Maciej Sobieraj

Lecture 1

Outline

1. **Introduction to computer programming**
    1. **Different languages for different purposes**
    2. Your first program
    3. Integer values, integer variables and comments
    4. Quiz

Natural language vs programming language

- At least one language accompanies us throughout our whole lives – it's our **native language**
- The languages we use to communicate with other people are called **natural languages**.
- The languages defined by international standards, and although they are understood by many people, the exchange of thoughts between human-beings is not their most important application – Such languages are **programming languages**.

Natural language vs programming language

- For example, rules determine which symbols (letters, digits, punctuation marks, and so on) could be used in the language. This part of the definition of the language is called **lexicon**.
- Another set of rules determines the appropriate ways of collating the symbols – this is the **syntax** of the language.
- We would also like to be able to recognize the meaning of every statement expressed in the given language – and this is what we call **semantics**.

Natural language vs programming language

- A computer it's like a well-trained dog – it responds only to a predetermined **set of known commands**.
- A complete set of well-known commands is called an **instruction list**, sometimes abbreviated to **IL.**
- The IL is in fact the alphabet of a language, commonly known as a **machine language**. This is the simplest and most primary language we can use to give commands to our computer.

Natural language vs programming language

- Computer programming is the act of composing selected commands (instructions) in the proper order so that a desired effect is produced.
- Programs written in machine language are very difficult for humans to understand, including experienced programmers.
- need for some kind of **bridge** between the human language (natural language) and the computer language (machine language). That bridge is also a language – an intermediate common language for humans and computers to work together. Such languages are often called **high-level programming languages**.

Natural language vs programming language

- A high-level programming language is at least somewhat similar to a natural language; it uses symbols, words and conventions readable to humans. This language enables humans to express complex commands for computers.
- We can just **translate** our program into machine language. Moreover, the translation can be done by a computer, making the whole process fast and efficient.

Natural language vs programming language

- The translation we are referring to is made by a specialized computer program called a **compiler**. The process of translating from a high-level language into a machine language is called **compilation**.
- The main task is to write a program in accordance with the rules of the chosen programming language. Such a program (which in fact is just text) is called the **source code**, or simply **source**, while the file which contains the source is called the **source file**.

Natural language vs programming language

- If the compiler doesn't notice any mistakes in your source, the result of its work will be a file containing your program translated into machine language. That file is commonly called an **executable file**.
- What is the most common use of "C"? It is the so-called **general-purpose programming language**, i.e., suitable for almost any programming project and at the same time not particularly predestined to any specific, narrow class of applications. It's best if used for coding drivers, embedded applications or operating systems (for example, the Linux kernel is mainly coded in "C").

Outline

1. **Introduction to computer programming**
    1. Different languages for different purposes
    2. **Your first program**
    3. Integer values, integer variables and comments
    4. Quiz

Your first program

- We want a short and rather meaningless text to appear on the screen.
  - „*It's me, your first program*”.
- What further steps should our first program perform? Let's try to enumerate them here:
  1. to **start**;
  2. to **write** the text on the screen;
  3. to **stop**;
- This sort of structured and semi-formal description of each step of the program is ca… an **algorithm**.

Your first program

```c
#include <stdio.h>

int main(void)

{

    puts("It's me, your first program.");

    return 0;

}
```

Your first program

- The character # (*hash*) at the beginning of the first line means that the content of this line is a **preprocessor directive**.
- The prefix “pre” suggests that these operations are performed **before** the full processing (compilation) takes place.
- **include** directive - when the preprocessor encounters that directive, it replaces the directive with the content of the file whose name is listed in the directive (in our case, this is the file ***stdio.h***).

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

Your first program

- The **stdio.h** file (defined by the standard of the “C” language) contains a collection of preliminary information about ready-made blocks which can be used by a program to write text on the screen or to read letters from the keyboard.

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

Your first program

- One of the most common types of blocks used to build “C” programs is **functions**.
- Imagine a function as a **black box**, where you can insert something into it (not always necessary) and take something new out of it as if out of a magic hat.
- Things that are put in the box are called function **arguments** (or function **parameters**).
- Things that are taken out of the box are called function **results**.

```
int main(void)
```

Your first program

- The standard of the “C” language assumes that, among the many different blocks which may be put into a program, one specific block must always be present, otherwise the program won't be correct. This block is always a function of the same name: **main**.
- Every function in “C” begins with the following set of information:
  - what is the **result** of the function?
  - what is the **name** of the function?
  - how many **parameters** does the function have and what are their **names**?

```
int main(void)
```

Your first program

- From int main(void) we can read:
  - the **result** of the function is an **integer value** (we read it from the word int which is short for *integer*)
  - the **name** of the function is **main** (we know why already)
  - the function **doesn't require any parameters** (which we read from the word void)
- A set of information like this is called a **prototype.** The prototype says nothing about what the function is intended for. It's written inside the function and the interior of the function is called the function **body**. The function body begins where the first opening bracket **{** is placed and ends where the corresponding closing bracket **}** is placed.

```
int main(void)
```

Your first program

- Inside the main function body we should write what our function (and thus the program) is supposed to do. We look inside and find a reference to a block called ***puts***. This is what we call a **function invocation**.
- Firstly, note the **semicolon** at the end of the line. Each instruction (precisely: each statement) in “C” **must end with a semicolon.**
- The text intended to be shown on the screen is passed to the function as a function parameter. Remember that the name of the invoked function **must** always be followed by a pair of parentheses **(** and **)**, even when function doesn’t expect any parameters from us.

```
puts("It's me, your first program.");
```

Your first program

- puts
- (
- "It's me, your first program."
-)
- ;

- In the “C” language you don’t have to write just one statement per line. It’s possible to place two (or more) statements on the same line, or split one statement into several lines, but bear in mind that the readability (for humans) is the most important factor. Compilers will never complain about your style.

```
puts("It's me, your first program.");
```

Your first program

- Besides the function invocation, this is another **statement** of the “C” language. Its name is just *return* and that’s exactly what it does. Used in the function, it **causes the end of the function execution**.
- If you perform return somewhere inside a function, this function immediately interrupts its execution.

```
return 0;
```

Your first program

- This is important – this is how your program tells the operating system the following message: *I did what I had to do, nothing stopped me and everything is OK*. If you were to write:
  - **return** 1;
- it would mean that something had gone wrong, it didn't allow your program to be successful and the operating system could then use that information to react in the most appropriate way.


```
return 0;
```

Outline

1. **Introduction to computer programming**
    1. Different languages for different purposes
    2. Your first program
    3. **Integer values, integer variables and comments**
    4. Quiz

Numbers and how the computers see them

- The **binary system** - it's the system computers use for storing numbers, and that they can perform any operation upon them.
- The numbers handled by modern computers are of two types:
  - **integers**, that is, those which are devoid of the fractional part;
  - **floating-point** numbers (or simply floats), that contain (or are able to contain) the fractional part.
- At this point we have made friends with two types of the "C" language – an **integer type** (known as ***int***) and a **floating point type** (known as ***float***).

Numbers and how the computers see them

- We would write the number like this:
  - 11,111,111
- or like this:
  - 11.111.111
- or even like this:
  - 11 111 111
- However, in “C” it’s prohibited. You **must** write this number as follows:
  - 11111111

Numbers and how the computers see them

- Positive numbers don't need to be preceded by the plus sign but you can do it if you want. The following lines describe the same number:
  - +123
  - 123

Numbers and how the computers see them

- There are two additional conventions, unknown to the world of mathematics. If an integer number is preceded by the *0* digit, it will be treated as an **octal value**. This means that the number must contain digits taken from the *[0..7]* range only.
  - 0123
- is an octal number with a decimal value equal to **83**.
- The second allows us to use hexadecimal numbers. This type of number should be preceded by a prefix written as *0x* or *0X*.
  - 0x123
- is a **hexadecimal number** with a decimal value equ **291**.

Numbers and how the computers see them

- To print an integer number, you should use (this is only a simple form):
  - printf("%d\n", IntegerNumberOrExpression);
- To print a floating point number, you should use (this is only a simple form):
  - printf("%f\n", FloatNumberOrExpression);
- In both cases, you should first include the stdio header file (as we did in the first program):
  - #include <stdio.h>

A variable is variable

- How to store the results of arithmetic operations in order to use them in other operations.
- There are special "**containers**" for this purpose and these containers are called **variables**.
- What does every variable have?
  - a **name**
  - a **type**
  - a **value**

A variable is variable

- If you want to give a name to a variable you must follow some strict rules:
  - the name of the variable must be composed of **upper-case or lower-case Latin letters, digits and the character _** (*underscore*);
  - the name of the variable must **begin with a letter**
  - the **underline character is a letter** (strange but true)
  - upper- and lower-case letters are treated as **different** (a little differently than in the real world – Alice and ALICE are the same given names but they are two different variable names, consequently, two different variables)

A variable is variable

- The type is an **attribute** that uniquely defines which values can be stored inside the variable.
  - integer (*int*) and floating point (*float*) types
- The variable comes into existence as a result of a **declaration**.
- A declaration is a syntactic structure that binds a name, provided by the programmer, to a specific type offered by the “C” language.
- The construction of the declaration (in other words – the declaration syntax) is simple:
  - just use the name of the desired type, then the variable name (or variable names separated by commas if there are more than one).

A variable is variable

- We can declare a variable of type *int* named *Counter*. The relevant portion of the program looks like this:
  - **int** Counter;
- What is declared by the following fragment of a program?
  - **int** variable1, account_balance, invoices;
- It declares three variables of type *int* named (respectively) *variable1*, *account_balance* and *invoices*.

A variable is variable

- And how do we give a value to the newly declared variable? You need to use the **assignment operator**.
- The assignment operator looks very familiar – here it is:
  - =
- Let's look at some examples:
  - Counter = 1;
- Another example:
  - Result = 100 + 200;
- And now a slightly more difficult example:
  - x = x + 1;

Keywords – why they are the keys?

auto	extern	short
break	float	signed
case	for	sizeof
char	goto	static
complex	if	struct
const	imaginary	switch
continue	inline	typedef
default	int	union
do	long	unsigned
double	register	void
else	restrict	volatile
enum	return	while

Keywords – why they are the keys?

- For example – you can't do this:
  - int int;
- You mustn't have a variable named *int* – it’s prohibited. But you can do this instead:
  - int Int;

Comments on the comments

- The developer may want to put in a few words addressed not to the compiler but to humans, usually to **explain** to other readers of the code how the tricks used in the code work, or the meanings of variables and functions and eventually, in order to keep stored information on who the author is and when the program was written.
- In the “C” language a comment is a text that begins with the pair of characters
  - ***/**
- and ends with the pair of characters
  - ***/**
- The comment can span several lines

Comments on the comments

```
/* Counter variable counts the number of sheep in the meadow */
int Counter;
```

Comments on the comments

```
/**********************************************************
Counting sheep version 1.0
Author: Ronald Sleepyhead, 2012
email: rs@insomnia.org

Changes:
2012-09-13: Ginny Drowsy: counting black sheep improved
**********************************************************/
```

Comments on the comments

- The question is: is the declaration of variable k a comment or not?

```
/* int i; /* int j; */ int k; */
```

Outline

1. **Introduction to computer programming**
    1. Different languages for different purposes
    2. Your first program
    3. Integer values, integer variables and comments
    4. **Quiz**

Quiz

The English language is an example of:

- a programming language
- a machine language
- a natural language

Quiz

A high level language is:

- a language spoken in the high society
- a type of a programming language
- a language spoken by mountain tribes

Quiz

A compiler is:

- an alternative name for a processor
- a computer program designed to translate programs from a high level language into a machine language
- a computer program designed to translate programs from a machine language into a high level language

Quiz

Data of type `int` is:

- a fractional number
- an internal number
- an integer number

Quiz

The following string:

```
ThisIsTheNameOfTheVariabla
```

- ◯ must not be used as a variable name
- ◯ may be used as a variable name

Quiz

The following string:

```
101Dalmatians
```

- ○ may be used as a variable name
- ○ must not be used as a variable name

Quiz

What is the value of the `var` variable after the execution of the following snippet of code:

```
int var;
var = 100;
var = var + 100;
var = var + var;
```

- ○ 300
- ○ 400
- ○ 100
- ○ 200

Quiz

A keyword is a word which:

- ○ functions as a password needed to launch a program
- ○ must not be used in the meaning other than defined in the language standard
- ○ is the most important word in a program

Quiz

A comment placed anywhere inside the code is a syntactical equivalent of:

- ○ a space
- ○ a keyword
- ○ a number

Quiz

Every variable has the following attributes:

- ○ header, footer, setter
- ○ type, name, value
- ○ variability, stability, readability