"Ss. Cyril and Methodius" University in Skopje
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Lesson 1

Introduction to C
Integrated Development Environemnts (IDE)

Structured programming

Contents

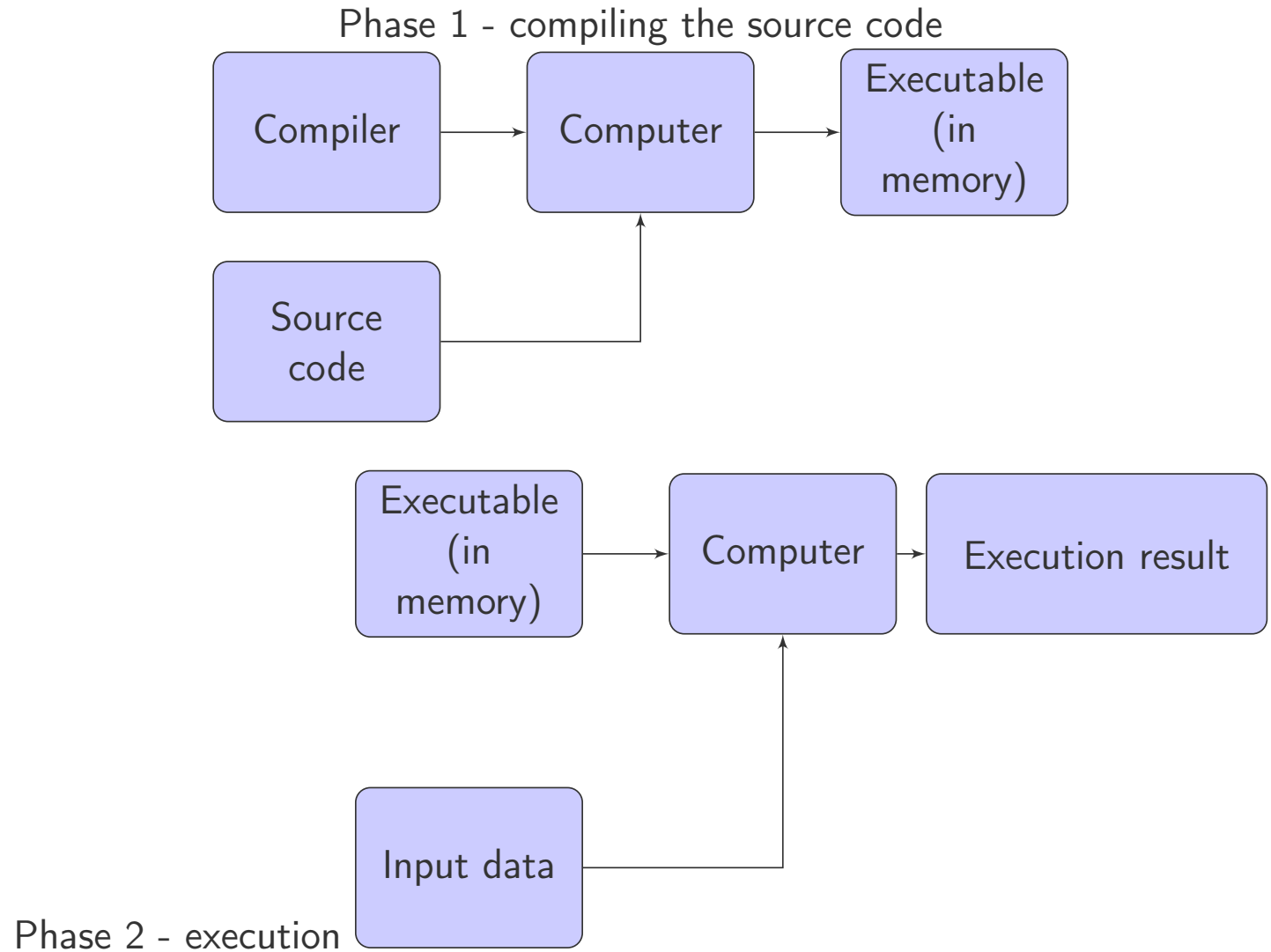
Programming (1)

- Executable programs are containing only **zeros and ones**, because that is the only language that computer understands
- Programmers write the programs in human readable languages, called **programming languages**
- Program written in programming language is called a **source code**

Programming (2)

- Usually for writing source code programmers use **development environments**
- The code is written in text editor
- Then the code is (**compiled**)
- At the end we have **executable** i.e. program written in machine language (the language of the computer)

Compiling and executing process



Introduction to C programming language

- Developed in Bell laboratories in period from 1969 to 1973 by Dennis Ritchie
- One of the most used general purpose programming languages of all time
- Has big influence in origins and development of many other programming languages
  - C++
  - Objective C
  - PHP
  - Java

C Syntax

Allowed characters in C:

```
a-z, A-Z, 0-9 and ~!@#$%^&*()-+={}[]:;'"<>?/._
```

Attention!

Compiler is case sensitive (makes difference between uppercase and lowercase)!

All characters in C are forming words that can be:

1. Keywords
2. Numerical and symbolic constants
3. Identifiers
4. Strings (char arrays)
5. Operators

C Syntax

Set of keywords (32)

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

Simple C program structure

All the source code written in C is organized in **functions**

Program in C

```
int main() {
    variable_declaration
        ;
    expressions;
}
```

Program in Pascal

```
Program program_name;
var variable_declaration
     ;
begin
    expressions;;
end.
```

C Functions

`main`

C main function

`()`

In parentheses we put the input arguments

`int`

Data type of the result it's before the function name

`{}`

The function body starts with {, and ends with }

`;`

All declarations and expressions are separated with ;

Comments usage

- Comments are used for extra explanation or documenting the source code
- C supports two types of comments
  1. one line comments
  2. multiple line comments

1. One line comments

```
// one line comment ;)
```

2. Multiple line comment

```
/* Comment
in multiple lines */
```

Examples

Example 1

```c
#include <stdio.h>
// main function
int main() {
    /* function for print on standard output (screen) */
    printf("Welcome to FINKI!\n");
    return 0;
}
```

C program structure (extended)

INCLUDE section	contains `#include` expressions for including external libraries, for using externally declared functions
DEFINE section	contains `#define` expressions for constants and data type declaration
...	defining global variables and functions
`int main()`	main function

Preprocessor

- In C compiling is done by:
  - preprocessor
  - compiler
- The preprocessor is used with directives
  - Each directive starts with #

Header files

- One usage of preprocessor directive is including the "header files"
  - Used for declaration of functions and variables in some predefined library
  - In order to use the external functions and variables users include the "header file"
- The inclusion is done with the preprocessor directive `#include`
  - Statement `#include` causes inclusion a copy of the given file in the place where the statement is written

include variations

- There are two types of this directive:
  - file that is included can be written in double quotes (""),
  - or in angle brackets (<>)

Example

```
#include <filename.h>
#include "filename.h"
```

- The difference is in the location in which the preprocessor searches the file that should be included
  - In angle brackets (files from the standard library are used)
  - With double quotes (the preprocessor first tries to find the file in the same directory with the C file that should be compiled)

Variables

- Variables are symbolic names for places in memory where the computer keeps values
- Before usage all variables must be *declared*
- With each new assignment of value, the old value is discarded

Types of variable declaration:

Variable type	Variable name	=	Initial value	;

Data types

Data types in C

Numbers	Letters	Decimal
`int`	`char`	`float`
`short`		`double`
`long`		

Defining variable names

- In naming variables allowed are:
  - lowercase letters from a to z;
  - uppercase letters from A to Z;
  - digits from 0 to 9 (can't start with digit);
  - underscore _ which is recognized as a letter (not recommended to start with _);

Be carefull!

- mostly names length is up to 32 characters
- C is case sensitive!

Examples

Example 2

```
#include <stdio.h>

int main() {
    int a, b, c;
    a = 5;
    b = 10;
    c = a + b;
    return 0;
}
```

Constants

- Constants are used for values that are unchanged during the program execution
- Each constant is from some data type
- C has several constant types:
  - decimal: `1, -23, 15`
  - octal: `015, 035, 0205`
  - hexadecimal: `0x25, 0xA4C`
  - real: `3.5F, -2.845F, 1.34e-9`
  - literals: `'a', '_', 'e'`
  - strings: `" ", "Koncepti za razvoj na softver"`

Determining constant type

- Determining the type of varialbes is simple (can be seen from the declaration)
- Constants aren't declared, so their type is determined by the way they are written:
  - Numbers that contain "." or "e" are `double`: `3.5, 1e-7, -1.29e15`
  - Instead of using `double`, to use `float` constants "F" is added at the end: `3.5F, 1e-7F`
  - For `long double` constants "L" is added: `1.29e15L, 1e-7L`
  - Numbers without ".", "e" or "F" are `int`: `1000, -35`
  - For `long int` constants "L" is added: `9000000L`

Named constants (1)

Named constants are created by using the keyworkd `const`

Example 3

```
#include <stdio.h>

int main() {
    const long double pi = 3.141592653590L;
    const int days_in_week = 7;
    const sunday = 0; /* by default int */
    days_of_week = 1; /* error */
}
```

Named constants (2)

Named constants can be created with the preprocessor, and usually uppercase letters are used

Example 3

```
#include <stdio.h>
#define PI 3.141592653590L
#define DAYS_IN_WEEK 7
#define SUNDAY 7
int main() {
    long double pi = PI;
    int day = SUNDAY;
}
```

Operators

- Operators are used in builing expressions, and operations are evaluated from right to left by applying the operator priority rules
- There are three types of operators
  - Arithmetic operators
  - Relational operators
  - Logical operators

Arithmetic operators

Usen on numbers (integers or reals)

Operator	Operation
+	Addition
-	Substruction
*	Multiplication
/	Division
%	Division by modulo

Relational operators

Applyed on any comparable data types, and the result is 0 (false) or 1 (true).

Operator	Meaning
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	equal
!=	not equal

Logical operators

Used mostly in combination with relational operators in forming complicated logical expressions, which again results in 0 or 1

Operator	Operation
&&	Logical AND
||	Logical OR
!	Negation

Additional operators

- Assignment operator =
- Increment and decrement operators

  `++, --`
- Unary usage of operators + and –

```
X = + Y;
X = - Y;
```

- Double operators
  - Combination of assignment operator and other operator

    `(+=, -=, *=, /=, %=)`

Assignment operator =

- All expressions have values, even those containing =
- Value of such expression is the value of the expression of the right side
- Because of that, this assignment is possible:

```
x = (y = 10) * (z = 5);
x = y = z = 20;
```

Double operators

Operator +=

```
a += 5; // a = a + 5;
a += b * c; // a = a + b * c;
```

Operator -=

```
a -= 3; // a = a - 3;
```

Operator *=

```
a *= 3; // a = a * 3;
```

Operator /=

```
a /= 3; // a = a / 3;
```

Operator %=

```
a %= 3; // a = a % 3;
```

Using variables and operators

Example 4

```
#include <stdio.h>
int main() {
    int a;
    float p;
    p = 1.0 / 2.0; /* p = 0.5 */
    a = 5 / 2;     /* a = 2   */
    p = 1 / 2 + 1 / 8; /* p = 0; */
    p = 3.5 / 2.8; /* p = 1.25 */
    a = p; /* a = 1 */
    a = a + 1; /* a = 2; */
    return 0;
}
```

Printing to the standard output

- In C doesn't exist printing command
- Already defined function from the input/output library `stdio.h` is used (**st**andar**d** **i**nput/**o**utput) `#include <stdio.h>`
- The function used is:

```
int printf(control_array, list_of_arguments)
```

- The control array contains any string, formating and printing arguments with leading % or special characters with leading \.
- Characters for formating are determined from the data type that should be printed.

Formatting characters

Character	Explanation
%d	integers
%i	integers
%c	single character
%s	char arrays
%e	real number in technical foramat (e)
%E	real number in technical foramat (E)
%d	real number in decimal foramat
%f	real number in shorter from the foramats %e and %f
%g	real number in shorter from the foramats %E and %f
%u	unsigned integer
%o	unsigned integer in octal
%x	unsigned hexadecimal integer
%X	unsigned hexadecimal integer
%p	pointer
%n	number of printed characters
%%	printing the character %

Function printf usage

Example 5

```
#include <stdio.h>
int main() {
   printf(" length is %d letters.\n", printf("Makedonija"));
   return 0;
}
```

Problem 1

Write a program that will compute the value of the mathematical expression: $x = \frac{3}{2} + (5 - \frac{46*5}{12})$

Solution

```c
#include <stdio.h>
int main() {
    float x = 3.0 / 2 + (5 - 46 * 5 / 12.0);
    printf("x = %.2f\n", x);
    return 0;
}
```

Problem 2

Write a program that for a given value of x (with the declaration assignment) will compute and print on standard output x^2.

Solution

```
#include <stdio.h>
int main() {
    int x = 7;
    printf("Number %d squared is: %d\n", x, x * x);
    return 0;
}
```

Problem 3

Write a program that for a given sides of one triangle, it will print the perimeter and area of the square (values are a=5, b=7.5, c=10.2).

Solution

```
#include <stdio.h>
int main() {
    float a = 5;
    float b = 7.5;
    float c = 10.2;
    float L = a + b + c;
    float s = L / 2;
    float P = s * (s - a) * (s - b) * (s - c);
    printf("Perimeter is: %.2f\n", L);
    printf("Area is: %.2f\n", P);
    return 0;
}
```

Problem 4

Write a program for computing the arithmetic mean of the numbers 3, 5 and 12.

Solution

```c
#include <stdio.h>
int main() {
    int a = 3, b = 5, c = 12;
    float am = (a + b + c) / 3.0;
    printf("Arithmetic mean is: %.2f\n", am);
    return 0;
}
```

Problem 5

Write a program that will print the remainder from the division of number 19 with 2, 3, 5 and 8.

Solution

```
#include <stdio.h>
int main() {
    int a = 19;
    printf("Remainder from division of %d with 2: %d\n", a, a %
           % 2);
    printf("Remainder from division of %d with 3: %d\n", a, a % 3);
    printf("Remainder from division of %d with 5: %d\n", a, a % 5);
    printf("Remainder from division of %d with 8: %d\n", a, a % 8);
    return 0;
}
```

Contents

Integrated development environments elements

Integrated development environment is mix of multiple programs, that are combined in order to simplify the development process

- text editor
- compiler
- debugger
- external library integration
- linker

Text editor

- Program for inserting and editing the text of the source code
- Enables saving and loading already written source code files for editing
- Syntax highlighting

Compiler

- Transforms (translates) the source code from the programming language (high level) to machine language
- Two types of code translators: **interpreters** and **compilers**
- Interpreter is a translator of source code which *translates each command separately*, checks for errors and executes, and goes to the next command.
- Compiler is a translator which is *processing the whole program*, checks for errors, and creates the executable program.
  - The resulted executable can be executed

Debugger

- Compilers and interpreters are detecting the (syntax) errors of the source code as a result from incorrect usage of the programming language
- Other type of errors are logical errors
  - The program doesn't execute as intended
  - Very difficult to discover
- Debugger is a program that helps discovering logical errors
  - Enables execution step by step

Integration with external libraries

- Integration and usage of already created and correct modules, also called functions
- This type of organization has many advantages
- Reuse of standard functions
- Example libraries
  - Standard input/output
  - Mathematical operations

Linker

- Sometimes the program is to large to be written in one file
  - different parts can be written by different programmers
  - some parts from one program can be used in some other program
  - Separately compiled parts must be united in on full executable with the help of the **linker**
  - Another role is to "link" standard functions with the executable

Integrated Development Environment - IDE

- All these elements of the development environment are integrated in IDE
- Code::Blocks is an example of IDE that can be used in this course

Contents

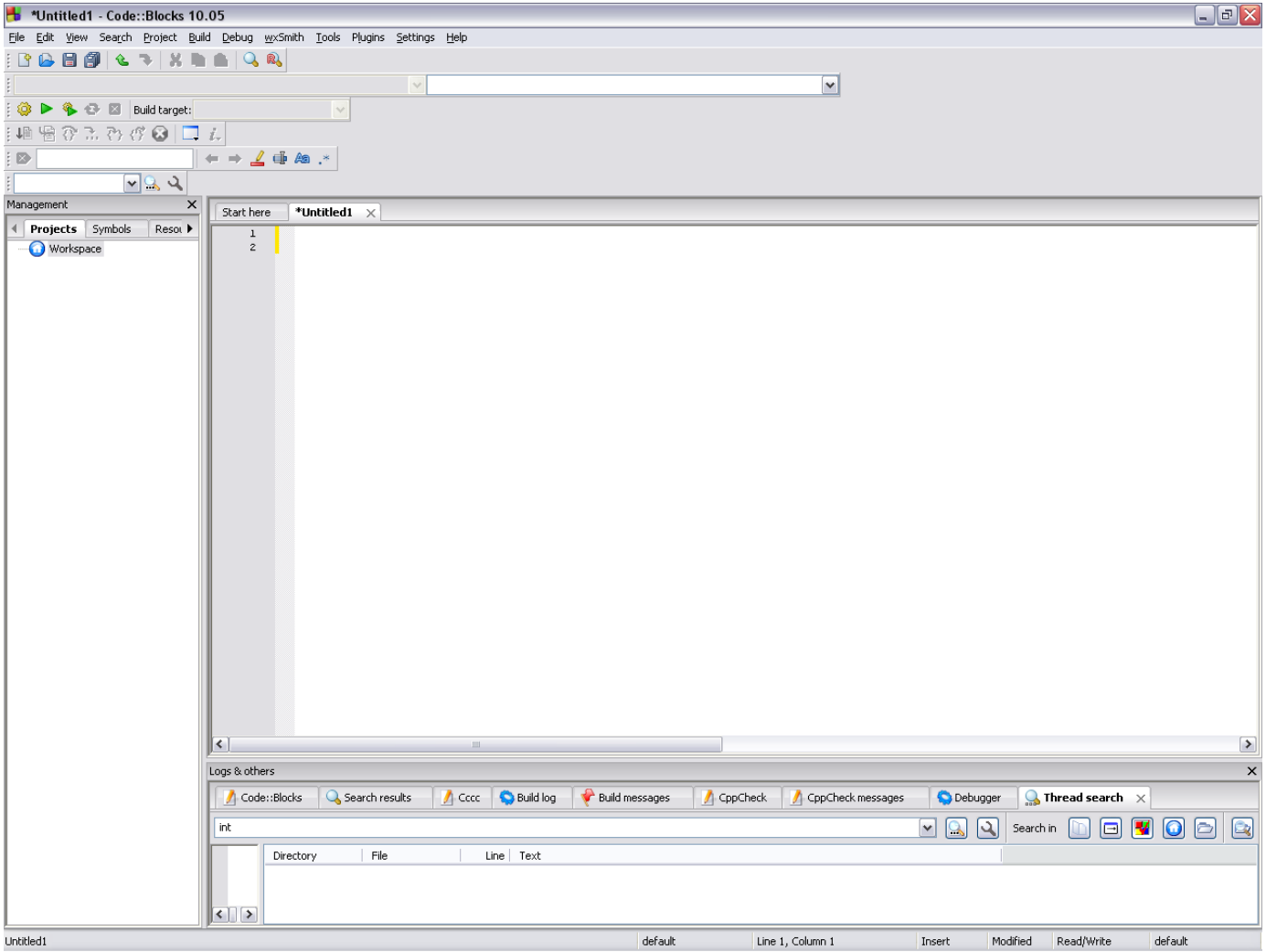
Code::Blocks - installation

- How to find and install Code::Blocks
- Code::Blocks is **free software** and can be find http://www.codeblocks.org/downloads
- In the central part of the page, there are three links: **Download the binary release**, Download the source code and Retrieve source code from SVN
- The first link is recommended for most simple installation - **Download the binary release**,

Code::Blocks – installation (2)

- For begginers is recommended the version that includes **MinGW** setup
  - at the moment that is the link **codeblocks-10.05mingw-setup.exe** which supports all **Windows** operating systems
  - By clicking the source Sourceforge.net opens up a new page, and after 5 seconds the page will show you option to save the file in some location
  - After saving the file follow the installation instructions

Code::Blocks – main window



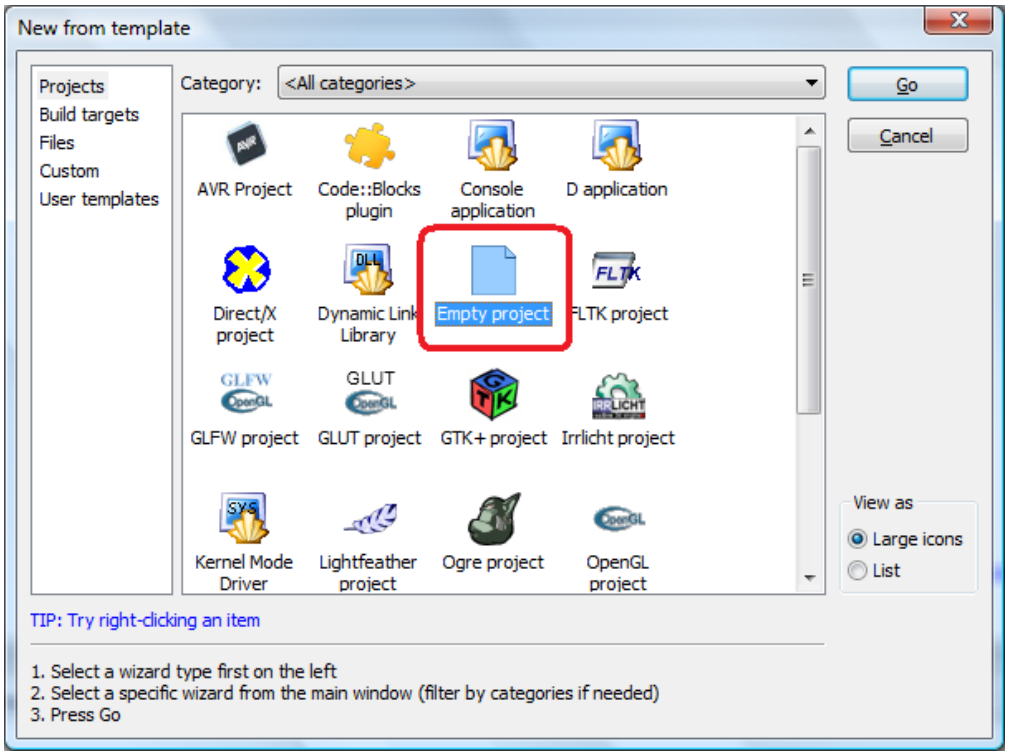
Elements of the main window

- Menu strip
  - Menu strip is in to top left corner of the window, right bellow the title
  - There can be found menues File, Edit, View, Search, Project, Build, Debug, wxSmith, Tools, Plugins, Settings, Help
- Toolstrip
  - Toolstrip (buttons for starting most often used commands) are just bellow the menu strip
- Workspace
  - Text editor
  - Info window for logs & others
  - Window for management

Programming in C with Code::Blocks

Creating project

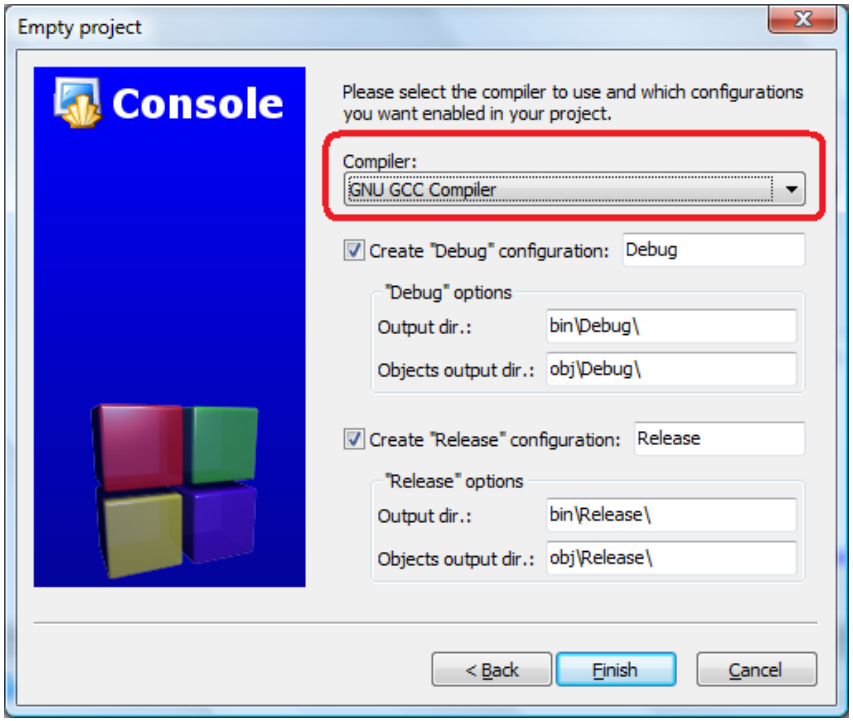
1. Start CodeBlocks
2. File -> New -> Project -> Empty Project -> Go



Programming in C with Code::Blocks

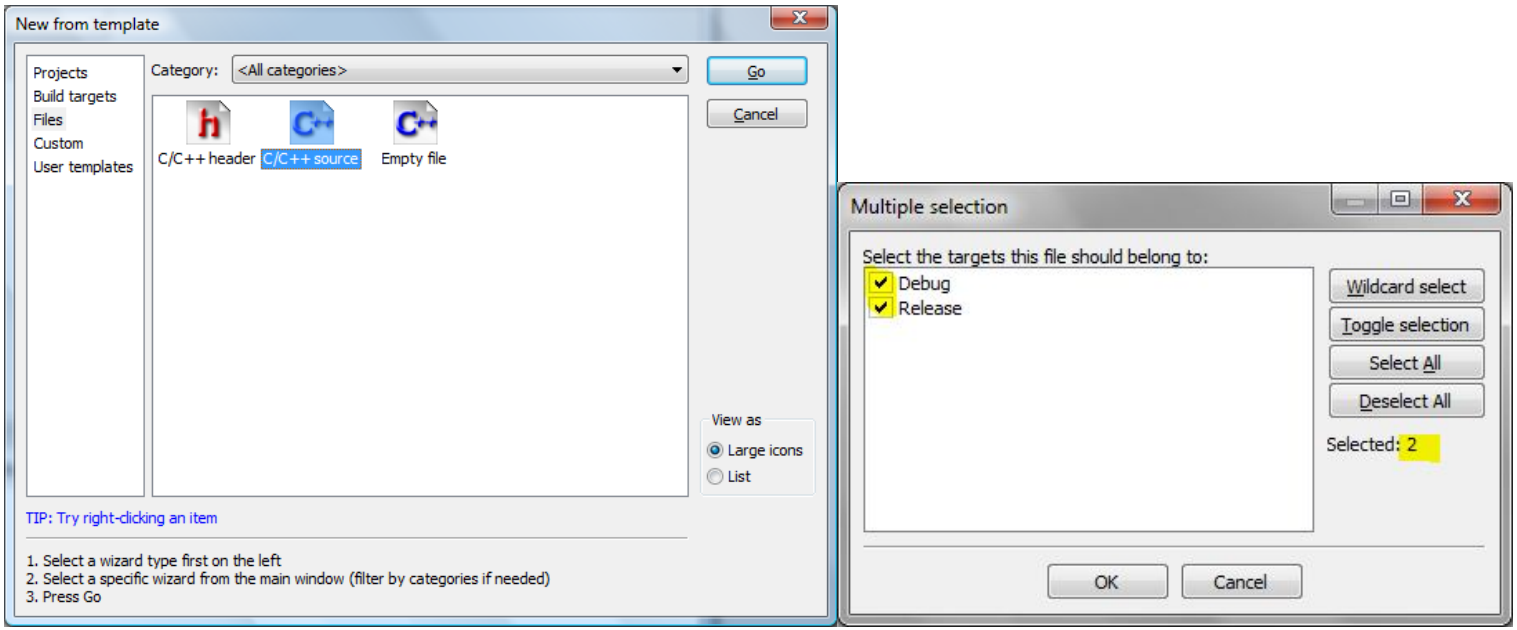
Creating project

3. Choose GNU GCC Compiler
4. Select the next 2 options if you want to create “debug” and “release” configuration



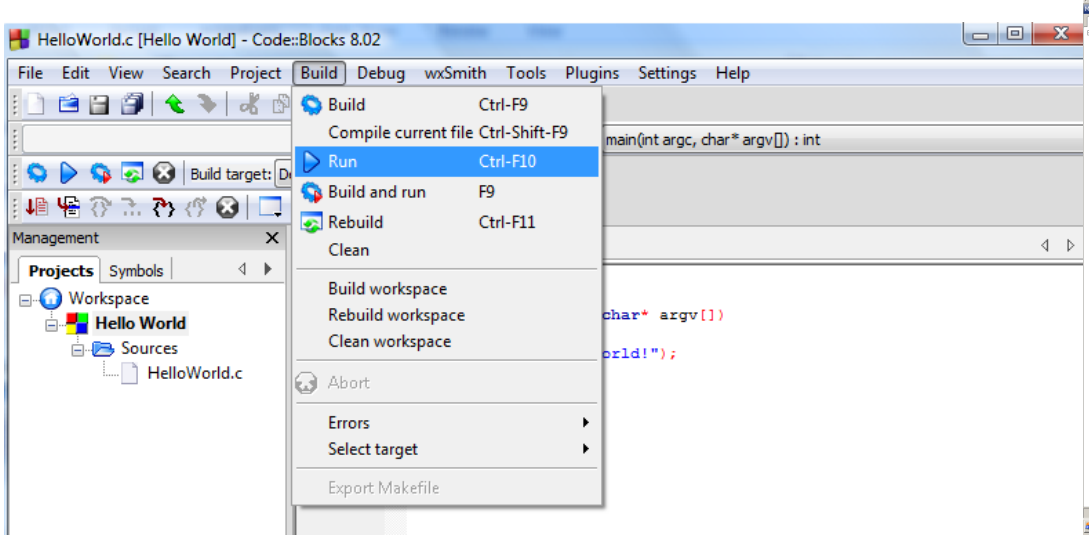
Adding source file

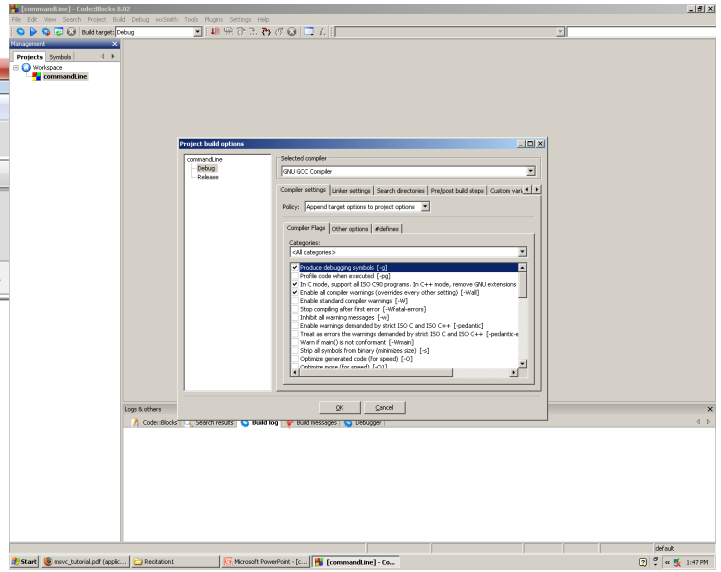
5. Add source file in the project: File -> New -> File -> C/C++ Source
6. Choose C as programming language
7. Enter the file name with the full path, and don't forget to check "Add file to active project"



Programming

- For each project following options can be checked “Project Build Options.. Compiler Flags”
- To build the project press Ctrl + F9
- To execute the project press Ctrl + F10





Homework

- In the next section are given some example problems that you should try to complete at home
- so you can be ready for next excersises

Problem 1

Try to create new project with one .c file with the following source code:

```c
#include <stdio.h>

int main() {
    printf("Hello, how are doing?\n");
    return 0;
}
```

Problem 1

- Execute the program
  - What result do you get?
- If you made some errors during writing the code, try to find, correct them and execute again.
- Make some errors, intentionally. Execute again!
  - What's happening now?

Problem 2

In the text of te program add the following line:

```
#include <stdio.h>
int main() {
    printf("Hello, how are you doing?\n");
    printf("So, you are not much of a talker...\n");
    return 0;
}
```

What is the result now?

Materials and Questions

Lectures, exsercises and announcements
courses.finki.ukim.mk

Source code of all examples and problems
https://github.com/tdelev/SP/tree/master/latex/src

Questions and discussion
forum.finki.ukim.mk