EL2310 – Scientific Programming

Lecture 6: Introduction to C

Andrzej Pronobis
(pronobis@kth.se)

Royal Institute of Technology – KTH

Overview

Lecture 6: Introduction to C

- Roots of C
- Getting started with C
- Closer look at “Hello World”
- Programming Environment

Schedule

- Last time (and before): MATLAB
- Today: Introduction to C - main part of this course
- Wed, September 19th: Deadline to submit your MATLAB project solutions
- Thu, September 20th: Project exam

Announcements

- New materials online:
  - Online courses
  - Books
  - Reference manuals
  - Forums
  - Coding convention guides
  - Linux and Emacs
- Virtual machine for C/C++ projects is online
- Homework until Wednesday:
  - Install and run the virtual machine (or use Linux...)
  - Start Emacs
  - Type, compile and run a Hello-world program
  - Check out coding conventions!

The roots of C

- First compiler developed by Dennis Ritchie at Bell Labs (1969-1973)
- Was based on two languages:
  - BCPL, written by Martin Richards at University of Cambridge
  - B, written by Ken Thompson at Bell Labs in 1970 for the first UNIX system
- Original C language was known as “K&R” C (Kernigan & Ritchie C) since the K&R book was the only language specification

ANSI C

- American National Standards Institute (ANSI) formed a committee
- Aim: to define "an unambiguous and machine-independent definition of the language C"
- Committee formed in 1983
- Work completed in 1988
- Resulted in ANSI C standard
- Extensions to the standard: C99, C11

The C language

- Developed for UNIX
- The system and most programs written in C
- “System programming language”
  - Constructs map efficiently to machine instructions
  - A replacement for the assembly language
- Many later languages borrow from C:
  - C#, D, Go, Java, JavaScript, Perl, PHP, Python, Unix C Shell
- Considered low level language (in contrast to e.g. MATLAB)

Types

Types:

- Classify type of data e.g. *integer*, *char*, *string*, etc.
- Determine the possible type values
- Machine data types: bits, words (32-bit/64-bit)
- Compiler maps language data types to machine data types

Operators:

- Interaction between objects of certain types (e.g. **+**,**-**)

Types

- Typing systems differ between programming languages
- Strongly / Weakly typed
  - Unclear definition
  - Restrictions on interaction between data types
  - MATLAB “weakly” typed
  - C/C++ “strongly” typed
- Statically / Dynamically typed
  - Type checking during compile time or run time.

Strongly, statically typed languages are more likely to catch errors at compile time while weakly typed languages allow further flexibility.

Learning C

- Practice!
- Practice!
- Practice!
- Practice!
- Practice!
- A very good idea: Define your own little project.

Steps to a running program

- **Write**
- **Compile**
- **Link**
- **Execute**

From: http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/compile.html

Compiling the code

- Parsing of the statements for syntax
- Translation of the statements into machine language
- Setting up the addresses of all variables
- Optimization of the code (if desired)

Linking

- Assembles the routines produced during the compilation
- Resolves missing calls to either language-specific libraries or system-wide functions

Optimization

- You can tell the compiler to optimize the code
- Better NOT to optimize until the program runs as expected
- Optimization changes the code internally for better efficiency
- Makes debugging much harder!
- Can typically specify different levels of optimization
- Optimization can in same cases change behavior of code

Hello world

- The Hello world program
- Typically the first program written in all languages
- First one written in B

- Input: nothing
- Output: prints “Hello world” on the screen

Hello world in C

```
#include <stdio.h>

main()
{
  printf(``Hello world\n'');
}
```

The gcc compiler

- GNU - Unix-like OS developed by the GNU Project
- GNU offers a freely available compiler
- `gcc`

Compiling a program

- `gcc hello.c`
- If the program is correct, will produce a binary file: `a.out`
- GNU/Linux naming controversy

Running the program in Linux

- `./a.out`
- The prefix `./` instructs the system to run the program `a.out` in the current directory
- Just like in MATLAB there is a `PATH` variable that tells the system where to look for programs to run
- In Unix/Linux systems this `PATH` does normally not contain the current directory.

Compiler arguments

- Compiler takes many arguments
  - `-o <output filename>`
  - `-Wall` - enable all warnings
  - `-O, -O1, -O2, -O3` - optimization level
  - `-c <filename.c>` - only compile filename.c (not link)
  - `-lname` - link to library called libname
  - `-L<directory>` - tell the linker where to find libraries
- For now let us focus on `-o`

Compiling a program cont'd

- To create executable `hello` from `hello.c`
- `gcc -o hello hello.c`

Analysis of the program

```
#include <stdio.h>

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

- A C program consists of *functions* and *variables* (like in MATLAB)
- Functions are built using statements (like in MATLAB)
- Program execution starts in the function `main`
- Each program must have a `main` function

Analysis of the program

- Program starts with `#include <stdio.h>`
- Instructs the compiler to include information from the standard library for input and output (I/O)
- These lines are typically found at the top fo the file

- The `main` function can, but does not have to have arguments
- The statements within a function should be placed between braces

The `printf` function

- `printf` is a command used to print to standard output
- The argument is a string (enclosed in double quotes)
- Will see later that it can take more arguments
- The last character in the string is `\n` which is C style for the newline character
- Other “hidden” characters can be obtained with an *escape sequence* (\)
- `\t` is a tab character

Virtual Machine

- Can be downloaded from the course materials page
- Ubuntu Linux guest running inside VirtualBox
- Preinstalled: gcc/g++/SDL/emacs
- VirtualBox can be installed in any host OS
- Go to: `www.virtualbox.org`, download and install
- Unpack the VM and use Machine-Add, then Start
- Use Shared Folders to exchange files with your host OS

Editing files

- We will use simple text editors, not full IDEs
- Emacs preferred, but you can use any text editor (e.g. if you prefer to edit text in Windows)
- Avoid rich text editors (e.g. Word) and save the file as text only
- Emacs pre-installed inside the VM and can be installed natively in Windows
- A short introduction to Emacs available from the course materials
- Use the interactive Emacs tutorial inside Emacs

Compiling in Linux

- Open the terminal
- Go to the folder containing source files (`cd <path>`)
- Run the compiler (`gcc -o hello hello.c`)
- Linux beginner tutorials available in the course materials

Homework

- Homework until Wednesday:
  - Install and run the virtual machine, or use:
    - Native Linux on your laptop
    - CSC computers
  - Start Emacs
  - Type, compile and run a Hello-world program
  - Check out coding conventions!
- Wednesday: Continue with C