Lecture 04: Introduction to C

Md. Taslim Arefin
Associate Professor, Department of ETE
Daffodil International University

What is C?

- C is a general-purpose, structured programming language that is powerful, efficient and compact
- Its instructions consists of terms that resemble algebraic expressions, supplemented by certain English keywords such as if, else, for, and do
- Features modern flow control and data structures, and a rich set of operators
- Contains additional features that allow it to be used at a lower level
- Used for writing system programs and application programs

History of C (1/3)

- C was developed in the 1970s by Dennis Ritchie at Bell Labs (Murray Hill, New Jersey) in the process of implementing the Unix operating system on a DEC PDP-11 computer
- 1960s, CPL (Combined Programming Language), (Barron et al., 1963)
  - Purpose: to create a language that was capable of both high level machine independent programming and would still allow the programmer to control the behavior of individual bits of data.
  - Drawback: it was too large for use in many applications

History of C (2/3)

- In 1967, BCPL (Basic CPL)
  - a scaled down version of CPL while still retaining its basic features
- In 1970, B
  - Ken Thompson developed the B language, which was a scaled down version of BCPL written specifically for use in systems programming
- In 1972, C
  - Dennis Ritchie returned some of the features found in BCPL to the B language and developed C

History of C (3/3)

- Limited to use within Bell Laboratories until 1978
- In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C in their book titled “The C Programming Language”, now known as the K&R C
- ANSI formed a committee in 1983 to establish a standard definition of C, now known as ANSI C (1989)
  - Updated in 1995
- New features added in 1999, now known as C99

Features of C

- Small:
  - C is a language of few words, containing only a handful of terms, called keywords, which serve as the base on which the language’s functionality is built
- Portable:
  - Portable means that a C program written for one computer system (an IBM PC, for example) can be compiled and run on another system (a DEC VAX system, perhaps) with little or no modification
  - C provides a standard library of functions that work in the same way on all machines

Features of C

- Middle-level language
  - C is often called a middle-level computer language because it combines the best elements of high-level language with the control and flexibility of assembly language
- Structured Language
  - C allows programmer to divide program into modules
  - C provides all basic control structures
  - Use of subroutines that employ local variables
  - Use of code block
  - No use of go-to statements

C Is a Programmer’s Language

- Not all computer programming languages are for programmers
- For example, COBOL was designed, in part, to enable nonprogrammers to read and presumably (however unlikely) to understand the program
- In contrast, C was created, influenced, and field-tested by working programmers
- C gives the programmer what the programmer wants: few restrictions, few complaints, block structure, stand-alone functions, and a compact set of keywords

Preparing to Program

- The Programming Process
  - Determine the objective of the program
  - Design your solution
    - Inputs, outputs and logical steps to achieve the outputs
  - Code your solution
  - Compile your program
    - Handling errors
  - Run and Test your program

Structure of a C Program

- A program is a sequence of instructions
- Instructions of a C program are written as a statement
- A statement is terminated by a semicolon (`;`)
- One or more statements forms a block (compound) statement with the individual statements enclosed within a pair of braces, i.e., `{ }`
- All executable statements must be inside a function

Structure of a C Program

- A function is where all program activity occurs
- Every C program consists of one or more functions
- Every C program must contain a special function named `main`
  - The statements within this function is the first one to be executed
- Comments are written within the delimiters `/*` and `*/`
  - E.g., `/* this is a comment */`

First Program: hello.c

```
/* A simple C program that outputs two lines of text */

#include <stdio.h>                /* I/O header file */

main()                            /* main function heading */
{
   printf("Hello, world\n");       /* call to printf */
   printf("Welcome to Cse122\n");
}
```

Comments in C

- The first line
  `/* A simple C program that outputs two lines of text */`
  starts with `/*` and ends with `*/`
- Anything written between `/*` and `*/` is called a ***comment***
- ***Comments*** are not executable statements and they are ignored by the compiler
  - They have no effect on the behavior of the resulting program
- Comment serves as documentation for the human reader of the program

Preprocessor Directives (1/2)

- The line
  `#include <stdio.h>`
  is called the preprocessor directive
- Lines that begin with the `#` (read hash) sign are preprocessor directives
- It is mostly written at the beginning of the program
- They are not executable code line but indications for the C preprocessor

Preprocessor Directives (2/2)

- The C preprocessor is a tool which filters your source code before it is compiled
- In this case, it tells the compiler's preprocessor that the contents of the file `stdio.h` should be included at the place where `#include` appears
- The file `stdio.h` is called a header file in C and it contains the declaration needed to perform standard input output operations

The main Function (1/2)

- The next line
  `main()`
  is the first line of a function `main`
- The function `main()` is required in all C programs
- The `main` function is the starting point of a C program
- It is independent from whether it is at the beginning, at the end or by the middle of the code - its content is always the first to be executed when a program starts

The main Function (2/2)

- `main` goes followed by a pair of parenthesis `()` because it is a function
- In C, all functions are followed by a pair of parenthesis `()` that, optionally, can include arguments within
- The content of the main function follows immediately to its header enclosed between braces `{}`, as in our example
- The code inside the braces `{}` are program statements that are to be executed

Main Function Body (1/2)

- The first statement
  `printf("Hello, world\n");`
  causes the text `Hello, world`, enclosed in quotes, to be printed in the standard output device (often known as the console)
- Here, `printf` is a C function that outputs the text
  - Anything written within the quotes is printed
- The `printf` function is defined in the file `stdio.h`

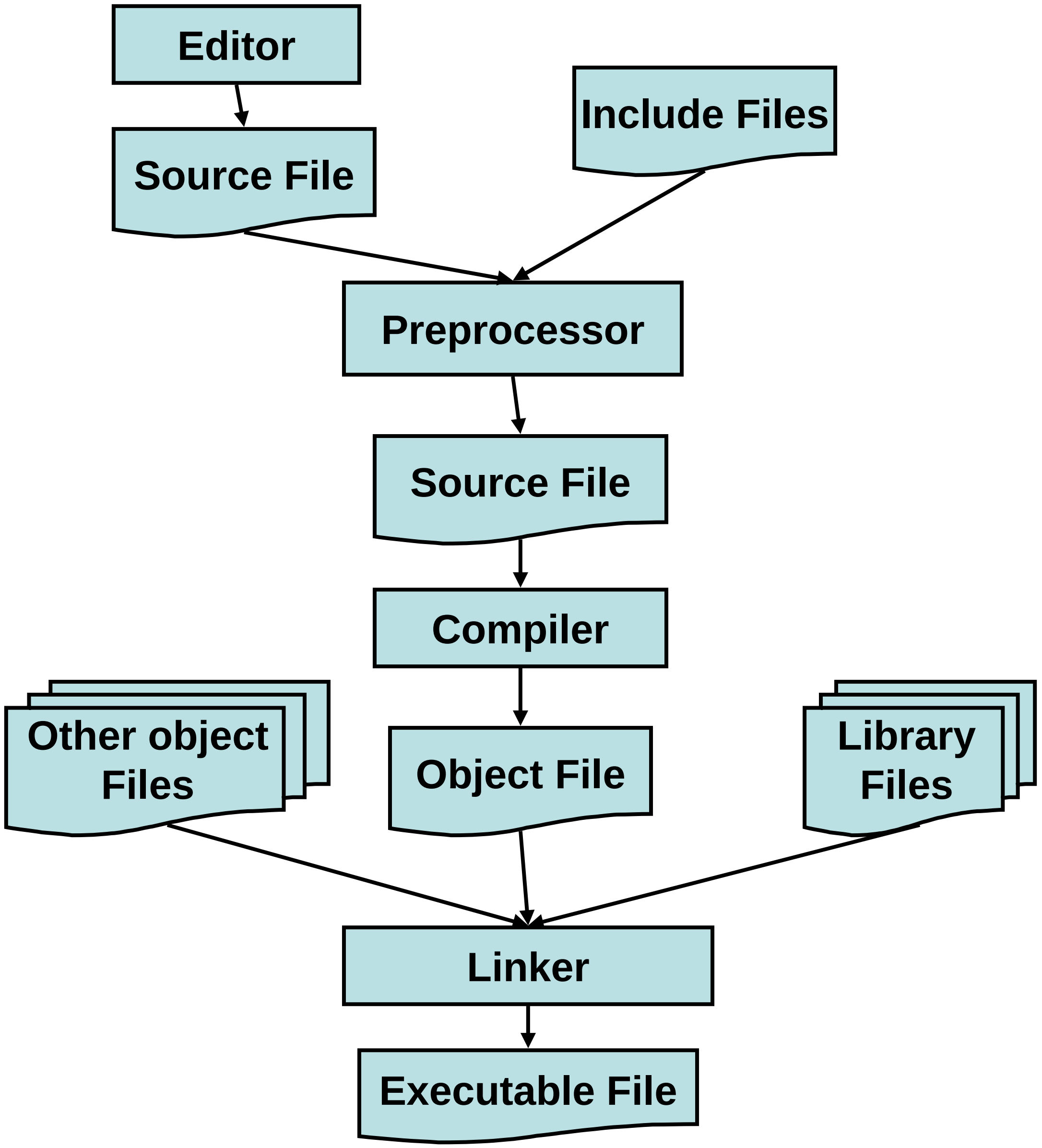
Main Function Body (2/2)

- Here `\n` is a nonprinting character and is one of the escape sequence of C
- `\n` tells to print a new line which causes the next text to be printed on next line
- Hence, the next `printf` causes the text `Welcome to ITC213` to be printed on next line
- The closing brace `}` at the last line program signifies the end of the `main()` function and hence the end of program

The Build Process

- An ***editor*** is a specialized word processor used to prepare source modules in the language of choice (e.g. C, C++, Java, Fortran)
- The ***preprocessor*** adds in standard pre-written code (boilerplate) from include files you specify to produce a complete source module
- The ***compiler*** produces object code for the target computer/ operating system
- The ***linker*** ties multiple modules together into a complete program
- An ***executable file*** is a program that will run on the computer. The editor, preprocessor, compiler and linker are all executables. So is your program

The Build Process



Compilation and Linker Errors

- A **compilation error** occurs when the compiler finds something in the source code that it can’t compile
  - A misspelling, typographical error, or any of a dozen other things can be a cause
- **Linker errors** are relatively rare and usually result from misspelling the name of a C library function
  - In this case, you get an Error: undefined symbols: error message, followed by the misspelled name (preceded by an underscore)

Another Example

```
/* Program to find the area and perimeter of a rectangle
   given its width and height */
#include <stdio.h>
main()
{
  int width, length; /* variable declaration */
  int area, perimeter;

  width = 5; /* assign the value 5 to the variable width */
  length = 7; /* assign the value 7 to the variable length */

  area = width*length; /* calculate the area */
  perimeter = 2*(width+length); /* calculate the perimeter */

  /* Print the results */
  printf("Area is %d\n", area);
  printf("Perimeter is %d\n", perimeter);
}
```