Data Representation

Programming Fundamentals

Dr Kevin Chalmers

School of Computing
Edinburgh Napier University
Edinburgh

k.chalmers@napier.ac.uk

Edinburgh Napier UNIVERSITY

Notes

Outline

Notes

But to start with...

- Assessment for this module!
- Two components
  - 60% Coursework
    - Two parts - currently planning week 7 and week 13 submissions
  - 40% Exam
    - Centrally timetabled. Will be week 14 or week 15

Notes

References

- Burd, Stephen (2006), *Systems Architecture*, 5th Edition
  - Chapter 3 "Data Representation"
  - Available in Library
- Kernighan & Ritchie (1988), *The C Programming Language*, 2nd Edition

Notes

Outline

Notes

There are 10 kinds of people in the world, those who understand binary, and those who don't

Notes

What is Data?

- As humans, we handle information every day.
- We see, we speak, we hear, we read, we perform mathematical operations etc.
- We can understand and adopt many languages, script systems, numerical systems, visual and audio stimuli.

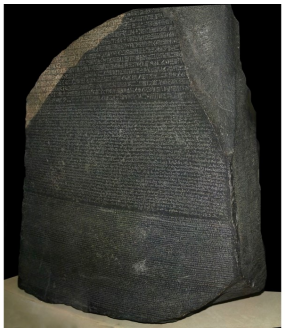

Figure : The Rosetta Stone, Wikimedia Commons ©Hans Hillwaert, CC BY-SA 4.0

Notes

What is Data?

- We build computers in order to help us process information ...
  - ... but computers are dumb! They only understand two states: on and off.
- This is because they process electric signals. For abstraction purposes we talk about zeros and ones.
- In science and computing, data has multiple definitions. In our context, it designates information in a format that can be processed by a computer.

Notes

Data *[mass noun]*
The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media. (Oxford Dictionary of English)

Notes

Data Representation

- Information has to be represented in a form that can be easily reproduced by electric signals: binary numbers.
- Most human civilisations use the decimal representation (base 10) where 10 digits are available (0 to 9) and we add powers of 10 together to form larger values (tens, hundreds, thousands etc).
- Binary (base 2) is a system where only two binary digits (bits) are availabe: 0 and 1. To form larger numbers we add powers of two together.

Notes

Binary Example

- Let's use the number 42, in base 10 it can be broken down:
  - $42 = 40 + 2 = 4 \times 10 + 2 \times 1 = 4 \times 10^1 + 2 \times 10^1$
- Its binary representation is 101010
  - $1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$
  - In base 10 $1 \times 32 + 0 \times 16 + 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1$
- This is a simplistic example, we want computers to handle more complex data: real numbers, characters, images, sounds etc.

Notes

Questions?

Notes

Outline

Notes

Data Types in C

- C can handle a large range of number types: unsigned integers, signed integers, floating point numbers.
- Each data type has a size, usually expressed in multiples of 8 bits (bytes).
- The `sizeof` operator returns the size of a data type in bytes. Example:
  - `printf("An int is %d bytes.\n", sizeof(int));`
  - `An int is 4 bytes` – in most modern platforms

Notes

Integers in C

- Common types to store integers in C are `char`, `short`, `int`, `long`, and `long long`
- On older platforms, an `int` would equate to a short (16 bits) but on most modern platforms an `int` is equivalent to a `long` (32 bits).
- They can be `signed` or `unsigned`, i.e. they can handle positive and negative values, or positive values only. The type is usually `signed` by default. To alter it, you can use the `unsigned` keyword, e.g.
  - `unsigned int example = 42;`

Notes

Example

- Let's go back to our example, 42. We can represent it as an unsigned char (8 bits). All bits are used for the value.
  - | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
- However, if we need to represent the value -42, we need to use one bit to represent the positive (0) or negative (1) sign.
  - | **0** | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
  - | **1** | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
- The *signing bit* alters the *width* of the number from 8 to 7 bits, however it allows us to work with negative values.

Notes

Integer Sizes and Ranges

Type	**Size**	`unsigned` **range**	`signed` **range**
`char`	8 bits (1 byte)	$0 \dots 255$	$-127 \cdots +127$
`short`	16 bits	$0 \dots 65,535$	$-32,767 \cdots +32,767$
`int`	16 bits or 32 bits	Same as `short` or `long`	Same as `short` or `long`
`long`	32 bits	$0 \dots 2^{32}-1$	$-(2^{31}-1) \cdots +(2^{31}-1)$
`long long`	64 bits	$0 \dots 2^{64}-1$	$-(2^{63}-1) \cdots +(2^{63}-1)$

Notes

Data Types - Floating Point

- Integers are useful, but in the real world we often work with real numbers and fractionals.
- C uses the types `float` (32 bits) and `double` (64 bits) and `long double` (128 bits), which are binary representations of real numbers using the *scientific notation*. They are called *floating point* because they are represented in a way where the decimal point's location can vary.
- When it comes to floating point numbers being represented as `float` or `double`, we talk about *single precision* or *double precision*.

Notes

Data Types - Alphanumerical

- So far we used `char` to represent one byte long numbers.
- This is where it gets interesting... don't we use `char` to represent a character?
  - We do, in fact characters are just binary data, like any number. It boils down to how we interpret it.
- In the following statement, what output would we get?

Outputting char Data

```
char a = 'A';
char b = 65;
printf(   a  = %c and b = %c\n   , a, b);
```

Notes

It's all Data!

- `a = A and b = A`
- This is because the character `A` in the ASCII table and the number 65 have the same binary representation over 8 bits.
- Below is a statement a C compiler will accept. What will the output be?

Outputting ASCII

```
for(char i = 'A'; i < 'A' + 26; i++)
{
    printf("%c %d\n", i, i);
}
```

Notes

It's all Data!

- This would display the entire uppercase alphabet and their ASCII codes.
- It is because we use different place-holders in our `printf` statement, forcing the function to display the same binary data in different forms.
- %c displays a char as a character, %d as a number.
- Other place-holders:
  - `%u` for `unsigned`
  - `%l` for `long`
- They can even be combined:
  - `%lu` is an `unsigned long`
  - `%ll` is a `long long`

Notes

Types and Placeholders

- What happens is we want to force display a `signed int` as `unsigned`? e.g.

Printing `signed` as `unsigned`

```
int n = -42;
printf("Value %u\n", n);
```

- We would get 4294967254 as a result. This is because the actual binary data in memory is 11111111111111111111111111010110 which represents different `signed` and `unsigned` values.
- Note that the `signed` representation of -42 is different from that given previously. This is because the actual representation of negative numbers differs from our theoretical approach earlier.

Notes

Questions?

Notes

Outline

Notes

Data Sizes - 32-bit vs. 64-bit

- One type of data varies according the hardware architecture - *pointers*.
- This is because pointers are variables that contain a memory address, not an actual value.
  - On a 32-bit system `sizeof(char *)` would return 4 and on a 64-bit system it would return 8.
- This is why 32-bit systems cannot handle more than 4GB of memory, as the largest possible value on a 32-bit number is roughly 4.2 billion
  - Therefore the computer can only address 4.2 billion memory locations, or 4.2 billion bytes

Notes

Casting

- C provides a way to convert a data type to another - *casting*.
- Casting can happen implicitly under some conditions, but has to be performed explicitly in some others. The syntax is the following:
  - `(type)expression`
- For example, we can convert the `int 42` to the `float 42.0f` as follows:

Casting from `int` to `float`

```
int a = 42;     // Signed integer 42
float b = (float)a; // Floating point 42.0
```

Notes

Casting

- Casting at our current level is useful to convert different number types to each other. As you will see in the workbook, there are limitations which must be considered.
- We will make greater use of casting when working with dynamically allocated memory, and later with object-orientation in C++.

Notes

structs

- C allows the creation of more complex data types. One of them is structures.
- Structures are aggregates of basic data types, with the idea to create a more complex data type.
- To create a structure, we use a type modifier, `struct`.

Notes

Defining a struct

- This an example of a structure for a student:

student struct

```
struct student
{
    unsigned int matric;
    char *name;
    char *address;
};
```

Byte	Data
0	matric
1	matric
2	matric
3	matric
4	name
5	name
6	name
7	name
8	address
9	address
10	address
11	address

- The basic types used are all 4 bytes on a 32-bit system, therefore we would have 12 bytes in memory.
- Note - these are just pointers to the strings. The strings are stored elsewhere. More on this in further units.

Notes

Using structs

- To create a `struct` we use the following syntax:
  - `struct student s; // Note the use of struct`
- To access its members we use a dot `.` notation:

Accessing struct Members

```
s.matric = 12345678;
s.name = "Kevin";
s.address = "Edinburgh";
```

- structs are useful in C to create more complex types. When we reach C++ and object-orientation, we will have more possibilities (classes).

Notes

Type Modifiers

- Previously we saw the use of a type modifier `struct`.
- Other type modifiers exist in C, such as:
  - `const` defines a constant value (one that cannot be changed)
  - `static` defines a value in the static context. This means that the value will always exist and not be deleted
  - `extern` defines that a value is created within another file (i.e. header or code file)
- These are keywords that, attached to a type, change the nature of the variable or data, e.g.
  - Declaring `const float PI = 3.14159;` will create a non-modifiable variable.
  - `extern int variable;` refers to a variable that was declared in a different source file.

Notes

Questions?

Notes

Outline

Notes

Summary

- Computers use binary representation of data in memory.
- Because it is an old language, C offers multiple sizes for similar types of data, in a view to optimise memory usage.
- Unlike more modern languages such as Java or C#, data types in C give direct access to their representation in memory. The data type is only a way to represent or interpret binary data.

Notes

To do...

- In the lab, workbook provides practical experience working with data representations. Also includes makefiles, `enums` and type modifiers.
- Tutorials - flowcharts!
- Complete the test quiz for unit 02. Same as before - do it multiple times as you will get different questions.
- Next we will look at how to integrate assembly code into our C programs in a bid to understand how your C code relates to the machine's code.

Notes

Notes