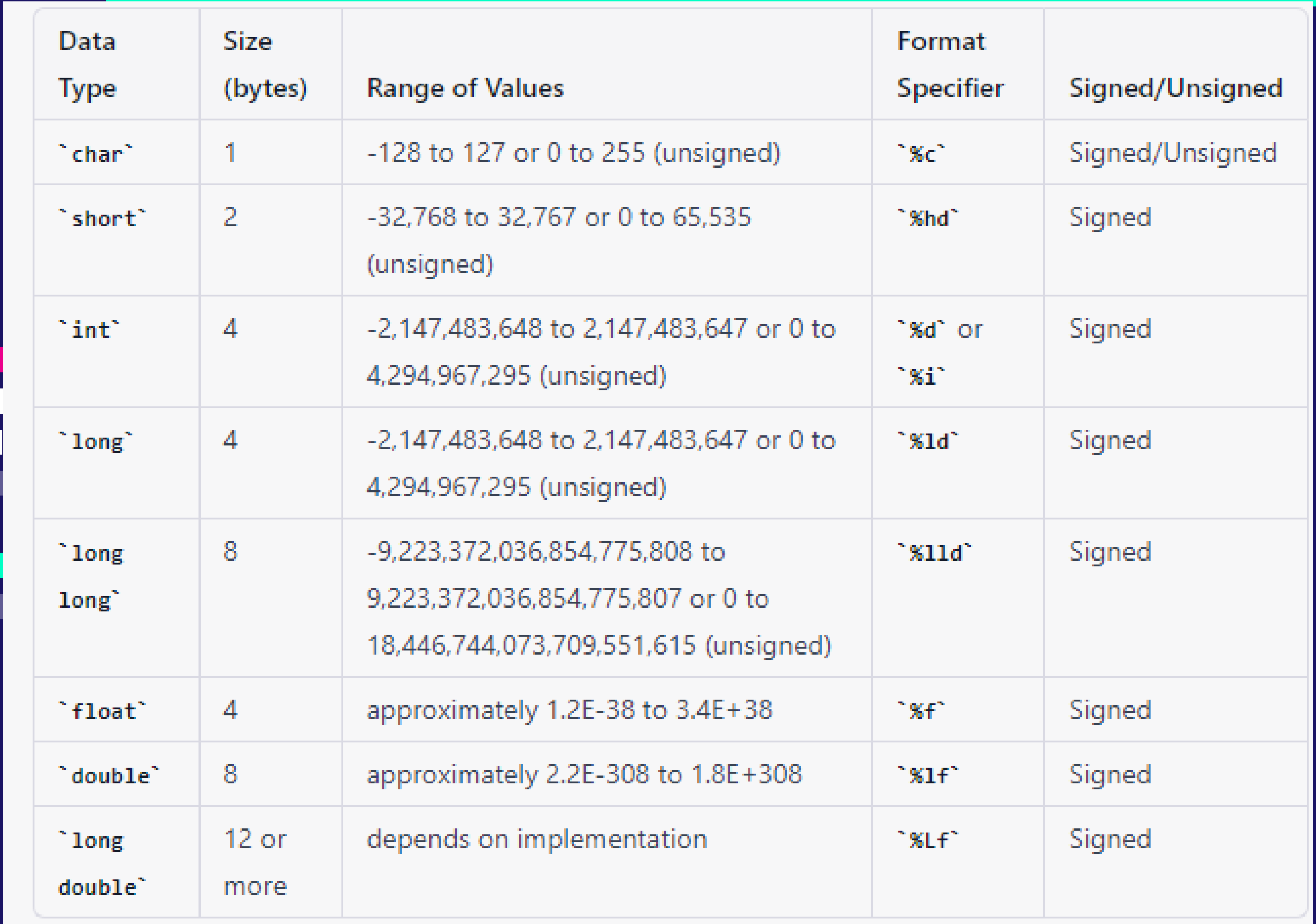
```
The size of my_var is 4 bytes
```

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

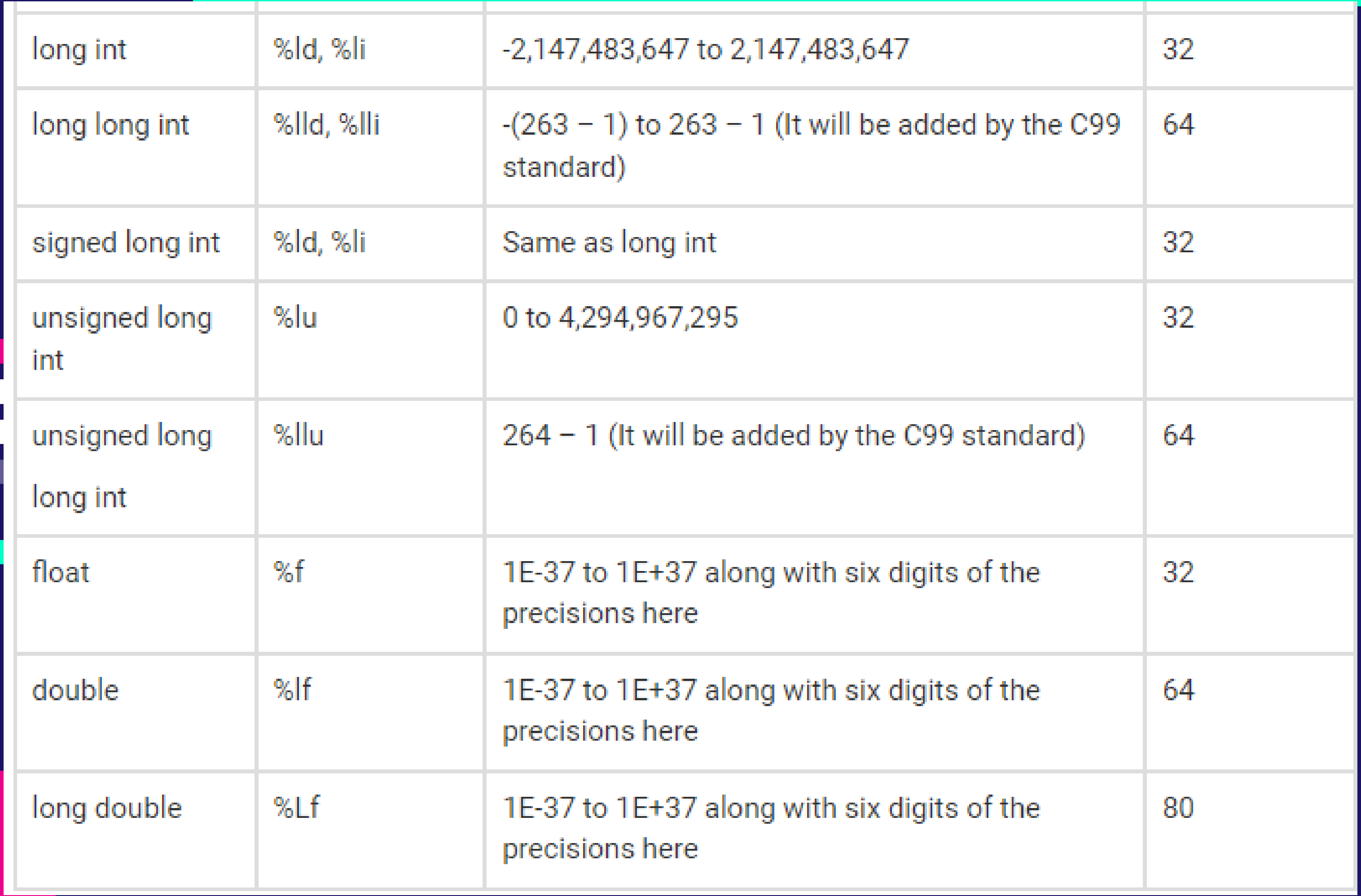


Data Type	Size (bytes)	Range of Values	Format Specifier	Signed/Unsigned
`char`	1	-128 to 127 or 0 to 255 (unsigned)	`%c`	Signed/Unsigned
`short`	2	-32,768 to 32,767 or 0 to 65,535 (unsigned)	`%hd`	Signed
`int`	4	-2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 (unsigned)	`%d` or `%i`	Signed
`long`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

signed or unsigned ?

data types can be either signed or unsigned. The signedness of a data type determines whether it can represent negative values or not.

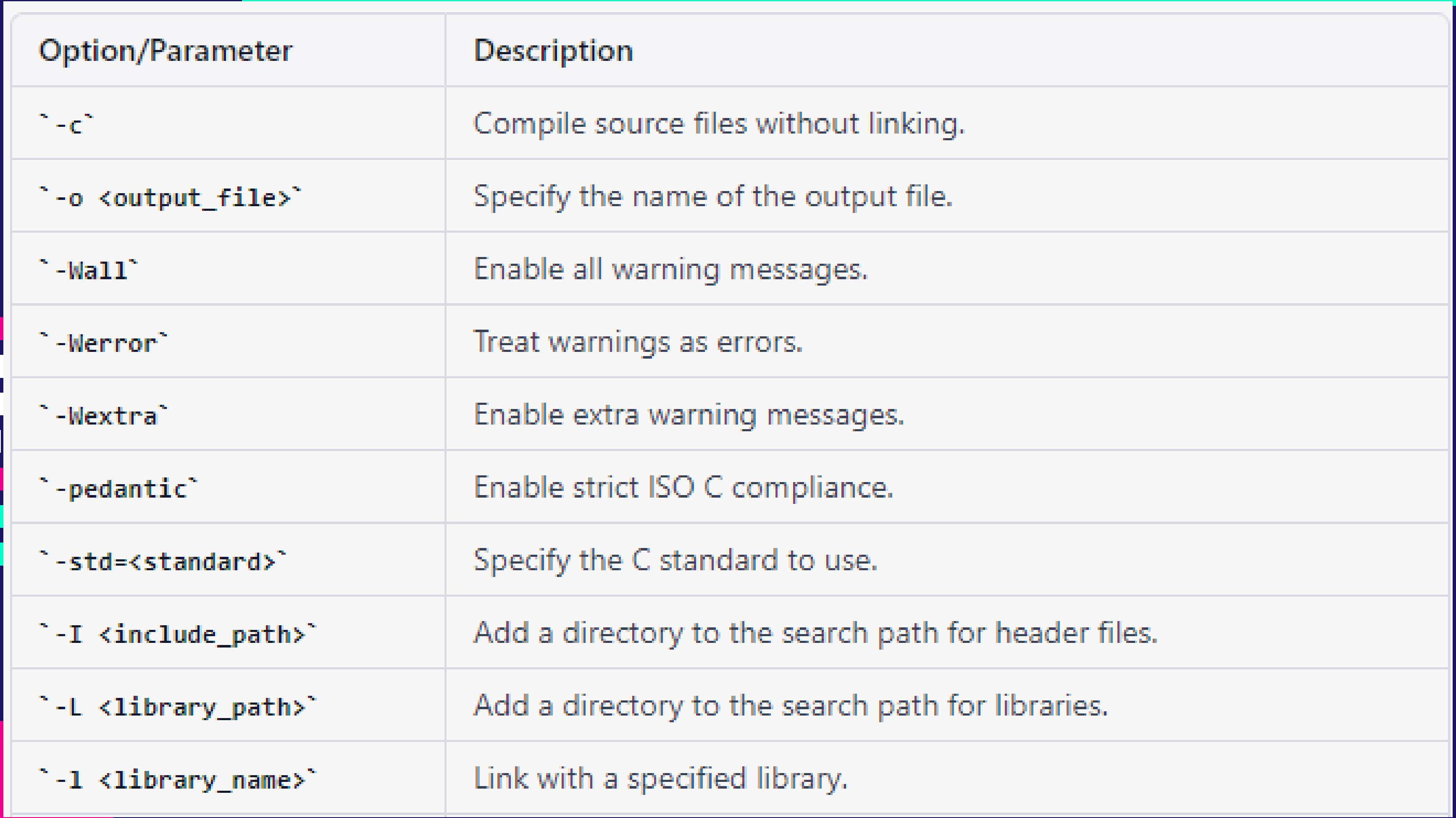
A signed data type can represent both positive and negative values, with one bit reserved for the sign (positive or negative). such as -5, -10, 0, 10, and 5.

An unsigned data type can only represent non-negative values (i.e., zero and positive values). such as 0, 10, and 5, but cannot store negative values such as -5.

signed or unsigned ?

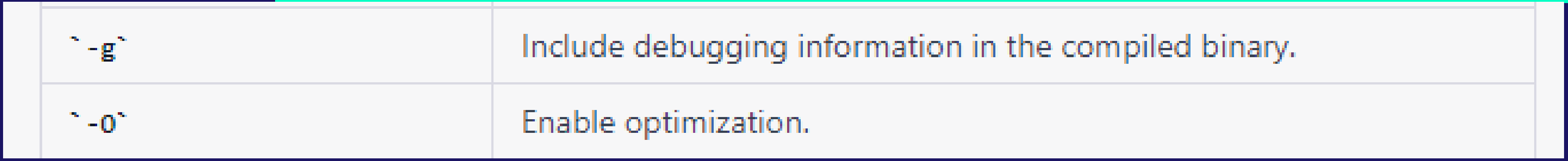
```c
#include <stdio.h>

int main() {
    // Signed integer variable
    int signedNum = -10;
    printf("Signed integer value: %d\n", signedNum);

    // Unsigned integer variable
    unsigned int unsignedNum = 10;
    printf("Unsigned integer value: %u\n", unsignedNum);

    return 0;
}
```

GCC parameters

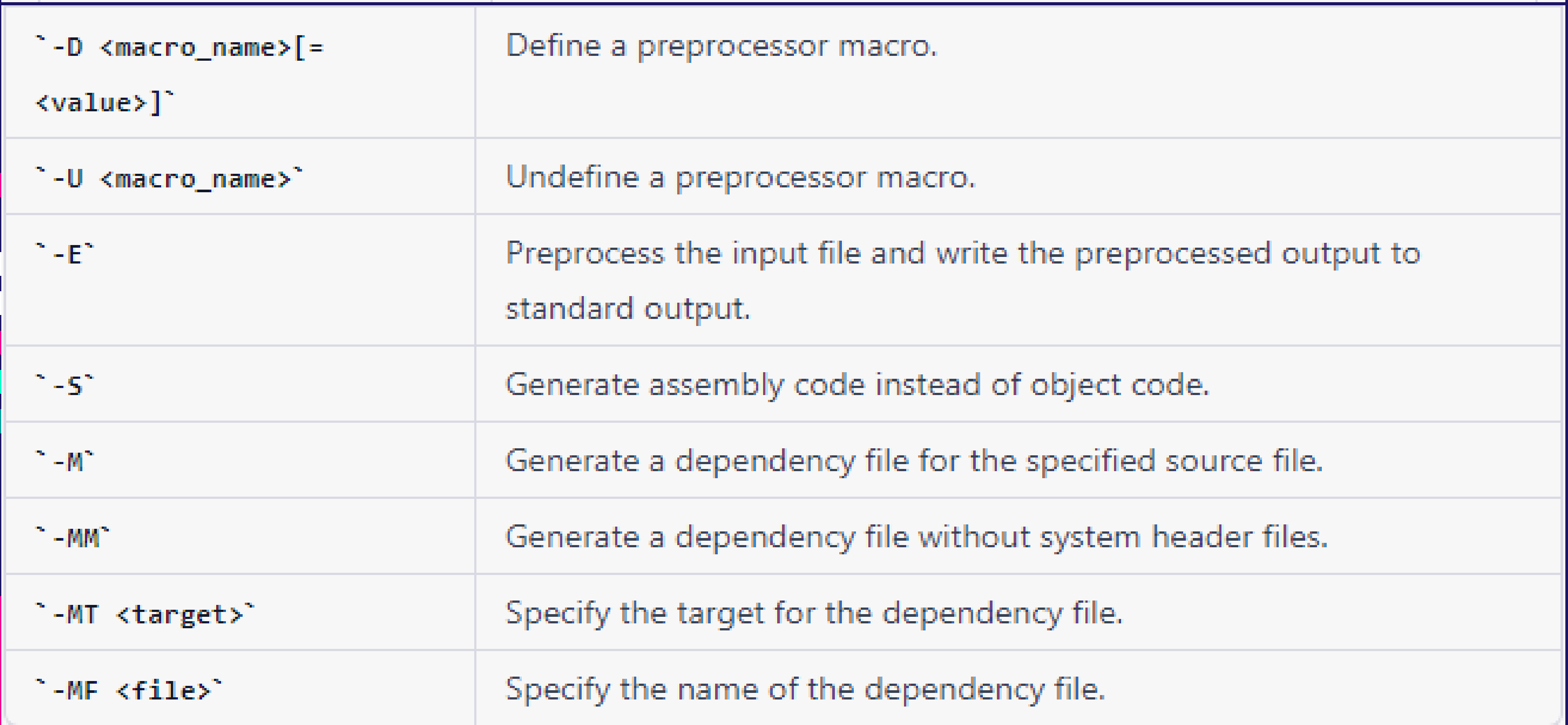


Option/Parameter	Description
`-c`	Compile source files without linking.
`-o <output_file>`	Specify the name of the output file.
`-Wall`	Enable all warning messages.
`-Werror`	Treat warnings as errors.
`-Wextra`	Enable extra warning messages.
`-pedantic`	Enable strict ISO C compliance.
`-std=<standard>`	Specify the C standard to use.
`-I <include_path>`	Add a directory to the search path for header files.
`-L <library_path>`	Add a directory to the search path for libraries.
`-l <library_name>`	Link with a specified library.

GCC parameters



`-g`	Include debugging information in the compiled binary.
`-O`	Enable optimization.



`-D <macro_name>[=<value>]`	Define a preprocessor macro.
`-U <macro_name>`	Undefine a preprocessor macro.
`-E`	Preprocess the input file and write the preprocessed output to standard output.
`-S`	Generate assembly code instead of object code.
`-M`	Generate a dependency file for the specified source file.
`-MM`	Generate a dependency file without system header files.
`-MT <target>`	Specify the target for the dependency file.
`-MF <file>`	Specify the name of the dependency file.

How to compile using gcc

to compile a C program using GCC (GNU Compiler Collection), you can use the following command in the terminal:

```
gcc <source_file.c> -o <output_file>
```

For example, if you have a C source file named hello.c and you want to compile it into an executable file named hello, you can use the following command:

```
gcc hello.c -o hello
```

What is the default program name when compiling with gcc

When compiling a C program with GCC (GNU Compiler Collection), the default name for the output program file is a.out. This stands for "assembler output" because it was originally designed to be used with the Unix assembler program as.

For example, if you compile a C program called main.c with the following command:

```
gcc main.c
```

The resulting executable file will be named a.out. You can then run the program by typing ./a.out in the terminal.

However, you can specify a different name for the output program file using the -o option, followed by the desired name. For example:

```
gcc main.c -o myprogram
```

This will generate an executable file named myprogram, which can be run by typing ./myprogram in the terminal.

what is masm

The -masm option in GCC is used to specify the assembly language syntax that should be used for the generated assembly code. The option takes a single argument that specifies the syntax convention to use, and it is typically used in conjunction with the -S option, which tells GCC to stop after generating assembly code.

There are two possible arguments for the -masm option:

-masm=intel: This option specifies that the generated assembly code should use Intel syntax, which is the syntax convention used by Intel and some other processor manufacturers in their documentation and programming examples. In Intel syntax, the destination operand is listed first, followed by the source operand, and instructions are generally easier to read and write.

-masm=att: This option specifies that the generated assembly code should use AT&T syntax, which is the syntax convention used by the GNU assembler (GAS) and some other assemblers. In AT&T syntax, the source operand is listed first, followed by the destination operand. Additionally, register names in AT&T syntax are prefixed with a percent sign (%), while in Intel syntax they are not.

By default, GCC uses the AT&T syntax convention for generated assembly code, but the -masm=intel option can be used to switch to Intel syntax if desired.

What is the official C coding style and how to check your code with betty-style

tThere is no one official C coding style that is universally accepted, as different companies, organizations, and individuals may have their own preferred style. However, there are several widely used coding style guidelines for C programming, including the GNU coding standards, the Linux kernel coding style, and the Google C++ Style Guide.

One popular C coding style guideline is the Betty style, which was developed for use in the Holberton School's curriculum. Betty is a set of rules and guidelines for writing clean and easy-to-read C code.

To check your C code using the Betty style, you can use the betty-style checker tool. This tool analyzes your code and checks for violations of the Betty style guidelines.

To use the betty-style checker, you can follow these steps:

Clone the Betty repository from GitHub:
git clone https://github.com/holbertonschool/Betty.git
you're working with.

Change into the Betty directory:

```
cd Betty
```

Run the betty-style script followed by the name of the C file you want to check:

```
./betty-style path/to/your/file.c
```

The tool will analyze your code and provide feedback on any style violations that it finds. You can then review the feedback and make any necessary changes to bring your code into compliance with the Betty style guidelines.

It's worth noting that while following a coding style guideline can help make your code more readable and maintainable, it's also important to balance this with your own personal preferences and the requirements of the project or organization you're working with.

How to find the right header to include in your source code when using a standard library function

When using a standard library function in your C code, you may need to include the appropriate header file to ensure that the function is properly declared and defined. Here are some steps you can follow to find the right header file to include:

1- Consult the documentation: The first step in finding the right header file to include is to consult the documentation for the function you want to use. The documentation should provide information on any header files that need to be included in your code.

2- Search online: If you don't have access to documentation or if it doesn't provide information on the required header file, you can try searching online for information on the function and its required headers. A search engine query like "C standard library function [function name] header file" can often yield helpful results.

3- Check the standard library headers: If you're not able to find the required header file through the documentation or online searches, you can try checking the standard library headers themselves. The headers are typically located in the include directory of your C compiler installation. For example, on a Unix-based system, the headers may be located in /usr/include. You can search through the headers for the function you're using to find the appropriate header file to include.

How to find the right header to include in your source code when using a standard library function

4- Use trial and error: If you're still having trouble finding the required header file, you can try including different headers and seeing if the compiler throws any errors. If it does, you can look at the error message to determine which header file is missing or incorrect.

It's important to note that including the wrong header file or failing to include a required header file can result in compilation errors or undefined behavior at runtime, so it's important to ensure that you've included the appropriate header file for each standard library function you use in your code.

How does the main function influence the return value of the program

The main function can return an integer value to the operating system, which can then be interpreted as an indication of the success or failure of the program. A return value of 0 typically indicates success, while a non-zero value indicates failure. This convention is widely used in Unix-like operating systems, but it's not universal across all operating systems.

For example, consider the following main function:

```
int main() {
    // program code
    return 0;
}
```

How does the main function influence the return value of the program

On the other hand, if the program encounters an error during execution, the main function can return a non-zero value to indicate the error. For example:

```
int main() {
    // program code that encounters an error
    return 1; // return non-zero value to indicate failure
}
```

In this case, the main function returns the integer value 1, which indicates to the operating system that the program has encountered an error and failed to execute successfully.

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