C language

Imad Kissami[1]
[1]Mohammed VI Polytechnic University, Benguerir, Morocco

Outline of this lecture

- Introduction
- Your first program
- Variables and expressions
- Repetition and Looping
- Array Subtasks
- Structures
- Strings
- Pointers / Dynamic memory allocation
- File input / output

Introduction

C Programming Language

- C was designed in 1972 by Dennis Richie, at AT&T's Bell Laboratory
- It is the culmination of two languages:
  - BPCL developed in 1967 by Martin Richards
  - B developed in 1970 at AT&T by Ken Thompson.
- Standardized by American National Standards Institute (ANSI)
- Compiled language
- Powerful
- High efficiency

Because C is a hardware-independent, widely available language, applications written in C can run with little or no modifications on a wide range of different computer systems.

Your first program

Program Basics

- The source code for a program is the set of instructions written in a high-level, human readable language.

```
1 X = 0;
2 MOVE 0 TO X.
3 X := 0
```

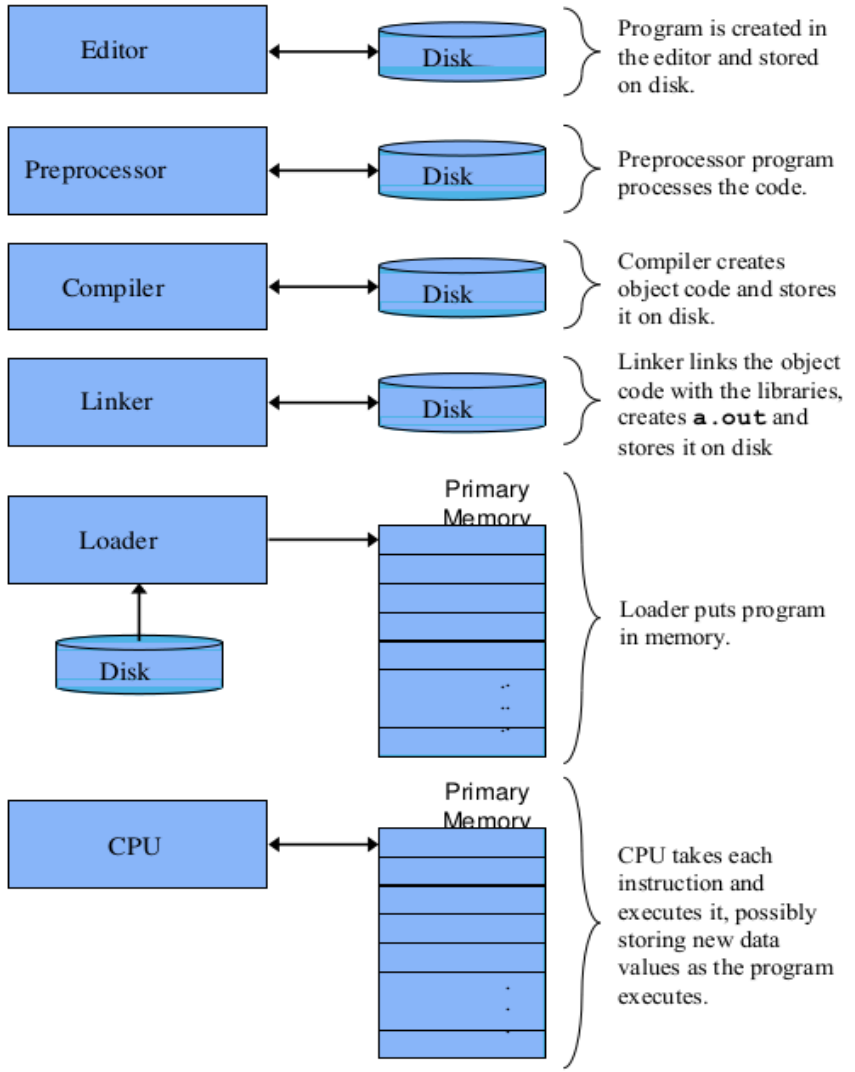
- The source code is transformed into object code by a compiler. Object code is a machine usable format.
- The computer executes a program in response to a command.

Your first program

Basics of a Typical C Environment

Phases of C Programs:

1. Edit
2. Preprocess
3. Compile
4. Link
5. Load
6. Execute



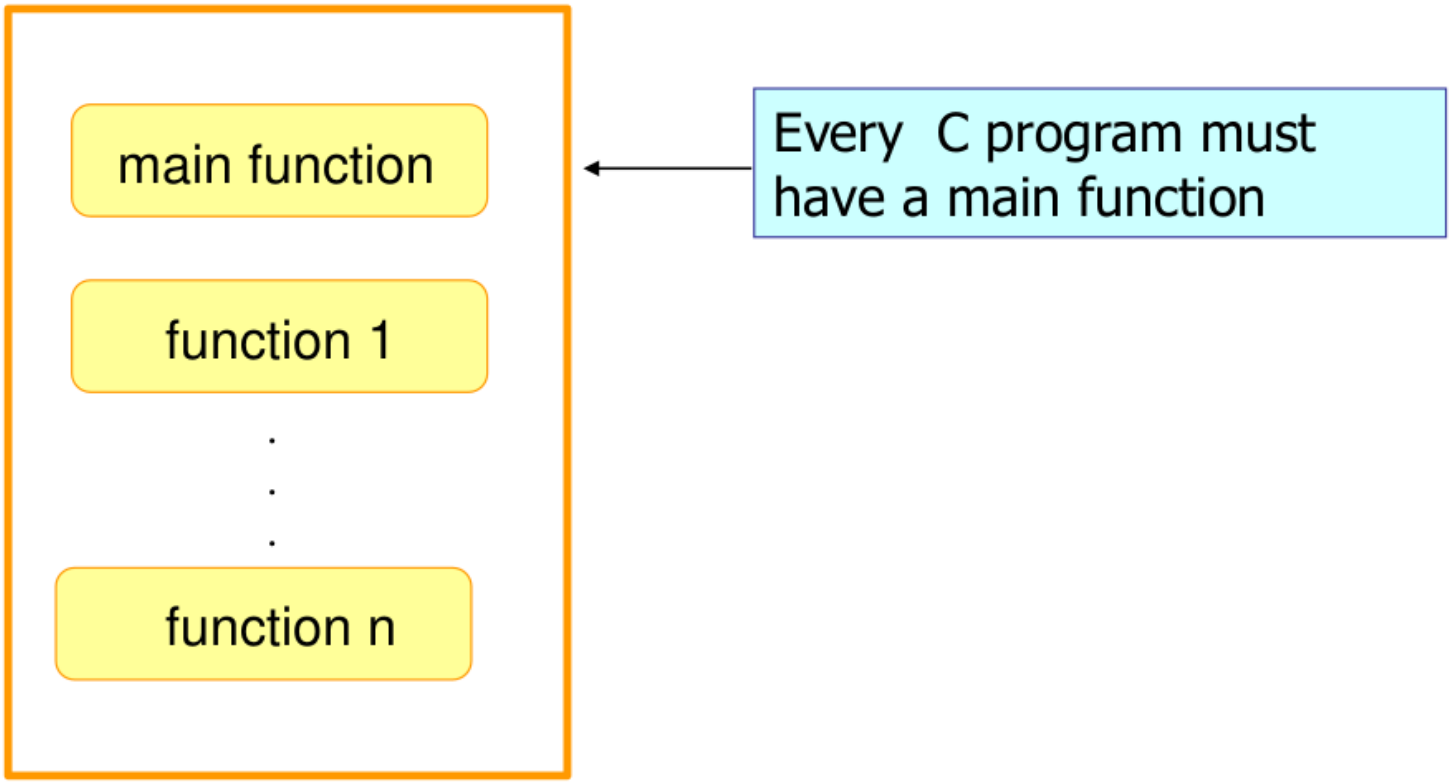
Your first program

GCC Program Basics

- The basic program writing sequence:
  1. create or modify a file of instructions using an editor (gedit, emacs, vi, ...)
  2. compile the instructions with GCC
  3. execute or run the compiled program
  4. repeat the sequence if there are mistakes

Your first program

Structure of a C Program



Your first program

Functions

- Each function consists of a header followed by a basic block.
- General format:

```
1 <return-type> fn-name (parameter-list)
2    basic block
```

Your first program

The Basic Block

```
1 {
2     declaration of variables
3     executable statements
4 }
```

- A semi-colon (;) is used to terminate a statement
- A block consists of zero or more statements
- Nesting of blocks is legal and common
  *Each interior block may include variable declarations

Your first program

Return statement

- return expression
  1. Sets the return value to the value of the expression
  2. Returns to the caller / invoker
- Example:

```
1 int main(){      // header
2     // begin of basic block
3     // ...
4     return 0;     // program ending successfully
5 }                 // end of basic block
```

Your first program

Unix Commands: mkdir & cd

- mkdir repo
  - Creates a new directory / folder
- cd repo
  - Changes the current directory
- vim toto.c
  - Edit toto.c using vim editor

Your first program

Our First Program

```c
// Program: toto.c
// Purpose: A first program in c printing Hello world
// Author:  Imad
// Date:     mm/dd/yy

#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("Hello world!\n");
    return 0; // program ending successfully
}
```

Your first program

Compiling and Running a Program

- To compile and print all warning messages, type

```
1 gcc -Wall prog-name.c
```

- If using math library (math.h), type

```
1 gcc -Wall prog-name.c -lm
```

- By default, the compiler produces the file a.out

Your first program

Compiling and Running a Program

- to execute the program

```
1 ./a.out
```

  - (The ./ indicates the current directory)
- To specify the file for the object code, for example, p1.o, type

```
1 gcc -Wall prog1.c -o p1.o
```

  - then type

```
1 ./p1.o
```

Your first program

Comments

- Make programs easy to read and modify
- Ignored by the C compiler
- Two methods:
  1. // - line comment
     - everything on the line following // is ignored

```
1 // Purpose:  Display Hello world
```

  2. /* */ - block comment
     - everything between /* */ is ignored

```
1 /*
2 Program:    toto.c
3 Purpose:     A first program in c printing Hello world
4 Author:      Imad
5 Date:          mm/dd/yy
6 */
```

Your first program

Preprocessor Directive: #include

- A C program line beginning with # that is processed by the compiler before translation begins.
- #include pulls another file into the source

```
1 #include <stdio.h>
```

  - causes the contents of the named file, stdio.h, to be inserted where the # appears. File is commonly called a header file.
  - $<>'s$ indicate that it is a compiler standard header file.

```
1 #include "myfunctions.h"
```

  - causes the contents of myfunctions.h to be inserted
  - $"'s$ indicate that it is a user file from current or specified directory

Your first program

Introduction to Input/Output

- Input data is read into variables
- Output data is written from variables.
- Initially, we will assume that the user
  - enters data via the terminal keyboard
  - views output data in a terminal window on the screen

Your first program

Program Input/Output

- The C run-time system automatically opens two files for you at the time your program starts:
  - stdin – standard input (from the keyboard)
  - stdout – standard output (to the terminal window in which the program started)
- Later, how to read and write files on disk
  - Using stdin and stdout
  - Using FILE's

Your first program

Console Input/Output

- Defined in the C library included in $< stdio.h >$
  - Must have this line near start of file:

```
1 #include <stdio.h>
```

  - Includes input functions scanf, fscanf, ...
  - Includes output functions printf, fprintf, ...
- printf
  - Print to standard output, typically the screen
  - General format (value-list may not be required):
    printf("format string", value-list);

```
1 printf("Hello world!");
```

Your first program

Console Output

- What can be output?
  - Any data can be output to display screen
    * Literal values
    * Variables
    * Constants
    * Expressions (which can include all of above)
  - Note
    * Values are passed to printf
    * Addresses are passed to scanf

Your first program

Console Output

We can

- Control vertical spacing with blank lines
  - Use the escape sequence "\n", new line
    - Should use at the end of all lines unless you are building lines with multiple printf's.
    - If you printf without a "\n" and the program crashes, you will not see the output.
- Control horizontal spacing
  - Spaces
  - Use the escape sequence "\t", tab
    - Sometimes undependable.

Your first program

Terminal Output - Examples

```
1 printf("Hello world!\n");
```

- Sends string "Hello World" to display, skipping to next line

```
1 printf("Good morning\n Ms Smith.\n");
```

- Displays the lines Good morning
  Ms Smith.

Your first program

Program Output: Escape Character \

- Indicates that a "special" character is to be output

Escape Sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
\a	Alert. Sound the system bell.
****	Backslash. Used to print a backslash character.
\"	Double quote. Used to print a double quote character.