C Programming under Linux

P2T Course, Martinmas 2003–4
C Lecture 1

Dr Ralf Kaiser

Room 514, Department of Physics and Astronomy

University of Glasgow

r.kaiser@physics.gla.ac.uk

Summary

- The History of C
- Basics of C Program Writing
- Creating an Executable Program
- Coding Style
- References

`http://www.physics.gla.ac.uk/~kaiser/`

Programming Languages

Computers operate on *binary numbers*, both the data and the program are that is executed are a long series of `1` and `0`. The program in this case is called machine code

```
1001 1100
0011 1011
1100 1111 ... and so on ...
```

Unlike computers, people don't think in numbers. Therefore *assembly language* was developed, and at first used by programmers to write programs that were then translated by hand:

```
MOV A,47  1010 1111
ADD A,B   0011 0111
HALT      0111 0110
```

As the cost of computers went down and that of programmers went up, programs (*assemblers*) took over the translation.

Programming Languages

- The next step were *high level* languages, that are even easier to understand for humans, leaving more translation work for the computer. For this translation the computer uses a program called a *compiler*.
- Examples of high level languages are
  - FORTRAN FORMula TRANslation
  - Pascal
  - COBOL (still used in some banks)
  - Java
- C is also a high level programming language. Like most programming languages it is based on English (thankfully).

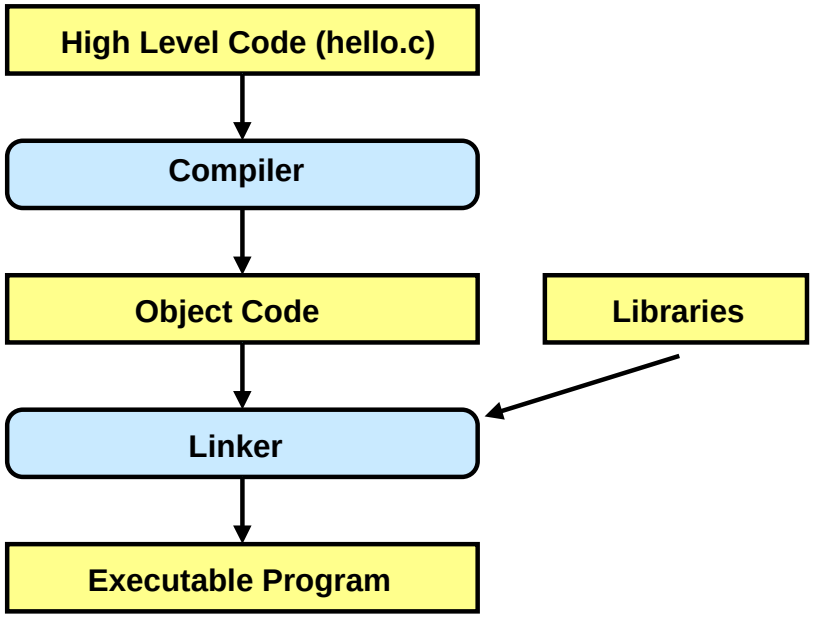
A Short History of C

- BCPL (Basic Computer Programming Language), Martin Richards, 1967
- B, Bell Labs, Ken Thompson, 1970
- C, Bell Labs, Dennis Ritchie, 1970+
- *The C Programming Language*, B.Kernighan/D.Ritchie, 1978
- C++, Bjarne Stroustrup, 1980
- ANSI C, American National Standards Institute, 1989
- ISO/IEC 9899 C, International Organisation for Standardization, 1999, the current Standard C
- C#, Anders Hejlsberg, Microsoft, 2000

Which C are we using ?

- There is a large number of different C compilers for different operating systems. Many of them today are C++ compilers that also compile C code: Borland C++, Microsoft Visual C++, Turbo C++.
- We will use gcc, the Gnu C Compiler, open source software from the Free Software Foundation. gcc supports Standard C, but contains some extended features (that can be disabled). Details at `http://gcc.gnu.org/`.
- C code is portable, i.e. the source code (as long as it complies with Standard C) can be compiled by any compiler on any platform. The compilers are platform-dependent.
- Unix, including Linux, has a special connection with C, because Unix was written in C.

From High-Level Code to Executable Program



- source code written by programmer in high-level language, in our case in C
- compiler translates it into an object file
- linker combines object code with predefined routines from libraries and produces the executable program
- This is usually not done 'by hand', but using a wrapper program that combines the functions of compiler, assembler and linker.

Basic Building Blocks of C

- **Data**
  are stored in variables that can be of different types. The type of a variable defines the set of values it can have. Variables have to be declared in a declaration statement before they can be used:

```
type name;
int variable1;
int variable2;
int sum;        /* sum of variable1 and variable2 */
```

- **Instructions**
  tell the computer what to do with the data. Operators act on one or connect two variables to form an expression that can be part of an assignment statement. In addition there area also control statements that regulate the flow of the program. All C statements end with a semi-colon, not with the end of the line.

```
sum = variable1 + variable2
```

Basic Building Blocks of C

- **Functions**
  are the building blocks of a C program. They have a type (like variables), arguments or parameters listed in round brackets and a function body enclosed in curly braces:

```
type function(parameter1,parameter2){
    first statement;
     ...
    last statement;
}
```

- **Files**
  The C code is contained in source files that conventionally have the ending `.c`, e.g. `test.c`. One file can contain more than one function and several files can be compiled together into a single executable. This helps with being organised.

- We will later look at variable types, operators and functions in a lot of detail with many examples.

Hello World

Let's now write our very first program. Using a text editor of your choice (e.g. `emacs` or `xemacs`), write a file `hello.c`:

```
#include <stdio.h>
int  main()
{
      printf ("Hello World\n");
     return  (0);
}
```

- `int main(){}`
  is the `main` function, the function that is always executed first, it returns a value of type `int` and has no arguments
- `#include <stdio.h>`
  includes a header file that contains the definitions of standard input/output functions, e.g. `printf()`.
- `printf("Hello World\n");`
  prints the words 'Hello World' to the screen and then goes to the next line
- `return (0);`
  sets the return value of the function `main` to zero. Note the semi-colon at the end of each statement.

Compiling a Program

- There are two types of compilers that are very different in look-and-feel: command-line compilers and Integrated Development Environments (IDEs).
- Under Linux command-line compilers are more common; IDEs exist (e.g. KDevelop), but they typically are front-ends for command-line compilers and assume some knowledge of the command-line compiler and Linux. Almost every Windows compiler contains an IDE.
- We will use the command-line compiler `gcc`. To compile our example `hello.c` we type

```
gcc -o hello hello.c
```

- This tells the `gcc` compiler to read, compile and link the the source file `hello.c` and write the executable code into the output file `hello`.

gcc command-line options

- Like many Linux programs also `gcc` has a (large) number of command-line options, or flags, that turn features off/on. To compile `hello.c` we might actually use something like

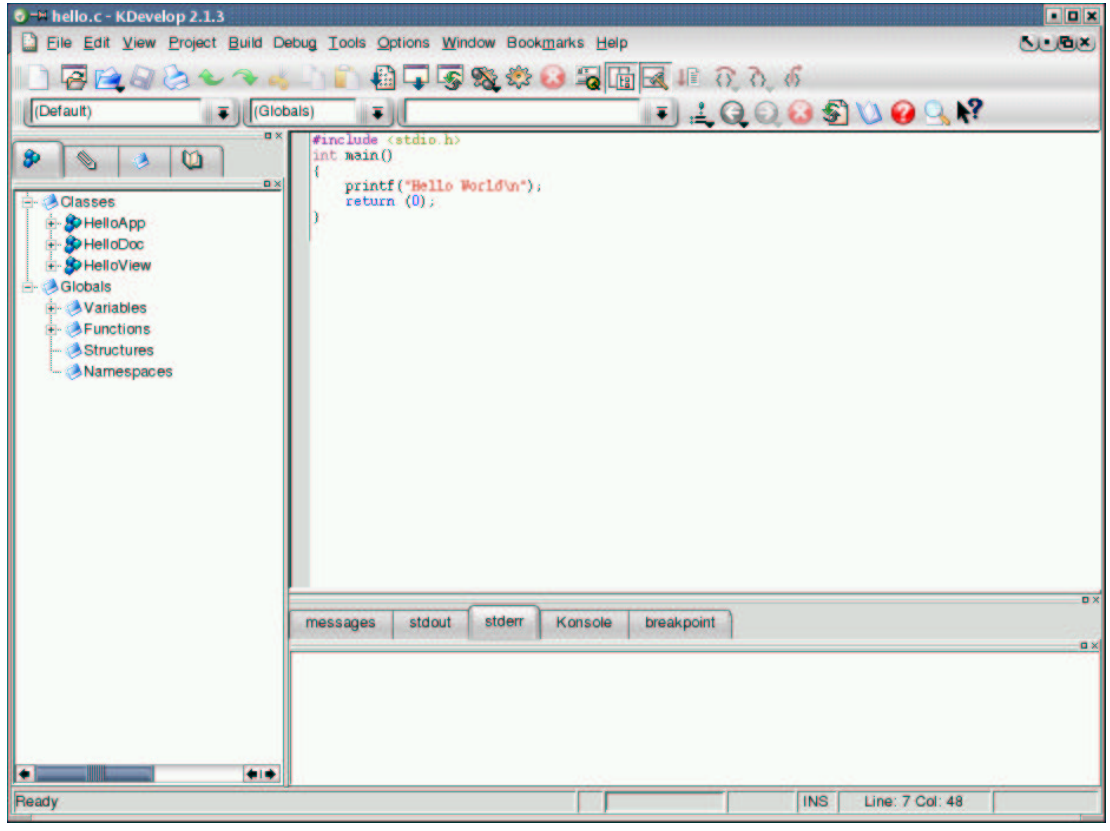
```
gcc -g -Wall -ansi -pedantic -o hello hello.c
```

`-g`	enables debugging
`-Wall`	turns on warnings
`-ansi`	turns off `gcc` features that are incompatible with ANSI C
`-pedantic`	issue warnings for any non-ANSI feature

- We will get back to `gcc` flags later. We also will see that the `make` utility saves us from having to remember all the flags we need.

KDevelop IDE

One example for a C programming IDE under Linux is KDevelop, which is part of the KDE desktop. Unfortunately KDevelop assumes some familiarity with Linux, `make`, C (and C++), so that it is perhaps a useful tool after completion of this course ...



Coding Style - Comments

- A lot more time is spent on upgrading, maintaining, debugging and adapting code than on actually writing new code ‘from scratch’. For example, it’s three years ago that I wrote the last completely new C program - a simple detector simulation based on the Bethe-Bloch equation.
- For this reason code must be commented and the absence of comments will be treated as a programming error in the marking of assignments.
- Comments in C start with `/*` and end with `*/`, for example

```
/* Say hello to the world */
printf("Hello World\n");
```

- It is customary that longer comments are put into boxes, e.g.

```
/*=======================================*\
 * this could be a section header        *
\*=======================================*/
```

Coding Style - Comments

Besides the comments in the code, that explain the meaning of variables, functions and sections of the code, there should also be a comment box at the begin of each file that contains all relevant information:

- name of the program and what it does
- author and how to reach him/her
- usage: how do you call it, what are the options
- revision history: who edited the file when and why
- file formats, input/output files
- references, i.e. from whom did you copy what
- restrictions: what the program doesn't do
- known bugs and anything else that's relevant

Coding Style - Comments

This is what our `hello.c` looks like when it's properly commented:

```
/*************************************************************
 * hello -- program to print out "Hello World".              *
 *                                                           *
 * Ralf Kaiser, September 2003                               *
 *                                                           *
 * Reference:  Steve Oualline, Practical C Programming,      *
 *             O'Reilly                                      *
 *                                                           *
 * Purpose:  Demonstration of comments                       *
 *                                                           *
 *************************************************************/

#include <stdio.h>

int main()
{
    /* Say Hello to the World */
    printf("Hello World\n");
    return (0);
}
```

Coding Style - Comments

Since C99 also C++ style comments are possible in C. In this case the comment starts with // and extends to the next line break.

```
#include <stdio.h>

// C++ Style Comment
// program that says hello to the world

int main()
{
    printf("Hello World\n");
    return (0);
}
```

This is typically already no problem if the compiler is a C++ compiler that also compiles C (C being a subset of C++). However, it is not recommended for use in C programs in this course, because `gcc -ansi -pedantic` will give an error message and not compile.

Coding Style - Indentation

To make programs easier to read, most programmers indent their programs according to the level of the statement. This is not necessary for the program to compile, but it makes the code easier to understand. Two styles of indentation are frequently used:

```
int main(){
    if (morning) {
        printf("Hello World\n");
    } else {
        printf("Good Night\n");
    }
    return (0);
}
```

```
int main()
{
    if (morning)
    {
        printf("Hello World\n");
    }
    else
    {
        printf("Good Night\n");
    }
    return (0);
}
```

The style of curly braces is up to you. Editors like `emacs` and `xemacs` will help you with the indentation.

References

- By now you know that all the information you need is available online, either directly on your PC or on the web. There are the `man` and `info` pages for each Linux command, including `gcc`
- Manuals in postscript or .pdf format are available on the web for `emacs` and other programs.
- There are websites with 'Frequently Answered Question' (FAQ) sections and Usenet newsgroups that you can turn to with questions.

References

Books that have been used in the preparation of the C-programming part of this course:

- Practical C Programming, S.Oualline, O'Reilly
  (O'Reilly specialises in Unix related programming handbooks for professionals; took examples from this book, can be found on the web)
- The C Programming Language, B.Kernighan, D.Ritchie
  (the original C bible)
- Programming with C, U.Glasgow
  (C handouts from previous P2T courses)
- ISO 9899, the actual standard definition