Computer Programming: Skills & Concepts (CP)
Imperative Programming and C

J. Bradfield

19 September 2017

Yesterday's Lecture

- Structure of the CP course
- What is programming?
    - Imperative Programming?
    - C programming?
- Hello World
- Any questions?

Today's lecture

- Examining the finicky details of `hello.c`
- *Imperative programming*.
- Assigning values to variables.

The structure of ‘Hello World’

```
#include <stdlib.h>      /* Include Standard Library */
#include <stdio.h>       /* Include Input/Output Library */

int main(void) {            /* Exactly one ‘main’ function */
  printf("Hello, World.\n"); /* Output function printf */
  return EXIT_SUCCESS;      /* ‘All ok’ signal returned */
}
```

- Includes *headers* verbatim into the program text.
- <*filename*> files are in the system directories (often `/usr/include`).
- "*filename*" files are in the current directory.
- `<stdio.h>` contains the `printf` function.
- `<stdlib.h>` contains the definition of `EXIT_SUCCESS` (it is 0).

Comments

```
#include <stdlib.h>       /* Include Standard Library */
#include <stdio.h>        /* Include Input/Output Library */

int main(void) {            /* Exactly one ‘main’ function */
  printf("Hello, World.\n"); /* Output function printf */
  return EXIT_SUCCESS;       /* ‘All ok’ signal returned */
}
```

- 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

```
#include <stdlib.h>       /* Include Standard Library */
#include <stdio.h>        /* Include Input/Output Library */

int main(void) {            /* Exactly one 'main' function */
  printf("Hello, World.\n"); /* Output function printf */
  return EXIT_SUCCESS;      /* 'All ok' signal returned */
}
```

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

printf

```
#include <stdlib.h>      /* Include Standard Library */
#include <stdio.h>       /* Include Input/Output Library */

int main(void) {          /* Exactly one ‘main’ function */
  printf("Hello, World.\n"); /* Output function printf */
  return EXIT_SUCCESS;      /* ‘All ok’ signal returned */
}
```

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

Template for almost every program ...

```
/* 'Template' for most programs you'll write */

#include <stdlib.h>      /* Include Standard Library */
#include <stdio.h>       /* Include Input/Output Library */

int main(void) {             /* Exactly one 'main' function */


      THIS IS WHERE YOUR SPECIFIC CODE WILL GO


  return EXIT_SUCCESS;       /* 'All ok' signal returned */
}
```

Of course, there will be lots of extra *things (the particular code to execute; some variable declarations; maybe some extra functions coded up).*

What is Imperative Programming?

- The original style of programming-languages, and closest to what actually is done by computers at the “machine-level”.
- In Imperative programming, the expressions of the language are instructions or commands to the machine to perform some action:
  - Often the actions involve a change of state in the environment of the program (more about this in week 2);
  - Or other times we use input or output actions.
- Writing an imperative program is like creating a ‘recipe’:
  - The instructions will be carried out one at a time;
  - The order of the instructions is important.

‘Change of State’

- We can add, multiply, ... numbers of the `int` type using operations in the kernel of our language.
  - `2+3`
  - We could also output values involving these operations
    `printf("8*8 is %d", 8*8);`
- We will also want to work with numbers whose value we *don’t know* in advance.
  - Use VARIABLES to store numbers.
  - Must declare the variables *in advance*.
  - The *current values stored in the program variables* at any time is the (current) STATE.
  - The state will change as the program commands are executed.

Discussion

- What are the commands of the ‘Hello, World’ program?
- The variables of the ‘Hello, World’ program?
- What does its program environment look like?

Squaring a number

Want to read in a number from the user (so value is not known when writing our program), square the number (multiply by itself), and print this out.

Need a variable for the number which is unknown.

- Must DECLARE the variable first.
  - `int x;`
  - Above is the declaration
  - The name of the variable is `x`
- Will want to ASSIGN a value to the variable.
  - `x = 100;`
  - Above is *one* way to give a variable a value.
  - We will want to *read the value for* `x` *from the user* - we will show how in week 2 (full program for squaring).

Important: In C, the '=' means '*becomes* equal to', **not** 'is (already) equal to'. Regrettable, but we're stuck with it.

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*).

Example of an error:

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

```
zagreb: gcc -Wall hello.c
hello.c:3:19: warning: extra tokens at end of #include direct
```

Common errors

- `\*` instead of `/*`
- Mis-spelling
- Missing punctuation
- Additional symbols
- Wrong punctuation
- Missing `#include`
- No `main` function
- `return` statement forgotten from a function
- `printf` written as `pritnf`
- `("\n")` written as `('\n')`
- `#include <stdio.h>;`

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! MAYBE ...
  - To show all the warnings: `gcc -Wall`
- Error while running the program:
  - 'Segmentation fault'.
  - The wrong result.

Labs

First Lab is 10:00 Thursday of week 1 for students in the Thursday Lab; other students will have Lab on Monday at 15:10 or Tuesday at 12:10.

- The handout for Lab 1 will be available on the webpage soon (will be announced); and handed out in hardcopy at the lab.
- Details of how to set your DICE password, have been or will be emailed to you at your University email account. Please check your email for this information before the first lab (if it's not in your email folder, come to your lab anyway).

Wrapping Up

- Please email your Personal Tutor immediately and ask them to enrol you, if have not been registered yet.
  - You will not be allocated an Informatics (DICE) account unless you are formally registered.

Assigned Reading

Kelley and Pohl sections 1.1, 1.2, 1.3, 1.5, 1.6.