
Computer Programming: Skills & Concepts (INF-1-CP1) The C Programming Language

29th September, 2009



Administrative business

- If you don't have your password yet, you can collect it from the ITO, AT level 4, during office hours.
- Practical 0. Remember to submit the required files by 2pm, Monday 5th October.

Lecture Theatres for the rest of term

Owing to unexpectedly high registrations, we've been moved.

Unfortunately, we've been moved to a different place on each day . . .

- **Monday** AT LT3 (except 09/11/09 (wk 8), venue then t.b.a.)
- **Tuesday** Minto House LT1 (on Chambers St)
- **Thursday** DHT LTA

Tutorials

- Start in week 3.
- Tutorial groups are mostly arranged, and can be viewed from the course webpage.
- Contact the ITO if your tutorial group clashes with another lecture, or if you have not been assigned to any group (and are officially registered for CP1).

Our Lab on Monday 28th September

- Please remember to finish and submit Practical 0 (before Monday 5th), if you did not complete the work at the Lab.
- Please do not switch off machines in the AT Labs after finishing work - just log off.
- *Printing*: The [File → Print Buffer] option in emacs will not work until you set up a default printer on your DICE account.
To find out how to set a default printer (probably at3 or at14), check out the "Printing" section of the online User Support FAQ:
<http://www.inf.ed.ac.uk/systems/support/FAQ/>

The C Programming Language

- Developed by Dennis Ritchie in 1972 at Bell Labs, in conjunction with the UNIX operating system.
- The American National Standards Institute (ANSI) formed a committee to develop a standardised version of C. The main standard was published in 1989 and is known as ANSI-C.
- An *imperative* programming language - programming task is achieved by a list of *commands* acting on a set of program *variables*.

Getting a working C program

- *Write the code.*
- Use gcc to translate your C program into something the computer will understand.
- *Run* the program, once we have a version which has successfully compiled.

edit → *compile* → *run*.

What to do when it doesn't work

"Right first time" is not a reasonable strategy for programming

- Some 'debugging' usually necessary.
- You can learn a lot from trial-and-error.
- Spending time working on the logical structure of your code, and the typographical details, will minimize debugging time.
- (for assignments) You are only assessed on the final version that you submit.

A simple C program

```
/* Simple hello program */  
  
#include <stdio.h>  
#include <stdlib.h>  
  
int main(void) {  
    printf("\n");  
    printf("Hello world!");  
    printf("\n");  
    return EXIT_SUCCESS;  
}
```

The Edit-Compile-Run cycle

- **Edit:**
 - emacs hello.c
- **Compile:**
 - gcc -Wall hello.c
 - (gcc stands for **G**nu **C** Compiler)
- **Run:**
 - ./a.out (./a.exe, if you are using cygwin)

The Edit-Compile-Run cycle

- **Edit:**
 - Where do I write this C stuff?
- **Compile:**
 - How do I get my C program translated into something the computer will understand?
- **Run:**
 - How do I start my program?
 - Where do the results get output?

The structure of “Hello World”

Header Files

```

/* Simple hello program */

/* ----- */
#include <stdio.h>
#include <stdlib.h>
/* ----- */

int main(void) {
    printf("\n");
    printf("Hello world!");
    printf("\n");
    return EXIT_SUCCESS;
}

```

- Includes *headers* verbatim into the program text.
- *<filename>* files are in the system directories (often /usr/include).
- "filename" files are in the current directory.

Comments

```

/* ----- */
/* Simple hello program */
/* ----- */

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

int main(void) {
    printf("\n");
    printf("Hello world!");
    printf("\n");
    return EXIT_SUCCESS;
}

```

- Everything in-between the /* and /* is ignored.
- You should always comment (well) every program that you write.
- Include the author name, and the date.

main

```

/* Simple hello program */

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

/* ----- */
int main(void) {
    printf("\n");
    printf("Hello world!");
    printf("\n");
    return EXIT_SUCCESS;
}
/* ----- */

```

- A *function* called main.
- Contrast with "+".
- (void) In this case main takes no arguments.
- int In this case, main returns an integer.
- main is always the first function to execute.

Functions

- Integer
 - A positive or negative whole number or zero.
- Functions: Consider '+'
 - 1+2 - evaluates to the value 3
 - plus(1,2) - returns the value 3
 - plus(A,B) - returns the value C

printf

```
/* Simple hello program */
```

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

```
int main(void) {
/* ----- */
    printf("\n");
    printf("Hello world!");
    printf("\n");
/* ----- */
    return EXIT_SUCCESS;
}
```

- printf is a *library* function.
- It has a manual page: man 3 printf.
- Contrast to man printf (remember the 3...)
- \n = new line.

Programming Errors

- Most programs fail to work correctly the first time.
- Tracking down the errors requires *time + patience + attention to detail*.
- Skill in debugging is gained from experience (and *attention to detail*).

return

```
/* Simple hello program */
```

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

```
int main(void) {
    printf("\n");
    printf("Hello world!");
    printf("\n");
/* ----- */
    return EXIT_SUCCESS;
/* ----- */
}
```

- Remember that main returns an integer.
- EXIT_SUCCESS is the integer that it returns.
- stdlib.h defines EXIT_SUCCESS as 0.
- Numbers are often used in programming to represent a 'status'.

Example

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

```
.....
```

```
[fletcher]mccryan: gcc -Wall hello.c
hello.c:3:19: warning: extra tokens at end of #include directive
```

Common errors

- Mis-spelling
- Missing Punctuation
- Additional symbols
- Wrong punctuation
- Missing `#include`
- No `main` function
- `return` statement forgotten in a function
- `Printf` → `Pritnf`
- `("\\n")` → `('\\n')`
- `#include <stdio.h>;`
- `("\\n")` → `("\\n)`

Manifestations of an error

- Compiler *error* messages:
 - Fatal mistake - cannot continue.
- Compiler *warning* messages:
 - A mistake was found, the compiler 'guessed' what you meant, and continued.
 - Your program may still manage to work!
 - To show all the warnings - "`gcc -Wall`".
- Error while running the program:
 - "Segmentation fault".
 - The wrong result.