COMP1511 PROGRAMMING FUNDAMENTALS

LECTURE 2

Variables and Constants - oh my!

LAST LECTURE...

ON MONDAY, WE TALKED:

- Tech issues at UNSW (an extra special welcome)
- Welcome and Introductions
- Course Administration
- How COMP1511 works
- How to get help and the best ways to approach learning Programming
- What is programming?
- What is Linux and working in Linux

IN THIS LECTURE

TODAY...

- Variables and how we store information
- Constants
- Maths in C!

WHERE IS THE CODE?

Live lecture code can be found here:

HTTPS://CGI.CSE.UNSW.EDU.AU/~CS1511/25T1/LIVE/WEEK01/

A BRIEF RECAP

OUR FIRST PROGRAM

```c
// A demo program showing output in C
// Welcome to COMP1511 :)
// Buckle, it is going to be a wild ride with
// a steep learning curve that we will hit in
// about Week 3!

#include <stdio.h>

int main(void){
    printf("Welcome to COMP1511!\n");
    return 0;
}
```

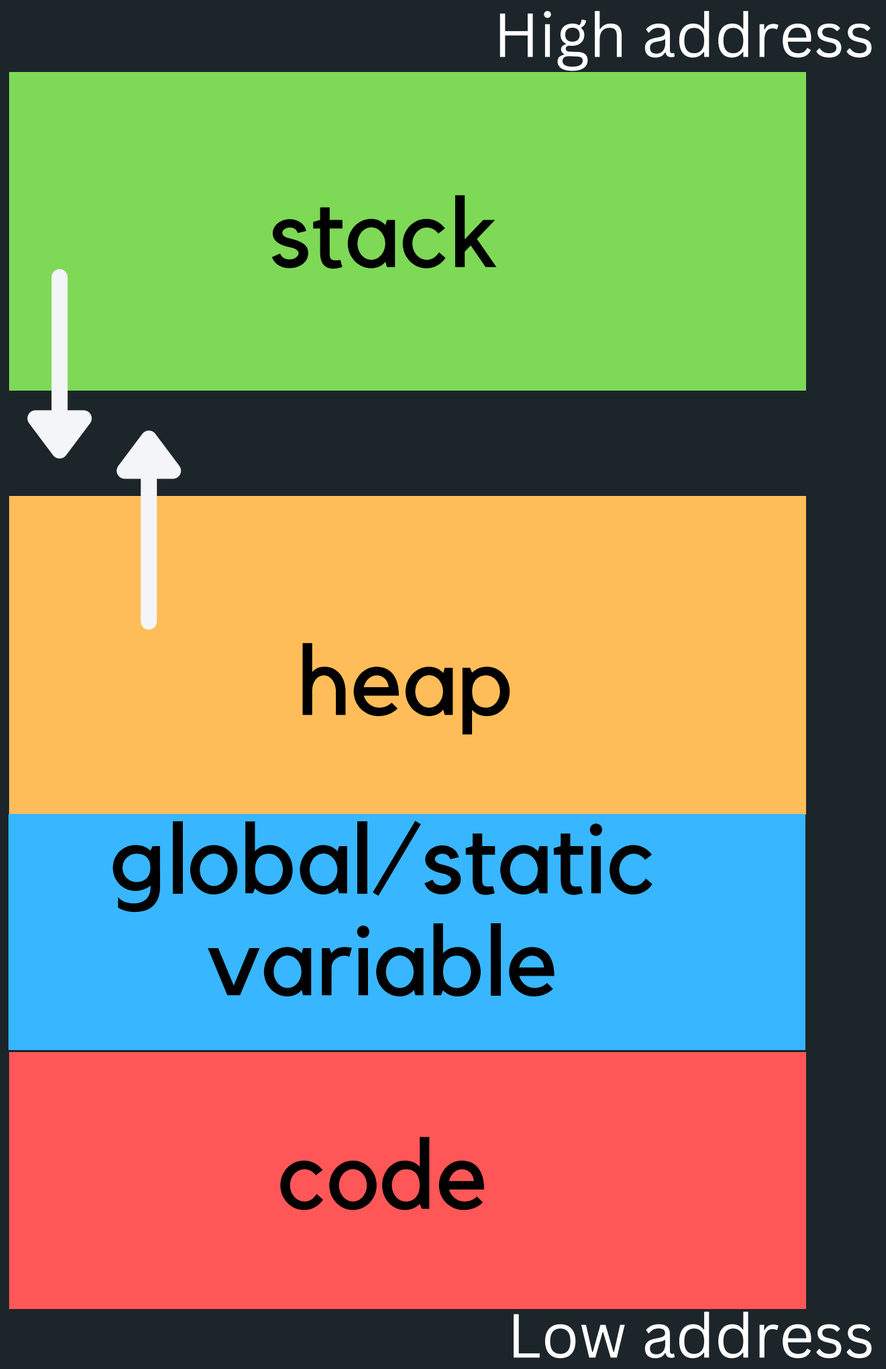
HOW DOES A COMPUTER REMEMBER THINGS?

ONES AND ZEROS!

- Computer memory is literally a big pile of on-off switches
  - We call these bits (smallest possible unit in computing, a bit is a choice between two things a 0 or a 1)
- We often collect these together into bunches of 8 bits
  - We call these bytes

WHAT DOES THIS LOOK LIKE?

When we execute code, the CPU will actually process the instructions and perform basic arithmetic, but the RAM will keep track of all the data needed in those instructions and operations.



WHAT IS A VARIABLE?

- Our way of asking the computer to remember something for us
- Called a ”variable” because it can change its value
- A certain number of bits that we use to represent something
- Made with a specific purpose in mind

WHAT KINDS OF VARIABLES WILL WE LEARN TODAY?

We're going to start out with three data types of variables:

`int` integer, a whole number (eg: -1,0,1,2,3)

`char` a single character (eg. 'a', 'A', etc)

`double` floating point number (eg: 3.14159, 8.534, 7.11)

Each of these has a different number of bytes that are allocated in memory once the program is run...

NAMING OUR VARIABLES

IT IS AN ART - CALL IT LIKE YOU SEE IT, LIKE YOU USE IT AND SOMEONE ELSE HAS TO SEE IT!

- Names are a quick description of what the variable is
  - Eg: “answer” and “diameter”
  - Rather than “a” and “b”
- We always use lower case letters to start our variable names
- C is case sensitive:
  - “ansWer” and “answer” are two different variables
- C also reserves some words
  - “return”, “int” and “double” can't be used as variable names
- Multiple words (use snake_case)
  - We can split words with underscores: ”long_answer”

NAMING OUR VARIABLES

STYLE GUIDE

We name our variables in ways that make it obvious what they are representing. Remember someone else has to be able to skim your code and know what you are saying/doing!

https://cgi.cse.unsw.edu.au/~cs1511/25T1/resources/style_guide.html

INTEGER

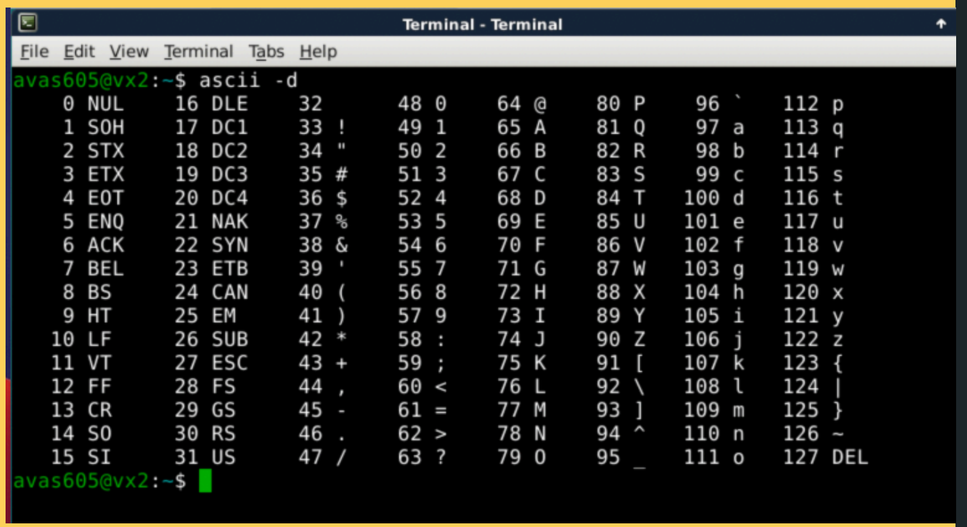
DATA TYPE `int`

- A whole number, with no fractions or decimals
- Most commonly uses 32 bits (which is also 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 ($2^{31} - 1$)

CHARACTER

DATA TYPE `char`



- A single character in C can also be represented as an int!
- This is because a single character variable holds an ASCII value (integers 0-127), as opposed to the character itself
- The syntax to assign a single character is to put the character in single quotes: ‘a’
- So for a capital letter ‘A: the character is ‘A’ and the int stored is 65
- You use a char to declare a character: char letter = ‘a’-- this will assign 97 to the variable letter

DOUBLE

DATA TYPE `double`

- A double-sized floating point number
- A decimal value - ”floating point” means the point can be anywhere in the number
- Eg: 10.567 or 105.67 (the points are in different places in the same digits)
- It’s called ”double” because it’s usually 64 bits, hence the double size of our integers (or 8 bytes)

LET'S TRY SOME CODE

DECLARE AND INITIALISE A VARIABLE

```c
// This program shows how to declare
// and initialise a variable

// Sasha Week 1

#include <stdio.h>

int main(void){
    // Declare a variable
    int answer;
    // Initialise a variable
    answer = 42;
    // Give the variable a different value
    answer = 13;

    // We can also declare and initialise together
    int answer_two = 42;

    return 0;
}
```

PRINTING OUT TO TERMINAL

printf()

```c
// Printing a variable
int number = 13;
printf("My number is %d\n", number);
```

- Not just for specific messages we type in advance
- We can also print variables to our display!
- To print out a variable value, we use format specifiers
  - this is a % symbol followed by some characters to let the compiler know what data type you want to print..
  - **%d** where the output you'd like to put an int (decimal value, hence **%d**)
- After the comma, you put the name of the variable you want to write

PRINT OUT MANY VARIABLES

WHY NOT?

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

```c
// Printing out two variables

int number_one = 13;
int number_two = 31;

printf("My first number is %d and second number is %d\n", number_one, number_two);
```

LET'S TRY DIFFERENT TYPES OF NUMBERS

INTS AND DOUBLES - OH MY!

- The **`%d`** and **`%lf`** are format specifiers that are used in printf statement to let the compiler know what data type we need to output.
  - **`%d`** stands for “decimal integer”
  - **`%lf`** stands for “long floating point number” (a double)
- Remember that we have to be very prescriptive when we tell the computer what to do and that extends to even telling it what types we are printing in C

```
1 // Print an int and a double
2 int diameter = 5;
3 double pi = 3.141;
4 printf("The diameter is %d, pi is %lf\n", diameter, pi);
```

WHAT ABOUT CHAR?

CAN'T FORGET THE LONELY CHAR

- The **%c** format specifier can also be used in printf statement to let the compiler know what data type we need to output (character).
- **%c** stands for “character”
- Don’t forget that when you declare a char, you enclose it in single apostrophes to let the computer know that you are using a letter character

```
1 // Print an int as a character
2 char letter = 'A';
3 printf("The letter %c has the ASCII value %d\n", letter, letter);
```

BREAK TIME

TIME TO STRETCH

There has just been a heavy fall of snow, Baudouim goes outside and finds that there is twice as much snow in his garden as in his neighbour Gael’s garden. He does not, however, appear surprised. Why not?

GREAT, WE CAN PRINT TO TERMINAL, CAN WE TAKE SOMETHING FROM TERMINAL?

scanf()

- Reads input from the user in the same format as printf
- Format specifiers (**`%d`**, **`%lf`**, **`%c`**) are used in the same way as for the printf statement
- The **`&`** symbol tells scanf the address of the variable in memory (where the variable is located) that we want to place the value into (more details later in term)

```c
// Reading an integer
int input;
printf("Please type in a number: ");
scanf("%d", &input);

// Reading a double
double input_two;
printf("Please type in a number: ");
scanf("%lf", &input_two);
```

WHAT ABOUT OUR LONELY CHAR?

scanf()

- If you want scanf to read in a character, you will need to declare a character by using the keyword: **`char`**
- Even though you have declared a char to store your character into, it is still stored as an ASCII value... so you can move between **`%d`** and **`%c`** when you printf this variable

```c
// Reading a single character as a character
char character;
printf("Please type in a character: ");
scanf("%c", &character);
```

WHAT IF A VARIABLE NEVER CHANGES?

THEN IT IS MOST LIKELY A CONSTANT...

- Constants are like variables, only they never change!
- To define a constant, we use **`#define`** and follow it with the name of the constant and the value

```c
// Using constants
#include <stdio.h>

// Define them before your main starts
#define PI 3.1415
#define MEANING_OF_LIFE 42
#define MAX_NUMBER 13

int main(void) {

}
```

Style Guide: We name them in all caps so that we remember that they're not variables!

HOW DOES SCANF() REALLY WORK?

A MAGICAL POWER...

- Gives us the ability to scan stuff in from the terminal (standard input)
- We have to tell the computer what we expect to scanf() - is it an **`int`**, **`double`**, or **`char`** ?
- But since scanf() is a function does it return something?
  - Yes, scanf() returns the number of input values that are scanned
  - If there is some input failure or error then it returns EOF (end-of-file) - we will look at this more later on!
  - This can be useful to check for any errors

DID YOU NOTICE HOW A NEW LINE IS READ BY SCANF()?

BECAUSE /N IS A CHARACTER ON THE ASCII TABLE: 10 LF (LINE FEED)

- You may have noticed that:
  `scanf("%d", &number);`
- is able to ignore anything other than a number when it scans in - this is because whitespace is not a number and the function looks for a number
- But did you notice that this is not the case for
  `scanf("%c", &character);`
- This is because a new line (/n) is a character on the ASCII table, which means it is still a valid character to scan in (It is number 10 LF if you are interested!)
- To fix this, we can tell scanf() to ignore all preceeding whitespace by using a special magic trick:
  `scanf(" %c", &character);`

LET'S TALK ABOUT MATHS

WE LOVE MATHS, RIGHT? C ALSO LOVES MATHS (SOMETIMES WITH QUIRKS).

- A lot of arithmetic operations will look very familiar in C
  - adding +
  - subtracting -
  - multiplying *
  - dividing /
- These will happen in their normal mathematical order
- We can also use brackets to force precedence

```c
// Using brackets to force precendence
int x = 5;
int y = 10;
int result;
result = (x + y) * x;
printf("The result is %d\n", result);
```

SUPER FUN FACT, YOU CAN DO MATHS WITH CHAR BECAUSE THEY ARE JUST INTS!

- Because characters are represented as ints inside the variable, you are able to move around the ASCII values by adding or subtracting to them.
- For example, if you are at ‘a’ and you want to get to ‘b’, you can add 1

```c
// Some basic maths!
char letter = 'a';
char next_letter = letter + 1;
printf("Original letter: %c with ASCII value %d\n", letter, letter);
printf("Next letter %c with ASCII value %d\n", next_letter, next_letter);
```

THE QUIRKS OF INTEGERS...

INTEGER OVERFLOW/ INTEGER UNDERFLOW

- Check out Boeing 787 that had to be rebooted every 248 days (2^{31}-hundredths of a seconds) https://www.engadget.com/2015-05-01-boeing-787-dreamliner-software-bug.html

https://www.theguardian.com/business/2015/may/01/us-aviation-authority-boeing-787-dreamliner-bug-could-cause-loss-of-control

THE QUIRKS OF INTEGERS...

INTEGER OVERFLOW/ INTEGER UNDERFLOW

- If we add two large ints together, we might go over the maximum value, which will actually roll around to the minimum value and possibly end up negative (Check out Ariane 5 explosion), a simple error like this caused a rather large problem: https://www.bbc.com/future/article/20150505-the-numbers-that-lead-to-disaster)

THE QUIRKS OF INTEGERS...

INTEGER OVERFLOW/ INTEGER UNDERFLOW

- In a less destructive example, the video Gangham Style on YouTube maxed out the views counter : https://www.bbc.com/news/world-asia-30288542

THE QUIRKS OF INTEGERS...

INTEGER OVERFLOW/ INTEGER UNDERFLOW

- Ints are not always 32-bits!

THE QUIRKS OF DOUBLES...

OFFENDING REPEATERS

- No such thing as infinite precision
- We can't precisely encode a simple number like ⅓
- If we divide 1.0 by 3.0, we'll get an approximation of ⅓
- The effect of approximation can compound the more you use them

NOW A LITTLE BIT ABOUT DIVISION

IT IS INTERESTING IN C...

- Remember that C thinks in data types
  - If either numbers in the division are doubles, the result will be a double
  - If both numbers are ints, the result will be an int, for example, 3/2 will not return 1.5, because ints are only whole numbers
  - ints will always drop whatever fraction exists, they wont round nicely, so 5/3 will result in 1
- % is called Modulus. It will give us the remainder from a division between integers, eg. 5 % 3 = 2 (because 5/3 = 1 rem 2)

Feedback please!

I value your feedback and use to pace the lectures and improve your overall learning experience. If you have any feedback from today's lecture, please follow the link below. Please remember to keep your feedback constructive, so I can action it and improve the learning experience.

https://forms.office.com/r/ZCjsg2JxX7

WHAT DID WE LEARN TODAY?

RECAP

Hello World!
our first program

VARIABLES

They come in different shapes and sizes - int, double and char
Printing from variables (printf)
Reading user input into variables (scanf)
Using maths with variables

REACH OUT

CONTENT RELATED QUESTIONS

Check out the forum

ADMIN QUESTIONS

cs1511@unsw.edu.au