Programming I

Introduction

Dimitrios Michail

Dept. of Informatics and Telematics
Harokopio University of Athens

Bibliography

- Deitel & Deitel, "C How to Program", 7/3, 2013.
- B. W. Kernighan & D. M. Ritchie, "The C Programming Language", Prentice Hall, 2/e, 1988.

Material

The lecture material is available at:

e-class platform

`http://eclass.hua.gr/courses/DIT135`

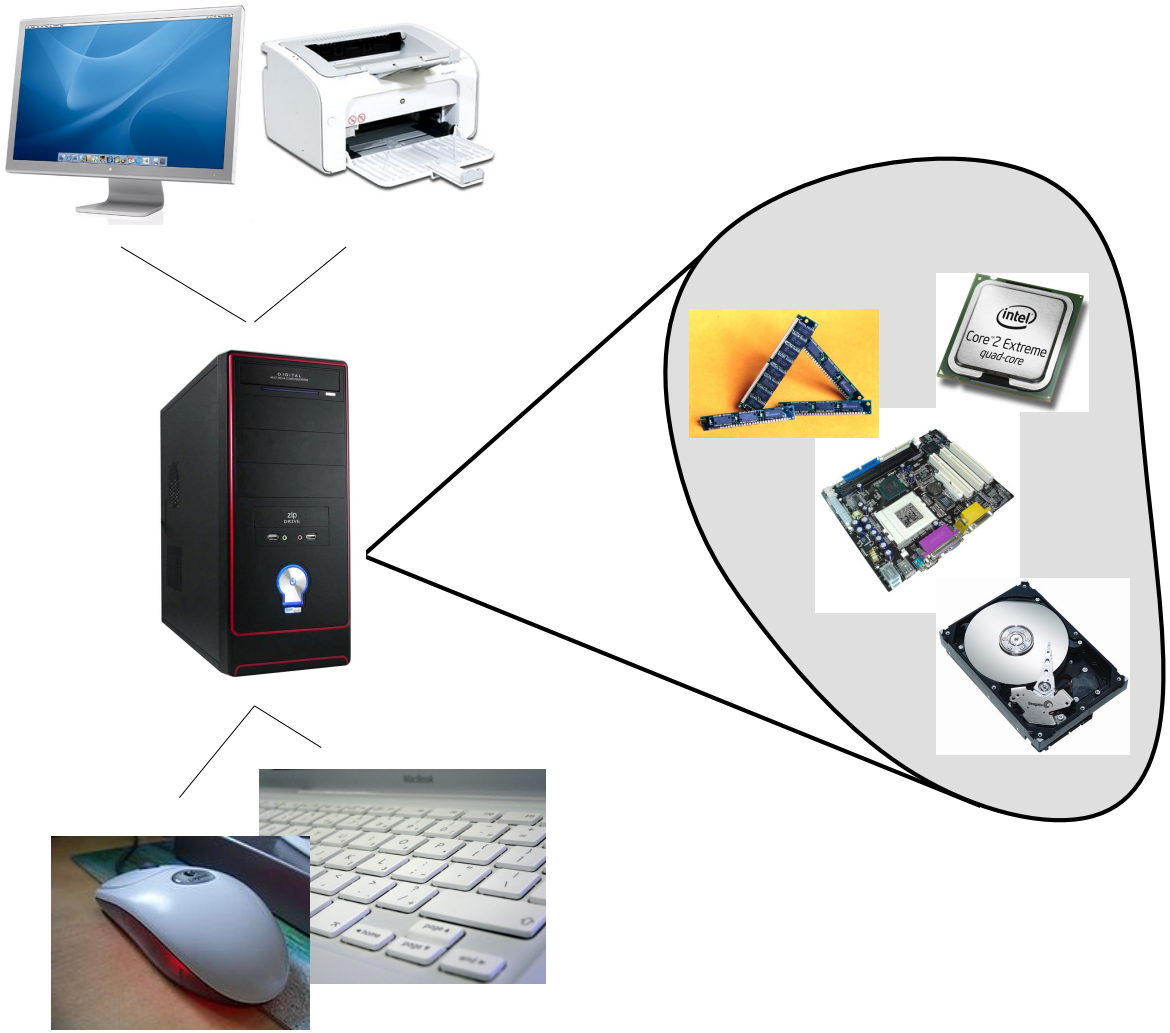
What is a computer?

A computer is a device which can execute calculations and take decisions, billion times faster than humans.

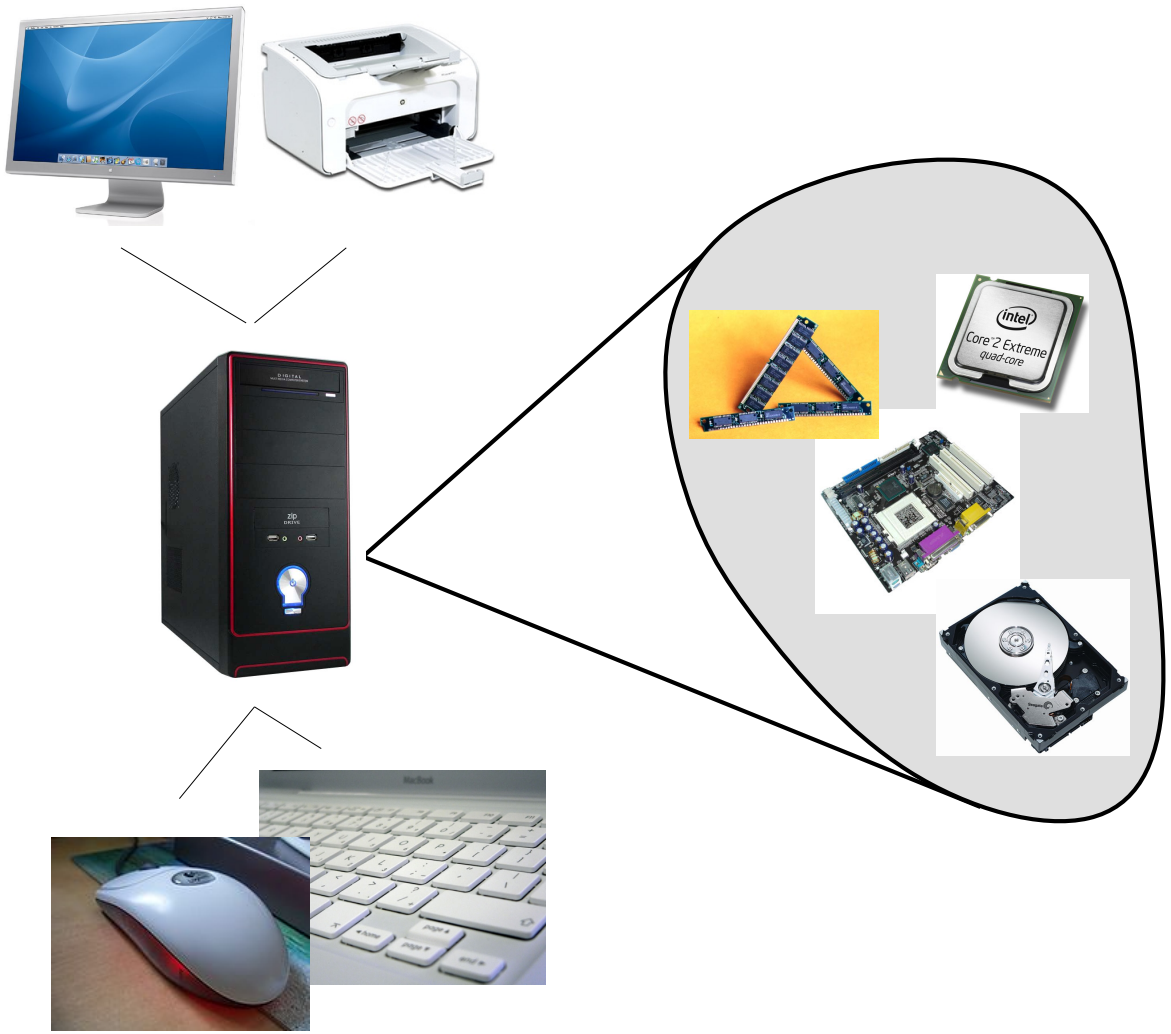
But in order for a computer to know what to do, someone must provide the instructions.

In this lecture we will learn how to give instructions to a computer.

The Anatomy of a Computer



The Anatomy of a Computer



Input Devices:
keyboard, mouse, touch screen, etc.

The Anatomy of a Computer

Output Devices:
Screen, printer, etc.

The Anatomy of a Computer

Main unit: central processing unit (CPU), memory unit, secondary-storage (e.g. SSD, Hard-Drive), arithmetic and logical unit (usually inside the CPU), bus for communication between different units, etc.

Software

The computer needs instructions. This role is played from different software components:

- **BIOS:** program which helps the computer boot

Software

The computer needs instructions. This role is played from different software components:

- **BIOS:** program which helps the computer boot
- **operating system:** program which takes control after the booting sequence and manages the resources of the computer, its input and output devices and generally takes care about the execution of other programs (e.g. Linux, MacOSX, Windows, SunOS, etc.)

Software

The computer needs instructions. This role is played from different software components:

- **BIOS:** program which helps the computer boot
- **operating system:** program which takes control after the booting sequence and manages the resources of the computer, its input and output devices and generally takes care about the execution of other programs (e.g. Linux, MacOSX, Windows, SunOS, etc.)
- **general applications:** programs build for a specific purpose, e.g. playing movies, image processing, word editing, games, etc.

What can a Computer Understand?

A computer can understand two states: 0 and 1

What can a Computer Understand?

A computer can understand two states: 0 and 1

But a human cannot easily talk to a computer using 0s and 1s.
e.g.

000000 00001 00010 00110 00000 100000

in some architecture tells the computer to read register 1 and register 2,

add them up and write the result into register 6.

From Machine Language up to today

- machine language
- *assembly language* contains english shortcuts for commands but still very close to the machine instructions, e.g.

  ADD32 6, 1, 2

  A program called the *assembler* takes the responsibility of translating between assembly and machine code.
- *high-level languages* which allows us to which instructions in almost spoken english which contain common mathematical symbols, e.g.

  x6 = x1 + x2;

  A program called *compiler* translates between a high-level language and an assembly language (or sometimes directly to machine code).

The History of the C Language

1969-1973	AT & T Bell Labs, Dennis Ritchie
1978	"The C Programming Language" K & R: Kernighan & Ritchie
1983	ANSI Standardization Committee X3J11
1989-1990	Acceptance of ANSI/ISO Standard (ANSI C)
1999	Revised the standard C9X (C99)
2011	C11 add numerous new features (atomic operations, multi-threading, etc.)
2018	C18 includes no new features, only technical corrections and clarifications to defects in C11.

Characteristics of C

- Small size

Characteristics of C

- Small size
- Usage of functions

Characteristics of C

- Small size
- Usage of functions
- Loose type system

Characteristics of C

- Small size
- Usage of functions
- Loose type system
- Structured language

Characteristics of C

- Small size
- Usage of functions
- Loose type system
- Structured language
- Low-level programming

Characteristics of C

- Small size
- Usage of functions
- Loose type system
- Structured language
- Low-level programming
- The programmer has full control (and power) but is responsible for her/his mistakes.

Characteristics of C

C is well known for various reasons:

- has high level structures
- can be used for low-level programming
- produces very efficient programs (speed and memory footprint)
- can be compiled into many many platforms

The syntax of a lot of languages which you will learn like C++, Java, C#, Javascript, etc. is based on C.

Structured Programming

A program is organized into small, self-contained entities. Each such entity must have a well defined entry and exit point.

In structured programming we use the following mechanisms in order to implement algorithms:

1. sequence
2. selection
3. iteration
4. functions

Structured Programming

A program is organized into small, self-contained entities. Each such entity must have a well defined entry and exit point.

In structured programming we use the following mechanisms in order to implement algorithms:

1. sequence
   in C all statements are executed one after another in the order that they appear in the source code.
2. selection
3. iteration
4. functions

Structured Programming

A program is organized into small, self-contained entities. Each such entity must have a well defined entry and exit point.

In structured programming we use the following mechanisms in order to implement algorithms:

1. sequence
2. selection
   one or a number of statements is executed depending on the state of the program. This is implemented using **if-then-else**.
3. iteration
4. functions

Structured Programming

A program is organized into small, self-contained entities. Each such entity must have a well defined entry and exit point.

In structured programming we use the following mechanisms in order to implement algorithms:

1. sequence
2. selection
3. iteration
   a statement or a block of statements can be executed multiple times until the program reaches a certain state. C provides several structures for loops like **for, do..while**.
4. functions

Structured Programming

A program is organized into small, self-contained entities. Each such entity must have a well defined entry and exit point.

In structured programming we use the following mechanisms in order to implement algorithms:

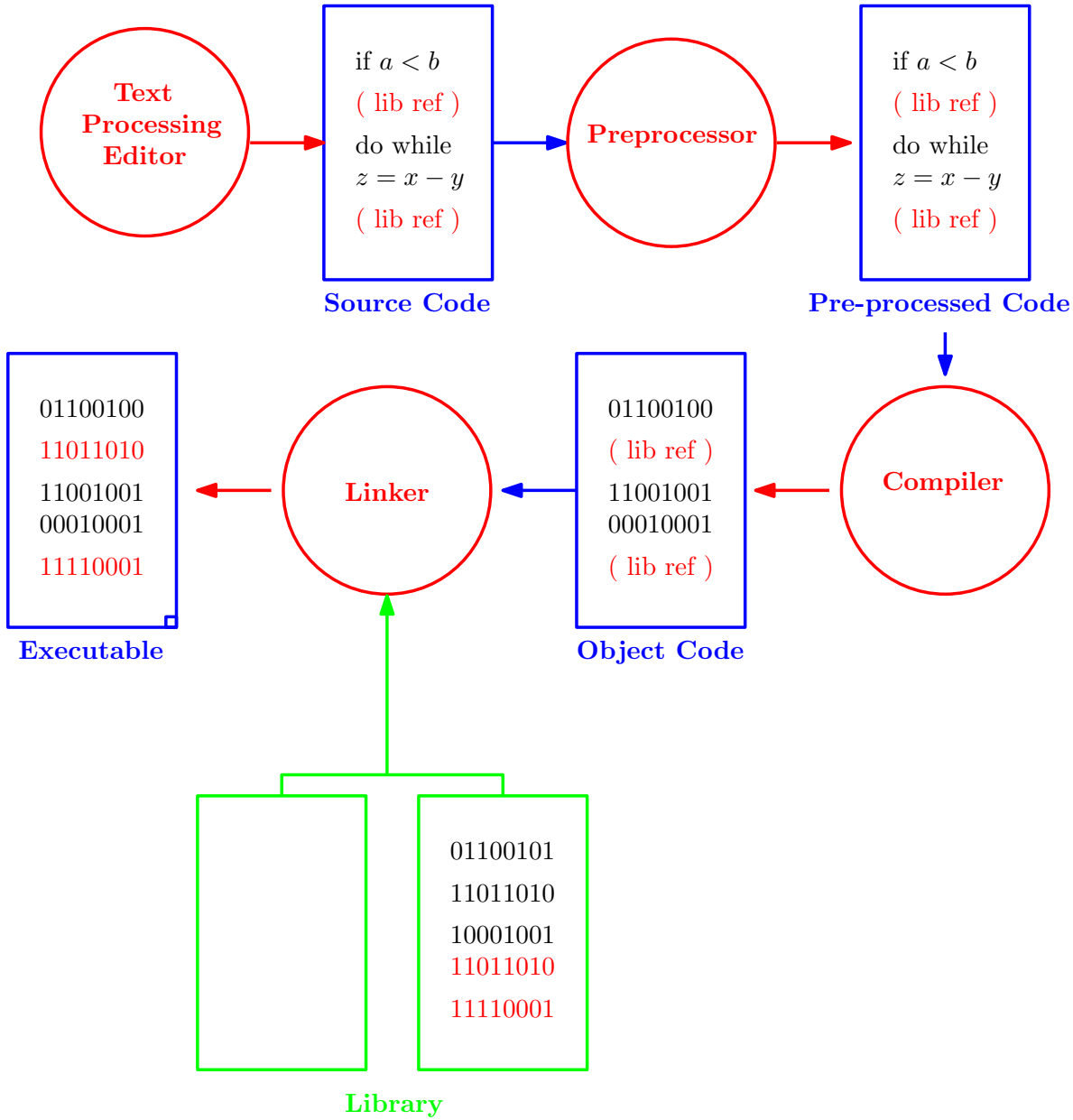
1. sequence
2. selection
3. iteration
4. functions
   a sequence of program instructions that performs a specific task, packaged as a unit. We have a separate chapter for functions in C.

My First Program in C

```c
/*
 * Hello world program in C
 */
#include <stdio.h>

int main() {

    printf("Hello, world!\n");

    return 0;
}
```

which tells the computer to print the message "Hello, world!" to the screen.

Compilation



Detailed Explanation

```
/*
 * Hello world program in C
 */
```

Anything inside

```
/*
*/
```

are comments which are completely removed and ignored by the compiler (pre-processor).

Detailed Explanation

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

Gives the instruction to the compiler (more specifically to the pre-processor) to find and read file *stdio.h* which includes declarations of several function which help print to the screen.

In this particular case, function `printf()`.

Detailed Explanation

```
int main()
{
```

and

```
}
```

is the definition of function `main()` which must exist in every C program and designates the start of a program.

Function `main()` returns an integer (`int`) after it finishes executing to the user.

The curly brackets denote the beginning and end of the function.

Detailed Explanation

```
printf("Hello, world!\n");
```

calls a function (declared in the file stdio.h) which prints to the screen the character sequence (string)

Hello, world!

the sequence `\n` denotes a special character in C which means newline and forces the computer to switch line in the screen.

The whole **statement** must end with a semicolon.

Detailed Explanation

```
return 0;
```

this statement tells the program to finish the execution of function `main()` and return to the user the value `0`.

General Guidelines

- use comments (where appropriate) to make code readable to others
- use approriate names for identifiers, functions, types, etc. which clearly express the functionality
- always write code which is properly indented (or use a proper editor which automatically does it for you)
- write simple programs, try to avoid "programming tricks"

Example of Bad Programming

```
#include <stdio.h>

main(t,_,a)
char *a;
{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86, 0, a+1 )+a)):1,t<_?main(t+1, _, a ):3,main ( -94, -27+t, a
)&&t == 2 ?_<13 ?main ( 2, _+1, "%s %d %d\n" ):9:16:t<0?t<-72?main(_,
t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+\
,/+#n+,/#;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!/\
+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;# ){n\
l]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#\
n'wk nw' iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;\
#'rdq#w! nr'/ ') }+}{rl#'{n' ')# }'+}##(!!/")
:t<-50?_==*a ?putchar(a[31]):main(-65,_,a+1):main((*a == '/')+t,_,a\
+1 ):0<t?main ( 2, 2 , "%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc \
i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```

The above code can be compiled!!