COMP1911 - Computing 1A

5. Numeric Types

In this lecture we will cover:

- Number systems you understand (decimal)
- New number systems (Binary, Hexadecimal)
- Converting between number systems
- Negative binary numbers
- C types

Number Systems

To understand "types", we need to understand number systems.

- System we're all familiar with:
  - Decimal system - made up of 10 unique digits
  - {0,1,2,3,4,5,6,7,8,9}

- System computers are familiar with:
  - A number system to denote "on and off" (binary)
  - {0,1}

  - binary digits are also called bits.

Decimal (Base 10)

- Once we get to 9 in decimal, we start combining digits to create 2 digit numbers that go from 10..99.
- Once we get to 99 we need to use 3 digit numbers, such as 100..999 etc.
- When we have a number such as 4913 what does it actually mean?

  - ➢ Why is it different from 9413 or 1349 which contain the same digits?
  - ➢ The positions of the digits are very important
  - ➢ The number 4913 actually means
    - ✓ 4000 + 900 + 10 + 3 which can also be expressed as
    - ✓ $4 * 10^3 + 9*10^2 + 1*10^1 + 3*10^0$

When poll is active, respond at **PollEv.com/binghaoli068**

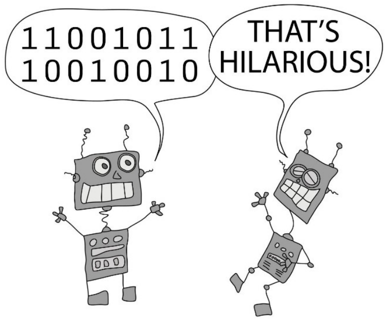
UNSW SYDNEY

Using 10 fingers, how many numbers you can represent?

Total Results: 0

Binary (Base 2)

- Since we can only use 0 and 1, once we get to 1 in binary, we start combining bits to create 2 bit numbers then 3 bit numbers, then 4 bit numbers etc.
- When we have 8 bits we call it a byte!
- The number 1011 actually means
  - $1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$
  - 8 + 2 + 1 = 11



What is the largest number you can represent in 4 bits?

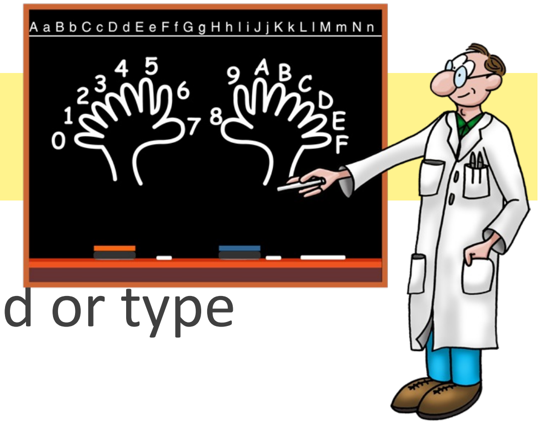
Binary (Base 2)

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

Decimal	Binary
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

- so 4 bits can represent 16 different numbers from 0..15
- what about in a byte (8 bits)? Is there a pattern to help us ?
  - 4 bits represents 2^4 different numbers from $0..2^4 - 1$
  - 8 bits represents 2^8 different numbers from $0..2^8 - 1$

Hexadecimal (Base 16)



- Binary numbers are not easy for humans to read or type correctly.
- We can map 4 bits to 1 hexadecimal digit which is more readable for humans.
- In hexadecimal we have 16 unique digits {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
- Once we get up to F we need 2 digits numbers, when we get to FF, we need 3 digit numbers etc.
- We use the prefix 0x to make it clear we are treating our number as hexadecimal (not decimal). e.g.
    - 0x8FA1
    - 0x1021 which is very different from 1021

Hexadecimal (Base 16)

- The number 0x8FA1 actually means
  - $8 * 16^3 + 15 * 16^2 + 10 * 16^1 + 1 * 16^0$
  - 32768 + 3840 + 160 + 1 = 36769
- To convert 0x8FA1 to binary just convert each hexadecimal to 4 bit binary numbers and join them together e.g.

8	F	A	1
1000	1111	1010	0001

- You can do the same in reverse to convert from binary to hexadecimal
- (Not in scope of course) How do you think Octal aka Base 8 works?

Converting from Decimal to other Bases

An example:

To convert 13 to binary

- 13/2 = 6 remainder 1 (this will be the right most bit)
- 6/2 = 3 remainder 0
- 3/2 = 1 remainder 1
- 1/2 = 0 remainder 1 (this will be the left most bit)
- Answer: 1101

Summary of "Algorithm" (A process of specified rules or process used to solve a problem):

- While (the quotient is not zero)
- Divide the decimal number by the new base
- Make the remainder the next digit to the left in the answer
- Replace the original decimal number with the quotient

Exercise: convert 26 to binary

Converting from Decimal to other Bases

To convert 100 to hexadecimal

- 100/16 = 6 remainder 4 (this will be the right most bit)
- 6/16 = 0 remainder 6 (this will be the left most bit)
- Answer: 0x64

Alternatively, you can convert to binary and then from binary to hexadecimal if you don't like dividing by 16.

Try converting 48 to hexadecimal yourself!

Convert 127 (decimal) to Binary and Hexadecimal.

Total Results: 0

Binary and Negative Numers

- 4 bits can represent 16 different integers.
  - ➢ If we only need positive numbers we can represent 0..15.
  - ➢ If we want negative numbers, we only have space to represent -8..7
- On computer systems when we want to be able to represent negative numbers we use a binary format called two's complement.

Sign–magnitude
Ones' complement

Positive-only Numbers

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal	Binary
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Positive and Negative Numbers: Sign–magnitude

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Simply change the sign

Decimal	Binary
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

Positive and Negative Numbers: Ones' complement

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Bitwise NOT applied

Decimal	Binary
-7	1000
-6	1001
-5	1010
-4	1011
-3	1100
-2	1101
-1	1110
-0	1111

Positive and Negative Numbers: two's complement

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Decimal	Binary
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111

So with 4 bits we can either have:

- 2^3 -1 positive numbers 2^3 negative numbers and 0.
- 2^4 -1 positive numbers and 0

Note: All negative numbers start with a 1

Converting positive to negative

- To negate a number just reverse all 0's and 1's and add 1.
- 0101 (5):
  - with bits reversed is 1010
  - then add 1 and you get 1011 (-5)

- More examples: 7, 0x17

Finding the value of a negative number

- Just reverse all 0's and 1's and add 1.
- 1001 (-7):
  - with bits reversed is 0110
  - then add 1 and you get 0111 (7), the original sign is 1, negative number
- More examples: 1000 1011; 1111 1101

Convert binary 0101 1111 and binary 1001 1111 (signed) to decimal?

- 95, -97
- 94, -96
- 95, -31
- 95, 159

Total Results: 0

Exercise

Suppose we only have 4 bits. What happens if we add 1 to -1?

$$\begin{array}{l} 1111\ + \\ 0001 \\ \hline \\ 0000 = 0 \end{array}$$

That is what we would hope to get

What happens if we add 1 to 7?

$$\begin{array}{l} 0111\ + \\ 0001 \\ \hline \\ 1000 = \text{-}8 \end{array}$$

Overflow!

Numeric Types in C: int

- The simplest numeric type is int.
- Variables of type int store integer-valued numbers in a constrained range, often -2^{31} to $2^{31} - 1$,

  i.e. -2,147,483,648 to +2,147,483,647
- These bounds are a consequence of **4 bytes** (32 bits) being used to store the variable.

Numeric Types in C: int

Hard question: What is the output by this program fragment?

```
int big, bigPlus1, bigTimes2, reallyBig;

big = 2147483647;
bigPlus1 = big + 1;
bigTimes2 = big * 2;
reallyBig = bigPlus1 * 2;
printf("big=%d bigPlus1=%d\n", big, bigPlus1);
printf("bigTimes2=%d ", bigTimes2);
printf("reallyBig=%d\n", reallyBig);
```

Numeric Types in C: int

Not what you think!

If we compiled with dcc, then run it, we will get a run-time error saying
"runtime error: signed integer overow"
with gcc we will get

```
big=2147483647 bigPlus1=-2147483648

bigTimes2=-2 reallyBig=0
```

The computation has exceeded the ability of the computer to represent integers.

Beware. Most C implementations don't check for integer overflow leading to incorrect values propagating through the remainder of any computation without any warning message.

Overflow and Other Types

Possible solution if int is not sufficient

- Use variables of type long. On lab machines these are 8 bytes (64 bits). This is sufficient to avoid overflow for many applications.
- If this is insufficient you could try an unsigned long if you were working with large positive numbers (Why?)
- If this is still insufficient, you can use a floating point types (double) although double will also reach its limits, so it does not completely solve the problem.

Other integer Types

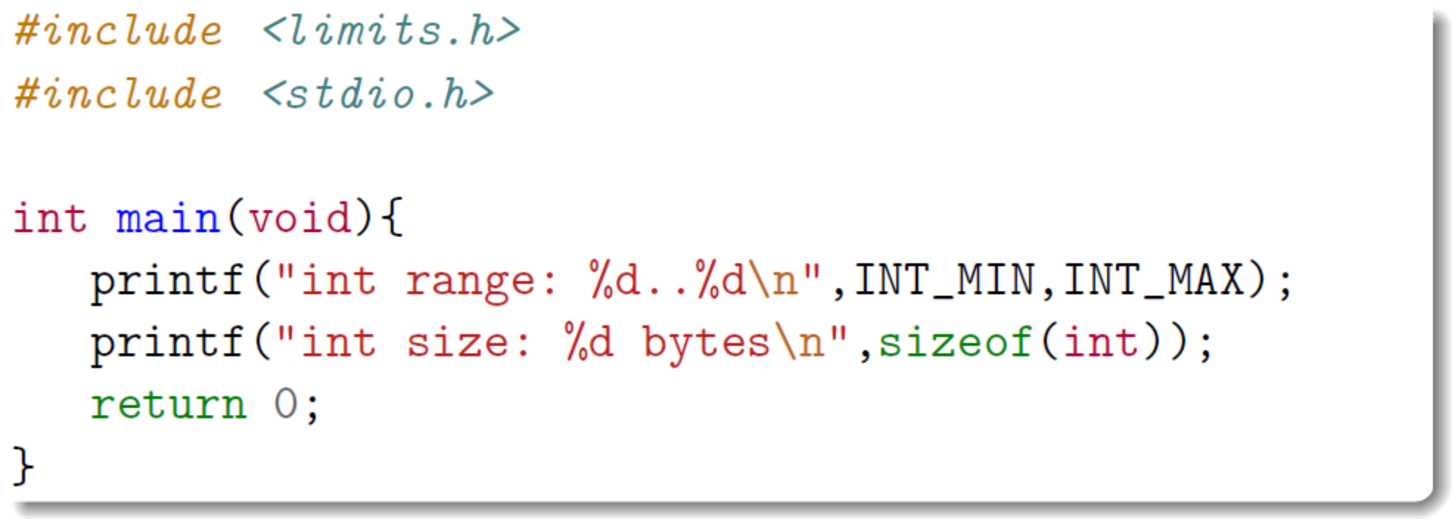
Note: all integer types are signed by default except char, which may signed or unsigned by default depending on the system.

type	bytes	range
signed char	1	-128..127
unsigned char	1	0..255
short	2	-32768..32767
unsigned short	2	0..65535
int	4 *	-2147483648..2147483647
unsigned int	4 *	0..4294967295
long	8 *	-2147483648..2147483647
unsigned long	8 *	0..4294967295
long long	8 *	$-2^{63}..2^{63}-1$
unsigned long long	8 *	$0..2^{64}$

* The sizes (and thus ranges) given for int, long and long long may vary on different systems. These are the sizes for the cse machines.

Other integer Types

The library *limits.h* defines constants that represent the largest and smallest values of the different integer types on the particular system you are running.
You can also find out the number of bytes each type is stored in on your system by using the *sizeof* command.



```c
#include <limits.h>
#include <stdio.h>

int main(void){
   printf("int range: %d..%d\n",INT_MIN,INT_MAX);
   printf("int size: %d bytes\n",sizeof(int));
   return 0;
}
```

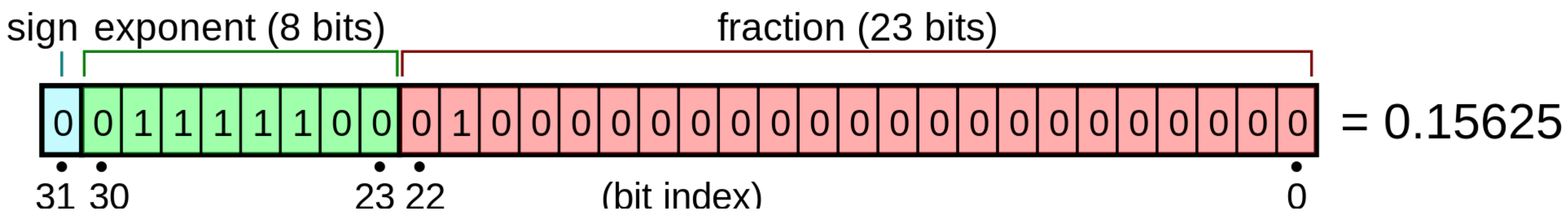
For this course we recommend you use int for integer values unless there is a good reason not to.

Types for Real Numbers

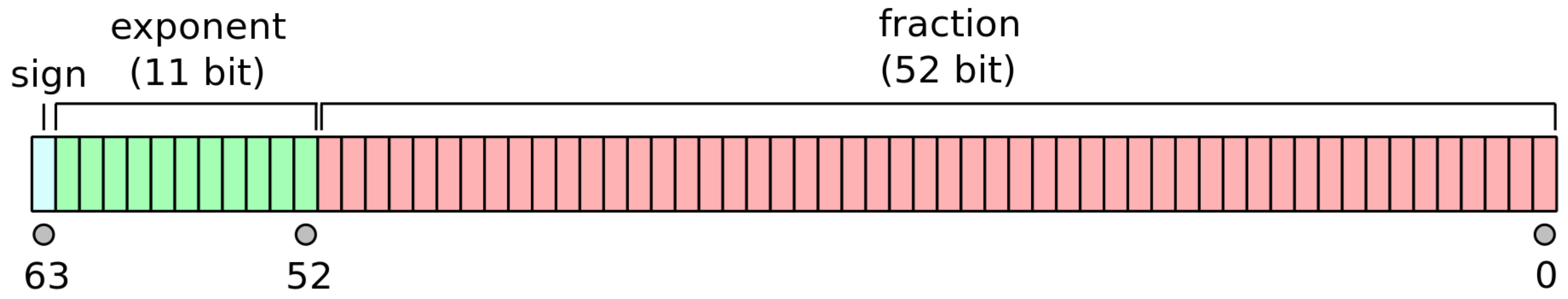
- If we want to store real numbers then we could use a float or a double
- floats are stored in 4 bytes while doubles are stored in 8.
- By default we recommend you use double rather than float as double has better precision.
- Note: There will still be precision errors even with double. Be careful with these types

Floating Number

layout for 32-bit floating point



layout for 64-bit floating point



Value: $$(-1)^{\text{sign}}\left(1+\sum_{i=1}^{52} b_{52-i}2^{-i}\right)\times 2^{e-1023}$$

Exponent (e): from 2^{1023} to 2^{-1022}

Floating Number Examples

32-bit floating point

2	=	$1*2^1$	=	4000 0000
-2	=	$-1*2^1$	=	C000 0000
π/180	=	1.745329e-02	=	3C8E FA35
0	=	$1.0*2^{-128}$	=	0000 0000

64-bit floating point

2	=	$1*2^1$	=	4000 0000 0000 0000
-2	=	$-1*2^1$	=	C000 0000 0000 0000
π/180	=	1.745329251994330e-02	=	3F91 DF46 A252 9D39

- If you are interested in this, Google IEEE 754 Standard or do COMP1521. This is not examinable in this course.

I/O

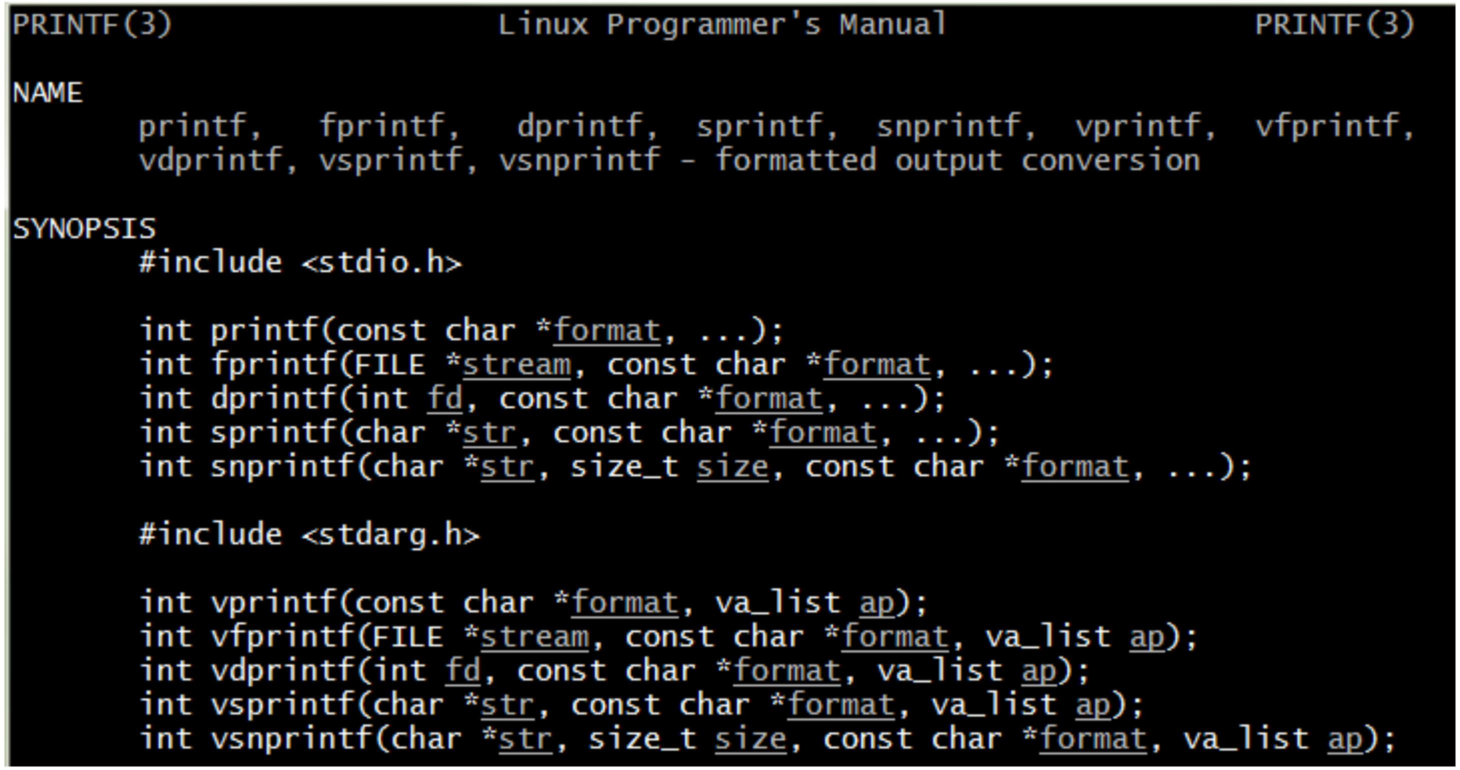
scanf Conversion Specifiers

%d	corresponds to the	char type
%d	corresponds to the	short type
%d	corresponds to the	int type
%u	corresponds to the	unsigned int type
%ld	corresponds to the	long type
%lld	corresponds to the	long long type
%llu	corresponds to the	unsigned long long type
%f	corresponds to the	float type
%lf	corresponds to the	double type

I/O

The Unix man command

To find out more check out your textbook and/or use the man command in a Unix terminal, e.g., man 3 printf.



```
PRINTF(3)                  Linux Programmer's Manual                  PRINTF(3)

NAME
       printf,   fprintf,   dprintf,  sprintf,  snprintf,  vprintf,  vfprintf,
       vdprintf, vsprintf, vsnprintf - formatted output conversion

SYNOPSIS
       #include <stdio.h>

       int printf(const char *format, ...);
       int fprintf(FILE *stream, const char *format, ...);
       int dprintf(int fd, const char *format, ...);
       int sprintf(char *str, const char *format, ...);
       int snprintf(char *str, size_t size, const char *format, ...);

       #include <stdarg.h>

       int vprintf(const char *format, va_list ap);
       int vfprintf(FILE *stream, const char *format, va_list ap);
       int vdprintf(int fd, const char *format, va_list ap);
       int vsprintf(char *str, const char *format, va_list ap);
       int vsnprintf(char *str, size_t size, const char *format, va_list ap);
```

Thinking Question

You have 200 bottles of milk. One of them is poisonous while the other 199 are non-poisonous. There are rabbits which dies exactly after 10 hours of drinking the poisoned bottle. You need to find out the poisoned bottle in 10 hours use minimal number of rabbits. How many rabbits you need?