Fundamentals of Programming

Lecture 2

Chamila Karunatilake

Department of Information and Communication Technology

Faculty of Technology

University of Sri Jayewardenepura

chamilakarunatilake@sjp.ac.lk

What is programming?

How to make a cup of tea?

Ingredients

2 cups of water

1 spoon of Ceylon Tea (Black Tea)

2 spoons of sugar.

Method

Boil 2 cups of water. Meanwhile add tea leaves into a tea pot.

Once water is boiled, pour it in to the tea pot. Allow teas for brewing for 03 minutes

After 03 minutes stir once. Then leave it for further 02 minutes making it total of 5 minutes.

Then pour the teas into the respective serving cups and add sugar.

- Similar to a tea recipe, a computer program presents a systematic set of instructions to achieve a specific task.
- It gives the data that the program needs and how to process those data to achieve the given task.

What is programming?

- A computer program is a sequence of instructions to perform a specified task by the computer.
- These instructions should be given to the computer in a language that computer understands.
- The only language that computer understands is **Machine Language** which consists of 1s and 0s.
- Suppose you want to give instructions to add two numbers which are currently resides in register 1 and register 2 and put the answer into the register 6. It will be written in Machine code like follows :

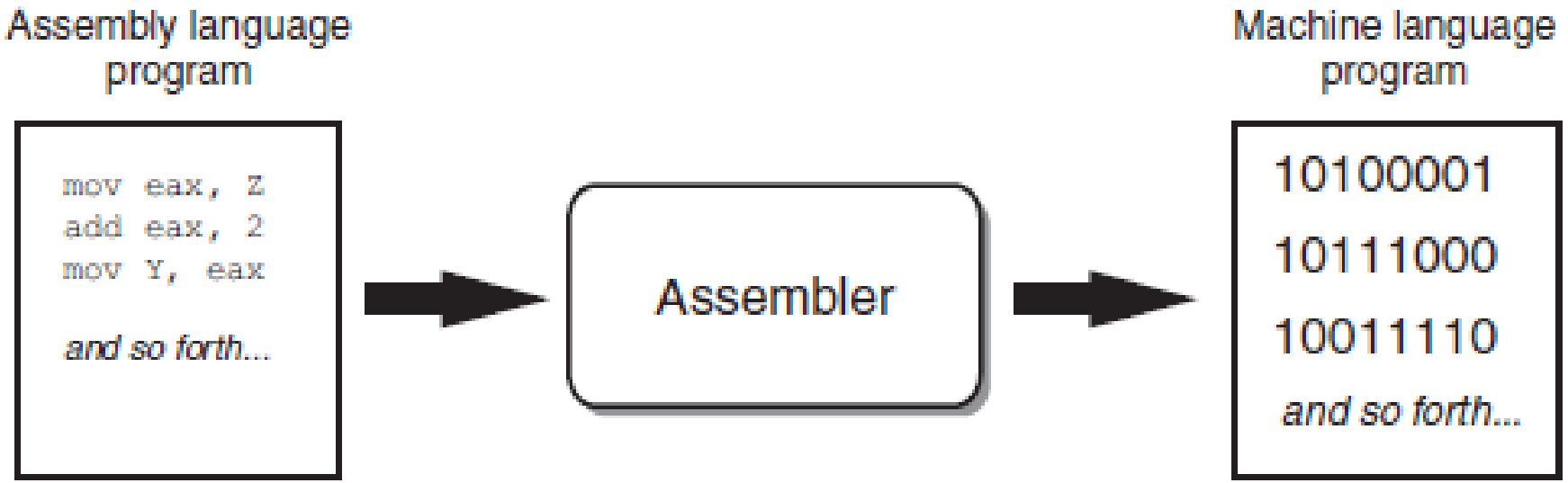
  000000 00001 00010 00110 00000 100000

- The machine code for the same set of instructions in different type of machines would be different. It depends on the type of the processor, memory organization and many other factors.

Assembly Language

- Writing large programs with thousands or even millions of binary instructions would be very tedious and time consuming.
- Programming in machine language would also be very difficult because putting a 0 or a 1 in the wrong place will cause an error.
- **Assembly language** was created in the early days of computing as an alternative to machine language.
- Instead of using binary numbers for instructions, assembly language uses short words that are known as **mnemonics.**
- For example, in assembly language, the mnemonic **add** typically means to add numbers, **mul** typically means to multiply numbers, and **mov** typically means to move a value to a location in memory.

Assembly Language

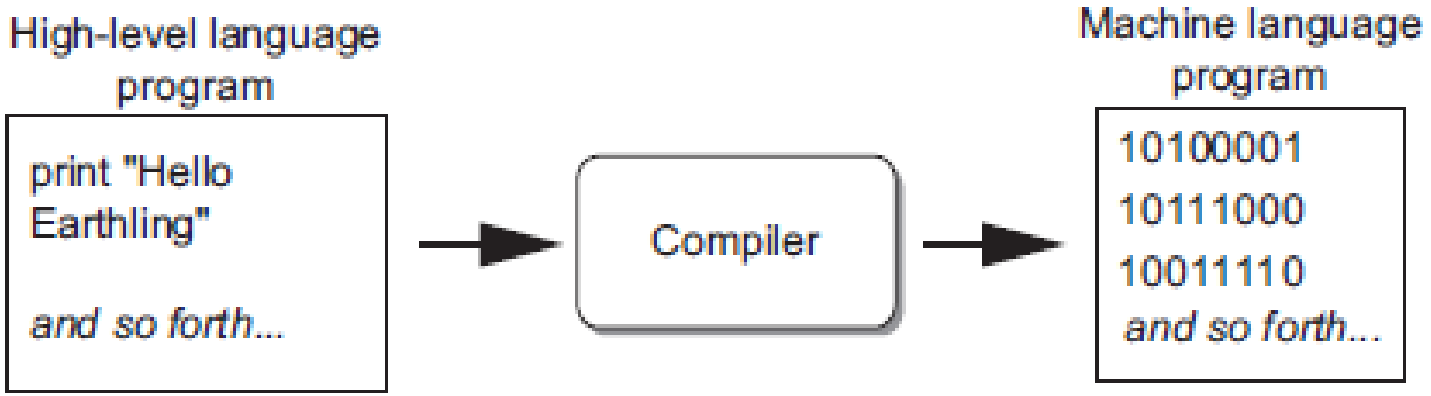


High-Level Languages

- Since assembly language is a direct substitute for machine language, it still requires a through knowledge of the CPU when writing a program.
- Assembly language also requires a large number of instructions for even a simple program.
- Machine language and Assembly languages are called low-level Languages.
- In the 1950s, a new generation of programming languages were appeared which are known as **high-level languages**.
- These new type of languages use words that are easy to understand.
- In addition, High-level languages provide the capability of writing powerful and complex programs without the knowledge of how the CPU works, and without writing large numbers of low-level instructions.

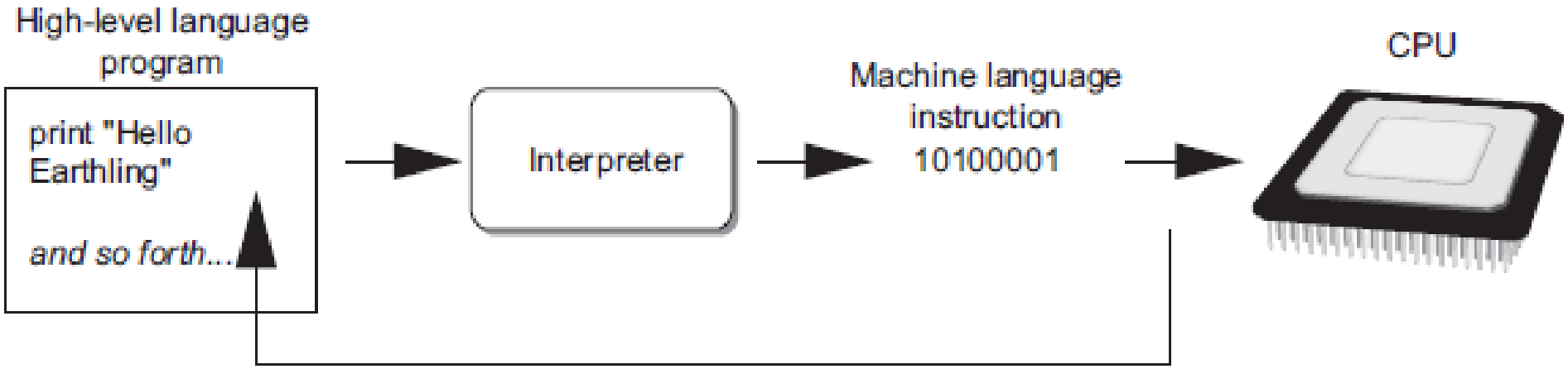
Compiler and Interpreter

- CPU understands only machine language Therefore, programs that are written in a high-level language must be translated into machine language.
- Depending on the language that a program has been written in, the programmer will use either a compiler or an interpreter to make the translation.
- A *compiler* is a program that translates a high-level language program into a separate machine language program.



Compiler and Interpreter

- Interpreter is also a program that both translates and executes the instructions in a high-level language program.
- As the interpreter reads each individual instruction in the program, it converts it to machine language instructions and then immediately executes them.
- This process repeats for every instruction in the program.



Introduction to C programming

Introduction to C Programming

- C programming language was developed by **Dennis M. Ritchie** in **1972** at Bell Telephone Laboratories
- It was a great event in the history of programming and it shaped most of the later advancements of computer programming during last four centuries
- Most of the modern programming languages were created based on fundamentals of C programming language.
- Almost all the modern operating systems were developed using C. Also, there are many other programming languages were developed using C.
- C was created based on BCPL and B. The first C compiler was developed using B language(Specifically NB – New B).

Features of C Programming Language

- C is a **general-purpose** programming language.
- C provides the fundamental control-flow constructions required for well-structured programs.
- C is a **relatively "low level"** language. that means it creates code that's a lot closer to what machines really understand.
- Although C matches the capabilities of many computers, it is independent of any particular machine architecture.
- Originally C is not a strongly-typed language but it has evolved.
- C is an **imperative**, **procedural** language.
- The C language is small in size, and simple in structure.

C Standards

There are three standards were developed with the evolution of C.

- ANSI C
- C99
- C11

ANSI C is from the late 1980s and is used for the oldest code. A lot of things were fixed up in the C99 standard from 1999. Some cool new language features were added in the current standard, C11, released in 2011.

The differences between the different versions are not huge.

Why we learn C?

- C is used where **speed**, **space**, and **portability** are important.
- C was initially used for system development work, particularly the programs that make-up the operating system.
- C was adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language.
- Some examples of the use of C might be **Operating Systems**, **Language Compilers** , **Assemblers**, **Language Interpreters**, **Text Editors**, **Print Spoolers**, **Network Drivers**, **Databases**, and Etc.
- Even though it is the best feature of C, the usage of it is not limited to **system development**. Since it is a general purpose language, it can be used to write **general programs** as well.

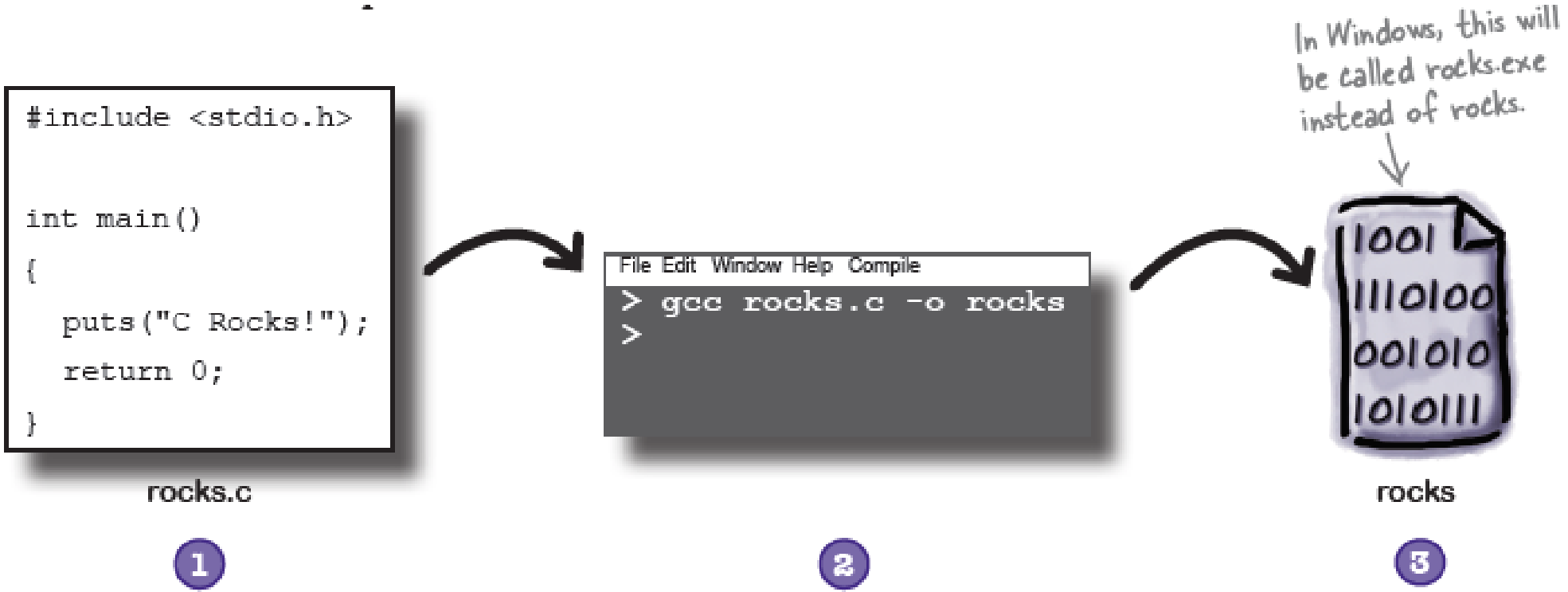
Our First C Program

```
#include <stdio.h>
int main()
{
   printf("Hello, world!\n");
   return 0;
}
```

How to run it?

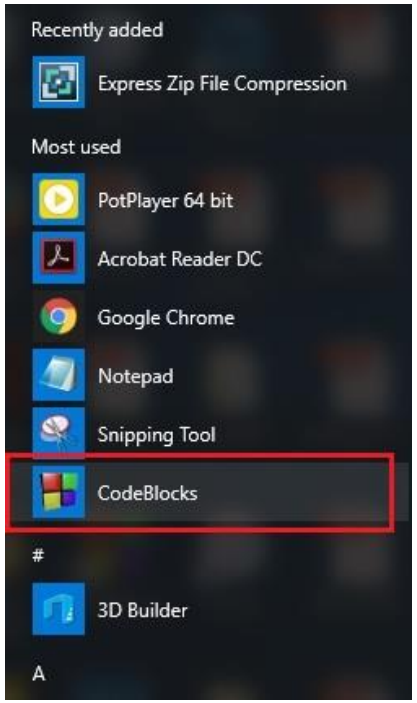
- The mechanism of running a C program depends on the system it is written on.
- For example, on the UNIX operating system, the program should be written in a file whose name ends in ". c“ (such as **hello.c**).
- Then compile it with the command **cc hello.c**
- If there are no **syntax errors**, such as omitting a character or misspelling something, the compilation will proceed silently, and make an executable file called **a. out**.
- If you run a. out by typing the command **a.out**, it will print **hello, world** on screen.

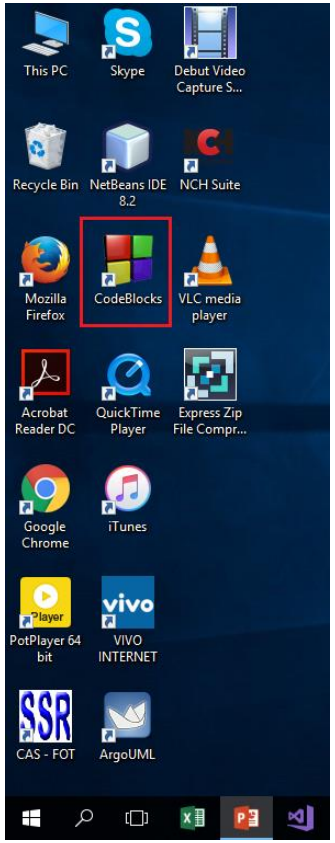
How to run it?


#include <stdio.h>
int main()
{
puts("C Rocks!");
return 0;
}
rocks.c
1
File Edit Window Help Compile
> gcc rocks.c -o rocks
>
2
In Windows, this will be called rocks.exe instead of rocks.
1001
1110100
001010
1010111
rocks
3

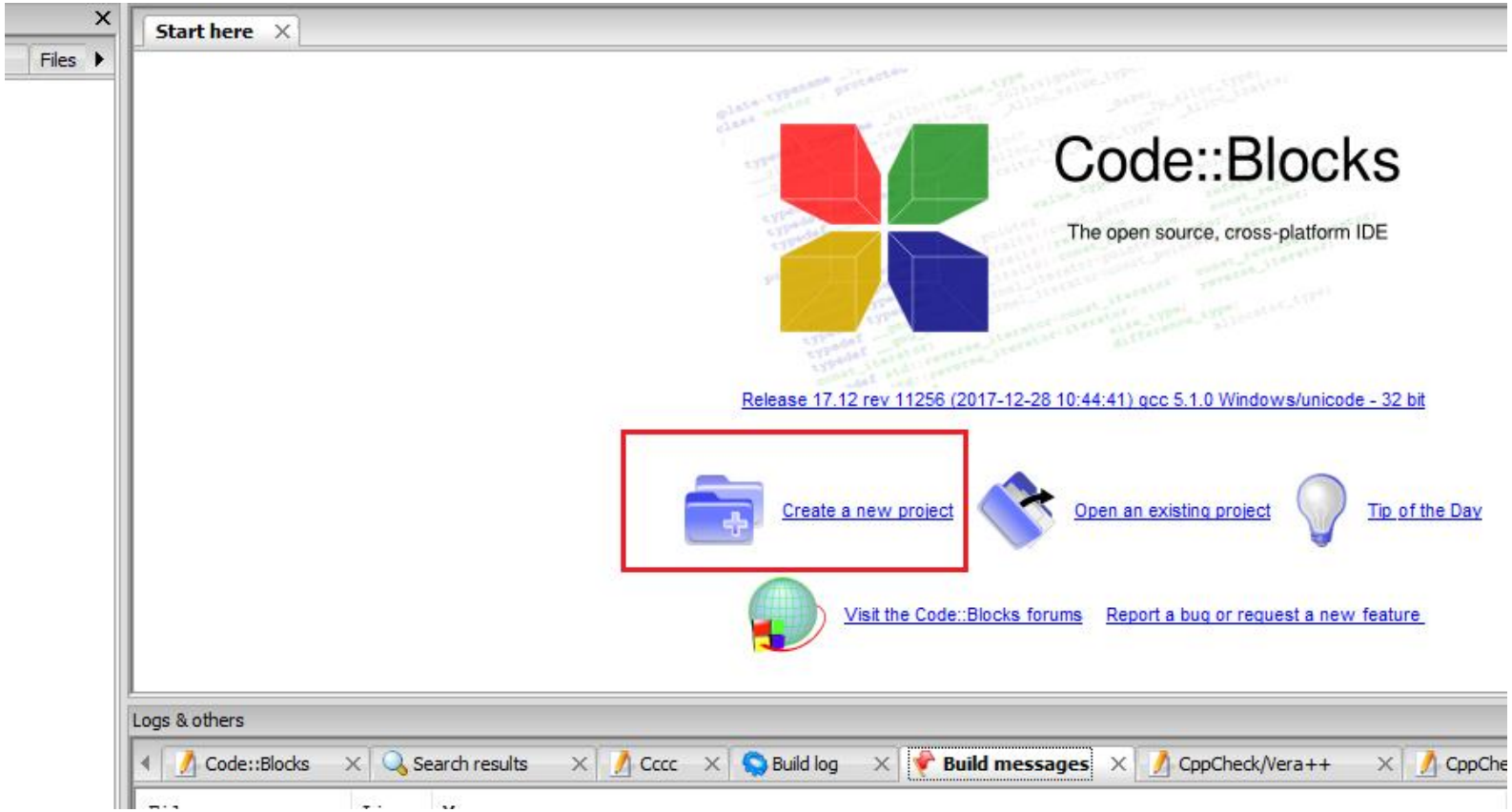
How to do it on Code::Blocks 1

Open Code blocks using Windows Start Menu or Clicking the Desktop short cut



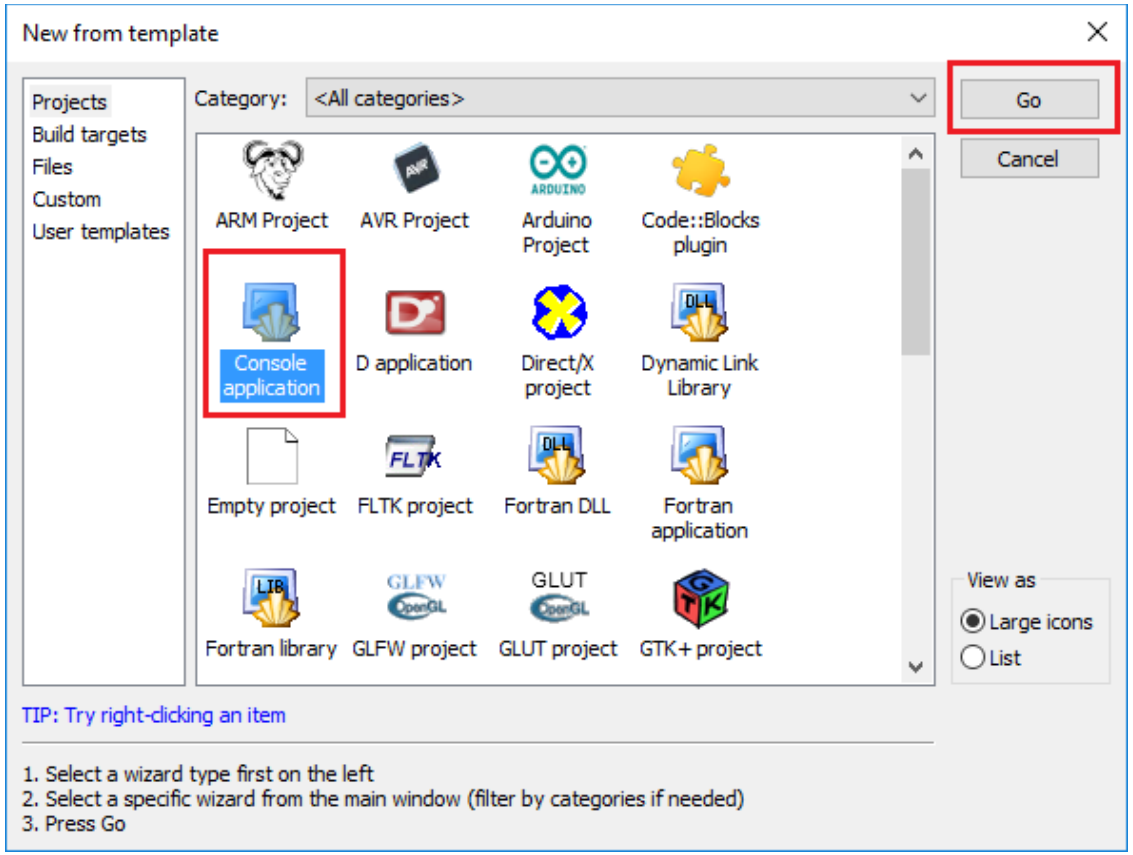


How to do it on Code::Blocks 2



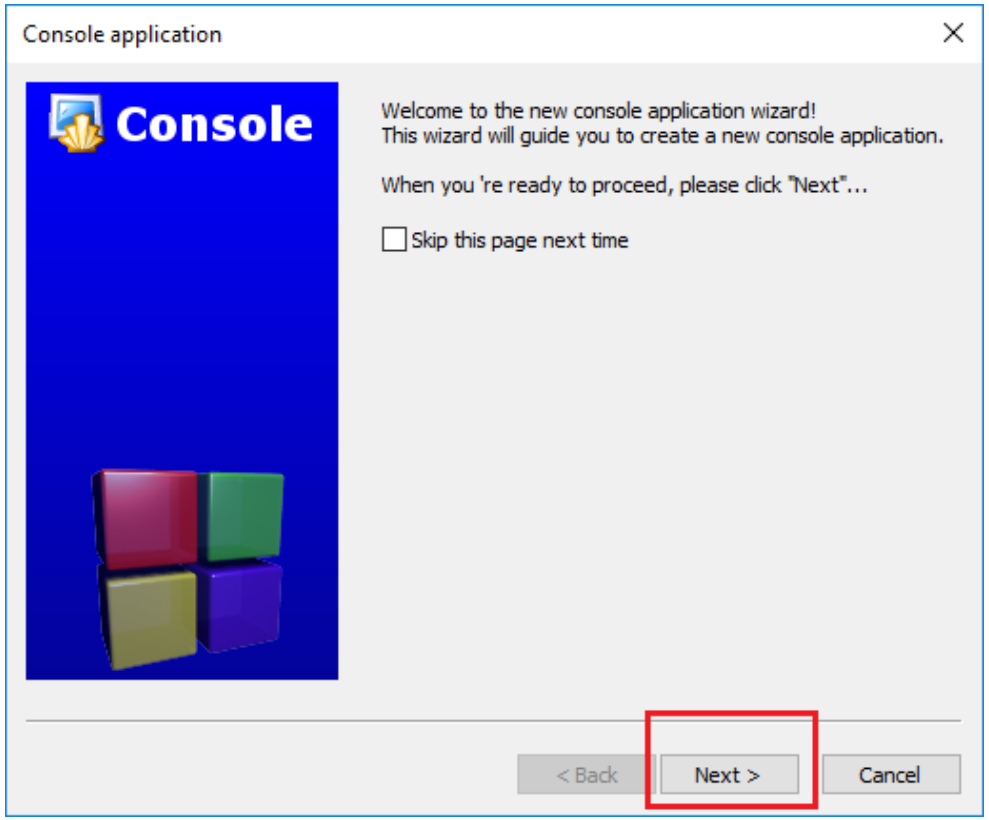
Click on “Create a new project” link to create new C project.

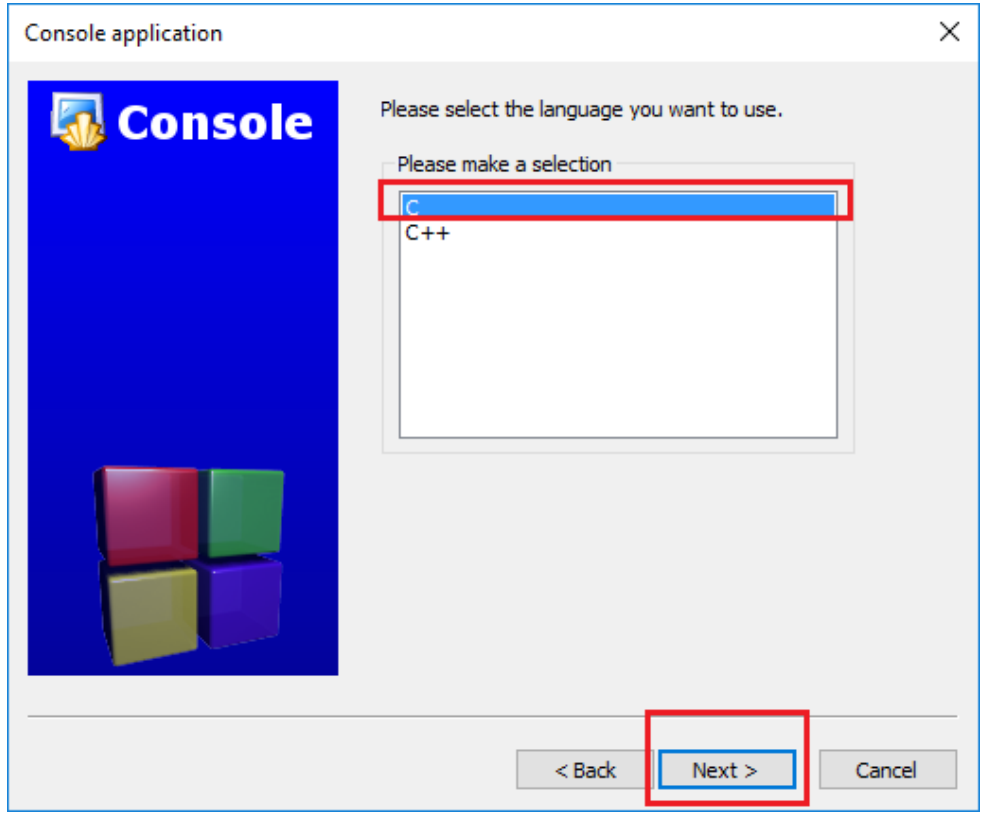
How to do it on Code::Blocks 3



Select “Console Application” in the list and Click on “Go” button.

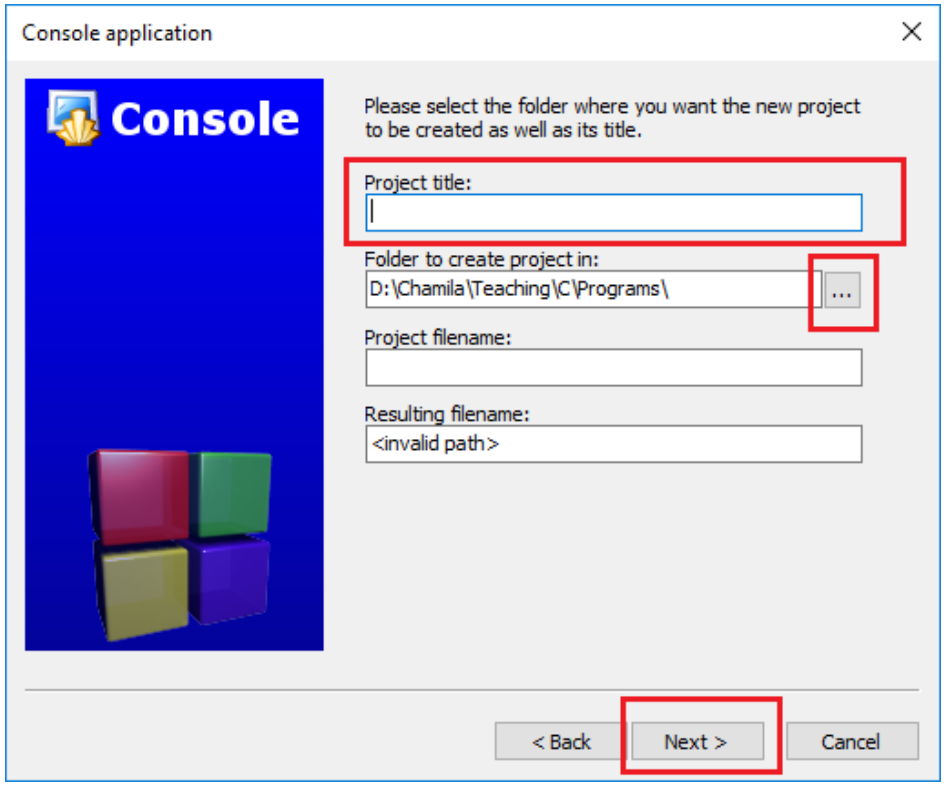
How to do it on Code::Blocks 4

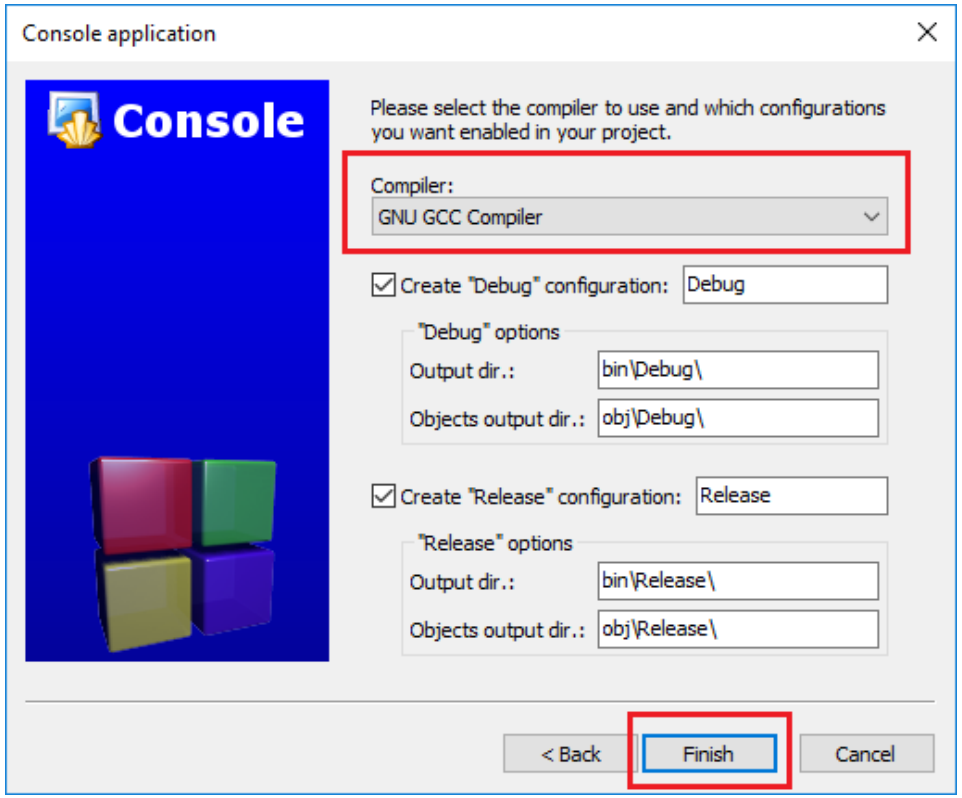




Select the “C” language as the language you want to use and then click on “Next” button.

How to do it on Code::Blocks 5

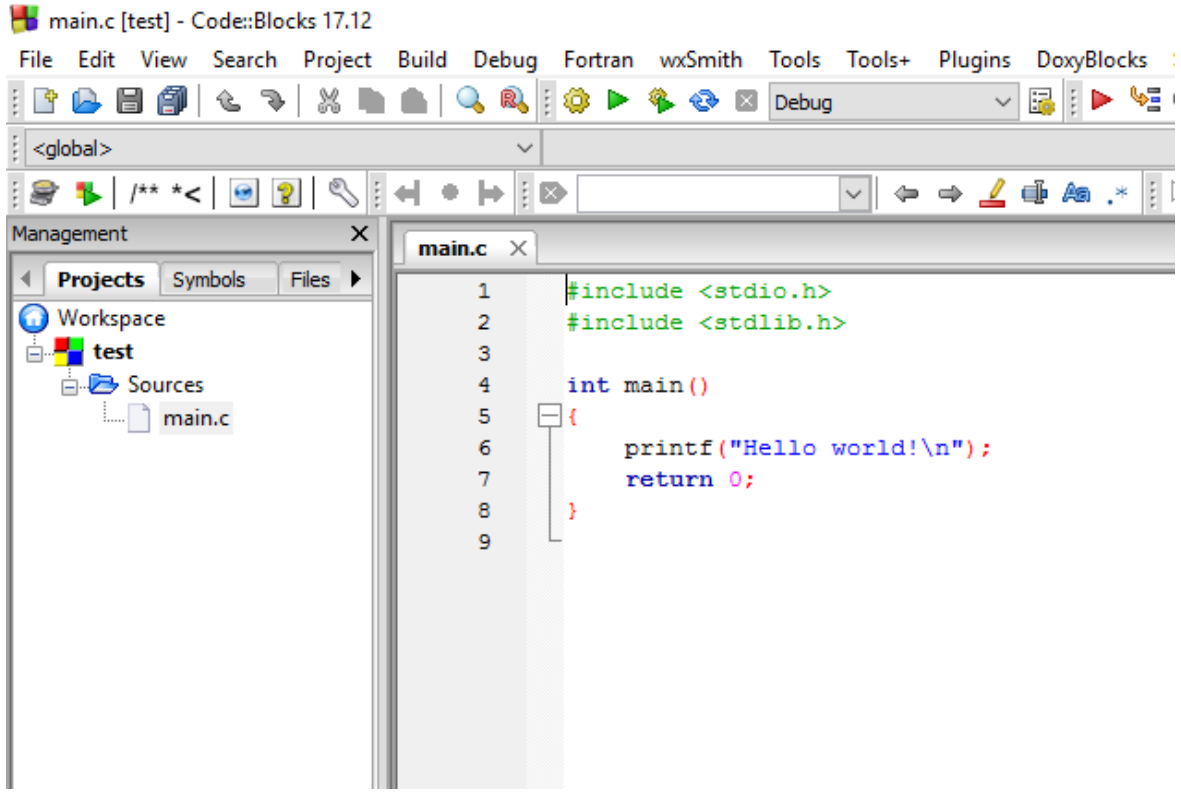




Give any name for your C project and select the folder to be saved.

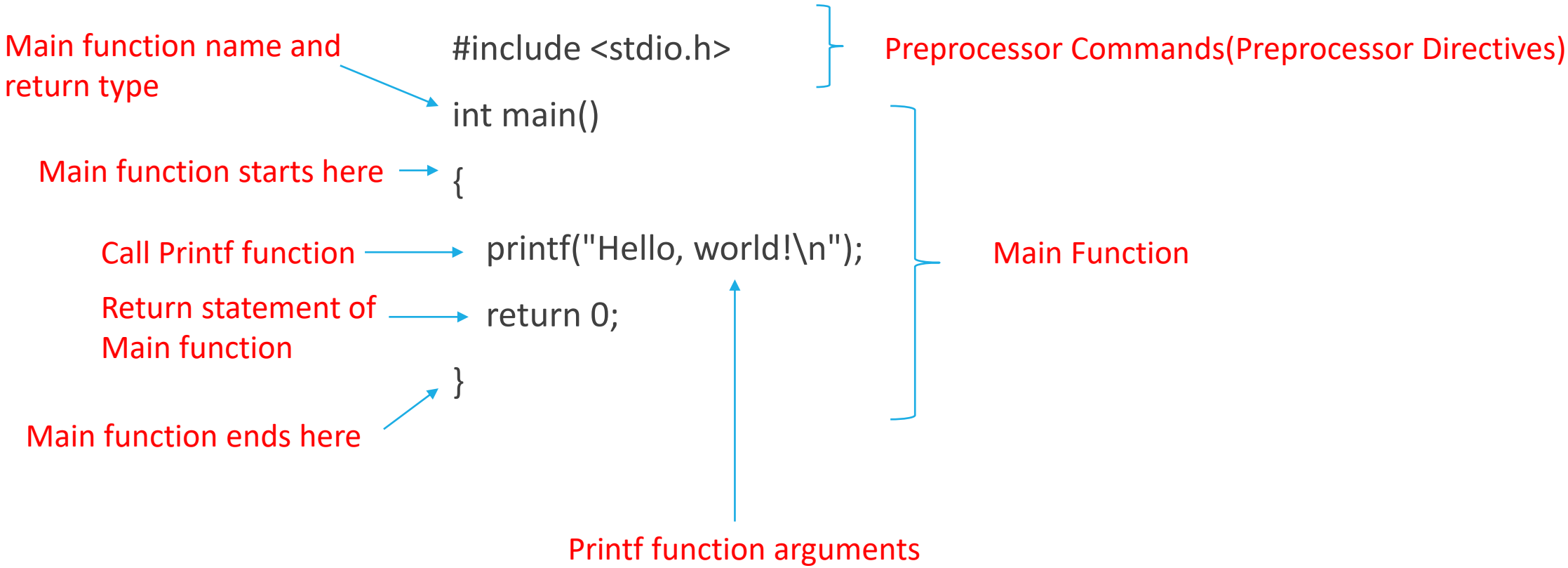
In next window, select the compiler as “GNU GCC Compiler” and click “Finish” button.

How to do it on Code::Blocks 6



Code::Blocks editor would load auto created main.c program as above.

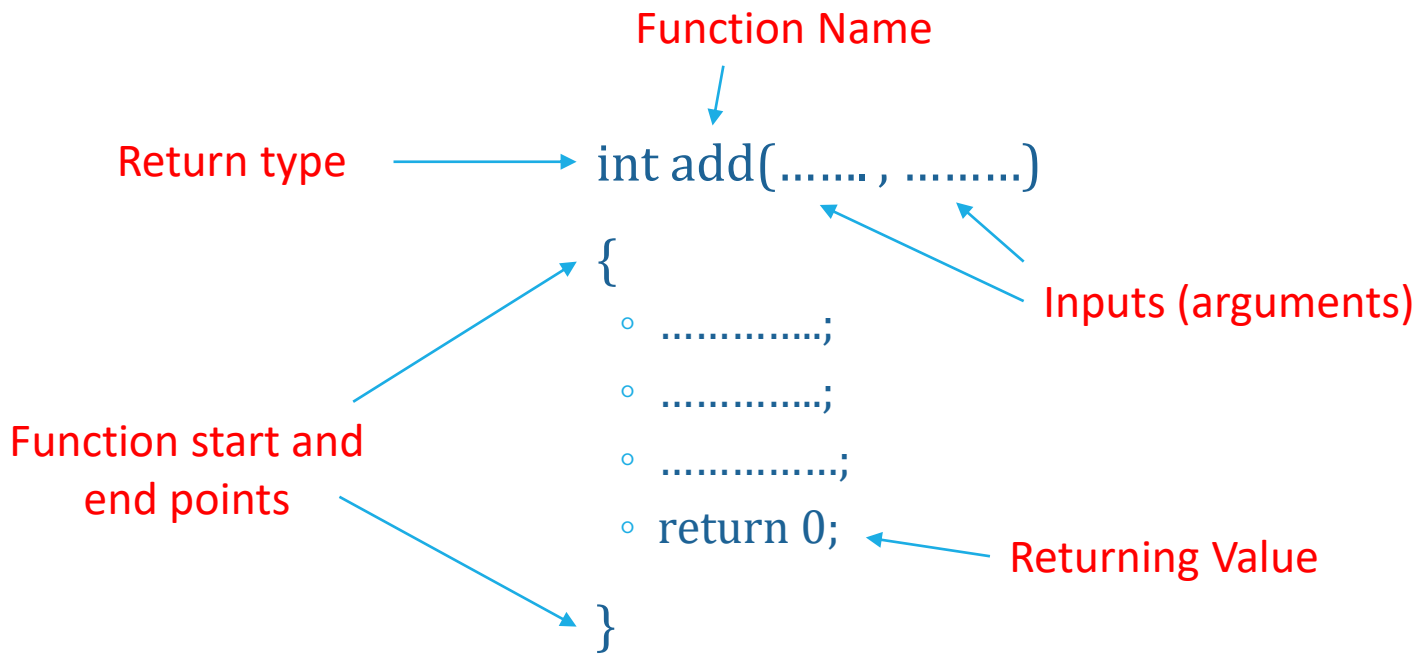
Analysis of the Program



Brief Introduction to Functions

- Writing a computer program is a process of writing a set of instructions to perform specific task.
- In C, these instructions(statements) are organized into groups which are called **functions.** Each function is a block of instructions which are related. They may carry out a sub task.
- Any number of functions can be added to a C program depending on the purpose of the program. Function, has a **name**, inputs(**arguments**), an output (**return value**) and a **body**.
- As the programmer, we can define the name of the function. However, there are several rules to be followed when naming a function.
- Function may have **zero or any number** of arguments. Function may have a **zero or one** return value.

Anatomy of a function



Main Function

- C program can have many number of developer defined functions however, a program that intended to be run **must** have a special function called **main function**.
- When the operating system runs a program, it passes control of the computer over to that program.
- The operating system needs to know where inside the program the control needs to be passed.
- In the case of a C language program, it's the main function that the operating system is looking for.
- main is the gate way to the program, and running of the program starts there.
- Inside main function we can **call** to other functions, which are inside the same program or other programs.
- Main function has **no arguments** and it usually returns an integer. In some programs it returns nothing (void).

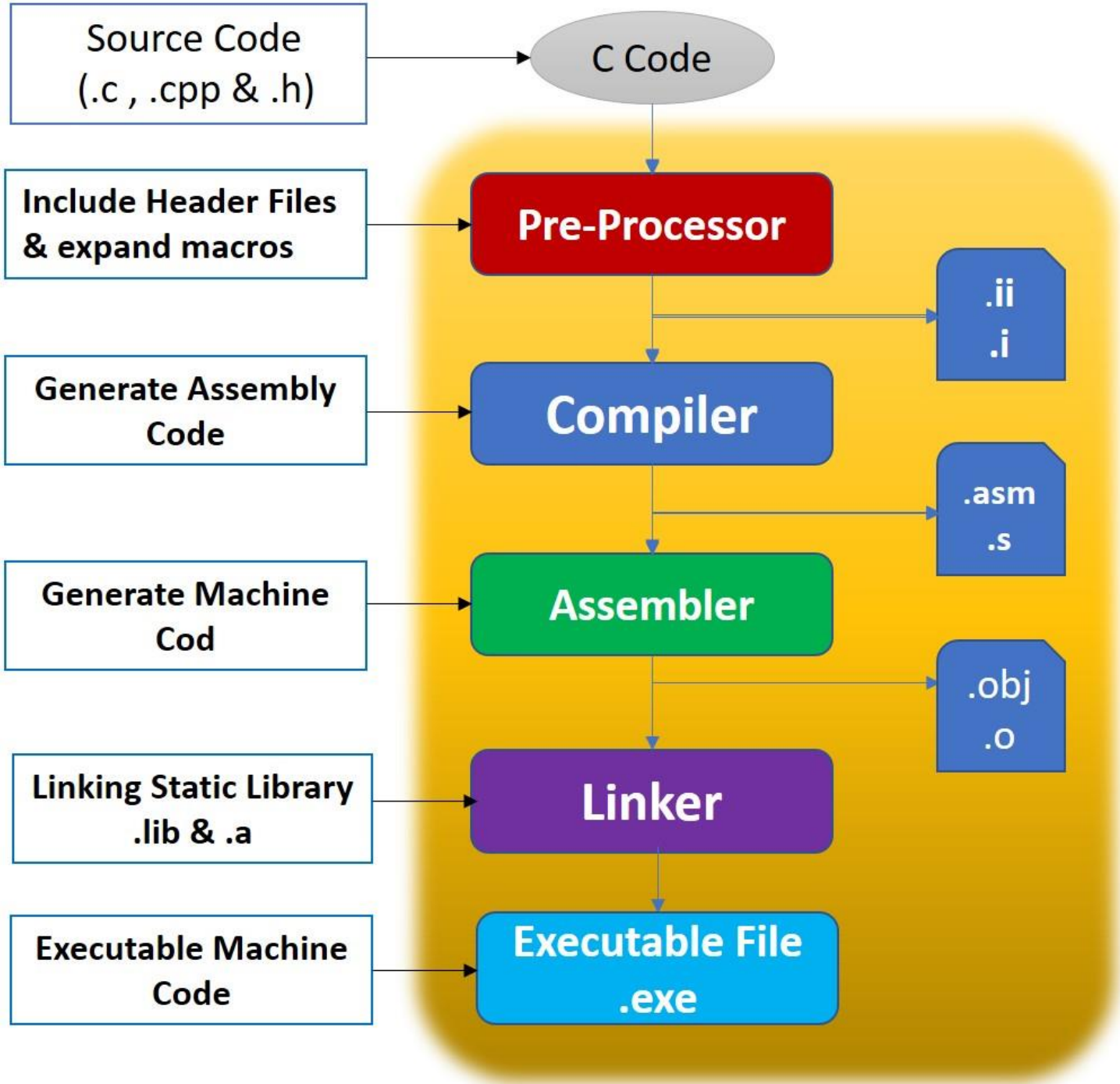
Preprocessor Directives

- The **#include <stdio.h>** is a preprocessor command.
- This command tells compiler to include the contents of **stdio.h** (standard input and output) file in the program.
- The stdio.h file contains functions such as **scanf()** and **print()** to take user input from keyboard and display output to the screen respectively.
- If printf() function is used without writing #include <stdio.h>, the program will not be compiled.

C Compilation Process

- The compilation is the process of converting the **source code** into **object code**.

- The compilation process can be divided into four steps
  - **Preprocessor**
  - **Compiler**
  - **Assembler**
  - **Linker**


Source Code
(.c , .cpp & .h)
C Code
Include Header Files
& expand macros
Pre-Processor
.ii
.i
Generate Assembly
Code
Compiler
.asm
.s
Generate Machine
Cod
Assembler
.obj
.o
Linking Static Library
.lib & .a
Linker
Executable Machine
Code
Executable File
.exe

Preprocessor

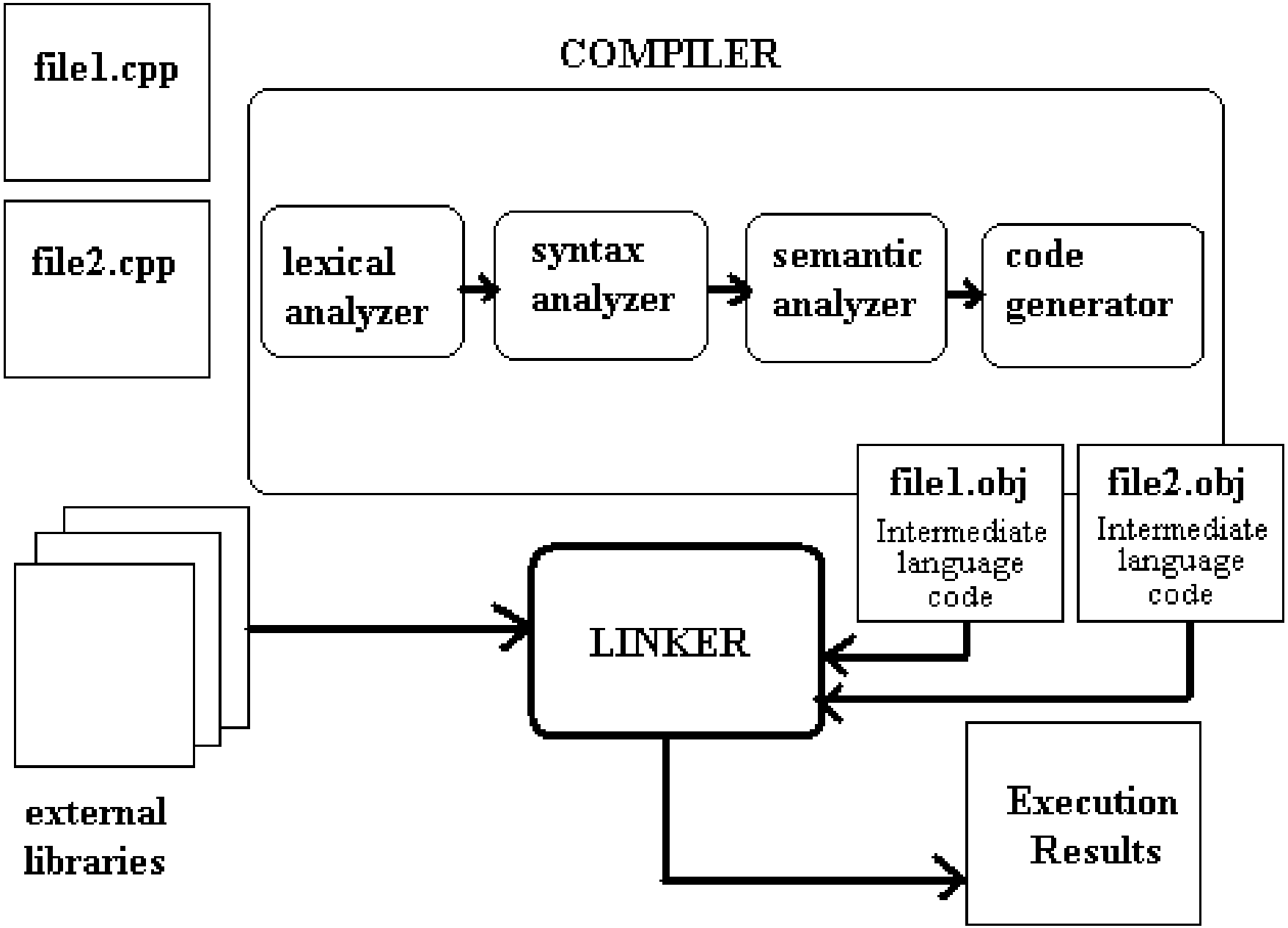
- Preprocessor prepares the source for compiling process by replacing the **pre-processor directives**.
- **Preprocessor directives** can be identified in code by the **prefix #**.
- One example is **#include** directives which are used to link code in one file with that in another.
- When compiling the C program, the pre-processor takes this #include and copies and pastes the code defined in that header file into the file that is including it.

Compiler

- The compiler converts this code into assembly code.
- Assembly language is a low-level programming language that more closely resembles the machine instructions of a CPU
- At the compilation stage, the C compiler looks for errors.
- If an error is found, then **compilation is stopped entirely**. You won't be able to compile your C program until all errors are fixed.

Assembler

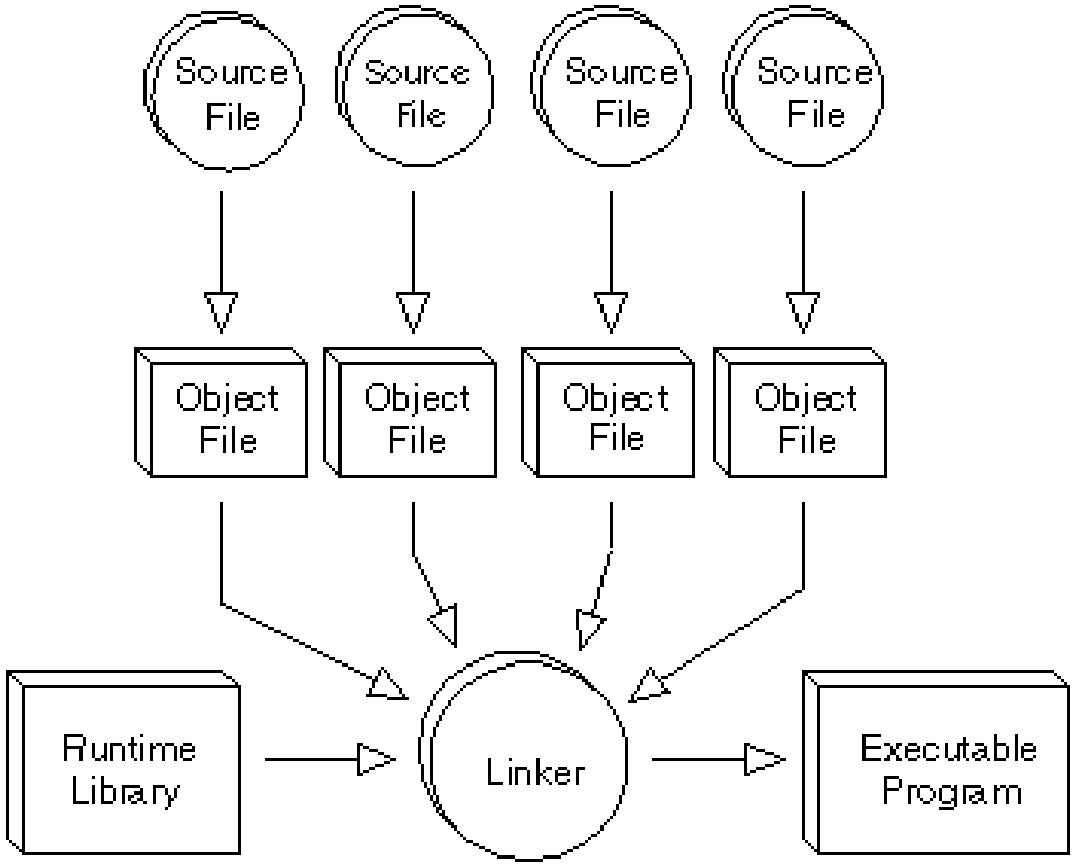
- The assembly code is converted into object code by using an assembler.
- The name of the object file generated by the assembler is the same as the source file.
- The extension of the object file **'.obj**,' (usually in DOS) or **'.o**' (usually in UNIX).
- If the name of the source file is **'hello.c',** then the name of the object file would be **'hello.obj**' or '**hello.o**'.


file1.cpp
file2.cpp
COMPILER
lexical analyzer
syntax analyzer
semantic analyzer
code generator
file1.obj
Intermediate language code
file2.obj
Intermediate language code
LINKER
external libraries
Execution Results

Linker

- These library functions which are used by our C program are pre-compiled, and the object code of these library files is stored with '.lib' (or '.a') extension.
- The main working of the linker is to combine the object code of library files with the object code of our program.
- Sometimes the situation arises when our program refers to the functions defined in other files
- Then linker links the object code of these files to our program.
- The output of the linker is the executable file

Linker


Source File
Source file
Source File
Source File
Object File
Object File
Object File
Object File
Runtime Library
Linker
Executable Program

Loader

- Whenever we give the command to execute a particular program, the loader comes into work.
- The loader will load the .exe file in RAM and inform the CPU with the starting point of the address where this program is loaded.

Questions?