Programming

CT103

Week 2a

Lecture Content

- Last lecture (Week 1b):
  - Computer programs.
  - Data types.
  - Example C program.

- Today's first lecture (Week 2a):
  - C basic variable types and their size in bytes.
  - Naming variables.
  - Declaring and initialising variables.
  - Comments.
  - C program.

Before We Start

- What did ‘\n’ do in the C code example from week 1b?
- Visual Studio 2022
- Xcode for Mac.

Before We Start

- What did ‘\n’ do in the C code example from week 1b?
  - Answer: ‘\n’ starts a new line when printing text to the screen.
  - E.g., **printf (“1 \n 2 \n 3");** will print “1”, “2” and “3” on new lines.

- Can I use Visual Studio 2022, instead of 2019?
  - Yes, this should be fine.

Before We Start

- Xcode for Mac.
  - If anyone is still having difficulty getting Xcode set up for Mac, there are plenty of online tutorials that can help you:
    - https://www.youtube.com/watch?v=_gwPhmyiuVo
    - https://www.youtube.com/watch?v=_cDXKReugEU
    - Search ‘Mac Xcode C’ on YouTube and you will find many results.

VARIABLES

Variables

- We need to be able hold data in our programs and change it as we do calculations
- Variables are pieces of memory that C reserves to hold our data
- Data is stored in binary form
- The more memory a variable uses, then the more data (1's and 0's) it can hold – hence the bigger numbers need more memory, e.g. (on a 64-bit windows machine):

  - *char* 1 byte
  - *short int* 2 bytes
  - *int* 4 bytes
  - *float* 4 bytes
  - *double* 8 bytes

Bits and Bytes

- What is a bit?
  - A bit is the most basic unit of information, i.e. 1 or a 0.
- What is byte?
  - A byte is 8 bits, e.g. 10101100.
- A kilobyte is 1024 bytes, i.e. 1 KB = 1024 B ($2^{10} = 1024$)
- A megabyte is 1024 kilobytes…
- Etc.
- However…

Bits and Bytes

- However…
- In International System (SI) Units, kilo means 1000.
- A kilobyte is 1000 bytes, i.e. 1 KB = 1000 B
- A megabyte is 1000 kilobytes… etc.
- These SI units are used for:
  - Data transfer rates
  - Hard drive capacities
- Other definition (1KB = 1024B) used for operating systems.

Also..

- Half a byte is called a “nibble”.
- As we saw, 1 byte = 8 bits.
- 1 nibble = 4 bits.

Types of C variables

Name	Description
char	Holds character data such as ‘x’ and ‘*’
short int	Holds integer data such as 1, 32, -456 Stores data between -32768 and 32767 Or 0 to 65535 if unsigned
int	Holds integer between -2,147,483,648 and 2,147,483,647 (double this if unsigned)
long int	Same as for int on a 32-bit compiler, but on 64 bit compiler: −9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
float	Holds floating point data such as 0.003, -12.4
double	Holds extremely large and small floating point data (bigger/smaller than $\pm 3.4 \times 10^{38}$!!)

Floating point

- A floating point number is one with a decimal number after the point.
- Decimal fractions difficult to represent exactly as binary fractions, so the binary is as close as possible, but there will be an approximation, but as we usually only display to a certain number of decimal places, we don’t usually notice.
- So we are usually seeing the rounded display of the actual machine value.

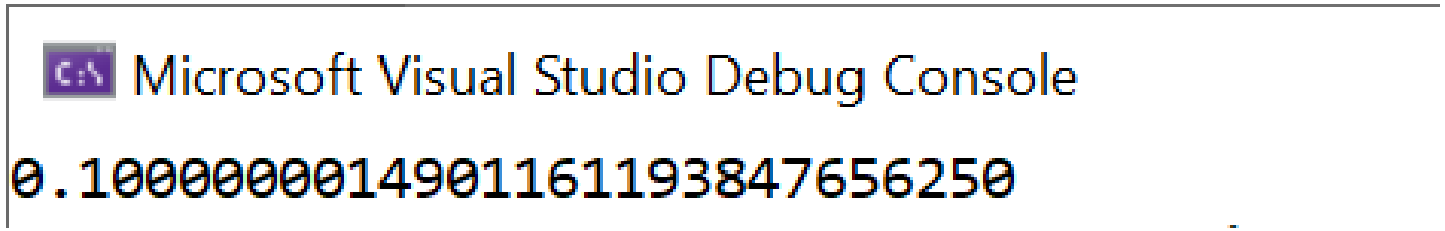
Rounding

- So for example if I execute this:

```
float f = 0.1;

printf("%30.28f", f);
```

- I see this output (printf allows me to specify the number of decimal places I see)!
  - In this case I am telling printf to print the variable ***f*** 30 characters wide, of which 28 are after the decimal point



Float vs double

- The double (64 bits) occupies twice the memory of a float (32 bits), therefore can store bigger, and more precise, numbers.
- So you can convert from a float to a double, but not necessarily the other way, without loss of precision.
- Depending on the platform you might use float if you don’t need doubles, to save on memory, performance and bandwidth.

NAMING VARIABLES

Naming variables

- Every variable you want to use needs a different name.
- A variable name can be from 1 to 32 characters long.
- The name must begin with a letter followed by any letter, number, underscore combination.
- The following are valid examples of variable names:

  myData pay94 age_limit amount
- The following are invalid examples of variable names :

  95Pay my age rate*pay printf

Variable naming conventions

- These are just some of the variable naming conventions (also called ‘cases’ or identifier formats).

- Many companies have their own conventions.

```
double annualsalary; // flat case
double annualSalary; // camel case
double annual_salary; // snake case
double Annual_Salary; // camel snake case
```

Declaring and Initialising variables

- We usually **declare** variables at the start of the program and we can optionally **initialise** them at the same time

```
float salary, pension; // variables declared but not initialised
char initial = 'c'; // declared and initalised
int departmentNumber; // not initialised
int age = 0; // declared and initalised


// now assign a value to the variable 'salary'
salary = 35000.00;
```

- Note how we can put comments at the end of a line.
- We will talk more about these later today!

Storing data in variables

- We use the *assignment operator* (=) to put data in variables

```
age = 34;
salary = 50000;
pension = salary + age*1000;
```

In general, we take what is on the right hand side (or what it evaluates to if it is an equation or a function call), and put it into the left hand side (usually a variable)

Printing out values of variables

- We can use printf() to do the work for us here
- For example:

```
int age = 25;
float salary = 34000.00;
char initial = 'D';

printf("age = %d \n", age);
printf("salary = %.2f \n", salary);
printf("initial = %c \n", initial);

printf("you are %d years old, you earn %.2f and your middle
initial is %c \n", age, salary, initial);
```

Using printf

- The printf function takes in a number of inputs
- The first input is always the text you want to print out, which may include placeholders (actually called *conversion characters*) for 1 or more pieces of data
- The data is supplied in the inputs following the formatted text input, with inputs separated by commas, for example:

```
int age = 35;
float salary = 35000.00;

printf("you are %d years old and earn %.2f per year", age, salary);
```

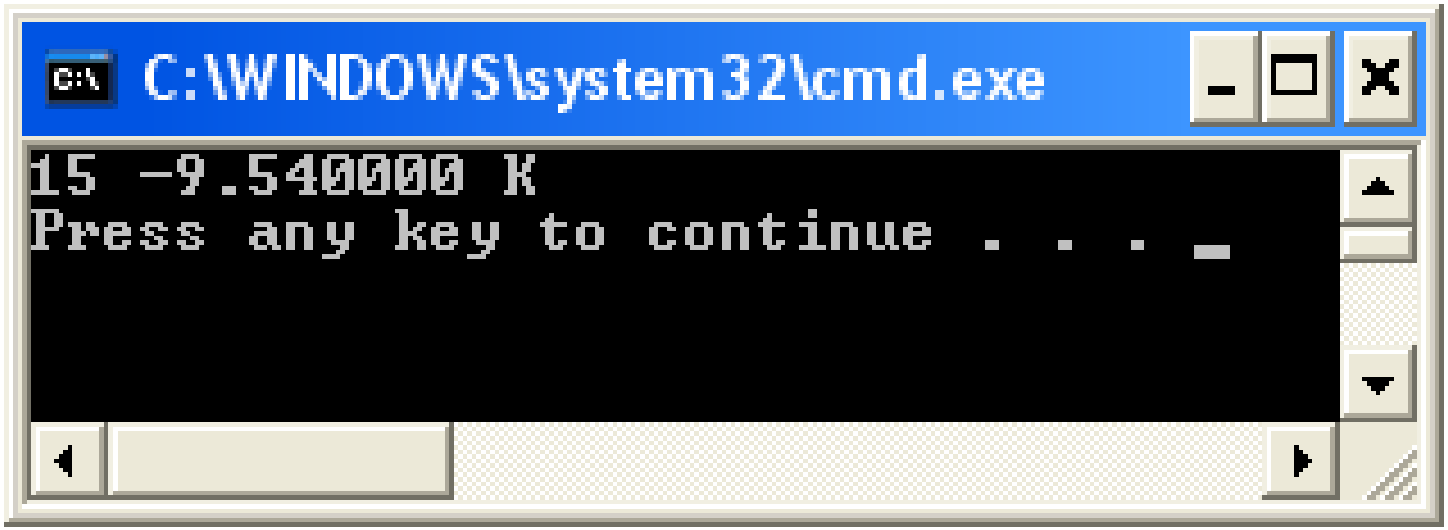
Conversion Characters

- Remember that we have to tell C exactly how to print numbers and characters
  - We have to use *conversion characters* (also called *format specifiers*)

Conversion Character	Description
%d	Integer
%f	Floating point
%c	Character
%s	String
%lf	Double
%X	Hexadecimal

Example

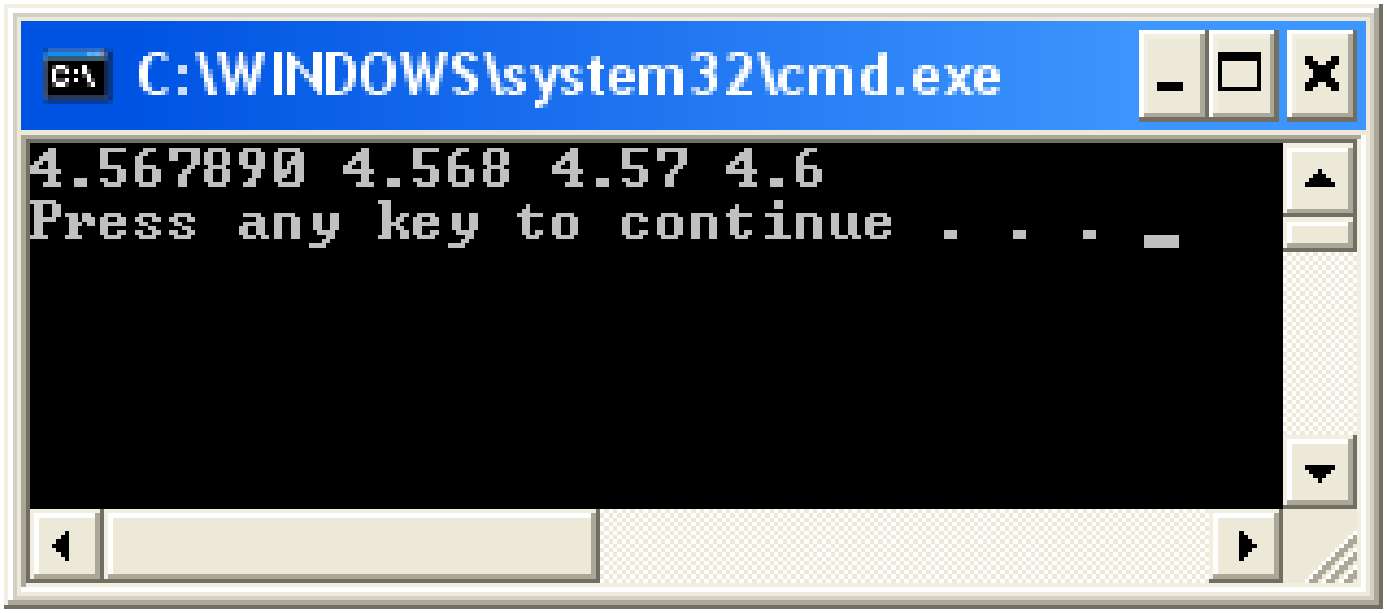
printf ("%d %f %c\n", 15, -9.54, 'K');



Note: if we don't specify the number of decimal places, C automatically puts in 6!

Example

```
printf ("%f %.3f %.2f %.1f\n", 4.56789, 4.56789, 4.56789, 4.56789);
```



Note: C rounds to the number of decimal places specified

Escape Sequences

- C uses Escape Sequences a lot to represent characters that can't easily be represented in text. They are converted into the correct character for example when output to screen.
- They are just special characters – we already used \n which gives us a new line.
- Some other ones are:

\t	tab
\\	just a backslash
\"	double quote
\'	single quote
\a	beep or alarm

Sample Program

- Try out this program yourself

```
#include <stdio.h>
void main()
{
        float grade1, grade2, grade3;
        float average = 0.0;

        printf("Enter 3 grades separated by spaces: ");
        scanf_s("%f %f %f", &grade1, &grade2, &grade3);

        average = (grade1 + grade2 + grade3) / 3.0;

        printf("average grade = %.2f", average);

}
```

How *scanf_s* is used

- The first input in scanf_s is the format text which tells scanf_s what the text the user inputs will contain, and how to parse it.
- In this example we are telling scanf_s that the input text will contain 3 floating point numbers, separated by spaces.
- After you enter the text via the keyboard and press enter, scanf_s *parses* the input to find the 3 floating point numbers.
- It then stores them in the variables which you provide to it.
- Putting the & in front of the variable name gives scanf_s access to the address of the variable, so it knows where to put the value it parses from the input text.

```
scanf_s("%f %f %f", &grade1, &grade2, &grade3);
```

&var gives you the address of var !

- So this code prints out the value of myInt and also the memory address where it is stored

```
int myInt = 44;

printf("myInt contains the value %d, which is stored at
location %X \n", myInt, &myInt);
```

```
myInt contains the value 44, which is stored at location C4FD70
```

- See what happens when I run it again – different memory location used when program is ‘reloaded’

```
myInt contains the value 44, which is stored at location 6FF7AC
```

So….

- To repeat…when I run this command, I am giving scanf_s the addresses of the three variables (grade1, grade2, grade3) so that it can store values there when it reads from the input (keyboard)

```
scanf_s("%f %f %f", &grade1, &grade2, &grade3);
```

COMMENTS

What are Comments?

- Comments are non-code text that you can add into your program.
- They are generally used to make the code more readable.
- You should use these to explain what your code is doing.

Using Comments

- In C, you can write a comment using //
- Anything that comes after // will be ignored by the compiler.
- E.g.

```
int age = 55; // This is a variable to store age
```

Comment Blocks

- In C, you can comment multiple lines of code using /**/
- Anything that comes in between /* and */ will be ignored by the compiler.
- E.g.

```
int age = 55;
/* I have created a variable to store age.
Next I will create a variable for salary.
*/
float salary = 35000.00;
```

Comments Example

- You can see how comments make the following code easier to understand:

```
/*
* Name: Karl
* Date: 1 October
*/
#include <stdio.h>
void main() {

    int age = 55; // variable for age

    float salary = 35000.00; // variable for salary

    // next I will print out the age and salary to the screen
    printf("print age %d, salary %f \n",age, salary);

}
```

CODE EXAMPLES

C Program Example

- Lets now look at a C program.