Computer Programming: Skills & Concepts (INF-1-CP1) Variables; `scanf`; Conditional Execution

30th September, 2010

Tutorials

- Start next week.
- Tutorial groups can be viewed from the appropriate webpage:
  `https://www.inf.ed.ac.uk/admin/itodb/mgroups/stus/cp1.html`
- Contact the ITO if your tutorial group clashes with another lecture, or if you have not been assigned to any group (and are officially registered for CP1).

Summary of Lecture 4

- Integer arithmetic in C.
- Converting pre-decimal money to decimal.
- The `int` type and its operators.
- Numeric *variables*.

Today's lecture

- Assigning and Re-assigning variables;
- The `if`-statement.
- Fixing the `lsd` program.
- Input using `scanf`.

Reprise: Variables in C

Variables are "boxes" to store a value

- Bit like variables in mathematics (may have varying assignments);
- A C variable holds a single value;
- *Have to define what type of item a variable will hold*, eg:
  `int x;` or maybe `int x = 2;`
- In C, the value can change over time as a result of *program statements* which act on the variable, eg:
  `x = x + 1;`

Reprise: Updating Variables

```
int n;                          <-- n is defined

n = 2 * n;                      <-- n is doubled (from what? ERR
n = 9;                          <-- n gets the value 9
n = n + 1;                      <-- n gets the value 9+1, ie 10
n = 22 * n + 1;                 <-- n gets the value ?
++n;                            <-- n gets the value ?
n++;                            <-- n gets the value ?
```

The Assignment Statement

A variable is updated by an *assignment statement*

```
n = 22 * n + 1;
```

The left-hand side `n` is the variable being updated.
The right-hand side `22 * n + 1` is an *expression* for the new value.
First compute the expression, *then* change the variable to the new value.

The Assignment Statement

A variable is updated by an *assignment statement*

```
n = 22 * n + 1;
```

The left-hand side `n` is the variable being updated.
The right-hand side `22 * n + 1` is an *expression* for the new value.
First compute the expression, *then* change the variable to the new value.

WARNING: C also allows assignments as *expressions*:

```
(n = 22 * n + 1)
```

is an expression which computes `22 * n + 1`, sets `n` to the result, and overall computes to the new value of `n`.
So you can write:

```
m = (n = 2*n) + 3;
```

DON'T do this! You may see assignment expressions, but they are never necessary.
Main danger is doing it by accident!

Shorthand Assignment Operators

C programmers are lazy! C provides shorthand for some very common assignments, for example:

```
x += 7;          // same as   x = x + 7;
x *= 2;          // same as   x = x * 2;
x -= 3;          // same as   x = x - 3;
x /= 3;          // same as   x = x / 3;
```

Note that, e.g. `x *= y + z;` means `x = x * (y + z);`.

Use these only if you're completely confident with them.

Shorthand Assignment Expressions

For even greater laziness, C provides some special assignment *expressions*. Unlike general assignment expressions, these are very commonly used.

`n++`

is an expression which computes to the value of `n`, and **afterwards** increases `n` by 1.

```
int n = 2, m = 3;

n++;                    // n is now 3.
m = n++;                // m is now 3, n is now 4
```

Shorthand Assignment Expressions (2)

Similarly `n--` computes to value of `n` and **then** decreases `n` by 1.
Much less often you will see `++n` and `--n`:
first increase/decrease `n` by 1, and then compute to the new value of `n`.

Warning: Easy to get confused, and/or run into subtleties of C. Suggest using these only in **for**-loops etc. (See later.)

`if` statement – basic form

```
if ( ⟨condition⟩ ) {
    ⟨statement-sequence⟩
}
else {
    ⟨statement-sequence⟩
}
```

- Allows two different strands of execution, depending on the result of evaluating ⟨*condition*⟩.
- ⟨*condition*⟩ is any boolean expression.
- ⟨*statement-sequence*⟩ is any legal sequence of C statements.
- The `else {... }` is optional.

MAX of two integer variables

```
if (x > y) {
   printf("MAX is %d: ", x);
} else {
   printf("MAX is %d: ", y);
}
```

- `(x > y)` is the condition to be evaluated. It evaluates to True only if `x` is larger than `y`.
- *where did we get the values* `x` *and* `y`?

Conditions on integers

C has the standard mathematical relations <, >, ==, <=, >=.
Remember that 'is equal to' == is a double equals sign!

Examples:

```
a < 0                  // a is negative
a == 2*b
a + c >= b
x % 6 == 0             // x is a multiple of 6
```

Fixing the old money $\longrightarrow$ new money calculation

We did (this year: should have done)

```
totaloldpence = oldpence + shillings * OLD_PENCE_PER_SHILLING;
newpence = ( totaloldpence * NEW_PENCE_PER_POUND )
                / OLD_PENCE_PER_POUND;
```

Probably we don't like the rounding:
2 old pence converts to $(2 * 100)/240 = 0$ in integers.
But 2d is really $\frac{5}{6}$p, so we should round to 1p.

Standard rounding is round $\frac{1}{2}$ or greater up, less than $\frac{1}{2}$ down.
We can add the lines

```
if ( ( totaloldpence * NEW_PENCE_PER_POUND ) % OLD_PENCE_PER_POUND
    >= (OLD_PENCE_PER_POUND/2) ) {
    newpence += 1;
}
```

Exercise: do the same without using `if`.

Harder exercise: what hidden assumption have I made above?

Fixing the printing of new pence

We did:

```
printf("is %d.%d in new money\n",pounds,newpence);
```

But this prints 4 pounds and 1 penny as 4.1, not 4.01. Fix:

```
printf(" is %d.");
if ( newpence < 10 ) {
    printf("0%d",newpence);
else {
    printf("%d",newpence);
}
printf(" in new money\n");
```

Fixing the printing of new pence

We did:

```
printf("is %d.%d in new money\n",pounds,newpence);
```

But this prints 4 pounds and 1 penny as 4.1, not 4.01. Fix:

```
printf(" is %d.");
if ( newpence < 10 ) {
    printf("0%d",newpence);
else {
    printf("%d",newpence);
}
printf(" in new money\n");
```

Actually, there's an easier way, with fancier features of printf.

```
printf("is %d.%02d is new money\n",pounds,newpence);
```

Input with `scanf`

`scanf` is the twin of `printf`. Reads numbers from input and stores them in variables.

But `scanf` requires a "`&`" before its arguments.
(Explanation later in the course...)

For example:

```
int x;
scanf("%d", &x);
printf("%d", x);
```

max.c

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

int main(void) {
   int x, y;
   printf("Input the two integers: ");
   scanf("%d", &x);
   scanf("%d", &y);
   if (x > y) {
      printf("MAX is %d: ", x);
   } else {
      printf("MAX is %d: ", y);
   }
   return EXIT_SUCCESS;
}
```