Introduction to the C/C++ module

Hayo Thielecke
University of Birmingham
`http://www.cs.bham.ac.uk/~hxt`

January 11, 2016

Progression: position of this module in the curriculum

First year	Software Workshop, functional programming, Language and Logic
Second year	C/C++
Final year	Operating systems, compilers, parallel programming

Why a module on C/C++

- Fun fact 1: This module exists because students suggested having a module on C
- Fun fact 2: C/C++ module title is the shortest and most techie
  unless the mathematicians have something called $\int$
- Fun fact 3: `C/C++` is syntactically valid in both C and C++, but very bad code. Why?

- C and C++ are widely used in industry
- C is a prerequisite for operating systems and others
- Knowing C is more than just knowing another language
- In C you need to understand how things work
- Not something you could easily teach yourself
- That is what universities are for

What this module is not

- In many universites, C/C++ are taught outside CS departments
- E.g., Elec Eng, Mathematics, Finance
- C taught to non-CS is necessarily superficial
  e.g. do some numner crunching, no pointers, not much recursion
- This module is totally different from non-CS modules on C
- Here we want C as part of CS; see progression
- This is not "C for beginners/dummies/whatever"
- This is not a re-run of Software Workshop
- This is not a manual

Outline of the module (provisional)

I am aiming for these blocks of material:

1. pointers+struct+malloc+free
   ⇒ dynamic data structures in C as used in OS
2. pointers+struct+union+tree
   ⇒ trees in C
   such as parse trees
3. object-oriented trees in C++
   composite and visitor patterns
4. templates in C++
   parametric polymorphism

An assessed exercise for each.

Teaching and assessment

- We have very few PhD students available as demonstrators
- Even fewer who know C
- The demonstrators we have are concentrated on first year workshop
- There will be much less hand-holding compared to Software Workshop
- You need to be more independent in this module

Exercises	20%
Exam	80%

Books

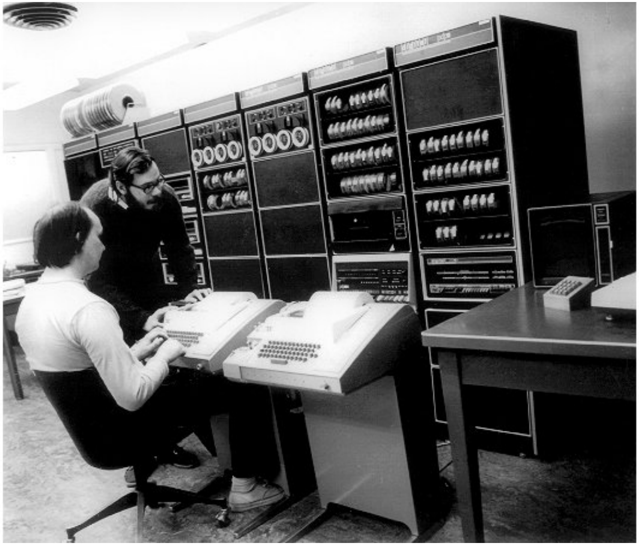
I recommend reading the books by the designers of the languages: K&R and Stroustrup.

- K&R: Brian Kernighan and Dennis Ritchie: The C Programming Language (1988)
- The C Book, available for free: `http://publications.gbdirect.co.uk/c_book/`
- Bjarne Stroustrup: The C++ Programming Language (2013)
- C language standard: `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2014/n4296.pdf`
- Gamma et al: Design patterns: elements of reusable object-oriented software

The C programming language

- Designed by Dennis Ritchie at Bell Labs
- based on earlier B by Ken Thompson
- Evolution: CPL $\rightarrow$ BCPL $\rightarrow$ B $\rightarrow$ C $\rightarrow$ C++ $\rightarrow$ Java
- C is typical of late 1960s/early 1970 language design (compare Pascal or Algol-W)
- C is minimalistic, much is reduced to pointers
- C is above all a systems programming language
- Other systems programming languages are largely forgotten
- C took over the world by accident, due to Unix
- Easy to implement, not easy to use
- Never intended for beginners
- Aimed a users who write their own compiler and operating system

Thompson and Ritchie and their mini-computer in 1972



That explains why C is efficient and concise.

The C++ programming language

- Designed by Bjarne Stroupstroup and then committees
- C++ aims to be more modern
- No Garbage Collection, unlike Java, OCaml, Haskell, Javascript
- both high-level and low-level
- C (is essentially) a subset of C++
- C is NOT a subset of Java, not even close
- C++ is the most complicated major language
- C++ has object-orientation but does not force you to use it
- C++ keeps evolving, e.g. lambda

C, C++, and Java

C

core imperative language (assignment, while, functions, recursion)
+ malloc and free; no garbage collector
+ pointers combined with other language features

C++

core imperative language (assignment, while, functions, recursion)
+ new and delete; no garbage collector
+ object orientation
+ templates

Java

core imperative language (assignment, while, functions, recursion)
+ simplified version of C++ object-orientation
+ garbage collector $\Rightarrow$ programmer can be naive about memory

Why C is (still) important

- Bits have not gone out of fashion (though there are more of them)
- systems programming
- "portable assembly language"
- $\Rightarrow$ prerequisite for OS module
- compilers: Clang is written in C++
- security: buffer overflow $\Rightarrow$ catastrophic failure
  see also Heartbleed bug
- extensions of C, e.g. CUDA C, OpenCL for programming graphics processors
- different view of programming than higher level languages

High level languages and C

C is half way between high level languages and machine code
Compiled code is a mess of pointers, not unlike C.
Even if you program in Haskell, it helps to understand what happens at this level.

Learning C

- C is concise
- C (not C++) is a small language, made from a few fundamental constructs
- C/C++ is unforgiving (very different from Java)
- C/C++ code often does difficult things, e.g. in compilers and operating systems

Therefore, in this module:

- use small code examples
- not use APIs
- need to understand what the code does

Moving from Java to C/C++

C code can be:

1. the same as in Java or other curly braces languages
2. slightly different from Java, but not rocket science
3. fundamentally different from Java and not trivial

This module will focus on the last.

Factorial in C

```
int factorial(int n)
{
  if(n == 0)
    return 1;
  else
    return n * factorial(n - 1);
}
```

A function in C is like a method in Java without any objects.
More or less like `public static`.

Factorial in C without recursion

```
int factorial2(int n)
{
    int res = 1;
    while(n > 0) res *= n--;
    return res;
}
```

Exercise

Take some of the examples of recursive methods in Java from the first year and translate them to C.

How C is unlike Java

Here is some Linux kernel code [1]

```
int (*open)(struct device *dev);
int (*stop)(struct device *dev);
int (*hard_start_xmit) (struct sk_buff *skb,
                        struct device *dev);
```

Does that look like Java?

[1] http://www.tldp.org/LDP/tlk/ds/ds.html

How C is unlike Java

Here is some Linux kernel code [1]

```
int (*open)(struct device *dev);
int (*stop)(struct device *dev);
int (*hard_start_xmit) (struct sk_buff *skb,
                        struct device *dev);
```

Does that look like Java?
Well, the is `int`. And semicolons. ☺

[1] `http://www.tldp.org/LDP/tlk/ds/ds.html`

How C is unlike Java

Here is some Linux kernel code [1]

```
int (*open)(struct device *dev);
int (*stop)(struct device *dev);
int (*hard_start_xmit) (struct sk_buff *skb,
                        struct device *dev);
```

Does that look like Java?
Well, the is `int`. And semicolons. 😜
Pointers to structures and functions, `*`.

[1] http://www.tldp.org/LDP/tlk/ds/ds.html

Header files in C

C programs may be split into separate files.
It is up to the programmer how to divide the code into files.
The purpose of a .h file in C is a bit like an interface in Java
It contains function declarations (sometimes called prototypes) and structure definitions.

```
#include <file>
```

System header files, such as stdlib.h or stdio.h

```
#include "file"
```

Header files for your own code.
Modern languages have much more elaborate constructs for organizing large programs.
C++ object, namespaces, templates, . . .

Compiling and running C with Clang on Linux

Edit `myprogram.c` in an editor, such as Emacs
To load LLVM, which includes Clang:

```
module load llvm
```

To compile with clang

```
clang -o myprogram myprogram.c
```

To run the compiled code:

```
./myprogram
```