Introduction to C programing
C PROGRAMING FOR BEGINERS
MASTER THE C LANGUAGE
Udemy.com course

David Gabriel Corzo Mcmath

2019-08-11 00:47

Índice general

Course content

Section 1	
Lecture # 001	Welcome to class
Lecture # 002	Class organization
Lecture # 003	Fundamentals of a program
Lecture # 004	Overview
Lecture # 005	Language Features
Lecture # 006	Creating a C program
Section # 2	
Lecture # 007	
Lecture # 008	
Section # 3	
Lecture # 009	
Lecture # 010	
Lecture # 011	
Lecture # 012	
Lecture # 013	
Lecture # 014	
Section # 4	
Lecture # 015	
Lecture # 016	
Lecture # 017	
Lecture # 018	
Lecture # 019	
Section # 5	
Lecture # 020	
Lecture # 021	
Lecture # 022	
Lecture # 023	
Lecture # 024	
Lecture # 025	
Lecture # 026	
Lecture # 027	
Lecture # 028	
Section # 6	
Lecture # 029	
Lecture # 030	
Lecture # 031	
Lecture # 032	
Lecture # 033	
Lecture # 034	
Lecture # 035	
Lecture # 036	

Capítulo 1

Introduction (Section I)

1.1. Fundamentals of a programming language

1.1.1. Basics

- For all the diferent problems that we need solved in the world the computer runs an algorithm.
- An algorithm is a set of instructions that tell the CPU what to do.
- To write a program you need to implement the instructions correctly in any kind of programming languages, Java, Python, Objective-C, C & C++.

1.1.2. Terminology

- CPU:
  - Central Processing Unit
  - Excecutes the computational work.
  - All instructionsare excecuted here.
  - components are:
    1. Control unit
    2. ALU
    3. Registers
- RAM:
  - Random Access Memory
  - Stores the data of a program while it's running.
  - A programs needs to have data to operate, this is $\neq$ to the harddrive.
  - RAM is another hardware component and is much faster.
- HardDrive:
  - Stroes files that contains programs source code, it stores that data even when the computer is turned off.
  - When a computer is turned off there is nothing in RAM but there is DATA in the harddrive.
  - Used to store program resources.
- Operating system:

- Makes a computer usable and easier.
- It's a program that runs everytime you use.
- Handles input, output, computer resources, storage, etc.
- Windows, Unix, Android, etc.

- Fetch / Excecute cycle:
  - Fetch instructions from memory and sending it to the CPU to be excecuted.
  - Gigahertz says the amount of times the CPU fetches and excecutes a program.
- Higher level programming language:
  - Oposite of assembly language
  - C is higher level programming language.
  - You don't understand the instructions sent to the CPU, but in C you don't have to wory about anything like this.
  - A **compiler** interpretes the code writen in the higher level programming language and upon compiling it will translate your statements in that programming language in to machine code tha will then be sent to the CPU.
  - It's esentially a translation between C to machine code.
  - The compiler will also check wether your program has the correct syntax, it won't compile if there are errors.

1.1.3. Writting a program

1. Define the program objetives.
   - Understand the requirements.
   - See what you want to do.
2. Design the program
   - Decide how to program will meet the requirements.
   - What will be the UI.
   - Organization in your program.
3. Write the code:
   - Start the implementation.
   - Use an IDE.
4. Compile:
   - Translate the code to CPU.
5. Run the program:
   - Execute.
6. Test and debug:
   - Running $\neq$ properly impliemented.
   - Debug your errors.
   - Test by **SMALL** batches.

7. Maintain and modify the program:
    - Get your program updated and continue dubugging.
    - Continue fixing and add new features.
    - Tipically the most expensive step in the process.
    - Use a methodology that works for you.

1.1.4. Observations

- Planning before coding is easier and permits more flexibility.
- You always want to plan eficiently.
- **Divide and conquer** for debugging sake.

1.2. Overview & history

1.2.1. Overview

- C is general purpose, you can use C to do anything you like, you can write OS programs.
- Uses statements that change a program's state.
- C is imperetive, it's going to focus how to get things done and using statements to do so.
- It's the most widely used language.
- C is a modern language:
    - Basic control structures
    - Top down planning (diferent steps can give you the same result, you don't have to follow the watefall aproach).
    - Organized aroud the use of funtions
    - Reliable, it won't crash for no reason, it's readable.
- It's used in minicomputers, Unix/Linux for example.
- C is preferred language forproducing word processing programs, spreadsheets and compilers.
- C is popular for programming embedding systems, for example you see C being used in car components.
- C plays a strong role in the Linux operating system and the Unix base of MacOS.
- C programs are easy to adapt to new models or languages.
- 1990, Software houses began turning to the C++ language for the large projects.
- C is a subset of C++ with object oriented programming tools added:
    - Any C program is a valid C++ program, C is organized around functions and C++ is object oriented.
    - By learning C you learn C++.
- C remains a very demanding skill for professional applications.
- It's not an old language it's used, it's a valuable skill.
- C provides constructs that map efficiently:

  - It provides low-level access to memory, low level capabilities.
  - Requires minimal run-timesupport.
- Invented by **Dennis Ritchie** of bell Laboratories, 1972; for the design of UNIX operating systems, C being old is an **advantage**.
- Was crated as a tool for working programmers, main goal is to be usefull language.
  - No more will you need to get 20 lines of code to add two numbers.
  - Realability and writability.
- C initially became as the development language of the UNIX operating system, all mayor operating systems are written in C and/or C++.
- C evolved from a previous programming language named B:
  - Uses many of the important concepts of B while adding data typing and other powerful features.
  - B was a typeless language, every data item occupied one word in memory, and the burden of typing variables fell on the shoulders of the programmers.
  - There is also a language called D.
  - B didn't have data types.
- C is available for most computers.
- C is hardware independent.
- 1970s C had evolved to traditional C.
- Variations of C, there are diferent variations of C:
  - C89/C90: is supported by all current C compilers.
  - C99: revises and and expends the capabilities of C, not all compilers suport it.
  - C11: isn't suported by compilers and some features, focus on C99.
- Standardizing C made the program portable and now any machine could run it.
- C is one of the most important languages.
- From C it's easy to migrate to any other programming language like Java, or objective-C.

1.3. Language features

1.3.1. Main features

- Efficiency and portability:
  - Compact and fast, similar to assembly language.
  - It is such a low level you can maximize efficiency.
  - You can run it in several operating system.
  - C is the leader in portability.
  - Compilers are widely available.
  - Linux and Unix come with a C compiler.
  - C compilers exist in an array of forms, there are compilers for automobile C.
- Powerful and flexible:

- The unix/linux kernel is written in C, the kernel is the brain of the operating system.
- Many compilers and interpreters for other languages like FORTRA, Perl, Python, Pascal, LISP, Logo, Basic, etc.
- C programs are also used for solving physiscs and engeneering problems and for movie animations and special effects.
- C is flexible:
  - You can solve problems in diferent ways.
  - It's the basis for other programming languages.
  - It's a trade off.
- It is used for mobile phone apps, a form of objective C.
- Easy toi learn if it is your first language.

- Programer oriented:
  - Gives you access to memory and to hardware.
  - You can manipulate individual bits in memory.
  - Contains a large selection of operators.
  - Less strict, this is an advantage and a disadvantage, you can make something much better but you need to be more responsable.
  - Implementations have a large library of usefull C functions.
- Other features:
  - Provides bit manipulation.
  - Provides features of lower level languages.
  - You can access memroy by bits.
  - You can have pointers, it's memory related.
- Disadvantage:
  - Flexibility and freedom also requires added flexibility, poiters especially, errors are dificult to trace.
  - Sometimes because of it's wealth of operators it's dificult to read.
- Advantages of C:
  1. Fast and efficient
  2. Portable
  3. Variety of data types and powerful operators
  4. Easy to extend
  5. Modularity
  6. Function rich libraries

1.4. Creating a C program

1.4.1. Four fundamental tasks in the creation of a C program

1. Editing:
   - Process of creating and modiffying your C source code.

- The source code is inside a file that contains the program.
- Choose a wise name for your base file name (.c).
- We'll use an IDE (code blocks) for the editing task.

2. Compiling:

- Converts the source code and turns it in to machine code and detects errors.
- The input to the compiler is going to be the source code.
- It's a two stage process
  a) preprocessing fase (your code maybe modified or added to)
  b) generation of the assembly language code.
- The compiler will look at all the statements and shecks it to ensure that it confirms to the syntax and syntax.
- The compiler will detect dead code, typical errors such as syntax or semantic errors, but **not** the logic errors.
- The object code is a .o or .obj will be the assembly language statement into actual machine instructions.
- Run the command cc -c myprog.c of gcc -c myprog.c .
- If you omit this flag will automatically be linked as well.

3. Linking:

- After the program has been compiled, linking is getting all the dependencies of the .o or .obj then it will take all of them and create a .exe or a.out file which is an executable.
- This fase will also detect errors.
- Non-existing library component for example will be an error.
- The program libraries suport and extend the C language by providing functionality already built and for me to use.
- If there is a linking failiure means you need to re-edit your source code.
- The programs that are big need to be modular.

4. Executing:

- Just run the program by clicking on it, or on the command line.
- This fase will be for executing your program.
- This fase also consists of debugging fase.

These processes are assentially the same to developing programs in any enviroment.

Capítulo 2

Starting to write code (Section III)

2.1. Exploring the Code::Blocks Enviroment

2.1.1. Main

- The main function is the main functions, functions have parenthesis and the contents of these functions are defined inside { } symbols.
- You can't declare a variable by the name main.
- You can't not return anithing with an intiger is expected.

Capítulo 3

Basic Concepts (Section IV)

3.1. Comments

3.1.1. Comments

- The comments are a way of documenting a program, increasing readibility.
- They remind you or someone else what the code is doing.
- Comments are ignored by the compiler.
- A comment can save significant amount of time, it can make the program readible in the sense that everyone who wants to understand.
- To write a comment:
  - "/*" marks the beggining of a multiple line comment, to close you use "*/"
  - The single line comment can be done with the "//".
- Don't use comments in excess.
- Add the comments while you are doing the code, it's easier.
- Comments help when there are errors.
- Meaningful names are a form of documentation.
- See: **Clean Code**.

3.2. Preprocessor

3.2.1. Overview

- It's part of the compiliation fase, it does things before compiling, basically doing stuff before compiling.
- The include is an example of a preprocesor command or directive; the preprocessors statements begin with a pound sign as first character in teh line, "# include ¡stdio.h¿".
- Preprocessor usage:
  - Create your constants and macros with the #define statement.
  - Include library files use the #include statement.
  - More powerful programs with the conditional #ndif, #else, #ifndef statements.
- This all happens before compiliation.

3.3. The include statement

- Notice the syntax: "#include ¡stdio.h¿", we know that this is a preprocesor directive.
- Directive = Statement
- Not strictly part of the program necesarily but the compiler will complain if you don't have it.
- These directives can be anywhere in the source code but it's convention to have them in the very first line.
- This statement says to the program to get the header files needed to compile correctly. The ".h" is a header files, they are called headers because they must be the first thing (by convention) in your source code.
- Header files:
  - They define information about some of the funcitons used in the source code.
  - The stdio.h is a C standard library file, for example "printf()", it comes with the C programming language.
  - stdio.h is short for standard input/output, which means that we are going to use this library for input output operation.
  - Header files specify infromation that the compiler uses to integrate any predefined functions within a program.
  - We'll create our very own header files.
  - It helps with the modularity feature of quality code.
- Syntax of header files:
  - Header files are case sensative, always use lower case.
  - You can call header files in one of two ways:
    - With angle brackets: #include ¡DavidsHeaderFile.h¿; this syntax is used to call for the header specified in **one or more standard system directories**.
    - With double quotes: #include "DavidsHeaderFile.h"; this double quote syntax tells C to look for the header file in the **current directory**.
  - The double quotes are usually for libraries made by developers.
  - Every C compiler that conforms to the C11 standard will have a set of standard header files supplied with it.
  -
- #ifndef and #define to protect against multiple inclusions of a header file; this is used to not import headers more than once.

3.4. Display output

- The printf statement displays output to the screen.
- It's a C standard library it outputs information to the command line, by default output is the command line.
- The information displayed is what is inside the parenthesis of the printf function.
- Usage: The printf can print values of variables, the results of computations, and for this it is a very powerful debugging tool.

3.5. Reading input from the terminal

- It's very useful to have user input in your program, there are an array of ways of doing this, one such way is the command line or terminal arguments, another is to ask explicitly in the terminal.
- We'll use scanf, it's the most general.
- scanf can be used to get text in a file as well.
- This function reads from the standard input stream "stdin" and scans that input according to the format provided; %s, %d, %c, %f, etc; for floats, strings, chars, integers.
- The stdin stream is by default trated by text and then convert to the desired dta type like %d.
- 2014 → chr 2 0 1 4 → convertion to int 2014.
- The scanf uses pointers to variables to store the inputed data.
- 3 Rules of scanf:
  1. It will return the number of items that is successfully reads.
  2. To store values in a variable get the variable and set it as scanf's second argument preceeded by an ampersand &.
  3. If you use scanf to read a string into a character arry, don't use an &.
- scanf has to have two arguments, the first is the data type you will need to be converted, the second will be the & variable, if the variable is anything but a string you **don't** use the ampersand.
- scanf uses whitespaces (newlines, tabs and spaces) separates into fields.
- Remember to specify the data you input.
- When a program uses scanf it pauses the collect the input and will continue upon detecting the return key.
- Example:

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

int main()
{
    char str[100]; // a string array of 100.
    int i;         // integer a

    printf("Enter a value :");

    scanf("%d %s", &i, str);

    printf("\nYou entered: %d %s\n", i, str);

    return 0;
}
```

- To read a double you can use the specifier %lf.
- Problems with scanf is that if you execute multiple scanf's they won't execute properly, flush the data out with getchar.

Capítulo 4

Variables and data types (Section V)

4.1. Overview

- When a program needs to store the instructions of it's program and teh data that it act's upon while your computer is executing that program.
- This information is stored in RAM.
- Harddrives store presistent data, RAM is eliminated when the computer is turned off.
- Think of RAM as an ordered sequence of boxes:
  - The bow is full when it represents a 1 or the bow is empty when it represents 0.
  - Eaxh bow represents one binary digit called a bit.
  - Bits in memory are grouped into sets of eight called a byte.
  - Each byte has a label, this is a number, this is referenced with this label.
  - Each byte has a unique reference.
- A program is only going to be powerful if it can manipulate data.
- So we need to understand the different data types you can use.
- Constants are types of data that do not change and retain their values throughout the life of the program.
- Variables are types of data that may change or be assigned values as the program runs, these values can change, this is the oposite of the constant, variables and constants are stored in memory and referenced using these hexadecimal adresses.
- Variable names reference these hexadecimal adresses so that you don't have to memorize them.
- You can access these variable by their name.

4.1.1. Naming variables

- In C all the names must begin a letter or underscore and what ever follows them.
- Valid:

```
David; myFlag; j5x7; _anotherVariable
```

- Invalid:

```
temp$value // has a dollar sign
my Flag // has a space
3jason // starts with number
int //reserved word
```

- Use meaningfull names, this is a documentation process as well.

4.1.2. Data types

- The computer needs a way to identify what dat type it's processing.
- Do this by preceding them by the data type for example "int myvar"; this also corrsponds to how much momory you need for the type being declared.
- Primitive data types , in C everything is a primitive data type, there are no objects in C.

4.1.3. Declaring variables

- When you declare the variable you must do this using the format of first the data type (int, float, double, char, etc) and then your variable name.
  - *type-specifier variable-name;*
- You can declare multiple varibles with comas.
- Variable declaration is just saying what your program has and will use.
- Example:

```
int x;
int x,y,z;
```

  C will allocate the memory for x, y and z; but they are null.

4.1.4. Initializing variables

- After declaration of the variables you assign a value to that variable:

```
int x = 0;
int y = 9, r = 5;
```

4.1.5. Example

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

int main()
{
    int david = 90; // declaration and initialization

    david = 8; // modification a that variable

    return 0;
}
```

4.2. Basic Data types

- C suports diferent types of data.
- The basic data types in C are the:
  - int
  - float
  - double
  - char
  - _Bool
- The diference between these types are the amount of memory that is going to be allocated to store them; this is important for the compiler and for the programmer.
- Difereces:
  - The amount of memory.
  - Machine dependent for memory allocation. A windows machine can allocate more or less memory depending on the machine.

4.2.1. Int

- Data type to store integral values only.
- You can use negative values, it can store all the values of $\mathbb{Z}$.
- You can assign a hexadecimal number, to do this use `int something = 0xFFEF0D;`.
- Rules:
  - No embedded spaces are permitted between the digits.
  - Big numbers can't be written with comma separators.

4.2.2. Float

- Data type for numbers with decimal places.
- Floating-point such as 3.14
- You can represent scientific notation in C, an example is `1.7e4` one point seven times 10 e to the power of four.

4.2.3. Double

- Very similar to the float, but can store a bigger float.
- A double allocates two times the memory allocated for the float.
- All floating point constant are taken as doubles by the compiler.
- To make somehting a float append f at the end ot the decimal number, 14.5f.

4.2.4. _Bool

- Data types to store the values true or false, 0 or 1, it's for binary choices.
- Will store a true or false value.
- 0 is used to indicate a false value, 1 indicates a true value.
- They are stored as 0 or 1 not as true or false.
- With C89 compiler you can use bool prefix for this data type.

4.2.5. Other data types

- There are other data types that allows you to have a more specified type of int for example, use:
  - short
  - long
  - unsigned
- This is an extra adjetive that allows you to be more eficient.
- Short int uses less memory than an int.
- Long use more memory than an int.
- A double is a type of long float.
- You can use long long adjectives.
- You can use them like this:

```
long int = 98L;
```

- If you dont want an int to store a negative number you can use the unsigned adjective, this is in theory, some compilers don't enforce it. Basically to extend the int to be $\mathbb{Z}^+$.
- You can combine these such as short int, signed short, long unsigned.

4.3. Enums and chars

4.3.1. Enums

- Enums are data type that allows a programmer to define a variable and specify the valid values that could be stored in that variable.
- It allows you to esentially create your own data type.
- To use enums you need to give it a name, use the enum keyword, after this provide a name of that enum, after list the values available.
- Example: `enum primaryColor {red, yellow, blue};` this can only store the values defined inside the curly braces.
- Variables that are declared enum have to follow the definition standards in the declaration.
- You can declare it the same way as a variable, you can declare them and them.
- If you specify a value and then call a diferent one we get a compiler error.

4.3.2. Enums as ints

- The enums are stored as ints and have to be called that way, this is important.
- Access this in the same way you would a list.
- Each element is asociated by intigers, you can assign values explicitly that are sociated, for example: `enum direction {up, down, left=10, right};` , you can control left and set it to 10.

4.3.3. Char

- Chars represent a single character such as the letter 'a'.
- You can declare the char data type, use the char keyword, and assign a value **inside single quotes ''**.
- **Single quotes** are for chars, **double quotes** are for strings.
- A char can be a number valid in the ascii table, for example `char grade = 65;` will be A in the ascii table.

4.3.4. Escape characters

- These characters represent an action.
- You can represent this actions to represent a newline for example; for example `char x = '\n';` this will be a new line, it won't print out in the console as `'\n'` it will just print a new line.
- See: C Primer Plus, Prata.
  - `\a` Alert (ANSI C)
  - `\b` Backspace
  - `\f` Form feed
  - `\n` Newline
  - `\r` Carrige return
  - `\t` Horizontal tab
  - `\\` Vertical tab
  - `\'` Backslash
  - `\"` Single quote
  - `\?` Question mark
  - `\0oo` Double qoute
  - `\xhh` Hexadeciml value, h is a hexadecimal digit.

4.4. Format specifiers

4.4.1. Overview

- They are used to display data as output sotred in a variable.
- They specify the type of data of the variable to be displayed.
- Example: `printf("%s", var);`
- printf is a function, a function is a block of code, a function can be invoqued and called to do something.
- When you print a variable you must specify the data type that will be outputed.

- printf takes two arguments, the first one is the text to be displayed, the variable will be formated to the %s, or %d for example, the second parameter is the variable.
- The percentage symbol is the format specifier, inmedietly after the percent symbol the letters represent an identity for a data type.
- Format specifiers:
  - char → `%c`
  - _Bool → `%i, %u`
  - short int → `%hi, %hx, %ho`
  - unsigned short int → `%hu, %hx, %ho`
  - int → `%i, %x, %o`
  - unsigned int → `%u, %x, %o`
  - long int → `%li, %lx, %lo`
  - unsigned long int → `%lu, %lx, %lo`
  - long long int → `%lli, %llx, %llo`
  - unsigned long long int → `%llu, %llx, %llo`
  - float → `%f, %e, %g, %a`
  - double → `%f, %e, %g, %a`
  - long double → `%Lf, $Le, %Lg`
  - strings → `%s`
- Format specifiers can be used more than once if you specify the data type, you can send more arguments, the new arguments (3 an so on) will be more variables to be formated using the format specifiers.
- Width specifier, in floating point numbers and intigers values that have lots of decimals you can specify the cuantity of digits it will print out, example:

```
int main(){
    float x = 3.999192;
    printf("%.2f", x)\neg // will print 3.9
}
```

  The width operator starts with a point and then a number that represents how many digits will be printed.
- This can be used for rounding floating points as well.
-

4.5. Command line arguments

4.5.1.

- Command line arguments are a way of passing data in to your program.
- Requires the user to enter small amounts of information in the terminal.
- You can pass this data in the command line, you can request this data in many ways using request the data in the program, or you can use the command line.
- The main function can recieve command line arguments, the first argument must be the number of arguments, this will be an intiger value; the second argument is an array of character pointers.

- First entry in this second argument is a pointer to the name of the program that is executing.
- The first argument is regarded as argc, the decond is regarded as argv.
- Example:

```
int main ( int argc /* this will be the first argument*/ , char *argv
↪  /*the argv is the second */)
{
    int numberOfArguments = argc;
    char *argument1 = argv[0];
    char *argument2 = argv[1];
    printf("Number of arguments: %d", numberOfArguments);
    printf("Argument 1 is the program name: %s", argument1);
    printf("Argument 2 is the program name: %s", argument2);

    return 0;
}
```

- This is a way of passing data to the program withut asking for them.
- You can access many arguments in the array (the second way), this way you don't have to enter anything.

4.6. Print the area of a triangle