

CS2002 – C Lecture 1 - Introduction 1

CS2002


Computer Systems

Lecture 1

Introduction

Jon Lewis (JC 0.26)
School of Computer Science
University of St Andrews

1




CS2002 – C Lecture 1 - Introduction 2

Overview

- Admin
- C Language
 - Background
 - Compiling Programs
- C Language Basics
 - Types and constants
 - Declarations
 - Operations + Expressions
 - #defines and the preprocessor
 - Basic I/O - printf and scanf

2




CS2002 – C Lecture 1 - Introduction 3

Admin

- C/SP Lectures
 - Thursdays and Fridays at 9.00
 - Recording later on Panopto
- Tutorials
 - Start in week 2
 - Please fill in enrolment on MMS to indicate times you cannot make

3




CS2002 – C Lecture 1 - Introduction 4

Text Books

- Kernighan and Ritchie
 - The C Programming Language, Wiley, 2nd Edition
- Szuhay
 - Learn C programming: a beginner's guide to learning C programming the easy and disciplined way
- Stevens and Rago
 - Advanced Programming in the UNIX Environment
- King K.N.
 - C Programming: A Modern Approach, 2nd ed. Edition, WW Norton, 2008.
- Prata S
 - C Primer Plus, 3rd Edition, The Waite Group, SAMS, 1999

4




CS2002 – C Lecture 1 - Introduction
5

Why Learn C?

(with assembly, logic, alongside)

- Interested in OS, Embedded Systems, ...
- Interested in how systems work
- Affinity for programming

5




CS2002 – C Lecture 1 - Introduction
6

C History

- Started in the early 70s by Ken Thompson and Dennis Ritchie
- In 1973 Unix was rewritten in C
- Systems programming language - Unix kernels are written in C.
- Most kernels are written in C or derivatives (C++)
- 1978 - K&R C. Standard based on a book.
- 1989 - ANSI C (C89) and ISO C (C90)
- 1999- C99 released, 2011-C11 released
- 2018-C17 published (also known as C18)
 - default for clang and gcc on lab machines

6




CS2002 – C Lecture 1 - Introduction
7

C Basics

- Design Features
 - Bit-level and byte-level operations.
 - Minimal language with libraries.
 - Weakly typed - easy to coerce between types.
- Properties
 - Available on every processor and OS
 - Very portable if written carefully
 - Efficient and powerful
 - Allows direct access to the CPU, memory and hardware.

7




CS2002 – C Lecture 1 - Introduction
8

C vs. Java

Java	C
object-oriented	function-oriented
strongly-typed	can be overridden
polymorphism (+, ==), overloading, generics	very limited (integer/float)
classes for name space	(mostly) single name space, file-oriented
layered I/O model	byte-stream I/O

8




CS2002 – C Lecture 1 - Introduction
9

C vs. Java

Java	C
automatic memory management	function calls (C++ has some support)
no pointers (a bit of a lie)	pointers (memory addresses) common
by-value parameters (value for objects is reference)	by-value and by-pointer reference
exceptions, exception handling	if (f() < 0) {error} OS signals
concurrency (threads)	concurrency (threads library)

9




CS2002 – C Lecture 1 - Introduction
10

C vs. Java

Java	C
length of array	on your own!
string as type	just bytes (char []), with 0 end
extensive standard library	small standard library

10




CS2002 – C Lecture 1 - Introduction
11

A Java program is...

- collection of classes
- class containing main method is starting class
- running `java StartClass` invokes `StartClass.main` method
- JVM loads other classes as required

11




CS2002 – C Lecture 1 - Introduction
12

A C program is...

- collection of functions
- one function – `main()` – is starting function
- running executable (default name `a.out`) starts main function
- typically, single program with all user code linked in – but can be dynamic libraries (`.dll`, `.so`)

12



CS2002 – C Lecture 1 - Introduction **13**

C vs. Java

```


public class Hello {
    public static void main (String[] args){
        System.out.println("Hello World");
    }
}

#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hello World\n");
    return 0;
}

```

13




CS2002 – C Lecture 1 - Introduction **14**

C versions

- In this course you are allowed to use either **clang** or **gcc**.
 - The lab machines have **both**
- Code you write should compile fine on both.
- **clang** defaults to compiling c17, mostly better warnings/errors
- **gcc** defaults to c17.
- You should find **gcc** and **clang** interchangeable.

14



CS2002 – C Lecture 1 - Introduction **15**

Basic compiling

```


$ gcc world.c
$ clang world.c

# My advised defaults:

$ gcc world.c -Wall -Wextra -g
$ clang world.c -Wall -Wextra -g

```

15




CS2002 – C Lecture 1 - Introduction **16**

Editor & IDEs

- The 'unix classics'
 - **ed**, **vi**, **vim** and **emacs**
- Simple GUI editors
 - **gedit** (gnome), **kate** (KDE), **jedit** (java), **smultron**, **atom**
- IDEs
 - **Visual Studio Code**
 - Good option for remote working as it lets you develop code that is stored on our servers
 - **Eclipse** (install C development tools from Market Place)
 - **CLion** (Jetbrains)

16




CS2002 – C Lecture 1 - Introduction 17

Using gcc

- Two-stage compilation
 - pre-process & compile: `gcc -c hello.c`
 - link: `gcc -o hello hello.o`
- Linking several modules:


```
gcc -c a.c → a.o
gcc -c b.c → b.o
gcc -o hello a.o b.o
```
- Using math library
 - `gcc -o calc calc.c -lm`

17




CS2002 – C Lecture 1 - Introduction 18

gcc errors

- Multiple sources
 - preprocessor: missing include files
 - parser: syntax errors
 - assembler: rare
- Compiling each module (file) separately
 - Produces object code for each
 - Assumes references to external names will be resolved later
- Undefined names will be reported when linking:


```
undefined symbol  first referenced in file
_print           program.o
ld fatal: Symbol referencing errors
No output written to file.
```

18




CS2002 – C Lecture 1 - Introduction 19

Error reporting in gcc

- If gcc gets confused, hundreds of messages
 - fix first, and then retry – ignore the rest
- gcc will produce an executable with warnings
 - don't ignore warnings – compiler choice is often not what you had in mind
- Depending on version, may not flag common errors
 - `if (x = 0)` vs. `if (x == 0)`

19




CS2002 – C Lecture 1 - Introduction 20

Char – the most basic type

- `sizeof(char)` is always 1
- On every computer you ever see, char will be 8 bits.
- unsigned char: `{0..255}`
- signed char: `{-128...127}`

20




CS2002 – C Lecture 1 - Introduction
21

Type sizes

- An `int` is the “normal” integer.
- Size of types are ordered:
 $\text{char} \leq \text{short} \leq \text{int} \leq \text{long} \leq \text{long long}$
- Every type (except `char`) is signed by default.
- There is also an unsigned, which does not allow negative values.

21




CS2002 – C Lecture 1 - Introduction
22

Type sizes (2)

	32-bit Unix	64-bit Unix	32-bit Windows	64-bit Windows
short	2	2	2	2
int	4	4	4	4
long	4	8	4	4
pointer	4	8	4	8

22




CS2002 – C Lecture 1 - Introduction
23

Implementation Defined (vs ‘undefined’ or ‘unspecified’)

- Several things in C are **implementation defined**.
 - This means each compiler can behave differently.
- The most obvious examples you will come across are
 - the size of datatypes
 - organisation of variables in memory

23




CS2002 – C Lecture 1 - Introduction
24

Literals

0, 7, 99, -100	Integers (int by default)
1.0, 3.141	Floating Point
1e25	(1*10 ²⁵) (double by default)
-13.41e-12	(-13.41*10 ⁻¹²)
100ul	unsigned long
1000000001l	long long
1.0f	float
'A', ' ', '&', '\\'	characters
"A", "Buffy", "@#!"	strings

24




CS2002 – C Lecture 1 - Introduction
25

Variable Declarations

- All variable declarations look like Java (primitive type) declarations
- Identifiers are case sensitive, any length, containing letters, digits and `'_'`. They start with a letter or `'_'`;
- WARNING: Only global variables are automatically initialised (to 0).
- You MUST initialise all other variables.
- Declarations can appear (almost) anywhere.
 - (In C89, they could only be global, or appear at the start of a block)

25



CS2002 – C Lecture 1 - Introduction
26

Example Declarations

```


int year = 2021, day = 28, month = 1;
float some_number, other_number = 6.7;
char letter = 'b'

const int foo = 200;
const float bar = 4;

int i = 1.5;           # Silently converted to int!
float f = 1/2;         # = 0!
double d = 1.0/2;     # = 0.5

```

26



CS2002 – C Lecture 1 - Introduction
27

printf


- Your basic string outputting friend!
- Outputs strings, integers and floating points.
- Ordinary characters in a format string are printed as they appear in the string; conversion specifications are replaced

```

printf("Sum: %d / %d = %f\n", 1, 2, 1.0/2.0);
Sum: 1 / 2 = 0.500000

```

27




CS2002 – C Lecture 1 - Introduction
28

Printf modifiers ('conversion specifiers')

- the number of conversion specifications in a format string may not be checked by the compiler

Meaning	Symbol
int	%i or %d
char	%c
String	%s
float (exp notation)	%f (%e)
Shortest of %f %e	%g
Hex, Octal	%x, %o
long int, double	%li, %lf

28




CS2002 – C Lecture 1 - Introduction
29

Operators

arithmetic	+	-	*	/	%
relational	<	<=	>=	>	== !=
boolean	&&		!		
increment	++	--			
bitwise	&		^	~	

29



CS2002 – C Lecture 1 - Introduction
30

Increment & Decrement

The unary increment and decrement operators, ++ and --, add or subtract a unit value to the variable.


```

int i = 0;
i++;      /* i = 1      */
j = i++;  /* j = 1, i = 2 */
j = ++i;  /* j = 3, i = 3 */
j = --i;  /* j = 2, i = 2 */

i += j;   /* j = 2, i = 4 */
j *= 3;   /* j = 6, i = 4 */
j -= 2;   /* j = 4, i = 4 */

```

30



CS2002 – C Lecture 1 - Introduction
31

Comments

- There are two kinds of comments:


```

/*
    Block comments can cover
    multiple lines
*/

// Single line comment

```

31



CS2002 – C Lecture 1 - Introduction
32

#include

(pre-processor directive)

- #include inserts a file into the source code. These are commonly called 'header files' and given the extension '.h'

```

#include <stdio.h>           - system header
#include "myheader.h"       - from current directory
#include <curl/curl.h>       - library header

```

32



scanf and printf

- scanf is the inverse of printf, it reads input.
- Formatted in (almost) the same way.

```
int i, j;
scanf("%d %d", &i, &j);
printf("You entered %d and %d", i, j);
```

- The unary & operator gives the address of where the variable is stored in memory – so scanf can write the value entered by the user to that address
- Much more on **pointers** later

33



Special Characters

Character	Symbol
Newline	\n
Single Quote	\'
Double Quote	\"
Backslash	\\
Tab	\t
Hex Constant	\x00, \x01, ... , \xff

34



Basic C Program

```
// My program in "circumference.c"!
#include <stdio.h>
#define PI 3.14

int main() {
    int radius = 0;
    float circ;
    printf("Enter the radius : \n");
    scanf("%i", &radius);
    circ = 2 * PI * radius;
    printf("radius = %i, circumference = %f\n",
           radius, circ);
}
```

35



Compiling and running

- **Compile** .c to .o
`$ clang -Wall -Wextra -g -c circumference.c`
- **Link** all .o files (only one in this case) with standard library to produce executable
`$ clang -Wall -Wextra -g -o circumference circumference.o`
- **Run** the executable code
`$./circumference`
enter the radius :
3
Circumference of circle with radius 3 is 18.840000

36