COMP20005 Workshop Week 9

1	Number Representation
2	Root Finding: Ex 9.5, Ex. 9.8
3	Strings Defining new data types Structs?
LAB	Small-group exercises OR Implement: • Ex 9.11 only if you think you remember physics and Newton’s laws • Output the bit-patterns of a double value

Numeral Systems: decimal, binary, hexadecimal

2 1 1 . 3 9

$2 \times 10^2 + 1 \times 10^1 + 1 \times 10^0 \quad + \quad 3 \times 10^{-1} + 9 \times 10^{-2}$

→ *base* = 10

Other bases: binary (base= 2), hexa (base= 16) ...

$1001_{(2)} = 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 = 9_{(10)}$

$25_{(16)} = 2 \times 16^1 + 5 \times 16^0 = 37_{(10)}$

$B3_{(16)} = 11 \times 16^1 + 3 \times 16^0 = 179_{(10)}$

$111001_{(2)} = ?_{(16)} = ?_{(10)}$

$10011101_{(2)} = ?_{(16)} = ?_{(10)}$

Changing Binary → Decimal

Just expand using the base=2:

$$\ldots b_3 \quad b_2 \quad b_1 \quad b_0 \,.\, b_{-1} \quad b_{-2} \ldots \;(2)$$

is $\ldots\; b_3 x 2^3 + b_2 x 2^2 + b_1 x\, 2^1 + b_0 \;+\; b_{-1} x 2^{-1} + b_{-2} x 2^{-2} \ldots$

so $1\;0\;1\;0\;0\;1$ *and* $1\,.\,1\;0\;1$

are $2^5 \;+\; 2^3 \;+\; 1 = 41$ *and* $1 \;+\; 2^{-1} \;+\; 2^{-3} = 1.625$

Practical advise: remember

128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

Changing Decimal → Binary

Remember and apply the power-of-two sequence:

128	64	32	16	8	4	2	1	.	0.5	0.25	0.125
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}

So:

$11_{(10)} = 8 + 0 + 2 + 1 = 1101_{(2)}$

$2.625_{(10)} = 2 + 0 + 0.5 + 0 + 0.125 = 10.101_{(2)}$

Algorithm: Decimal→Binary, Integer Part

Changing integer x to binary: Just divide x and the subsequent quotients by 2 until getting zero. The sequence of remainders, in reverse order of appearance, is the binary form of x.

Example: 23

operation	quotoion	remainder
23 :2	11	1
11:2	5	1
5:2	2	1
2:2	1	0
1:2	0	1

So: $23 = 10111_{(2)}$

Algorithm: Decimal → Binary, Fraction Part

Fraction: Multiply it, and subsequent fractions, by 2 until getting zero. Result= sequence of integer parts of results, in appearance order. Examples:

0.375				0.1		
operation	int	fraction		operation	int	fraction
.375 x 2	0	.75		.1 x 2	0	.2
.75 x 2	1	.5		.2 x 2	0	.4
.5 x 2	1	.0		.4 x 2	0	.8
				.8 x 2	1	.6
				.6 x 2	1	.2

So: $0.375 = 0.011_{(2)}$ $0.1 = 0.00011(0011)_{(2)}$

Now try convert: 6.875 to binary

Converting Decimal->Binary

- $17_{(10)} = ?_{(2)}$
- $34_{(10)} =$
- $6.375_{(10)} =$

Representation of integers

representation of integers in w bits

CASE	w data bits for:
$x \geq 0$	binary form of x (note: $x < 2^{w-1}$)
$x < 0$	twos-compliment form of $\|x\|$ (which is $2^w - \|x\|$)

Examples using `w=4`*, representation:*

- *of* 2 *is* `0010`
- *of* 5 *is* `0101`
- *of* -5 *is* `1011`
- *of* 9 *is*

Note: twos-compliment of $|x|$ is just $2^w - |x|$

Finding twos-complement representation in w bits for negative numbers in 2 step

Suppose that we need to find the twos-complement representation of negative `x` *in* `w=16` *bits. It can be done easily in 2 steps:*

1) *Write the binary representation of* |`x`| *in w bits*
2) *Locate the right-most "*`1`*" bit, then inverse all the bits on its left.*

   (inverse = turn 1 *to* 0 *and vice versa)*

***find the 2-comp repr of* –40**	**Bit sequence**			
1) binary of 40 in 16 bits	0000	0000	0010	1000
2) – locate right-most "1"	0000	0000	0010	**1**000
- inverse the left	1111	1111	1101	1000

Ex: 2-complement representation in w=16 bits

Q: What are `17`, `–17`, `34`, and `–34` as 16-bit twos-complement binary numbers, when written as (a) binary digits, and (b) hexadecimal digits?

Representation of floats (as in lec slides)

- Convert $|x|$ to binary form, and transform so that:

  $|x| = 0.b_1b_2b_3\ldots \text{ x } 2^{\mathbf{e}}$ *with* $b_1 = 1$
- **e** is called exponent, **m**= $b_1b_2b_3\ldots$ is called mantissa
- Three components: sign, **e**, **m** are represented as in the diagram.

Ex: Representation of float

Q: Using the same representation as on page 27text book ($w=16$, $w_s=1$, $w_e=3$, $w_m=12$) of the numericA.pdf slides (page 235 textbook), what are the floating point representations of `-3.125`, `17.5`, and `0.09375`, when written as (a) binary digits, and (b) hexadecimal digits?

Numerical Computations

`int`, `float`, `double` all have some range:

- `int` 32 bits: about $-2x10^9$ – $2x10^9$
- `int` 64 bits:
- `float` $(w_s=1,\ w_e=8,\ w_m=23)$
- `double` $(w_s=1,\ w_e=11,\ w_m=52)$

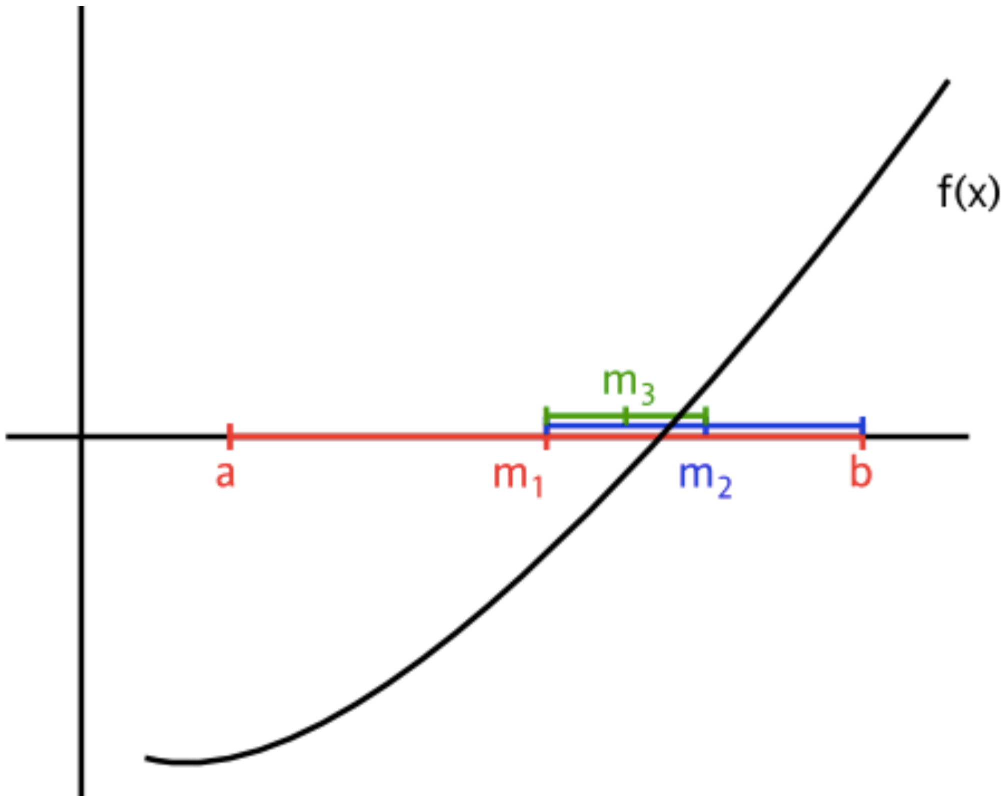
computation on `float/double` is imprecise, so

- use `if (fabs(x) < EPS)` instead of `if(x==0)`
- `if (fabs(a-b) < EPS)` instead of `if(a==b)`
- avoid adding a very large value to a very small value

Roof Finding for f(x)=0

Roof Finding for f(x)=0: bisection method

Ex. 9.5: The square root of z is the of equation $f(x)= x^2 - z$.
Evaluate the bisection method *by hand* (well, you can use calculators), start with a= 1 and b= 3. Stop when the length of the interval is 0.1 or less.



char, strings, struct, typedef

Defining new datatype

```
typedef int integer;
integer n;
```

Here we defined a new datatype, `integer`, which is the same as `int`. Hence, `n` is actually an `int` variable.

`char` and functions in `<ctype.h>`

getchar()

putchar()

isalpha(c)

isspace(c)

islower(c)

tolower(c)

Strings

- In the declaration:

```
typedef char name_t[21];
char s[10];
name_t name;
```

`s` is an array of 10 `char`. `s` can be understood as a string that can hold 9 characters as most.

Note that we can use:

```
name_t name= “David”;
```

but we cannot use:

```
name = “David”;      [WRONG]
```

for the latter, we should use:

```
strcpy(name, “David”);   [OK]
```

Important functions in <string.h>

```
strlen(s)          : returns length of s
strcpy(s1, s2)     : copies s2 to s1
strcmp(s1, s2)     : compares s1 and s2
```

Structures

Structures

- A structure is a compound object such as a student record. With the declaration:

```
typedef struct {
    char name[21];  /* name has 20 letters as most */
    int id;         /* student number */
    float score;    /* ATAR score such as 91.25 */
} student_t;
student_t student;
```

`student` is a variable which is a set of 3 independent variables

```
    student.name
    student.id
    student.score
```

and we can use these 3 variables as conventional variables.

Structures

- The best way to use structures is through `typedef` as follow:

```
typedef struct {
    char name[21];
    int id;
    float score;
} student_t;
student_t s1= {“Bob”, 1234, 97.75};
student_t s2, s3;
student_t ss[10]; /* array of 10 students */
scanf(“%s %d %f”, s2.name, &s2.id, &s2.score);
s3= s2;  /* YES, this assignment is allowed */
```

?

- How to write a function to sort an array of students in ascending order of name?

Lab: Group Work OR Implement:

1. Re-examine the `cube_root()` function on page 77 of the textbook, `croot.c` (you can copy it from the link provided in **LMS.WeeklySchedule.Week9**). What method does it use? Explore what happens if: (a) very large numbers are provided as input; (b) very small (close to zero) numbers are provided; and (c) `CUBE_ITERATIONS` is made larger or smaller.
2. Ex 9.11 if (and only if) you think you love physics. Warning: This exercise might be time-consuming!
3. Output the bit-patterns of a double value

Hints for 3): mimic Alistair's `loatbits.c`, or DIY by

a) using a "`unsigned long long`" pointer to access the "`double`" variable

b) using bitwise operations to get value of a bit. For example, if `n` is an "`unsigned long long`" then:

```
( n >> (63-i) ) & 1
```

would give the value of the `i`-*th* bit of `n` (from the left).

Function `cube_root` in `croot.c`

```
#define CUBE_LOWER 1e-6
#define CUBE_UPPER 1e+6
#define CUBE_ITERATIONS 25

double cube_root(double v) {
   double next=1.0;
   int i;
   if (fabs(v)<CUBE_LOWER || fabs(v)>CUBE_UPPER) {
      printf("Warning: cube root may be inaccurate\n");
   }
   for (i=0; i<CUBE_ITERATIONS; i++) {
      next = (2*next + v/(next*next))/3;
   }
   return next;
}
```

What method does it use? Explore what happens if: (a) very large numbers are provided as input; (b) very small (close to zero) numbers are provided; and (c) CUBE_ITERATIONS is made larger or smaller.

Buffering …

Hints for 2): mimic Alistair's f`loatbits.c`, or DIY by

Notes: Lectures in w8, 9

- Week 8:
  - lec 2: numeric
  - lec 3: finished: representation of int and float
- Week 9:
  - lec 1: Root finding – fimished on Mon
  - lec 2:
  - lec 3: