Textbook

- The material covered today is in chapters: 1 and 2 + pages 40 and 41 (beginning of ch. 3)
- Please go through this material before the lab
Credits + feedback

- The quizzes in these slides are from [http://www.cprogramming.com/](http://www.cprogramming.com/)

- The rest of the material is original, so feedback is welcome
C Compiler

- Converts C programs from plain text to executable format
- Already installed in the labs
- If you have your own computer, you should install it too
  - See instructions on the course website, try this before the lab, and if necessary ask for help during the lab session
Structure of C Programs

• The code to be executed is contained in **functions** (more details about functions in week 3)

• C functions have the following general structure
  `<return type> <function name> (<arguments>)`
  `{  
    statements to be executed  
  }

• One mandatory function: main
  `int main (void)`
Hello World!

- A very simple example C program:

```c
#include <stdio.h>

int main (void)
{
    /* this is a comment */
    printf("Hello World!\n");
    return 0;
}
```

- Can you guess what it does?
Colour coding

- Blue is for C-language keywords

```c
#include <stdio.h>

int main (void)
{
    /* this is a comment */
    printf("Hello World!\n");
    return 0;
}
```

- Currently coloured manually! So help me spot mistakes!
Compilation

- Converting a C program in plain text into executable binary code
- The process involves multiple steps: pre-processing, translating, assembling, and linking
- *Linking* is technically a separate step from compilation, but it is often referred as part of it
The Tools we Will Use

- GCC for compiling (& linking)
- Eclipse IDE for editing
- Cross-platform and open source, widely used
- Pre-installed in the lab computers
Tools outside the lab

- See instructions on the module's web site
Compilation Errors & Warnings

• The compiler is your friend: it will tell you when you make mistakes
  – However, it cannot guess what you want!

• It produces **errors** and **warnings**

• Fix warnings as if they were errors
Variables

• A variable is a container where you can store a single value

• Variable declaration: `<type> <variable name>;

• Type can be
  – char for an individual character
  – int for an integer number
  – float for a floating-point num. correct to 6 dec. places
  – double for a floating-point num. correct to 10 dec. places

• Variable name can be anything you want! (but careful..)
Variables (cont.)

- Examples:

```c
/* declaration examples */
int integerVariable;
double doublePrecisionVariable;
char characterVariable;
double voltage;
int nStudents;
int n_students;
int mickey_mouse; /* not a good idea? */
```
Variables Assignment

• Once a variable is declared, you can assign a value to it

• Examples:
  
  integerVariable = 100;
  doublePrecisionVariable = 33.3333;
  characterVariable = 'A';

• Values can be assigned multiple times
Declaration & Assignment in One Step

- Values can be assigned to variables right when they are declared.

- Examples:
  ```c
  /* declaration and assignment together */
  int integerVariable = 33;
  double doubleVariable = 1.001;
  ```

- This is generally a good idea, as it can prevent you from forgetting to initialize a variable.
Variables Names

- Names cannot start with numbers and cannot be C programming keywords
- Important: chose good names!
  - Good variable names can make a big differences about how you and others think about your code
Displaying Variable Values: `printf`

- Examples:
  ```c
  printf("the value is: %d\n", integerVariable);
  printf("value: %f\n", doublePrecisionVariable);
  ```

- `printf` takes as argument `text` (between inverted commas) and zero or more variables.

- The variables need to be specified in the text through `%` specifiers: `%d` for int, `%f` for float or double, `%c` for char.

- More in future lectures.
Inputting Values into Variables: scanf

• The input counterpart of printf

• Example:
  scanf("%d", &integerVariable);

• Very similar structure, except that variables need to be prefixed by '&'
  – & is the "addressof" operator

• More in future lectures
Variables scope

• When you declare a variable inside a function it can be accessed only inside that function
  – In the first few weeks we will only work with just one function: main

• A variable declared in a function is named a *local variable*
Variables scope (cont.)

- If you declare a variable outside any function it is accessible by all functions, and it is named *global*

- Using global variables is important in some cases, but it can lead to mistakes: use only when strictly necessary!
  
  - As much as you can, pass and return values instead of using global
More in the book

- Qualifying data types
- Register and volatile variables: very important for embedded C!
- Converting data types
Quiz to check your understanding

- What is the only function all C programs must contain?
  - A. start()
  - B. system()
  - C. main()
  - D. program()
Quiz to check your understanding

• Which of the following is not a correct variable type?
  – A. float
  – B. real
  – C. int
  – D. double
Quiz to check your understanding

• What punctuation ends most lines of C code?
  – A. .
  – B. ;
  – C. :
  – D. '
Quiz to check your understanding

• Which of the following is a correct comment?
  – A. */ Comments */
  – B. ** Comment **
  – C. /* Comment */
  – D. { Comment }
Arrays

- An array is a group or “list” of variables
- An array is declared this way:

  `<type> <array name>[<size>]`;
  
  - e.g. `int a[10];` or: `char name[32];`
  
  - the type can be int, float, double or char
  
  - the name can be anything you want (as for variables)
Arrays Access

- Elements of the arrays are accessed this way:
  
a[0] = 1;
a[1] = a[0] + 2;
printf("%d, %d\n", a[0], a[1]);
Array bounds

- It is the programmer’s responsibility to keep track of the array size and never go beyond it!

- E.g. if you declare a size 10 array and then try to access the 11th element you will get into troubles

- ..and the compiler will NOT tell you!!! Because it assumes you know what you are doing..

- `int array[10];
  array[10] = 33; /* BAD BAD BAD!!!! */`

(remember that counting starts from zero)
Array bounds (cont.)

- The worse part in doing something like:

  ```c
  int array[10];
  array[10] = 33; /* BAD BAD BAD BAD!!! */
  ```

  is that it may NOT always fail! Sometimes it may work!

- But it will generally fail

- Difficult to find the problem, until you ship your code (to the client or to the one who marks it...)
Arrays: Initialization shortcuts

• Different options:

  ```
  int count[5] = {0, 0, 0, 0, 0}; /* specify all elements */
  int count[5] = {1, 1}; /* specify first few (2) elements, all other ones set to zero */
  ```

• If you do not specify the size, the compiler will get the size from the data:

  ```
  int count[] = {0, 0, 0, 0, 0}; /* size: 5 */
  int count[] = {1, 1}; /* size: 2 */
  ```
Multi-dimensional Arrays

• Simply arrays of arrays:

```c
int m[3][4] = {
    {0, 1, 2, 3},
    {4, 5, 6, 7},
    {8, 9, 10, 11}
};
```

• If not all values are specified the other ones are set to zero
Multi-dimensional Arrays (cont.)

• Accessing values:

```c
m[1][2] = 33;
printf("%d\n", m[2][1]);
```
const

- If your program uses values that do not change during execution it's a good idea to let the compiler know
  
- The `const` keyword allows you to do so, e.g.:
  ```
  const int n = 5;
  ```

- Advantages:
  
  - Constant data can be placed in separate memory (you may run short of variable memory, especially on uC)
  
  - The compiler can warn you if you try to change a constant by mistake
Quiz to check your understanding

Which of the following correctly declares an array?

- A. int anarray[10];
- B. int anarray;
- C. anarray{10};
- D. array anarray[10];
Quiz to check your understanding

• What is the index number of the last element of an array with 29 elements?
  – A. 29
  – B. 28
  – C. 0
  – D. Programmer-defined
Quiz to check your understanding

Which of the following correctly declares a two-dimensional array?

- A. array anarray[20][20];
- B. int anarray[20][20];
- C. int array[20, 20];
- D. char array[20];
Quiz to check your understanding

• Which of the following correctly accesses the seventh element stored in foo, an array with 100 elements?
  – A. foo[6];
  – B. foo[7];
  – C. foo(7);
  – D. foo;
Summary

- The compiler converts programs into executable code
- Executable instructions sit inside functions
- Each C program must have one function called main
- Variables are single value containers, they are declared so: `<type> <variable name>;
- Arrays are containers for lists of values, declaration: `<type> <array name>[<size>];
- Read the text book & good luck with the 1st lab!