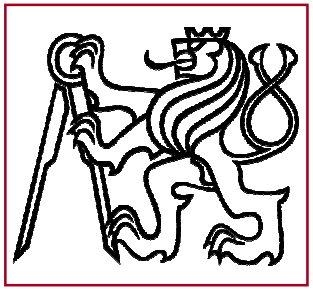
Y35PES

Programming for Embedded Systems

Ondřej Špinka, Pavel Němeček
spinkao@fel.cvut.cz nemecp1@fel.cvut.cz
http://dce.felk.cvut.cz/pes



Introduction to the C Programming Language

- 1966 – Martin Richards developed the **BCPL (Basic Combined Programming Language)** at the University of Cambridge – intended as a language for writing compilers of other languages
- 1970 – Dennis Ritchie from the Bell Telephone Laboratories, inspired by BCPL, develops the **C programming language**, probably the most widely used programming language ever.
- C is a **general-purpose**, **block structured**, **procedural**, **imperative** computer programming language, originally intended for system programming, but spread to application programming as well
- 1978 – Dennis Ritchie and Brian Kernighan publish the C programmers Bible – **The C Programming language**, known as **"K&R"**
- 1983 – Bjarne Stroustrup from the Bell Telephone Laboratories extends the C language by introducing the **object-oriented programming**, naming his clone the C++
- 1984 – C language was standardized by the ANSI C norm

Why to Learn the C?

- Old and proven language, very tightly related to Unix and Linux development
- The lowest high-level programming language (tight relations with assembler gives the programmer maximal control over the code and allow to produce the most efficient programs
- Compiler available for literally every platform
- High standardization of common libraries allows to create platform independent code with relatively little fuss (on the contrary to asm)
- Most of modern programming languages have their origins in C
- Most of modern OS were developed in C, therefore system calls and services are optimized for usage with C code
- Basic data types are defined flexibly to suit the target platform
- Very convenient and most common programming language for embedded applications

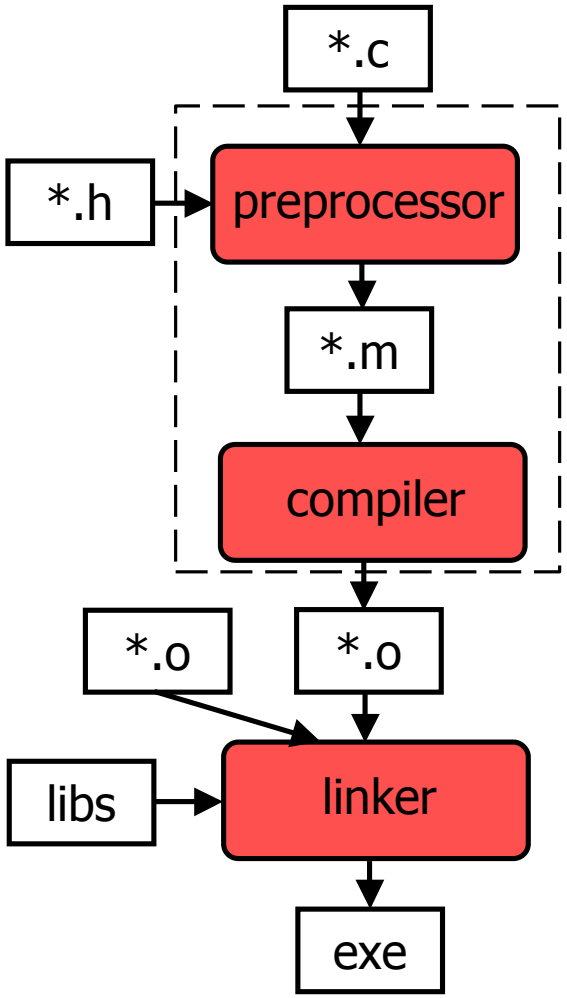
Main Differences and Similarities between the C and Java

(Assuming you know Java :-))

- Basic programming constructions are very similar (for example program flow control, such as if, then, else, while, switch…), because Java comes out of C
- ***C is not an object-oriented language***, on the contrary to Java; It is purely procedural language
- ***C does not perform any type of memory management by default***, the programmer is solely responsible for memory allocation/de-allocation and protection (however, libraries are available for that)

Phases of a C Code Compilation

- First, the source files are *preprocessed* by the **preprocessor** – i.e. the comments are erased, the constants and macros are replaced by respective values and definitions, header files are included and in-line functions are copied to the places where they are called
- Then the resulting file is *compiled* by the **compiler** – resulting into an **object file**, i.e. a binary, but with relative addressing
- Last, the **linker** *links* all the object files and libraries, creating an executable



Basic Structure of a C Program (1)

- The C is relatively brief language, with only some 30 **keywords**
- The basic programming unit in the C language is the **function**. Reusable functions are usually packed into **libraries**. Wide range of most common functions are provided within the **Standard Library (stdlib)**, which is defined by the ANSI / ISO standard. There are loads of various C libraries available.
- Each library has a so-called **header file** (*.h), providing basic information about the functions within the library (i.e. function prototypes, basic definitions and macros, external variables etc.)
- The header files must be *included* into the calling program, using the **preprocessor directive** *#include*

Basic Structure of a C Program (2)

- A **block structure** is a typical property of a C program. The functions are essentially functional pieces of programming code, denoted by the **function identifier** (name) and **parameters**, and delimitated by **braces {}**.
- Other essential building blocks of a C program are **constants**, **macros**, and **global data types** and **variables**. It is recommended to keep utilization of global data object to absolute minimum, due to program structuralization.
- Each executable C program must have an **entry point**, that is a function which is always started first. In C, this function is always called *main*.

Basic Structure of a C Program (3)

The general structure of a C program is as follows:

- Libraries inclusion (optional)
- Constants and macros definition (optional)
- Global data types and variables (optional)
- User function prototypes (or definitions) (optional)
- Application entry point (function main) (mandatory)
- User function definitions (optional)

Simple C Program Example (1)

```
#include <stdio.h>

void main ( ) {
   printf ( "Hello, world!\n" );
}
```

Simple C Program Example (2)

```
#include <stdio.h>

#define MESSAGE "Hello, world!"

char *message = MESSAGE;

void  print_string ( char *message );

void main ( ) {
    print_string ( message );
}

void print_string ( char *message ) {
    printf ( "%s\n" , message );
}
```

Simple C Program Example (3)

program.c

```
#include "mylib.h"

#define MESSAGE "Hello, world!"

void main ( ) {
    print_string ( MESSAGE );
}
```

mylib.c

```
#include "mylib.h"

void print_string ( char *message ) {
    printf ( "%s\n" , message );
}
```

mylib.h

```
#include <stdio.h>

void print_string ( char *message ) ;
```

Constants and Macros

Simple constant definition

```
#define ZERO 0
#define COND
```

Conditional constant definition

```
#ifdef COND
#define MYCONST 1
#else
#define MYCONST 0
#endif
```

Macro definition

```
#define ADD ( a,b )  (a + b)
#define PRINT_NUMBER ( num )  printf ( "number = %d\n" , num )
```

Data Types in the C Language

- bool <0 1> - not used very often, as any type of variable might be used as a logical type in C
- int (signed/unsigned, short/log) – variable platform-dependent bit length integer (ARM uses 32-bit int by default)
- char (signed/unsigned) 8-bit integer (may be used as ASCII code
- float – floating point variable (usually 32-bit)
- double – double-precision floating point variable (usually 64-bit)
- void – void variable

Data Type Modifiers

- **const** – constant, variable value cannot be changed
- **auto** – common for local variables, means that the variable is created dynamically during run-time (on the stack)
- **static** – local variable located in the RAM
- **extern** – pure declaration, defined elsewhere (often used when designing libraries)
- **register** – variable should be located in a register if possible (often used to speed-up computation, i.e. cycles)
- **volatile** – prevents optimization for asynchronously accessed variables

Variable Identifiers

- A **identifier** of a variable must be a string, containing letters, numbers and the bottom line "_" character, of a minimal length of 1 character and maximal length of 31 characters
- The first character of an identifier cannot be a number
- The string is **case-sensitive** (*sum* and *Sum* are treated as different identifiers)
- Identifiers must be composed of one word only, spaces are not allowed. If desirable, the bottom line character can be used as a word separator: *long_name_variable*
- The C language keywords cannot be used as variable identifiers (*if*, *else*, *while*…)

Lecture Summary – Essential Things to Remember

- C is a **general-purpose**, **block structured**, **procedural**, **imperative** computer programming language.
- C is **not object-oriented** and does not perform any type of **memory management** by default.
- There are three phases of C program compilation, i.e. **preprocessing**, **compiling** and **linking**.
- General C program consists of libraries inclusion, constants and macros definition, global data types and variables, user function prototypes (or definitions), application entry point (function main) and user function definitions.
- Each library must have a **header file** (.h), which includes declarations of public functions and variables, public defines and includes.
- The definition file of a library (.c) **must include its header**.