COMP1511: Introduction To C

Session 2, 2018

Variables

Variables are used to store a value (like “boxes” to hold values)

- The value a variable holds may change over its lifetime.
- At any point in time a variable stores one value (except quantum computers!)

C variables have a type,

- `int` for integer values, e.g.: 42, -10
- `float` for decimal numbers 3.14159, 2.71828 (single *precision*)
- `double` for decimal numbers 3.14159, 2.71828 (double *precision*)
- `char` for a character like ‘a’, ‘b’, ‘2’, ‘+’, ‘#’, etc.
- `Char *` for a string “Sydney” (more on this later)

Variables

Declare: The first time a variable is mentioned, we need to specify its type.

Initialise: Before using a variable we need to assign it a value.

```
// Declare
int answer;

// Initialise
answer = 42;

// Use
printf("Value is %d", num);
```

```
// Declare and Initialise
int answer = 42;

// Use
printf("Value is %d", num);
```

Variable Names (and other Identifiers)

- Variable names can made up of letters, digits and underscores
- Use a lower case letter to start your variable names
- Beware variable names are case sensitive,

  e.g. **hello** and **hEllo** are different names.
- Beware certain words can't be used as variable names:

  e.g.: **if**, **while**, **return**, **int**, **double**
- These keywords have special meanings in C programs.
- You'll learn what many of them are as we go on.

Output using printf()

- No variables:

```
printf("Hello World\n");
```

- A single variable:

```
int num = 5;
printf("num is %d\n", num);
```

- More than one variable:

```
int j = 5;
int k = 17;
printf("j is %d and k is %d\n", j, k);
```

Input using scanf()

- **scanf** uses a format string like **printf**.
- Notice **&** before the variable name.

Use **%d** to read an **int** (integer) value

```
int answer;
printf("Enter the answer: ");
scanf("%d", &answer);
```

Use **%f** to read a **float** value

```
float answer;
printf("Enter the answer: ");
scanf("%f", &answer);
```

Use **%lf** to read a **double** value

```
double weight;
printf("Enter weight: ");
scanf("%lf", &weight);
```

scanf() - read multiple values

Multiple variables (**space separated**):

```
int num1 = 0;
int num2 = 0;

printf("Enter two numbers, separated by a space: ");
scanf("%d %d", &num1, &num2);
printf("num1 = %d and num2 = %d\n", num1, num2);
```

Multiple variables (**comma separated**):

```
int num1 = 0;
int num2 = 0;

printf("Enter two numbers, separated by a space: ");
scanf("%d, %d", &num1, &num2);
printf("num1 = %d and num2 = %d\n", num1, num2);
```

scanf()

- scanf **returns number** of items successfully read
- **Use return value** of scanf to determine if scanf successfully read value(s)
- scanf **could fail** e.g. if the user enters letters instead of say expected numbers
- **OK for now** to assume scanf succeeds
- Good programmers **always check**! We will do it too later

Giving Constants Names

- It can be useful to give constants (numbers) a name.
- It often makes your program more readable.
- It can make your program easier to update particularly if the constant appears in many places
- One method is **#define** statement e.g.

```
#define   SPEED_OF_LIGHT   299792458.0
```

- **#define** statements go at the top of your program after **#include** statements
- **#define** names should be **all capital letters + underscore**

Division in C

- C division does what you expect if either operand is a **double**
  If either operand is a **double** the result is a **double** .
  $2.6/2 \implies 1.3$ (not 2!)
- C division may not do what you expect if both arguments are integers.
- The result of dividing 2 integers in C is an integer.
- The fractional part is discarded (not rounded!).
  $5/3 \implies 1$ (not 2!)
- C also has the **%** operator (integers only).
  computes the modulo (remainder after division)
  $14 \% 3 \implies 2$

Selection - Conditional Execution

- many problems require executing statements only in some circumstances
  e.g read two integers and print largest one

- sometimes called **control flow**, **branching** or **conditional execution**

- The C `if` Statement can do this.

The `if` Statement

```
if (expression) {
    statement1;
    statement2;
    ....
}
```

- **statement1, statement2, ...** are executed
  if **expression** is non-zero.
- **statement1, statement2, ...** are **NOT** executed
  if **expression** is zero.
- There is no "boolean" type in C.
  0 is regarded as "FALSE"
  anything non-zero is regarded as "TRUE"

The `if` … `else if` … Statement

Multiple **if** statements can be chained together:

```
int a, b;

printf("Please enter two numbers, a and b: ");
scanf("%d %d", &a, &b);

if (a > b) {
   printf("a is greater than b\n");
} else if (a < b) {
   printf("a is less than b\n");
} else {
   printf("a is equal to b\n");
}
```

Relational Operators

C has the usual operators to compare numbers:

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

- Be careful comparing doubles for equality using `==` or `!=`
- Remember doubles are approximations.

Relational Operators

- Many languages have a separate type for true & false.
- C just uses numbers.
- C convention is zero is false, other numbers true.
- relational operators return:
  the int **0** for false
  the int **1** for true
- For example:

```
5 > 4    ↦  1
5 >= 4   ↦  1
5 < 4    ↦  0
5 <= 4   ↦  0
5 != 4   ↦  1
5 == 4   ↦  0
```

Logical Operators

- C has logical operators:
  - **&&** for AND
  - **||** for OR
  - **!** for NOT

- Logical operators allow us to combine comparisons, eg:
  mark > 0 **&&** mark < 100
  mark < 0 **||** mark > 100

- logical operators return:
  - the int **0 for false**
  - the int **1 for true**

Logical Operators

- && is the **and** operator - true if both operands are true
  2 > 0 && 2 < 10 results in 1 && 1 results in 1 that is true

- || is the **or** operator - true if either operand is true
  24 > 42 || 2 < 10 results in 0 || 1 results in 1 that is true

- **!** is the **not** operator - true iff its operands is false
  ! (24 > 42) results in **!**0 results in 1

Logical Operators - Conditional evaluation

- The C operator && || have a useful property.
- They always evaluate their left-hand side first.
- They only evaluate their right-hand side if needed.
- && will not evaluate right-hand side if left-hand side is false (zero).
- || will not evaluate right-hand side if left-hand side is true (non-zero).
- For example we can write

```
x != 0 && y/x > 2
```

without risking division by zero.

De Morgan's Laws

- De Morgan's Laws are important because they help us to make logical reasoning. For example,

  NOT (A **OR** B) is equivalent to (**NOT** A) **AND** (**NOT** B)

  NOT (A **AND** B) is equivalent to (**NOT** A) **OR** (**NOT** B)

- By De Morgan's Law, the following are **equivalent in C**:
  - **!** (x > 50 **||** Y > 50)
  - **!**(x > 50) **&& !**(y > 50)
  - (x <= 50) **&&** (y <= 50)

De Morgan's Laws - Example

```
if (!(height <= 130 && width <= 240)) {
    printf("Envelope too large!\n");
}
```

.. is the same as ..

```
if (height > 130 || width > 240) {
    printf("Envelope too large!\n");
}
```

Common mistakes

In English, you say

x is not equal to 6, 7 or 8

In C (and many programming languages), you need to write

```
(x != 6) || (x != 7) || (x != 8)
```

`If` Example : Driving, Take 1

Write a program which asks the user to enter their age.

If they are at least 16 years old,

then, display “You can drive.”

Then, whether or not they can drive,

display “Have a nice day.”

`If` Example : Driving, Take 1

Write a program which asks the user to enter their age. If they are at least 16 years old, then, display “You can drive.” Then, whether or not they can drive, display “Have a nice day.”

If they are at least 16 years old, **then**, display “You can drive.”

Then, whether or not they can drive, display “Have a nice day.”

… Print “How old are you?”
… Read in their age.
… If their age is ≥ 16: print “You can drive”.
… Print “Have a nice day.”

```c
// Print out whether somebody can drive
// Andrew Bennett
// 2018-02-28

#include <stdio.h>

int main(void) {

    // ask their age
    int age;
    printf("What is your age?\n");
    scanf("%d", &age);

    // if they are 16+ print you can drive
    if (age >= 16) {
        printf("You can drive!\n");
        printf("you are over 16!\n");
    } else {
        printf("You are not allowed to drive!!! go back to school\n");
    }

    // print have a nice day
    printf("Have a nice day!\n");

    return 0;
}
```

Programming Style - Style Guide

- **Style Guide** is available on the class webpage, see the left panel on the class webpage.
- You should take time to read the Style Guide carefully, whenever new syntax is introduced in lectures.
- You should come back and read the guide as you learn new syntax and concepts in lectures, so don’t worry if you don’t understand everything in one go!

Programming Style - Style Guide

- Is it easy to understand the following code?
- What if there is a bug in the code and you need to address the bug?

```
#include <stdio.h>
int main(void) {
int x; printf("What is your x?\n"); scanf("%d", &x); if (x >= 16)
{printf("You can drive!\n"); printf("you are over 16!\n");
} else { printf("You are not allowed to drive!!! go back to school\n");}
printf("Have a nice day!\n");return 0;
}
```