Computer Programming: Skills & Concepts (INF-1-CP1) The C Programming Language: 2

28th September, 2010

Tutorials

- Start in week 3 (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 3

- *Edit* → *Compile* → *Run* cycle.
- "Hello World" example.
- Mistakes.

`printf`

- To output text to the screen: (`\n` means 'newline'):
  ```
  printf("This text will be output\n");
  ```
- To write out a variable:
  ```
  printf("The number is %d \n",number);
  ```
  `%d` is a placeholder meaning "print the next argument here"
  `%` introduces placeholders, `d` means "print an integer in **d**ecimal"
- To write several numbers, use several placeholders in order:
  ```
  printf("x is %d, and y is %d\n", x, y);
  ```

Overview

- Maths in C.
- Basic numeric types: `double` and `int`.
- Numeric *variables*.
- Common problems.

Today's problem

Convert pre-decimal British money to decimal

We know:

- The number of old pence in a shilling (12) and old pence in a pound (240).
- The number of new pence in a pound (100).

How to compute £4 7/8 in decimal?

Always do financial arithmetic with integers!

C program

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

const int OLD_PENCE_PER_SHILLING = 12;
const int OLD_PENCE_PER_POUND = 240;
const int NEW_PENCE_PER_POUND = 100;

int main(void)  {
    int pounds, shillings, oldpence, newpence;

    pounds = 4; shillings = 7; oldpence = 8;

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

    printf("%d %d/%d in old money ", pounds, shillings, oldpence);
    printf("is %d.%d in new money.\n", pounds, newpence);
    return EXIT_SUCCESS;
}
```


Integer arithmetic in C

Why did we write

```
newpence = ( oldpence * NEW_PENCE_PER_POUND ) / OLD_PENCE_PER_POUND;
```

instead of

```
newpence = oldpence * ( NEW_PENCE_PER_POUND / OLD_PENCE_PER_POUND );
```

Integer arithmetic is all integer – no fractions!
$(92 * 100)/240 = 9200/240 = 38$, but
$92 * (100/240) = 92 * 0 = 0$
Very common mistake – watch for it.

The `int` type in C

- An integer (whole number):
  - for example, $1, 2, -16000, 0$;
- 2^32 possible values $\{-2^{31}, \ldots, 2^{31} - 1\}$:
  - Some types of computer are more limited;
  - $2^{31} = 2,147,483,648$.
- Fully accurate within this range;
- Often used in indexing and status codes;
- Print with `printf("%d", integerVariable)`.
- Arithmetic operations:
  - plus: $12 + 7 = 19$
  - minus: $12 - 7 = 5$
  - times: $12 * 7 = 84$
  - divides: $12 / 7 = 1$ (**integer** division!)
  - remainder: $12 \% 7 = 5$ (N.B. $x = (x / y) * y + (x \% y)$ always.)

Precedence (of arithmetic operators)

```
oldpence = oldpence + shillings * OLD_PENCE_PER_SHILLING;
```
Means
```
oldpence = oldpence + ( shillings * OLD_PENCE_PER_SHILLING );
```
Not
```
oldpence = ( oldpence + shillings ) * OLD_PENCE_PER_SHILLING;
```

Precedence-based evaluation

- Multiplication (*), division (/) and remainder (%) are evaluated *before* addition (+) and subtraction (−).
- Use parentheses to force an evaluation order
- If in any doubt, USE PARENTHESES! *or just use them all the time!*

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 `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;`

VITAL TO REMEMBER: In C, a single equals sign = *always* means 'gets set to'; it *never* means 'is equal to'. **Beware** when people are mixing mathematical notation and C notation.
With `gcc -Wall`, the compiler will warn you any time it sees an = where it thinks you probably meant 'is equal to' (==), but it's not telepathic.

Updating Variables

```
int n;                      <-- n is declared as int

n = 2 * n;                  <-- n is doubled (from what? ERROR)
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 ?
```

Swapping Values

Aim: Swap the values of `x` and `y`

```
int x = 5;
int y = 10;

x = y;
y = x;
```


Swapping Values (Wrong)

Aim: Swap the values of `x` and `y`

```
int x = 5;
int y = 10;

x = y;
y = x;
```


Swapping Values (Correct)

```
int x = 5;
int y = 10;
int temp;

temp = x;
x = y;
y = temp;
```

We used an *auxiliary* variable ("box") to temporarily store `x`

Variable Names (Identifiers)

- Can be a letter, underscore, or a digit
- BUT first character CANNOT be a digit!
- See section 2.2 and 2.5 of "A Book on C"

OK: `EXIT_SUCCESS`, `Celsius`, `t0`, `n`.

Not OK: `hyper-modern`, `J@inf`, `4tet`.

Identifiers in Practice

- Use meaningful names
- (maybe) follow some convention:
  - `FunctionNames`
  - `variableNames`
  - `CONSTANT_VALUES`
- The particular convention is not so important
  ... But one convention per program please!
  If you're modifying someone else's program, follow *their* convention, even if it's silly.

C program again

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

const int OLD_PENCE_PER_SHILLING = 12;
const int OLD_PENCE_PER_POUND = 240;
const int NEW_PENCE_PER_POUND = 100;

int main(void)  {
    int pounds, shillings, oldpence, newpence;

    pounds = 4; shillings = 7; oldpence = 8;

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

    printf("%d %d/%d in old money ", pounds, shillings, oldpence);
    printf("is %d.%d in new money.\n", pounds, newpence);
    return EXIT_SUCCESS;
}
```


Type Modifiers: const

`const` tells the compiler

"this variable should never change"

```
const int OLD_PENCE_PER_SHILLING = 12;
```

const variables *must* be assigned at declaration ...
the = is mandatory

Why use const variables?

- To avoid mistakes typing the same number over and over.
- To make the program easier to read.
- Because some constants are not so constant ...

Questions