The C Programming Language

- Developed by Dennis Ritchie in 1972 at Bell Labs, in conjunction with the UNIX operating system.

- The American National Standards Institute (ANSI) formed a committee to develop a standardised version of C. The main standard was published in 1989 and is known as ANSI-C.

- An imperative programming language - programming task is achieved by a list of commands acting on a set of program variables.
Imperative Programming Languages specify HOW the processing must be done

- Have a collection of *commands* which can be used;

- Programmer is allowed to define named variables, of their own choice (of int or float or char);

- Programmer can write down an ordered sequence of commands;

- Commands might do things like *read input*, *print output*, and/or give new values to the *pre-defined variables*
Getting a working C program

- *Write the code.*

- Use `gcc` to translate your C program into something the computer will understand.

- *Run* the program, once we have a version which has successfully compiled.

  \[ EDIT \rightarrow COMPILE \rightarrow RUN. \]
What to do when it doesn’t work

“Right first time” is not a reasonable strategy for programming

• Some ‘debugging’ usually necessary.

• You can learn a lot from trial-and-error.

• Spending time working on the logical structure of your code, and the typographical details, will minimize debugging time.

• (for assignments) You are only assessed on the final version that you submit.
A simple C program

/* Simple hello program */

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("\n");
    printf("Hello world!\n");
    printf("\n");
    return EXIT_SUCCESS;
}

hello.c: **no** variables, **no** input commands. Only some printing (and return).
The Edit-Compile-Run cycle

• Edit:
  – Where do I write this C stuff?

• Compile:
  – How do I get my C program translated into something the computer will understand?

• Run:
  – How do I start my program?
  – Where do the results get output?
The Edit-Compile-Run cycle

- **Edit:**
  - `emacs hello.c`

- **Compile:**
  - `gcc -Wall hello.c`
  - *(gcc stands for Gnu C Compiler)*
  - `-Wall` is an *option* to ask gcc to write compile errors/warning to the “Wall”.

- **Run:**
  - `./a.out`

SEE NEXT LECTURE (and Monday’s LAB)
The structure of “Hello World”
/* Simple hello program */

/* --------------------- */
#include <stdio.h>
#include <stdlib.h>
/* --------------------- */

int main(void) {
    printf("\n");
    printf("Hello world!");
    printf("\n");
    return EXIT_SUCCESS;
}

- Includes *headers* verbatim into the program text.
- `<filename>` files are in the system directories (often `/usr/include`).
- “filename” files are in the current directory.
Comments

/* --------------------- */
/* Simple hello program */
/* --------------------- */

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    printf("\n");
    printf("Hello world!");
    printf("\n");
    return EXIT_SUCCESS;
}
main

/* Simple hello program */
#include <stdio.h>
#include <stdlib.h>

/* --------------------- */
int main(void) {

    printf("\n");
    printf("Hello world!");
    printf("\n");
    return EXIT_SUCCESS;

} /* --------------------- */
Every C program has exactly one main

• main is a function;

• main indicated to the compiler that the following section of code (within the parentheses {......}) is what gets executed when the program is run;

• main often has an empty input - this is indicated by (void)

• The name main is a reserved word in C (eg, cannot be used for variables);

• This output of this main is of type int ...
  but this is only a “flag” (computation ok/not-ok)
Functions

A function is any procedure which takes some (possibly empty) input, does some computation, and returns some (possibly empty) output

• Functions: Consider ‘+’
  – \(1+2\) - evaluates to the value 3
  – \(\text{plus}(1,2)\) - returns the value 3
  – \(\text{plus}(A,B)\) - returns the value C
printf

/* Simple hello program */

#include <stdio.h>
#include <stdlib.h>

int main(void) {
    /* --------------------- */
    printf("\n");
    printf("Hello world!");
    printf("\n");
    /* --------------------- */
    return EXIT_SUCCESS;
}

• printf is a library function.

• It has a manual page: man 3 printf.

• Contrast to man printf (remember the 3...)

• \n = new line.
/* Simple hello program */

#include <stdio.h>
#include <stdlib.h>

int main(void) {

    printf("\n");
    printf("Hello world!");
    printf("\n");
    /* --------------------- */
    return EXIT_SUCCESS;
    /* --------------------- */
}

• Remember that main returns an integer.

• EXIT_SUCCESS is the integer that it returns.

• stdlib.h defines EXIT_SUCCESS as 0.

• Numbers are often used in programming to represent a ‘status’. 
Programming Errors

• Most programs fail to work correctly the first time.

• Tracking down the errors requires time + patience + attention to detail.

• Skill in debugging is gained from experience (and attention to detail).
Example

```c
#include <stdio.h>
#include <stdlib.h>

.....
```

```
[fletcher]mcryan: gcc -Wall hello.c
hello.c:3:19: warning: extra tokens at end of #include directive
```
Common errors

• Mis-spelling
• Missing Punctuation
• Additional symbols
• Wrong punctuation
• Missing #include
• No main function
• return statement forgotten in a function
• Printf → Pritnf
• ("\n") → (\n)
• #include <stdio.h>
• ("\n") → ("\n")
Manifestations of an error

• Compiler *error* messages:
  – Fatal mistake - cannot continue.

• Compiler *warning* messages:
  – A mistake was found, the compiler ‘guessed’ what you meant, and continued.
  – Your program may still manage to work!
  – To show all the warnings - “gcc -Wall”.

• Error while running the program:
  – “Segmentation fault”.
  – The wrong result.