COMP1511/1911 Programming Fundamentals

Week 1 Lecture 2

Variables and Constants

Last Lecture

- Welcomes and Introductions
- How COMP1511/COMP1911 works
- How to get help
- What is programming?
- Working in Linux
- A first look into C
  - printf

Today's Lecture

- Memory and how we store data
  - Types and variables
- Printing out and reading in data
- Arithmetic Operators and Expressions
- Constants

Link to Week 1 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/24T3/live/week_1/

A Brief Recap: Our First Program

```c
// A program showing how to print output in C
// The first of many C programs you will C

#include <stdio.h>

int main(void) {
    printf("Hello COMP1511 and COMP1911\n");
    return 0;
}
```

Quick Question: What will this print out?

```c
// A tricky example with escape characters
// Warning: this may hurt your brain

#include <stdio.h>

int main(void) {
    printf("\\\"\\\n");
    return 0;
}
```

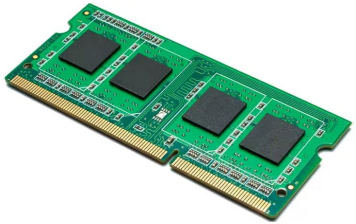
Basics of Computer Hardware

CPU:

- processes and executes our instructions
- performs arithmetic etc

RAM:

- Stores the instructions and the data we need
- What we call **memory** in this course



Hard Drive/Solid State Drive:

- Persistent storage of data e.g files

How Do Computers Store Data?

Computers store everything in binary : 0s and 1s

Why?

- Computer memory is a large number of on-off switches
- We use 0 and 1 to represent the off and on states
- We call these bits

We often collect these together into bunches of 8 bits

- We call these bytes

How can we use memory in our programs?

Variables

- A **name** for a piece of memory
- Can store a specific **type** of data
- Has a specific size (number of bytes)
- Called a **variable** as we can change what is stored in there!

Primitive Types

We will start out with 3 common primitive types

Type	Description	Examples
`int`	Integers (whole numbers)	1, 0, 999, -42
`char`	Individual characters	‘A’, ‘a’ ,‘?’
`double`	Floating point numbers	3.14159, -0.001

Declaring a variable

- Declaring a variable tells C to set aside a chunk of memory for the variable.
- We only need to do this once for each variable.
- To declare a variable, you use the syntax:

```
type name;
```

- E.g. the following declares a variable named age, of type int

```
int age;
```

Assigning values to Variable

- Before using a variable, we need to give it an initial value
  - Until then, it contains garbage values
- We use = (the assignment operator) to set values in variables

```
// Declare a variable
int my_age;
// Initialise the variable
my_age = 21;
```

```
// Declare and initialise
// a variable in one step
int my_age = 21;
```

Variable Names

- Should describe what the variable is storing
  - e.g. “age”, “radius”
  - rather than “a”, “b”
- C is case sensitive:
  - “ansWer” and “answer” are two different variable names
- We always use lower case letters to start our variable names
- We use snake_case
  - We can split words with underscores: E.g. “long_answer”
- C reserves some words
  - E.g. “return” , “int” and “double” can’t be used as variable names

Variable Names are Important

- Variable names are an important part of programming style
- We name variables to make it obvious what we are storing
- This makes our code more readable for
  - ourselves
  - others such as colleagues
  - and in your case, your tutors!
- We have a style guide for the course which you should follow
  https://cgi.cse.unsw.edu.au/~cs1511/24T3/resources/style_guide.html

`int` data type

- We can represent integers with the type **`int`**
  - whole number, with no fractions or decimals places
- Most commonly uses 32 bits (4 bytes)
  - This gives us exactly 2^{32} different possible values
- The maximum is very large, but it's not infinite!
  - Exact ranges from -2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31} - 1$)
  - Hmmm, what could possibly go wrong with this?

`char` data type

- We use **`char`** to store single characters
- The syntax is to put it in single quotes: **`'a'`**
- They are really integers under the hood
  - **`char`** stores a small integer
  - Usually 8 bits (1 byte)
  - Guaranteed to be able to store integers 0..127
- When we assign **`'a'`** to a char variable, it really stores the ASCII code 97.
- Type **`ascii`** on the command line to see ASCII codes.

`double` data type

- We use **`double`** to store Real numbers
  - can only represent a subset of all possible Real numbers
- Size is 64 bits (8 bytes)
- Warning: **`double`** are approximations and may not be exact!
  - Hmmm, what could possibly go wrong with this?

Coding with Variables

```c
int main(void) {
    // Declare a variable
    int my_age;
    // initialise a variable
    my_age = 21;

    // We can modify variable values
    my_age = 25;

    // We can also declare and initialise in same line
    double radius = 3.5;
    char grade = 'B';
    return 0;
}
```

How can we print out the values in our variables?

Printing out variables with printf

A format specifier is a `%` symbol followed by some characters to let the compiler know:

- what data type you want to print
- where to print the value

Type	Format Specifier
`int`	`%d`
`double`	`%lf`
`char`	`%c`

After the comma you put the variable name/s you want to print

```c
int my_age = 21;
printf("My age is %d\n", my_age);
```

Printing out many variables with printf

- The variables must match the symbols in the same order as they appear!
- You can have as many as you want and of different types also!

```c
int height = 21;
double radius = 3.5;
printf("Height is %d and radius is %lf\n", height, radius);
```

```c
char letter = 'A';
printf("The letter %c has ASCII value %d\n", letter, letter);
```

Code Demo

print_variables.c

print_errors.c

Quick Break

How can we read in input from the user?

Reading input with scanf

- Uses a similar format to printf
- Format specifiers `%d`, `%lf`, `%c` are used in the same way
- Difference is we need to use `&` before each variable
  - The `&` symbol tells scanf the address of the variable in memory (where the variable is located) so it knows where to store the value
- e.g. Reading in an integer

```
int age;
scanf("%d", &age);
```

Example scanf code

```c
#include <stdio.h>
int main(void) {
    char initial;
    printf("Please enter your first initial: ");
    scanf("%c", &initial);

    int age;
    printf("Please enter your age: ");
    scanf("%d", &age);

    double height;
    printf("Please enter your height in cm: ");
    scanf("%lf", &height);
    return 0;
}
```

Code Demo

scan_variables.c

scanf_confusion.c

scanf_magic.c

`scanf` magical tips and trips

- scanning an int ignores whitespace
  - `scanf("%d", &number);`
- scanning a char does not ignore whitespace
  - `scanf("%c", &character);`
  - This is good as sometimes we want to be able to read in spaces and newline characters as they are actually characters!
- We can ignore leading whitespace when working with chars with the following trick: (note the space before the `%c`)

```
scanf(" %c", &character);
```

Mathematical Expressions in C

- Arithmetic operators will look familiar!
- Warning: Division may not always give you what you expect…
  - Result depends on whether dividing integer types or doubles
- Modulus gives the remainder
  - integer types only

Operator	
+	addition
-	subtraction
*	multiplication
/	division
%	modulus

Mathematical Expressions in C

- **Precedence** is what you would expect from maths e.g.
  - a + b * c + d / e is the same as
  - a + (b * c) + (d / e)
- **Association** is what you would expect from maths e.g.
  - a - b + c is the same as
  - (a - b) + c
- We can also use brackets to force precedence e.g.
  - (a + b) * c

Precedence and Associativity

Precedence: Operators with higher precedence are executed before those with lower precedence
Associativity: The direction in which operators of the same precedence level are evaluated in an expression.

Operator	Associativity
* / %	left to right
+ -	left to right

https://cgi.cse.unsw.edu.au/~cs1511/24T3/resources/c-reference-sheet.pdf

Example Arithmetic Expressions Code

What do you think these will these print?

```c
int x = 4;
int y = 3;
int z = (x + y) * 10 - x;
printf("%d\n", z);
```

```c
int x = 3;
int y = 2;
int z = x / y;
printf("%d\n", z);
```

```c
char c1 = 'a';
char c2 = c1 + 1;
printf("%c\n", c2);
```

```c
int x = 3;
int y = 2;
double w = x / y;
printf("%lf\n", w);
```

Doing maths with char

- Characters are represented as integers
- You can add or subtract to get different ASCII values
- For example, you can add 1 to ‘a’ and get ‘b’ or add 2 to ‘a’ and get ‘c’ or subtract 1 from ‘e’ and get ‘d’

```c
char letter = 'e';
letter = letter - 1;
//This will print out 'd'
printf("%c\n", letter);
```

More about division

- If either operand is a **`double`** then the result is a **`double`**
  - `2.6/2` gives `1.3`
- If both operands are **`int`** then then result is an **`int`**
  - This is integer division and can be surprising when you first see it
  - `3/2` gives `1` not `1.5`
  - What would `1/2` be?
  - What would `1.0/2` be?
- You can do this to cast an **`int`** to the type of **`double`**
  - `1/(double)2;`
  - Note: Please only use casts between ints and doubles in this course

More about division and modulus

- `%` is called Modulus (or mod).
  - It will give us the remainder from a division between integers e.g.
  - `5 % 3` gives `2` (because 5/3 gives 1 remainder 2)
- We can tell if a number is even by checking the remainder when dividing by 2 e.g.
  - 10 % 2 is 0
  - 7 % 2 is 1
  - 4 % 2 is 0
  - 3 % 2 is 1

Double Division and Mod Examples

What will these print?

```c
double x = 5;
double y = 2;
int z = x / y;
printf("%d\n", z);
```

```c
int x = 5;
double y = 2;
double z = x / y;
printf("%.1lf\n", z);
```

```c
double x = 5;
double y = 2;
double z = x / y;
printf("%.1lf\n", z);
```

```c
int x = 5;
int y = 2;
int z = x % y;
printf("%d\n", z);
```

double precision

- There is no such thing as infinite precision
- We can’t precisely encode a simple number like 1/3
- If we divide 1.0 by 3.0, we'll get an approximation of 1/3
- The effect of approximation can compound the more operations you perform on them

Integer Overflow

- What happens if we take the largest int and add 1?
  - It can wrap around to the minimum value and give us smallest negative number
- What happens if we take the smallest int and subtract 1?
  - It can wrap around to the maximum value and give us largest positive number
- Doing maths on ints and going over the limits is called overflow
  - `dcc` helps us by giving us runtime errors when this happens with ints
  - This is better than it wrapping around and giving us hard to debug incorrect answers

Integer overflow disasters

- Boeing 787 that had to be rebooted every 248 days
  - 248 days is 2^{31} 100ths of a second

https://www.engadget.com/2015-05-01-boeing-787-dreamliner-software-bug.html

Integer overflow disasters

- A simple integer overflow error also caused the Ariane 5 rocket explosion
- The (different kind of) explosion of the video “Gangham Style” on YouTube maxed out the views counter. They have changed it to an 8 byte integer now.

https://www.bbc.com/future/article/20150505-the-numbers-that-lead-to-disaster

Constants

- Constants are like variables, only they never change!
- We use **`#define`** and follow it with the name of the constant and its value
- Style Guide: we use UPPERCASE so it is easy to recognise they are constants

```c
#include <stdio.h>
// Define constants after your #includes
// but before your main
#define MAX_SIZE 12
#define PI 3.1415
#define MEANING_OF_LIFE 42
```

Coding Time

Write a program **`convert.c`** that

- prompts the user enter the number of hours
- calculates how many minutes that is equivalent to
- prints out the number of minutes

See sample output below:

```
$ ./convert
Please enter the number of hours: 2.5
That is 150.00 minutes
```

Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.

https://forms.office.com/r/maiuL3wEkq

What did we learn today?

- Recap of escape characters: escaping.c
- Variables and types: int double char
- Printing variables using printf
  - print_variables.c, print_errors.c
- Reading values into variables using scanf
  - scan_variables.c, scanf_confusion.c, scanf_magic.c
- Creating arithmetic expressions (doing maths) with variables
  - expression_examples.c, tricky_expressions.c, type_troubles.c
- Defining constants in C
  - convert.c

Reach Out

Content Related Questions:
Forum

Admin related Questions email:
cs1511@unsw.edu.au