COMP10002 Foundations of Algorithms

Semester Two, 2017

Getting Started With C

Overview

Beginnings

Developed in 1969–1973 by Dennis Ritchie, at Bell Labs in the US. By 1973, expressive enough to support Unix O/S implementation (developed originally in assembler).

Critical factors in C's success:

- High-level control, but direct mapping to hardware, and relatively unrestricted access to memory
- Relatively light execution footprint
- Compilers robust, efficient, and portable
- Useful for both applications and systems programming
- Standards activity (K&R, 1978; ANSI, 1989; ISO, 1990)
- Right place, right time.

Development

Further standards activity took place in 1995, 1999, and 2011. C is now one of the most widely used programming languages.

Very few processors – right through to both ends of the spectrum – have not had a C compiler port undertaken.

Immense community effort invested in the gcc compiler, which has served as an exemplar of public software.

Development

C has served as basis for many subsequent developments in languages, including C++ and Java.

Compared to a wide range of modern languages, including Python, has benefit of being compiled rather than interpreted.

That makes it suitable for high-load applications: operating systems, network management, time- and scale-critical applications such as web search.

Drawbacks

Run time diagnostics and error checking (lack of) can trap careless programmers.

Low-level operation means that libraries are required for operations and types that may be native in other languages.

Powerful, but like a chain-saw; that brings a need to exercise good habits and follow safe work practices.

A few "what were they thinking?" issues that can trap the unwary, and/or cause confusion.

Hello world

Always start with something laughably simple: `helloworld.c`

On any new computer system, and in any new language, get the equivalent of this program compiled and running before you do anything else!

Simple compilation

In the first instance we will be compiling from a shell.

```
mac: gcc -Wall -o helloworld helloworld.c
mac: ./helloworld
Hello, boss!
mac:
```

(On a PC, use `helloworld.exe`)

The second most important thing you will do in your first workshop is to get this program compiled and running (and then extend it).

The first most important thing is to meet your tutor and make some new friends.

Once the first program is compiled...

Why rewrite when can reuse instead?

A standard boilerplate for C programs, `frame.c`:

```
/* Comment */

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

int
main(int argc, char *argv[]) {
    int n;
    double x;
    scanf("%d %lf", &n, &x);
    printf("n=%d, x=%f\n", n, x);
    return 0;
}
```

Then expand your repertoire

The standard set of control structures are all there:

- Chapter 2: arithmetic operators, precedence, `int` versus `double` (versus `float` versus . . .), `scanf` and `printf`
- Chapter 3: logical operators, `if` (and `switch`) statements for selection
- Chapter 4: `for` and `while` statements for iteration
- Chapter 5: functions, to provide abstraction.

And incorporate them into programs of increasing power.

Chapter 2 – Concepts

Variables and constants, type hierarchy.

Arithmetic operators and expression types.

Precedence.

Input using `scanf` and format descriptors.

Output using `printf` and format descriptors.

Chapter 2 – Program examples

- overflow.c
- rounding.c
- addorder.c
- format.c
- mixedvals.c

Chapter 2 – Exercise 1

Case Study:

Write a program that reads in the radius of a sphere, in meters, and calculates and outputs the volume of that sphere, in cubic meters. The volume of a sphere of radius r is given by $\frac{4}{3}\pi r^3$.

- `spherevol.c`

Chapter 2 – Exercise 2

Exercise:

Write a program that reads in a weight in pounds, and outputs the corresponding weight in kilograms. One kilogram is 2.2046 pounds.

Chapter 2 – Summary

Key messages:

- Use `#define` for *all* constants.
- `int` arithmetic will overflow silently
- `int` *op* `int` yields `int`, even if *op* is division.
- With `float` and `double`, operation order and rounding can affect the result
- Each type has corresponding *input* and *output* format descriptors, that may differ (doh!)

Chapter 3 – Concepts

Relational and logical expressions are of type `int`.

Non-zero is interpreted as being “true”.

Zero is interpreted as being “false”.

There is no Boolean type, and no Boolean constants.

There is no `elif`, you need to make use of `else if`.

Chapter 3 – Examples

Example 1:

```
if (n < 0)
    num_neg += 1;
```

Example 2:

```
if (scanf("%d%d%d", &n, &m, &r) != 3) {
    printf("scanf failed to read three items\n");
    exit(EXIT_FAILURE);
}
```

Chapter 3 – Examples

Example 3:

```
length_of_year = 365;
length_of_feb = 28;
if (year%4==0 && (year%100!=0 || year%400==0)) {
    /* need to allow for leap years */
    length_of_year += 1;
    length_of_feb += 1;
}
```

Chapter 3 – Examples

Example 4:

```
if (month==2) {
    length_of_month = length_of_feb;
} else if (month==4 || month==6 ||
        month==9 || month==11) {
    /* thirty days hath september, april, june,
        and november */
    length_of_month = 30;
} else {
    /* all the rest have 31, except february... */
    length_of_month = 31;
}
```

Chapter 3 – Program examples

- `equalinif.c`
- `danglingelse.c`
- `threetest.c`
- `switch1.c`
- `switch2.c`

Chapter 3 – Precedence

Operators	Operation class	Precedence
`++, --`	postinc, postdec	Highest
`!, -, (type)`	not, negation, casting	
`*, /, %`	multiplication	
`+, -`	addition	
`<, >, <=, >=`	comparison	
`==, !=`	equality	
`&&`	and	
`||`	or	
`=, +=, *=, etc`	assignment	Lowest

Chapter 3 – Evaluation order

Expressions involving `&&` and `||` have their logical elements evaluated from left to right. Execution ends when the outcome is known.

You may not assume that other expressions are evaluated strictly left to right. In `a*b + c*d`, the element `c*d` might be evalued before `a*b`; similarly, the element `b` might be evaluated before the element `c*d`.

Precedence says what happens last, not what happens first.

Beware of side effects.

Chapter 3 – Exercise

Exercise 3.4

Write a program that reads a date in `dd/mm/yyyy` format and prints, in the same format, the date that it will be tomorrow.

Chapter 3 – Summary

Key messages:

- Type cast rules also apply to relational operators
- Relational and logical operations are of type `int`, and slot into the precedence table below the arithmetic operators and above the assignment operators
- Use `==` for equality testing, beware of `=`
- Beware of side effects in expressions, only use the simple `i++` and `i--` ones at first
- `switch` should be avoided
- The `main` function should return a status value.