COMP1511/1911 Programming Fundamentals

Week 1 Lecture 2

Variables and Constants

Last Lecture

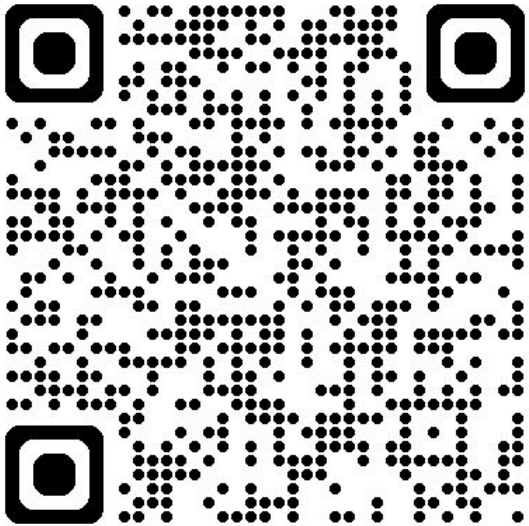
- Welcomes and Introductions
- How COMP1511/COMP1911 works
- Intro to Linux
- Compiling and running C code in Linux
  - printf
  - escape character \

Today's Lecture

- How can we **store data** in our programs
  - Variables, Types and Memory
- How can we **read in data** from users
  - printf and scanf
- How can we do **calculations** on our data
  - Arithmetic operators, expressions, constants

Link to Week 1 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/26T3/code/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;
}
```

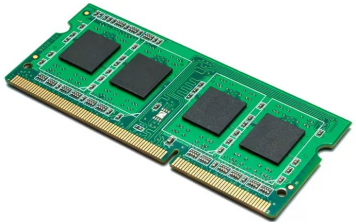
Basics of Computer Hardware

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 Variables

- 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 age;
// Initialise the variable
age = 21;
```

```
// Declare and initialise
// a variable in one step
int 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 **reserve**s some words
  - E.g. “return” , “int” and “double” can’t be used as variable names

Variable Names and Style

- 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/~cs1511/26T3/resources/style_guide.html

`int` data type

What `int` stores

- Whole numbers
- No fractional or decimal part

Typical size

- 32 bits (4 bytes)
- **2^32** possible values

Typical 32-bit range

-2,147,483,648
to
2,147,483,647

The range is large, but finite.

`char` data type

What `char` stores

- A single character
- Write the value in single quotes

```
char letter = 'a';
```

Under the Hood

- `char` is an integer type
- Usually 8 bits (1 byte)
- 'a' stores the ASCII code 97

Try printing the ASCII table

```
$ ascii -d
```

`double` data type

What `double` stores

- Numbers with fractional parts
- Typically 64 bits (8 bytes)

```
double height = 173.5;
```

Finite precision

- Only a subset of real numbers can be represented
- `double` values are approximations and may not be exact!

Coding with Variables

```c
int main(void) {
    // Declare a variable
    int age;

    // initialise a variable
    age = 21;

    // We can modify variable values
    age = 22;

    // We can also declare and initialise in same line
    double height = 173.5;
    char initial = 'B';
    return 0;
}
```

How can we print out the values in our variables?

Printing out variables with printf

- A format specifier starts with **%**
- It tells **printf** what type of value to print
- Put the variables after the format string

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

```c
int age = 21;

printf("My age is %d\n", age);
```

Printing out many variables with printf

- Each format specifier matches a variable after the comma
- They must appear in the **same order**

```c
int age = 21;
double height = 173.5;
printf("Age is %d and height is %lf cm\n", age, height);
```

```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. Back Soon…

How can we read in input from the user?

Reading input with scanf

- Uses format specifiers just like printf
  - `%d`, `%lf`, `%c`
- Difference is you put `&` before each variable
  - tells scanf where in memory to store the value

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

scanf demo

```c
#include <stdio.h>
int main(void) {
    int gold;
    int level;

    scanf("%d", &gold);
    scanf("%d", &level);

    printf("%d %d\n",
           gold, level);

    return 0;
}
```

What will the output be?

```
100
90
```

```
100          90 78 1000 hello
```

```
100
99.1231
```

```
100.5
99.7
```

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` tips and tricks

Reading an int

Whitespace is ignored.

```
scanf("%d", &number);
```

Reading a char

Whitespace is not ignored.
Spaces and \n are characters too.

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

Ignoring whitespace before a char

The space before %c skips leading whitespace.

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

Mathematical expressions in C

Operator	Meaning
+	addition
-	subtraction
*	multiplication
/	division
%	modulus

Division

- **int / int** gives an integer result
- If either value is a **double**, the result is a double

Modulus (%)

- Gives the remainder after division
- Works with integer types only

Precedence and Associativity

Precedence: which operators are evaluated first

Associativity: the evaluation direction when operators have the same precedence

Operators	Precedence	Associativity
* / %	higher	left to right
+ -	lower	left to right

See for full table: c-reference-sheet.pdf

Mathematical Expressions in C

- **Precedence**
  - a + b * c + d / e is the same as
  - a + (b * c) + (d / e)
- **Association**
  - a - b + c is the same as
  - (a - b) + c
- We can also use brackets to force precedence
  - (a + b) * c

Arithmetic expression examples

What will each program print?

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

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

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);
```

Integer Division Examples

What will each program print?

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

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

More about division

Integer division

- Both operands are **int**
- The result is an **int**
- The fractional part is discarded

```
3 / 2 gives 1
1 / 2 gives 0
```

Double division

- At least one operand is a **double**
- The result is a **double**

```
2.6 / 2 gives 1.3
1 / 2.0 gives 0.5
```

(double)1 / 2 casts one operand before dividing, so the result is **0.5**

More about modulus

What % gives us

- % gives the remainder after division
- Both operands must use integer types

5 % 3 gives 2

because 5 divided by 3 leaves a remainder of 2

Useful checks

Divisible

15 % 5 gives 0

A remainder of 0 means it divides exactly.

Even or odd

10 % 2 gives 0 even
7 % 2 gives 1 odd
-7 % 2 gives -1 odd

Division and Modulus Examples

What will each program print?

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

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

Division by zero with int

Division or mod by 0 gives a runtime error

```c
int x = 5;
int y = 0;
int z = x / y;
printf("%d\n", z);
```

```c
int x = 5;
int y = 0;
int z = x % y;
printf("%d\n", z);
```

double precision

- `double` values have finite precision
- `double` values are approximations
- Some values including 1.0/3.0 or even 0.1 cannot be stored exactly
- Small errors can grow across repeated operations
- Interesting fact: Dividing by 0 with `double` values gives **inf** (infinity)

Integer Overflow

- What happens if we take the largest **int** and add 1?
  - It can wrap around to the **minimum possible** int which is a large negative number.
- What happens if we take the smallest **int** and subtract 1?
  - It can wrap around to the **maximum possible** int which is a large positive number.
- Going outside the range of values an **int** can store is called **integer overflow**
  - `dcc` helps us by giving us a runtime error when this happens
  - This is better than silently producing an incorrect value

Integer overflow disasters

Boeing 787

- Needed rebooting every 248 days
- A signed 32-bit counter tracked time in hundredths of a second
- **2^31** hundredths of a second is 248 days

Source: Engadget, 2015

More Integer Overflow Failures

Ariane 5

An overflow during a number conversion caused the guidance system to fail and the rocket to self-destruct.

YouTube view counter

Gangnam Style passed the limit of a signed 32-bit counter. YouTube replaced it with a larger counter.

Source: BBC Future

Constants

- A constant is a name for a value that should not change
- Put **#define** after the includes and before **main**
- Use UPPERCASE names so constants are easy to recognise

```c
#include <stdio.h>

#define MAX_SIZE 12
#define PI 3.1415
#define MEANING_OF_LIFE 42
```

Coding Exercise

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

- prompts the user to 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 exercise: tricky_escape.c
- Variables and types: int double char
- Printing variables using printf
  - print_variables.c, print_errors.c
- Reading values into variables using scanf
  - scanf_demo.c 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:

Course Forum

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

Student Support - I Need Help With...

My Feelings and Mental Health Feeling depressed, overwhelmed or not your usual self?	**Mental Health Connect** student.unsw.edu.au/**mhc** **TalkCampus** student.unsw.edu.au/**Talkcampus**	**UNSW (24/7) Mental Health Support:** +61(2) 9385 5418 **Text support After Hours**: 0485 826 595 **Offshore** **Call 24-hour Medibank Hotline:** +61 2 8905 0307
Uni and Life in Australia Stress, financial, visas, Accommodation & More	**Student Support** **Indigenous Student Support**	student.unsw.edu.au/**advisors** nura-gili-centre-indigenous-programs
Reporting Sexual Assault/Harassment	**Equity Diversity & Inclusion (EDI)**	edi.unsw.edu.au/**sexual-misconduct**
Educational Adjustments To manage my studies and disability / health condition	**Equitable Learning Services (ELS)**	student.unsw.edu.au/**els**
Academic and Study Skills	**Academic Skills**	student.unsw.edu.au/**skills**
Special Consideration Because life impacts our studies and exams	**Special Consideration**	student.unsw.edu.au/**special-consideration**
Physical Health **Doctors visits, Vaccinations**	**Health Service**	student.unsw.edu.au/**hsu**

You are not alone!

What to do in the event of a problem or concern with the course

1. In the first instance, please try to resolve the issue with the **immediate party** - which in most cases will be your **tutor**
2. If unresolved, please escalate to the **course admins and lecturer**
3. If unresolved, please escalate to the CSE Student Representatives at **stureps@cse.unsw.edu.au** (or anonymously through our website)
4. If unresolved, please escalate to the CSE Grievance Officers at **grievance-officer@cse.unsw.edu.au**
5. If unresolved, please escalate to **UNSW Complaints** via the UNSW website

Brought to you by the CSE Student Representatives | **Find us at https://cgi.cse.unsw.edu.au/~stureps/**