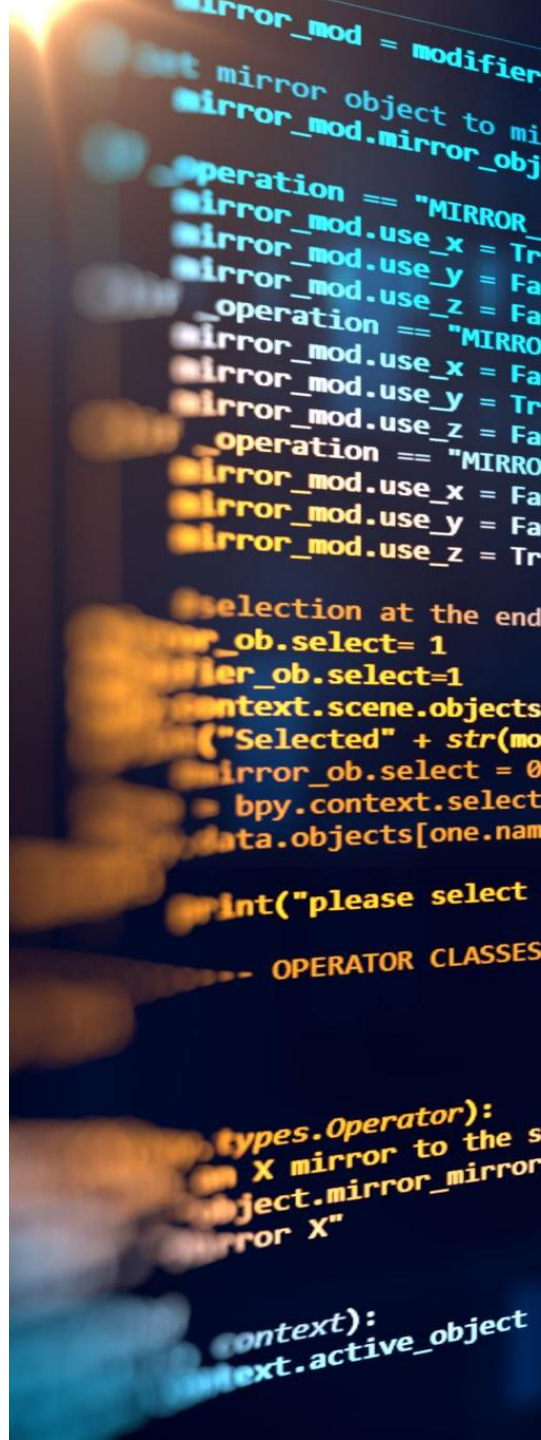
CS 1037
Fundamentals of Computer Science II

C Fundamentals

Ahmed Ibrahim



Agenda

- Introduction
  - Characteristics of C
  - Applications of C
  - C for data structures
- Compiling and Program Structure
- Data Types and Variables
- Operations and Expressions
- Flow Controls
- Functions

Introduction

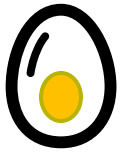
- Background of C
  - C, developed by Dennis Ritchie between 1969-1973, enabled UNIX portability and application program compatibility across platforms, making C widely versatile.
- Standardization
  - The first book on C, The C Programming Language by Brian Kernighan and Dennis Ritchie (1978), became the standard reference for learning C, known as K&R. C evolved throughout the 1980s, necessitating a standardized version to ensure compatibility across compilers.
  - The first C standard (ANSI C/C89) was approved in 1989 by ANSI and later by ISO. C99 and C18 followed. We will use C89 and the GNU C compiler (GCC) in this course.

Characteristics of C

Several characteristics make C the de facto standard programming language.

- **C is a high-level programming language**
  - Programmers write C source code programs using any text editor. However, C source code programs are not executable; they must be converted (**compiled**) into executable (**machine code**) programs and saved as files. These files can then be loaded into memory to run.
  - **C compilers** are programs that convert C source code programs into executable programs.
- **C is a low-level language**
  - It supports operations such as **memory addressing** and **bitwise operations**. This unique feature of the C programming language enables it to perform low-level operations efficiently.
- **C is a portable (or cross-platform) language**
  - A program written in C is compiled by a platform-dependent compiler and then runs on the computing platform's computers or operating systems. This differs from Java programs, which are **compiled once** and can run on any platform where the Java Virtual Machine (JVM) is installed.

Characteristics of C (cont.)



- **C is extensible**
  - C is extensible, using libraries of functions to enhance its features and functionalities.
  - C++ was designed as an **extension of C** to support object-oriented programming, adding many new features and paradigms such as classes, inheritance, polymorphism, and templates.
  - C++ is suitable for large-scale software development and more complex applications, while C remains a better choice for simpler, low-level, or performance-critical tasks.
- **C is a stable**
  - With only 32 keywords, a limited number of primitive data types, and a minimal set of operations, C is a language that's easy to grasp and work with.
  - This is why C is known for its stability as a programming language, with minimal changes over time due to standardization.

Applications of C

- C is used for operating systems and high-performance computing.
  - C was **instrumental** in developing operating systems such as UNIX, Linux, Windows, macOS, iOS, and Android.
  - Application programs written in C can seamlessly integrate with the operating system without significant overhead. Therefore, C is used for writing device drivers, utility programs (such as compilers), system programs (such as shells/command consoles/terminals), and native applications that run on top of operating systems.
- C is used for interpreters of other programming languages.
  - C has been used to create **interpreters** for several programming languages, such as **Perl**, **PHP**, and **Python**.

C for Data Structures

- The characteristics of C make it a primary language for implementing data structures.
- C allows for accessing and operating on data components within data structures efficiently.
- Understanding data structures at the **memory level** enables efficient design and implementation for algorithms and applications.
- In this course, we use C to gain a deep understanding of how data structures function and to learn how to utilize them for high-performance algorithm design and implementation.

Compiling and Program Structure

- To learn a new programming language, we need to start writing codes:

```c
#include <stdio.h>
void main() {
  printf("hello, world\n");
}
```

- This program might be stored in a file named hello.c
- The file name doesn't matter, but the .c extension is often required.
- Type the following command to compile (assuming the GCC compiler was installed):

```
gcc hello.c
```

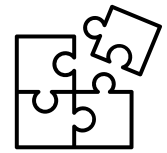
Compiling A Program

- Compiling will create an executable program named **a.exe** in the same folder under Windows OS or **a.out** in Linux or macOS.
- Under Windows, we run **a.exe** by command: **a.exe** or simply a. Under Linux, we run a.out by command: **./a.out**
- The output on the console will be:

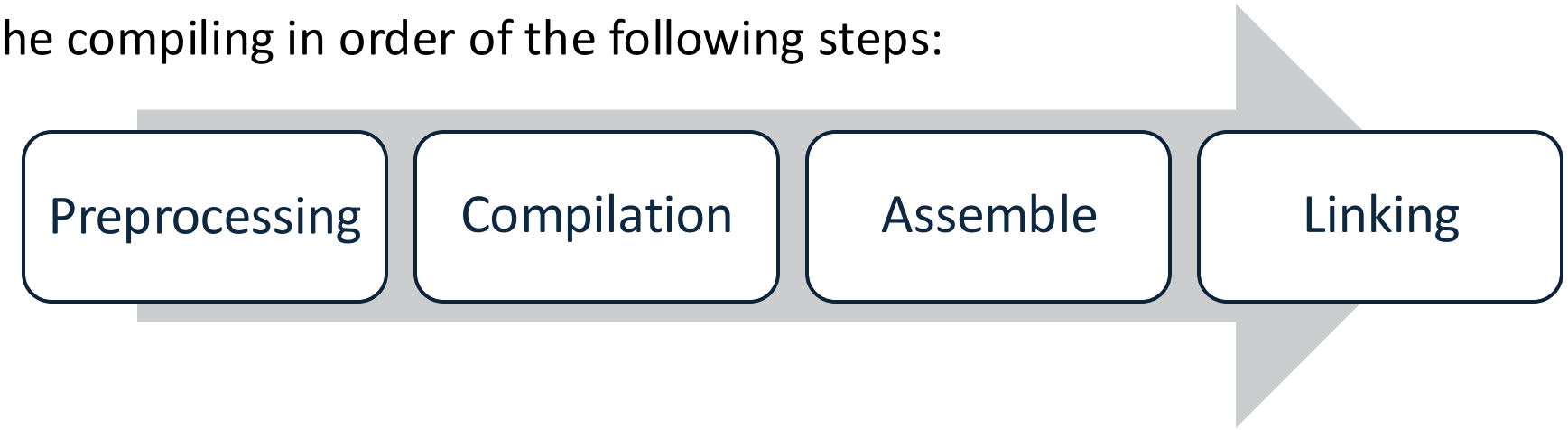
```
hello, world
```

- You have created, compiled, and run your first C program.

What happens with compiling?



- The compiling converts a C source code program to an executable program. The compiler does the compiling in order of the following steps:



- Resolve lines with preprocessor directives
- Convert the C source code program into an assembly program
- Convert the assembly program to a binary machine program, called: **object program**
- Combine the necessary object programs together to create an executable program of binary machine code

Common Compiler Options

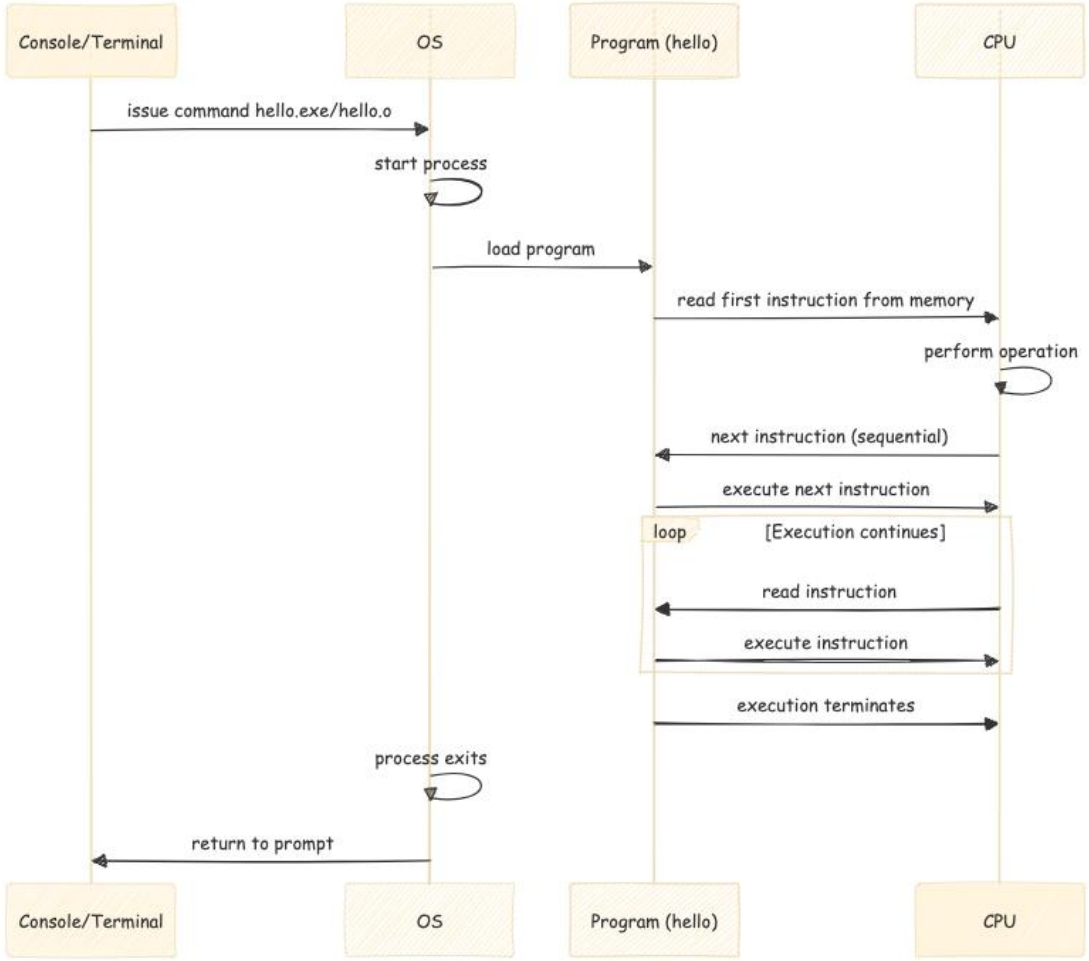
- GCC compilers have options to do compiling.
- Command gcc --help will list available options for gcc.
- The following options are commonly used:

Option	Purpose	Example
-c	Suppress linking step, just generate an object file	gcc –c hello.c will create the object program file hello.o
-g	Embed diagnostic information into the object file for debugging	gcc –g hello.c
-o	Specify output file name	gcc –o hello.exe hello.c creates hello.exe file.
-S	Suppress the assemble and linking steps and output the assembly program file.	gcc –S hello.c will create assembly program file hello.s

File Extension Convention

Extension	File type
.c	C program source code file, e.g. hello.c
.h	C program header file, e.g., sdtio.h
.exe	Default executable program file for Windows OS, .out for Linux and macOS.
.o	Object file, e.g. hello.o
.s	Assembly program source file, e.g., hello.s
.lib	Static library of object modules (.a for UNIX/Linux/macOS): contains pre-compiled code (object modules) that can be linked with your program during the compilation/linking process
.dll	Dynamic library file for Windows OS (.so for UNIX/Linux/macOS): The code is linked at runtime

What happens when running an executable program?



What happens when running an executable program?

- When the command hello.exe/hello.o is issued, the console/terminal starts a new process and loads the program 'hello' into memory.
- The program execution begins from the first instruction of the main function.
- An instruction is read from memory to the CPU, which then performs the operation.
- After each instruction is executed, program control moves to the next instruction, which could be sequential in most cases.
- Execution terminates when the main function ends, the process exits, and the console returns to the prompt for the next command.

A Problem Statement!

Create a C program with two functions: add to calculate the sum of two numbers, and minus to calculate their difference.

In the main function, call add(a, b) and minus(a, b), then print the results. The expected output is a + b = 3 and a - b = -1.

C Program Structure

- A C program is structured as a collection of functions, with the main function defining the starting point.
- functions may call other functions, and function declarations are typically made before definitions to avoid ordering issues in the code.

```c
/*
C program structure example
*/
#include<stdio.h>          // preprocessor directive include
int a;                     // global variable declaration
int add(int, int);         // function declaration
int minus(int, int);       // function declaration
int main()                 // main function
{
  a=1;                     // assign/set value 1 to global variable a
  int b=2;                 // declare local variable b and initialize/set it to value 2
  printf("a+b=%d\n", add(a, b));   // function calls
  printf("a-b=%d\n", minus(a, b)); // function calls
  return 0;
}
// definition/implementation of function add(int, int)
int add(int x, int y)      // function header
{
   return x+y;             // function body
}
// definition/implementation of function minus(int, int)
int minus(int x, int y)    // function header
{
  return x-y;              // function body
}
```

Steps to Solve the Problem

1. **Understand the Task –** Create two functions: add (sum of two numbers) and minus (difference of two numbers).
2. **Write Function Prototypes –** Declare add(int, int) and minus(int, int) above main.
3. **Plan the Program**:
    1. Define global variable a and local variable b inside the main function.
    2. Call both functions in the main function.
4. **Implement the Functions**:
    1. Write add to return the sum.
    2. Write minus to return the difference.
5. **Call Functions in main –** Call add(a, b) and minus(a, b) inside main.
6. **Print Results** – Use printf to display results in the specified format.
7. **Test and Debug** – Compile, run, and ensure correct output (a + b = 3 and a - b = -1).

C Program Organization

- The following command compiles the above programs to create and output the executable program named as 'testmain':

```
gcc addsub.c addsub_main.c -o testmain.exe
```

```c
/**
 * addsub.h
 * header file contains function headers
 */
/**
 * @param x - int value
 * @param y - int value
 * @return - sum of x and y, int value
 */
int add(int x, int y);
/**
 * @param x - int value
 * @param y - int value
 * @return -  subtract y from x, int value
 */
int minus(int x, int y);
```

```c
/**
 * addsub_main.c
 * test driver program containing the main function
 */
#include <stdio.h>
#include "addsub.h"
int a;
int main()
{
  a=1;
  int b=2;
  printf("a+b=%d\n", add(a, b));
  printf("a-b=%d\n", minus(a, b));
  return 0;
}
```

```c
/**
 * addsub.c
 * implementation of header functions
 */
#include "addsub.h"

int add(int x, int y)
{
   return x+y;
}

int minus(int x, int y)
{
  return x-y;
}
```

Keywords (Reserved Words)

- C has 32 keywords. We put them into five categories:

Category	Keyword
Basic data types	char, int, float, double, short, long, signed, unsigned, void
Define data types	typedef (create an alias), struct, union, enum - >all user-defined data types
Modifiers	const, auto, static, extern, volatile, register
Flow control	if, else, switch, case, default, goto, for, while, do, break, continue
Function	return, sizeof (returns the size (in bytes) of a data type or a variable.)

Please use the following QR code to check in and record your attendance.