C Programming

Lecture 1 : Introduction

Based on slides designed by Shay Artsi, Aythan Avior, Gitit Rockstein and Saher Ismir for the department of Computer Science, Technion, Israel Institute of Technology
Translated and updated by Anne Weill-Zrahia
Contact info

Anne Weill-Zrahia
Phone : 8294997
Email : anne@cc.technion.ac.il
Administration

Course information:
Moodle.technion.ac.il >
International School of Engineering > Computer Science>
Introduction to Computer Language C - 234112
Frequently asked questions

Q: I have taken programming in language X (Java, C++, Visual Basic, you_name_it) should I attend the lecture?
A: Yes, there are features of C that you probably don’t know.

Q: I have taken a course in C programming – should I attend?
A: You can skip the first half of the semester. You should attend when we start talking about pointers.
Frequently asked questions

Q: Do we submit assignments in pairs?
A: No, assignments are in single mode

Q: I have questions about an assignment, whom should I ask?
A: The TA

Q: I have another problem, whom should I ask?
A: Myself
What will be taught in the course

- Principles of computer programming
- Algorithmic approach to problem solving
- Translating algorithms to C programs
- The course is for beginners, it will not provide a complete presentation of the C language
Computing applied to science and engineering

- Physics: simulation of processes in new planet formation
- Biology: discovering the human genome
- Chemistry: discovery of special molecules
- Aeronautics: replacing construction of a wind tunnel by a computer simulation
- Engineering: programs for calculation of structures
- Environmental engineering: air pollution control, geophysical fluid dynamics
.. And vice –versa...

Different fields of science can help in the development of computer’s architectures:

- Physics: quantum computing
- Biology: cell processors
- Medicine: programs for calculation of structures
What will you learn in the course

Will I be able to program a wind tunnel simulation/strength of material calculation?

- Probably not yet.

The skills you will certainly acquire:

- Technology of computers: how they work and how to program them
- Basic knowledge of one programming language: C
- Being able to write, debug and run simple programs
Skills you will get in the course

- Writing code for simple calculations you may need in your work.
- For more complex programs, you will be able at least:
  1. To determine if there is a computational solution at all
  2. To better communicate with professional programmers who may be involved in your project: complex simulations are the result of team work between scientists, engineers and programmers
Programmer’s jargon

Some words that will be used a lot:

- **Source code**: The stuff you type into the computer. The program you are writing.
- **Compile (build)**: Taking source code and making a program that the computer can understand.
- **Executable**: The compiled program that the computer can run.
- **Language**: The core part of C central to writing C code.
- **Library**: Added functions for C programming which are bolted on to do certain tasks.
- **Header file**: Files ending in .h which are included at the start of source code.
Why learn C

- C is *small* (only 32 keywords).
- C is *common* (lots of C code about).
- C is *stable* (the language doesn’t change much).
- C is *quick running*.
- C is the *basis for many other languages* (Java, C++, awk, Perl).
- It may not feel like it but C is one of the easiest languages to learn.
Let’s get started ...
Hardware and Software

- **Hardware**  Refers as devices you can actually touch

- **Software**  : Computer instructions or data

Software exists as ideas and concepts. A computer without software is like a book full of blank pages -- you need software to make the computer useful just as you need words to make a book meaningful.
Input devices
The CPU (Central processing Unit) is the brains of the computer. The CPU is where most calculations take place.

Example: Intel i7 core 3.2 Ghz, phys. Size: 5x5 cm.
Memory

- Data storage that comes in the form of chips
- The computer can read from it or write to it.
- Several types of memory are classified by:
  1. Size (how much data can reside on it)
  2. Mobility
  3. Volatile/non volatile
  4. Speed
  5. Price
Main Memory

Also called RAM (Random Access Memory)

- Static
- Expensive
- Is erased when the computer is turned off
Secondary storage

- Usually the hard disk
- Keeps digital data
- Larger (x50 or 500) than the RAM
- Slow magnetic

SSD
Portable memory

- Can be slow (but not always)
- Comes in varying sizes
- Data is kept after switching off the computer
Bus

- Communication device to transfer data between different devices
Scientists from the RAND Corporation have created this model to illustrate how a “home computer” could look like in the year 2004. However, the needed technology will not be economically feasible for the average home. Also, the scientists readily admit that the computer will require not-yet-invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use and only
What is stored in the memory?

- The only information the computer can understand is of an electrical nature (charged/not charged).
- This can be expressed by ‘0’ or ‘1’.
- The memory is made of a huge number of binary digits (0 or 1) or bits (binary digits).
- Size of memory is the number of bits that can be stored on it:
  1. A CD can contain 6,000,000,000 bits of data.
  2. A common PC (with a memory of 2 GB) contains billions of bits in its RAM memory.
How can we use binary numbers

- We can do the same as for decimal numbers

```
  1  9  7  4
```

- thousands $(10^3)$
- hundreds $(10^2)$
- tens $(10^1)$
- units $(10^0)$
How can we use binary numbers

- Instead of base 10, we are using base 2:
- Instead of multiplying by powers of 10, we multiply each digit by powers of 2

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>thousands (2³)</td>
<td>hundreds (2²)</td>
<td>tens (2¹)</td>
<td>units (2⁰)</td>
</tr>
</tbody>
</table>
Expressing numbers in binary

From binary to decimal

\[ 1 \ 0 \ 1 \ 0 \]

\[ 2^3 \ 2^2 \ 2^1 \ 2^0 \]

\[ \frac{1 \times 2^3}{2} + \frac{0 \times 2^2}{2} + \frac{1 \times 2^1}{2} + \frac{0 \times 2^0}{2} = 10 \]

From decimal to binary

\[
\begin{array}{c}
0 \\
/ 2 \quad 1 \\
/ 2 \quad 2 \\
/ 2 \quad 5 \\
/ 2 \quad 10 \\
\end{array}
\]

remainder remainder remainder remainder remainder

\[
\begin{array}{cccc}
1 & 0 & 1 & 0 \\
\end{array}
\]
More and more bits ...

- The data you can express through single bits is limited.
- Therefore in computing we use a new unit: the byte (8 bits) through which we can express numbers from 0 (00000000) to 255 (11111111).

For bigger volumes of data we have multiples of $2^{10}$:

- 1 KB = 1,024 bytes
- 1 MB = 1,024 KB = 1,048,576 bytes
- 1 GB = 1,024 MB = 1,048,576 KB = 1,073,741,824 bytes

- For instance, the RAM of your PC can be 2GB.
How are these binary numbers used in computing?

In many ways ...
For instance...

- We can decide that every number describes a color: (1: black, 2: light green, 3: dark green etc.) and digitize a picture as below:

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>96</td>
<td>249</td>
<td>181</td>
<td>215</td>
<td>40</td>
<td>7</td>
<td>28</td>
<td>226</td>
</tr>
<tr>
<td>49</td>
<td>156</td>
<td>28</td>
<td>43</td>
<td>145</td>
<td>90</td>
<td>90</td>
<td>40</td>
</tr>
<tr>
<td>250</td>
<td>207</td>
<td>208</td>
<td>141</td>
<td>137</td>
<td>219</td>
<td>227</td>
<td>130</td>
</tr>
<tr>
<td>236</td>
<td>184</td>
<td>77</td>
<td>157</td>
<td>238</td>
<td>242</td>
<td>42</td>
<td>167</td>
</tr>
<tr>
<td>254</td>
<td>58</td>
<td>115</td>
<td>48</td>
<td>151</td>
<td>250</td>
<td>205</td>
<td>92</td>
</tr>
<tr>
<td>110</td>
<td>102</td>
<td>22</td>
<td>36</td>
<td>72</td>
<td>150</td>
<td>92</td>
<td>10</td>
</tr>
<tr>
<td>246</td>
<td>135</td>
<td>181</td>
<td>128</td>
<td>71</td>
<td>208</td>
<td>220</td>
<td>112</td>
</tr>
<tr>
<td>91</td>
<td>253</td>
<td>140</td>
<td>142</td>
<td>27</td>
<td>133</td>
<td>81</td>
<td>28</td>
</tr>
<tr>
<td>179</td>
<td>145</td>
<td>159</td>
<td>248</td>
<td>215</td>
<td>237</td>
<td>62</td>
<td>244</td>
</tr>
<tr>
<td>227</td>
<td>252</td>
<td>137</td>
<td>23</td>
<td>189</td>
<td>97</td>
<td>215</td>
<td>203</td>
</tr>
<tr>
<td>229</td>
<td>96</td>
<td>250</td>
<td>216</td>
<td>34</td>
<td>181</td>
<td>163</td>
<td>11</td>
</tr>
<tr>
<td>240</td>
<td>132</td>
<td>47</td>
<td>37</td>
<td>152</td>
<td>141</td>
<td>174</td>
<td>69</td>
</tr>
</tbody>
</table>
```
Operating system

- Operating system: a bundle of programs which:
  1. Manages computer hardware resources and services
  2. Starts up computing tasks (your code for instance)
  3. Allocates resources to tasks

- The operating system performs transactions between programs and peripherals

- Examples:
Machine language – contents of memory

- The memory contains binary numbers. Most of these numbers are data.
- Some are code names for instructions to be executed.
- These instructions are very basic: for instance, 0 - read from memory, 1 – add 2 numbers ...
- Each type of processor has its own “machine language” – numbers are translated in a different way.

```
send_packet proc
    push cx
    push si
    SEND_CHAR SLIP_END
    jcxz @@bailout
@@nextchar:
    lodsb
    cmp al,SLIP_END
    jne @@check_esc
```

<table>
<thead>
<tr>
<th>229</th>
<th>96</th>
<th>250</th>
<th>216</th>
<th>34</th>
<th>181</th>
<th>163</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>132</td>
<td>47</td>
<td>37</td>
<td>152</td>
<td>141</td>
<td>174</td>
<td>69</td>
</tr>
<tr>
<td>96</td>
<td>249</td>
<td>181</td>
<td>215</td>
<td>40</td>
<td>7</td>
<td>28</td>
<td>226</td>
</tr>
<tr>
<td>49</td>
<td>16</td>
<td>242</td>
<td>28</td>
<td>224</td>
<td>58</td>
<td>155</td>
<td>190</td>
</tr>
<tr>
<td>149</td>
<td>156</td>
<td>28</td>
<td>43</td>
<td>145</td>
<td>90</td>
<td>90</td>
<td>40</td>
</tr>
<tr>
<td>250</td>
<td>207</td>
<td>208</td>
<td>141</td>
<td>137</td>
<td>219</td>
<td>227</td>
<td>130</td>
</tr>
<tr>
<td>91</td>
<td>253</td>
<td>140</td>
<td>142</td>
<td>27</td>
<td>133</td>
<td>81</td>
<td>28</td>
</tr>
<tr>
<td>236</td>
<td>184</td>
<td>77</td>
<td>157</td>
<td>238</td>
<td>242</td>
<td>42</td>
<td>167</td>
</tr>
</tbody>
</table>
What does the processor (CPU)?

- Keeps the address of the next operation to be executed (program counter)
- The address resides in a “cell” in memory
- The processor advances through the memory and performs the following:
  - Reads the number stored in that cell
  - Translates the number to a machine instruction
  - Performs the instruction
  - Advances to the next instruction
Clock rate of the processor

- Machine operations are performed under the control of the internal clock
- Speed of the processor is computed according to the clock frequency (cycles/second)
- Examples
  - Apple 1Mhz (1 million ticks/second)
  - Pentium 486 (33 Mhz)

- Today several instructions are performed during a single clock cycle
Software

- A series of operations for the execution of a task.
- At runtime, the operations are copied onto the main memory
- These operations are in machine language (as the processor only understands these)
- We, as programmers won’t write the code in machine language
- It will be written in some higher level language, and will be translated to machine language.
/* The traditional first program in honor of
Dennis Ritchie who invented C at Bell Labs
in 1972 */

#include <stdio.h>

int main()
{
    printf("Hello World!\n");
    return 0;
}
More about Hello World

```c
#include <stdio.h>

/* My first C program which prints Hello World */

int main () {
    printf("Hello World!\n");
    return 0;
}
```

- **Preprocessor**
- **Comments** are good
- **main()** means “start here”
- **Library command**
- **Brackets** define code blocks
- **Return 0 from main** means our program finished without errors
שפת עילית

(Pascal, C#, C++, Java, BASIC, C)

דモית אנגלית.

פשוקים מורכבים יוצר מהוזאות המוכנה.


text

int i, sum, value;
sum = 0;
for(i = 0; i < 10; i++) {
    scanf("%d", &value);
    sum = sum + value;
}

טכניית השפת עילית מתאימה לירידת על מחשבים-now_ineverגית ואינה תלולה.

بشפת המוכנה של מעבד ספציפי.

(compiler) מהדר (compiler): טכنية המתרגמת תכنية משפה עילית לתכنية

بشפת מוכנה של מעבד נתון.

את קוד המוכנה (קובץherits) המ получаול ניון לחרים עליונים רבוד.

rights reserved.
High level languages

Common programming languages include …

C    C++    Java    Pascal    Visual Basic
FORTRAN    COBOL    Lisp    Scheme    Ada

These high – level languages

• Resemble human languages
• Are designed to be easy to read and write
• Use more complicated instructions than the CPU can follow
• Must be translated to zeros and ones for the CPU to execute a program
An **assembly language** command such as

```
ADD X Y Z
```

might mean add the values found at x and y in memory, and store the result in location z.

- **Assembly language** must be translated to **machine language** (zeros and ones)
  
  0110 1001 1010 1011

- The CPU can follow machine language
First program in C - definition of the problem

- Write a program in C which will obtain 10 numbers from the user, add them up and print the result

Problem definition
- **Input**: series of 10 numbers
- **Output**: sum of the numbers
First program in C - the algorithm

- Zero the sum
- Perform (10 times):
  - Read next number
  - Add to sum
  - Print out the sum

Data structure:
- Input: series of 10 numbers
- Output: sum of the numbers
First program in C

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

    sum = 0;
    for( i = 0; i < 10; i++ ) {
        scanf("%d", &value);
        sum = sum + value;
    }
    printf("%d \n", sum);

    return 0;
}
```
Preparing a C++ Program for Running

C++ program

Compiler

Object code for C++ program

Object code for other routines

Linker

Complete machine language code ready to run
#include <stdio.h>

int main()
{
    int i, sum, value;
    sum = 0;
    for ( i = 0; i < 10; i++ ) {
        scanf("%d", &value);
        sum = num + value;
    }
    printf("%d \n", value);
    return 0;
}
Errors

Syntax errors – errors that the compiler will easily find:

```
sum = sum + value;
sum = num + value;
```

Semantic errors – the text passes compilation but the result is not accurate

```
printf("%d \n", sum);
printf("%d \n", value);
```

Runtime errors – for instance division by 0 – discovered only at runtime

Logical errors – errors in the algorithm