Computer Programming: Skills & Concepts (INF-1-CP1)

`double`; `float`; quadratic equations

4th October, 2010

Practical 1

- Practical 1 is **out today**. :-)
  Pick up a copy before leaving the lecture.
- **due** by **2pm**, Monday 18 October.
- 4 Tasks:
  - Part A on Imperial-to-Metric distance conversion.
  - Parts B-D are basic geometric tasks, when input is given through an interactive graphics tool.
- Should be able to attempt Parts A-C right away!
- We discuss Parts B-D in detail on Tuesday 5 October.

Lectures 4 and 5 (Julian)

- Integer arithmetic in C.
- Converting pre-decimal money to decimal.
- The `int` type and its operators.
- *Variables*.
- The “swap” problem.
- Assigning and re-assigning variables;
- The `if`-statement.
- Conditional expressions.
- Fixing the `lsd` program.
- Input using `scanf`.

Today's Lecture

- More types: `float` and `double`.
- The `marathon.c` program.
- Quadratic Equations.
- General form of `if`-statement.
- Developing `quadratic.c` via nested `if`-statements.
- Boolean operators.

A tiny problem

Calculate the number of kilometres in a marathon

We know:

- The number of miles (26) and yards (385) that make up the marathon distance;
- How many kilometres correspond to a mile ($\sim$1.609);
- How many yards in a mile (1760).

How to compute the marathon distance in kilometres?

Types: `float`

- A signed floating-point number:
  - for example, 1.5, -2.337, 6×10^{23}, 0.0 (note the decimal points);
  - for example, a number in a pocket calculator.
- Accurate to about 7 significant digits:
  - Max value is $3.40282347 * 10^{38}$;
  - Requires the same amount of storage as int.
- Contrast with real numbers in mathematics?
- Print with `printf("%f", floatVariable)`.
  - `%f` means "float"

Types: `double`

- A float with double precision.
- Accurate to about 15 significant digits:
  - Max value is $1.7976931348623157 * 10^{308}$;
  - Requires twice the storage space as float;
  - The computer has to work harder when computing with doubles;
  - Values may depend on your computer.
- Print with `printf("%lf", doubleVariable);`
  - The `%lf` meams “long float”

Choosing a Type

- `float`
  - For engineering calculations: eg, $3.0/2.0 = 1.5$;
  - When small inaccuracies is acceptable: 0.9999999 may be 1.0;
  - When speed is important.
- `double`
  - When more precision is required.
- `int`
  - For indexing, status codes, etc.
  - When inputting/outputting values which are *naturally* integer.
- *Speed* depends on hardware - `int` math is not necessarily faster!

marathon.c

```
#include <stdio.h>
#include <stdlib.h>

const float KILOMETRES_PER_MILE = 1.609;
const float YARDS_PER_MILE = 1760.0;

int main(void)  {
  int miles, yards;
  float kilometres;

  miles = 26; yards = 385;

  kilometres = (miles + yards/YARDS_PER_MILE)* KILOMETRES_PER_MILE;
  printf("%d miles and %d yards ", miles, yards);
  printf("equals %f kilometres.\n", kilometres);
  return EXIT_SUCCESS;
}
```

Mixing Types and casting

What happens when we divide a float by an int?

$3.0/2 = ?$

Sometimes this will work, sometimes not.
Safest option is to cast the integer into a float:

$3.0/(\text{float})2 = 1.50$

marathon1.c (explicit casting)

```
#include <stdio.h>
#include <stdlib.h>

const float KILOMETRES_PER_MILE = 1.609;
const float YARDS_PER_MILE = 1760.0;

int main(void)  {
  int miles, yards;
  float kiloms;

  miles = 26; yards = 385;

  kiloms = ((float)miles + (float)yards/YARDS_PER_MILE)* KILOMETRES_PER_MI
  printf("%d miles and %d yards ", miles, yards);
  printf("equals %f kilometres.\n", kilometres);
  return EXIT_SUCCESS;
}
```

Mathematical Operators in C

+	Addition.
−	Subtraction *or* negation.
`*`	Multiplication (don't use 'x').
/	Division - order is important here!
%	Integer remainder (eg, `5 % 3 = 2`). % is an *overloaded* symbol).
++	Increment (`x++` means `x = x+1`).
−−	Decrement (`x--` means `x = x-1`).
`sqrt`	Computes the square-root of its argument, returning a `double` - eg `sqrt(64.0)` returns `8.0`.

Quadratic equations

Consider any quadratic polynomial of the form $ax^2 + bx + c$.
We know this equation has exactly two *complex* roots (solutions to $ax^2 + bx + c = 0$) given by:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Suppose we want real roots ONLY.
Three cases:

- If $b^2 < 4ac$, there are **no** real solutions.
- If $b^2 = 4ac$, there is **one** real solution: $-b/(2a)$.
- If $b^2 > 4ac$, there are **two** different real solutions.

quadratic.c - attempt 1

```
/* Compute the two roots of a quadratic.  */
#include <stdio.h>
#include <stdlib.h>
#include <math.h> // Need to include math.h to use sqrt.

int main(void) {
  /* Vars for the 3 co-efficients, and for the roots we'll find.*/
  int a, b, c;
  double x1, x2;
  printf("Input the x^2 co-efficient a: ");
  scanf("%d", &a);
  printf("Input the x co-efficient b: ");
  scanf("%d", &b);
  printf("Input the constant term c: ");
  scanf("%d", &c);

  x1 = (-(double)b - sqrt((double)(b*b - 4*a*c)))/((double)(2*a));
  x2 = (-(double)b + sqrt((double)(b*b - 4*a*c)))/((double)(2*a));
  printf("The solutions to %dx^2 +%dx +%d = 0 are ", a, b, c);
  printf("%lf and %lf.\n", x1, x2 );
  return EXIT_SUCCESS;
}
```

Assumptions :-(

We made some HUGE assumptions for `quadratic.c`

(A1) We assumed that `sqrt((double)(b*b +4*a*c))` would return a value! But $\sqrt{b^2 - 4ac}$ is *complex* if $b^2 < 4ac$, and hence C's `sqrt` function is UNDEFINED in this case.

(A2) By solving a quadratic, we (implicitly) assumed a is non-zero.

SOLUTION - use the (general) `if` statement.

if statement - general form

```
if (<condition-1>)
  <statement-sequence-1>;

else if (<condition-2>)
  <statement-sequence-2>;

...
else
  <statement-sequence-n>;
```

- `<condition-1>`, . . ., `<condition-(n-1)>` are all boolean expressions.
- `<statement-sequence-1>`, . . ., `<statement-sequence-n>` are all sequences of C-programming statements.

Boolean operators

Assume `e1` and `e2` are (usually arithmetic) expressions . . .
We can apply boolean operators to form a boolean expression.

`e1 == e2`	`e1` equal to `e2`
`e1 != e2`	`e1` not equal to `e2`
`e1 < e2`	`e1` less than `e2`
`e1 <= e2`	`e1` less than or equal to `e2`
`e1 > e2`	`e1` greater than `e2`
`e1 >= e2`	`e1` greater than or equal to `e2`.

note: We can compare `float` expressions in this way - but `int` comparisons are *most reliable*.

More complicated Boolean expressions

Assume `e1` and `e2` are boolean expressions ...
Can build more complicated boolean expressions iteratively.

0	false (always)
non-zero	true (always)
`!e1`	true if e1 is false
`e1 && e2`	true if (e1 is true and e2 is true)
`e1 || e2`	true if (e1 is true or e2 is true)

The expressions `e1, e2` are (formally) integer expressions.
Can think of integers as (informally) acting as boolean "type".

quadratic1.c - general if statement

```
if (b*b > 4*a*c) {
  x1 = (-(double)b - sqrt((double)(b*b -4*a*c)))/((double)(2*a));
  x2 = (-(double)b + sqrt((double)(b*b -4*a*c)))/((double)(2*a));
  ....
  return EXIT_SUCCESS;
}
else if (b*b == 4*a*c) {
  x1 = -((double)b)/((double)(2*a));
  ....
  return EXIT_SUCCESS;
}
else {
  printf("No real solns to %dx^2 + %dx +%d = 0.\n", a, b, c);
  return EXIT_SUCCESS;
}
```

Nested `if`-statements

- The `<statement-sequence>` place-holder in the general `if`-statement allows other `if`-statements to be part of the program fragment.
- This is a “nested” use of the `if`-statement.
- Example - refine our `quadratic.c` program further.

quadratic.c - header and input code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> // Need to include math.h to use sqrt.

int main(void) {
  int a, b, c;
  double x1, x2;

  printf("Input the x^2 co-efficient a: ");
  scanf("%d", &a);
  printf("Input the x co-efficient b: ");
  scanf("%d", &b);
  printf("Input the constant term c: ");
  scanf("%d", &c);
```

quadratic.c - a≠0 case

```
if (b*b > 4*a*c) {
  x1 = (-(double)b - sqrt((double)(b*b -4*a*c)))/((double)(2*a));
  x2 = (-(double)b + sqrt((double)(b*b -4*a*c)))/((double)(2*a));
  ....
  return EXIT_SUCCESS;
}
else if (b*b == 4*a*c) {
  x1 = -((double)b)/((double)(2*a));
  ....
  return EXIT_SUCCESS;
}
else {
  printf("No real solns to %dx^2 + %dx +%d = 0.\n", a, b, c);
  return EXIT_SUCCESS;
}
```

`quadratic.c` - what if a=0

If $ax^2 + bx + c$ is a quadratic, and a is 0, then we have a linear equation:

$$bx + c.$$

This has ...

- Exactly *one* root of value $-c/(b)$, if $b \neq 0$.
- No root at all, if $b = 0$

Now incorporate this case into our code:

quadratic2.c - all cases

```
  if (a != 0) {
     if (b*b > 4*a*c) {
       x1 = (-(double)b - sqrt((double)(b*b -4*a*c)))/((double)(2*a));
       x2 = (-(double)b + sqrt((double)(b*b -4*a*c)))/((double)(2*a));
       ...
     }
     else if (b*b == 4*a*c) {
       x1 = -((double)b)/((double)(2*a));
       ....
     }
     else {
       printf("No real solns to %dx^2 + %dx +%d = 0.\n", a, b, c)
;
       return EXIT_SUCCESS;
     }
  }
  else if (b != 0) {
       x1 = -((double)c)/((double)(b));
       printf("1 real soln to %dx^2 +%dx +%d = 0.\n", a, b, c);
       printf("It is %lf.\n", x1);
       return EXIT_SUCCESS;
  }
```