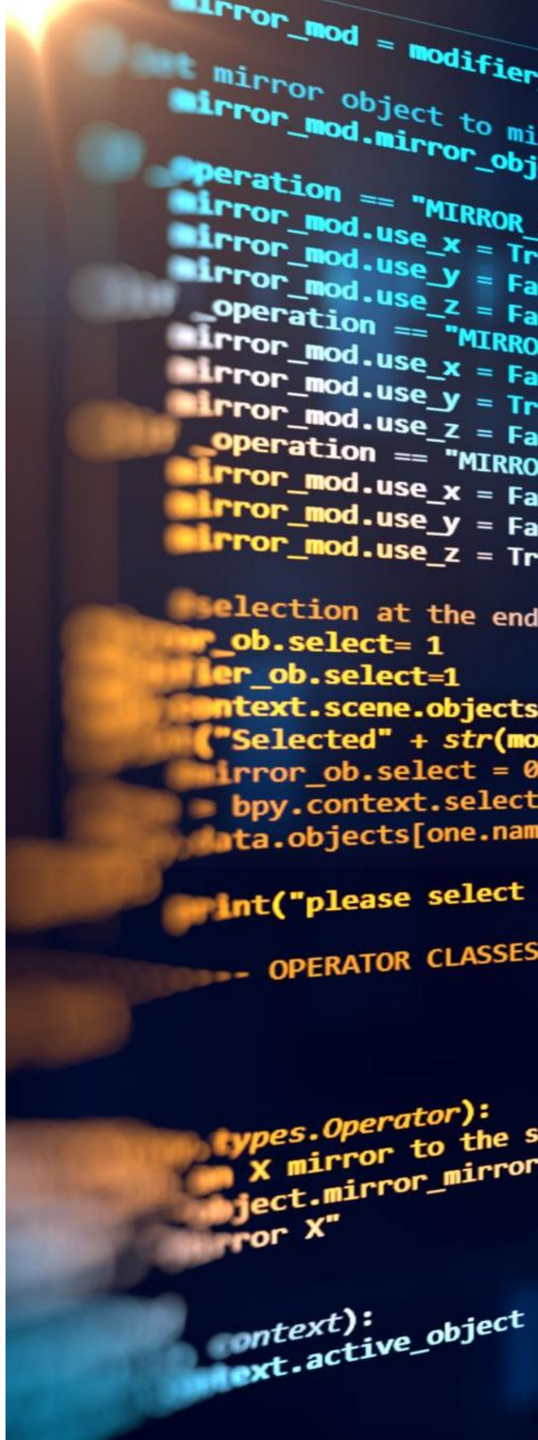
CS 1037
Fundamentals of Computer Science II

C Fundamentals (cont.)

Ahmed Ibrahim



Recap

- **Introduction to C**: Developed by Dennis Ritchie (1969-1973) for UNIX portability. Standardized in 1989 (ANSI C), followed by C99, C18.
- **Key Characteristics**: High-level, low-level (memory handling), portable, extensible (with libraries), and stable (32 keywords).
- **Applications**: Widely used in OS development, system utilities, device drivers, and interpreters.
- **Data Structures**: Ideal for understanding memory-level operations for algorithms.
- **Compilation Process**: Source code is compiled into an executable file via GCC. Includes steps: preprocessing, compilation, assembly, and linking.
- **Program Structure**: C programs are function-based, starting with main().
- **Common Compiler Options**: Flags like -c, -g, -o, and -S control various steps of compiling.
- **Problem-Solving Example**: Simple program with add and minus functions for summing and subtracting two numbers.

Recap: The compilation process

- The following diagram illustrates the C compilation process.
- The process begins with a preprocessed source code (.c) file, compiled into assembly code, and then assembled into an object file (.o). The linker combines this object file with libraries to create the final executable file, which can be run on the system.

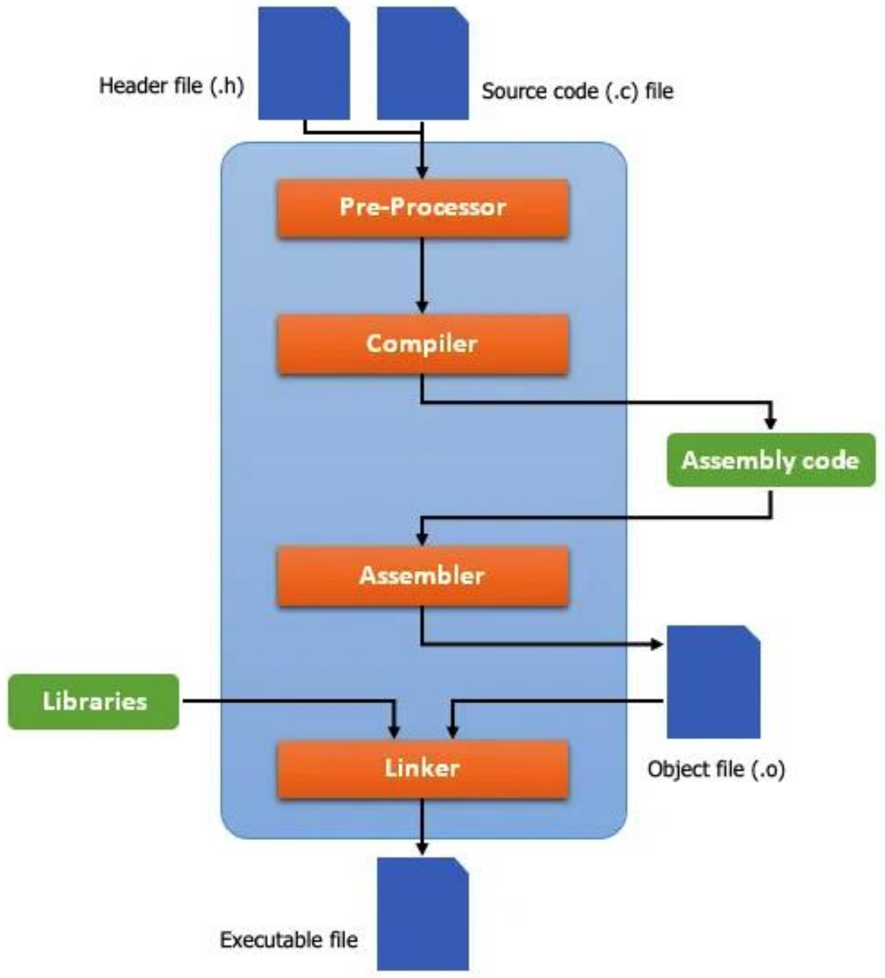
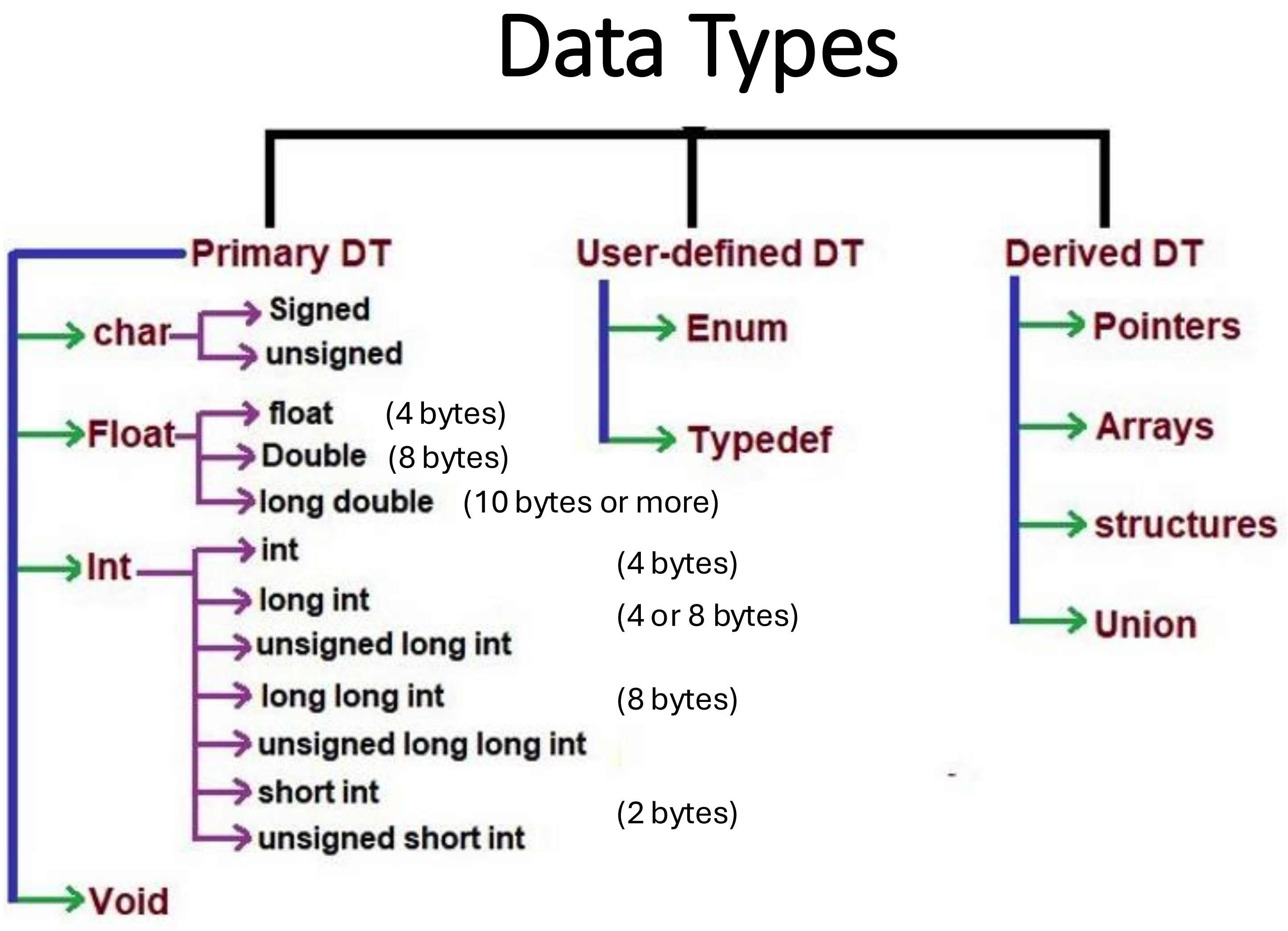


Image source: https://medium.com/@chiaracaprasi/the-compilation-process-in-the-c-language-22d46e8057d8

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

Data Types

- The following categorizes data types in C into three groups: **Primary** (e.g., char, int, float), **User-defined** (e.g., enum, typedef), and **Derived** (e.g., pointers, arrays, structures).



Variables

- A variable is a name (**identifier**) used to represent a data object that stores values at runtime.
- Memory is allocated for the variable at compile time and is instanced in memory when used at runtime.
- A variable allows the program to set or get values from its allocated memory block, with an absolute memory address assigned at runtime.

Variables (cont.)

- C is a typed language, meaning each variable must have a declared data type and be initialized before use.
- A variable declaration tells the compiler to allocate a relative memory block, while a value assignment generates instructions to store data in that block.
- At runtime, the compiler writes and reads values to/from the variable’s absolute memory location based on its declared type and assigned value.
- For example, `int x = 10;` declares `x` as an integer and initializes it with 10, with the compiler managing memory and instructions for accessing `x`.

Variables (cont.)

- Examples:

```
int a; // let compiler allocate 4 bytes memory for variable a
char c; // let compiler allocate 1-byte memory for variable c
float f; // let compiler allocate 4 bytes memory for variable f
a = 2; // let the compiler generate instructions to store value 2 to variable a
c = 'a'; // let compiler generate instructions to store 97 (0111001) to variable c
f = 1.41; // let compiler convert 1.41 to 32 bits single precision float number and store it to f
```

ASCII values

IEEE 754 standard

- It is a **good practice** to declare and initialize a variable in one statement as follows.

```
int a = 2; char c = 'a'; float f = 1.41;
```

- C allows to declare and/or initialize several variables of the same type in one line separated by a comma in one statement.
- For example, `int a=1,b=2,c;` is a valid statement.

Scopes

- A scope consists of a sequence of statements where identifiers (variables/functions) are declared and used.
- Scopes can be separated or nested, with a global scope encompassing the entire program.
- A **local variable** is declared within a code block **{}** and is accessible only within that block and its nested scopes.
- **A global variable** is declared outside any function and can be accessed by any function.
- The compiler binds variables to their respective scopes, allowing the same variable name in different scopes.

Question?

Consider the following code snippet:

```c
int x = 5;
void func() {
    int x = 10;
    printf("%d", x);
}
int main() {
    func();
    printf("%d", x);
}
```

What will be the output of this program?

A) 5 5

B) 10 10

C) 10 5

D) 5 10

Variable name convention

- C has restrictions on variable naming. C variable names **must** start with a letter, followed by letters, underscores and numbers.
- **It's important to note that variable names in C are case-sensitive**.
- C programming has two naming convention styles for program readability: **underscore_style** and **camelCaseStyle**.
    - The camelCaseStyle is also used in C++ and Java programming.
    - The underscore_style was used in classical C programming.
    - We will use the **underscore_style** in course code examples.

Constants

- Constants are fixed data values in programs, like Pi (3.1415926).
- Using a constant multiple times is easier by assigning it a simple name.
- **Define constants by macro**: The C preprocessor replaces constant names with actual values during preprocessing (e.g., `#define PI 3.1415926`).
- **Define constants by read-only variables**: Use the const keyword to declare a constant variable, which cannot be reassigned later (e.g., `const float pi = 3.1415926`).
- Constant variables offer single precision representation in compilation, reducing redundant conversions.

Operations and Expressions

- **Arithmetic operations** for character, integer, and floating types.
  - **Operators**: + (add), - (subtract), * (multiply), / (divide), % (modulus)
  - **Operands**: char, int, float, double (modulus only for non-floating types)
- **Boolean** and **bitwise** operations are also available.
  - Relational operators: <, <=, >, >=, ==, !=
  - Logical operators: && (AND), || (OR), ! (NOT)
  - Boolean expression: Evaluates 1 (true) or 0 (false).

Operations and Expressions Example

- This program demonstrates arithmetic operations, relational comparisons, and logical operators in C by performing basic calculations and evaluating conditions with simple output.

```c
#include <stdio.h>

int main() {
    int a = 10, b = 5, x = 1, y = 0;

    // Arithmetic Operations
    printf("Sum: %d, Product: %d\n", a + b, a * b);

    // Relational Operators
    printf("a > b: %d, a == b: %d, a != b: %d\n", a > b, a == b, a != b);

    // Logical Operators
    printf("x && y: %d, x || y: %d, !x: %d\n", x && y, x || y, !x);

    return 0;
}
```

format specifier

Bitwise Operations

- Operate on binary numbers at the individual bit level.
- This program demonstrates basic bitwise operations, including AND, OR, XOR, NOT, left shift, and right shift, using two integers to manipulate their binary values.

```c
#include <stdio.h>

int main() {
    unsigned int a = 6, b = 3;  // 6 = 110, 3 = 011

    //signed int: Used when negative numbers are needed.
    //unsigned int: Used when only non-negative numbers are needed
    //the default is signed int

    printf("a & b: %d\n", a & b);  // AND
    printf("a | b: %d\n", a | b);  // OR
    printf("a ^ b: %d\n", a ^ b);  // XOR
    printf("~a: %d\n", ~a);        // NOT
    printf("a << 1: %d\n", a << 1);  // Shift left
    printf("a >> 1: %d\n", a >> 1);  // Shift right

    return 0;
}
```

Thank you

References

- Data Structures Using C, second edition, by Reema Thareja, Oxford University Press, 2014.