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.

*CP1–6 – slide 2 – 4th October, 2010*
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 (∼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, $6 \times 10^{23}$, 0.0 (note the decimal points);
  - for example, a number in a pocket calculator.

- Accurate to about 7 significant digits:
  - Max value is $3.40282347 \times 10^{38}$;
  - 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 $1.7976931348623157 \times 10^{308}$;
  - 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` means “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!
```
#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/(float)2 = 1.50
#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_MILE;

    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).

\texttt{sqrt} Computes the square-root of its argument, returning a double - eg \texttt{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 \textit{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 \textbf{ONLY}.

Three cases:

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

\[
\begin{align*}
\text{if (\langle condition-1\rangle)} & \quad \langle\text{statement-sequence-1}\rangle; \\
\text{else if (\langle condition-2\rangle)} & \quad \langle\text{statement-sequence-2}\rangle; \\
\ldots & \\
\text{else} & \quad \langle\text{statement-sequence-n}\rangle;
\end{align*}
\]

- \langle condition-1\rangle, \ldots, \langle condition-(n-1)\rangle are all boolean expressions.
- \langle statement-sequence-1\rangle, \ldots, \langle statement-sequence-n\rangle are all sequences of C-programming statements.

\textit{CP1–6 – slide 16 – 4th October, 2010}
Boolean operators

Assume \( e_1 \) and \( e_2 \) are (usually arithmetic) expressions \ldots
We can apply boolean operators to form a boolean expression.

\[
\begin{align*}
    e_1 &= e_2 & \text{e1 equal to e2} \\
    e_1 &\neq e_2 & \text{e1 not equal to e2} \\
    e_1 &< e_2 & \text{e1 less than e2} \\
    e_1 &\leq e_2 & \text{e1 less than or equal to e2} \\
    e_1 &> e_2 & \text{e1 greater than e2} \\
    e_1 &\geq e_2 & \text{e1 greater than or equal to e2}.
\end{align*}
\]

\textbf{note:} We can compare float expressions in this way - but int comparisons are most reliable.
More complicated Boolean expressions

Assume \( e_1 \) and \( e_2 \) are boolean expressions . . .
Can build more complicated boolean expressions iteratively.

<table>
<thead>
<tr>
<th>0</th>
<th>false (always)</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-zero</td>
<td>true (always)</td>
</tr>
<tr>
<td>!( e_1 )</td>
<td>true if ( e_1 ) is false</td>
</tr>
<tr>
<td>( e_1 ) &amp; ( e_2 )</td>
<td>true if ( e_1 ) is true and ( e_2 ) is true</td>
</tr>
<tr>
<td>( e_1 )</td>
<td></td>
</tr>
</tbody>
</table>

The expressions \( e_1 \), \( e_2 \) are (formally) integer expressions.
Can think of integers as (informally) acting as boolean “type”.

*CP1–6 – slide 18 – 4th October, 2010*
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.
", 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.
#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);
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;
}
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;
}
else {
    printf("No real solns to %dx^2 + %dx +%d = 0.\n", a, b, c);
    return EXIT_SUCCESS;
}