The C programming language

- What is C and why we learn it
- History of C
- Running the *Hello World* program
- Generating output!
- Language components: Data types, literals, identifiers, variables
- References:
  - Introduction to C programming (link on Web site)
  - C reference books (K & R, Afzal)

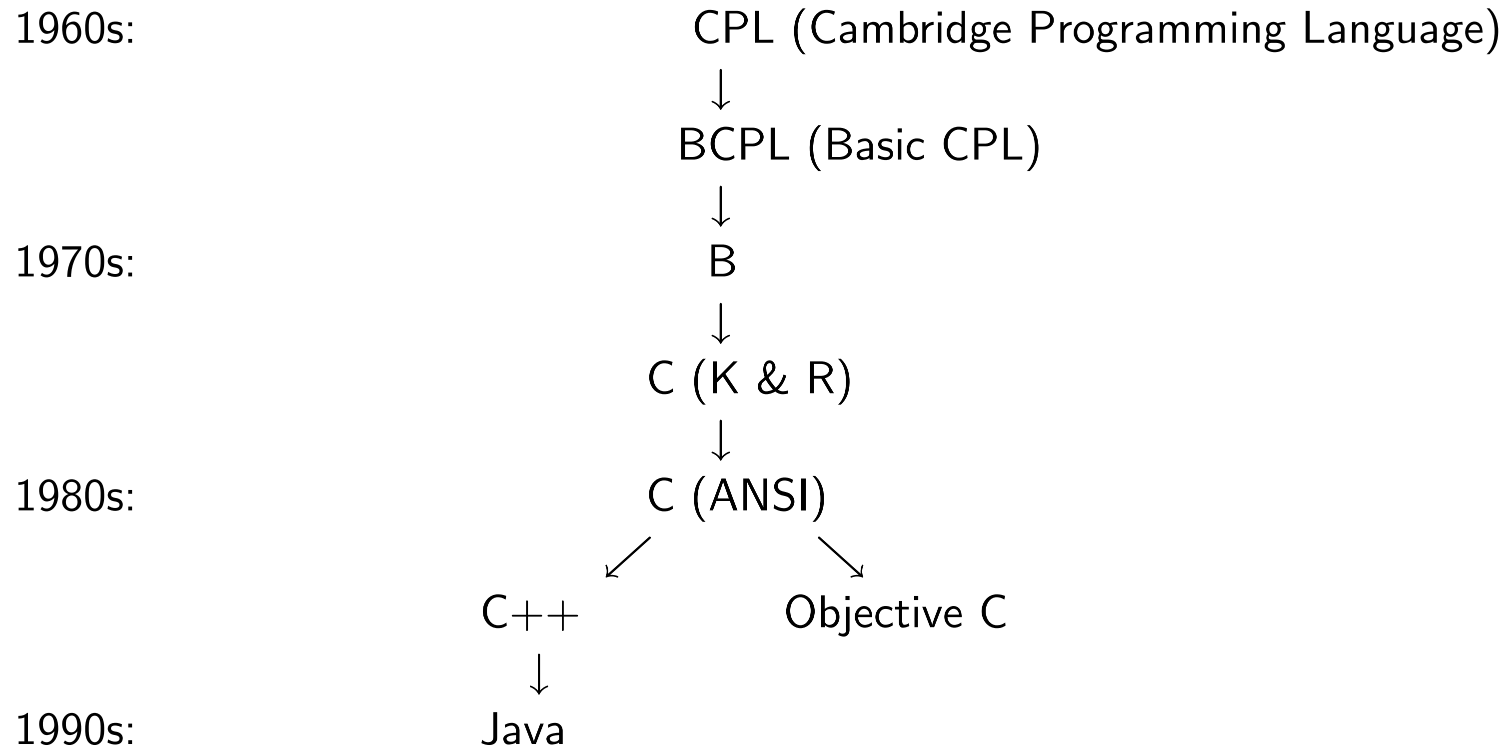
What is C?

- An **imperative** programming language
  - Contains a list of instructions or commands (Latin imperare *to command*)
- The emphasis is on saying $what$ a program has to do, instead of objects (like Java, Eiffel or Smalltalk) or functional relationships (like ML or Lisp)
  - Functional languages emphasise evaluation of expressions rather than execution of commands
- It has statements and basic data-types
- The same programming paradigm as assembly language, which also has instructions and basic data-types (e.g. byte, integer, float)
  - A "mid level language" (Afzal, Table 1.2)

Why do we learn C?

- Introduction to imperative programming
- Used later in COMP (and other courses)
- Wide usage in the “Real World”
- Other languages will be easier to understand and learn:
  C++, Java, Modula-2, Pascal, csh, Python, Perl, Matlab, etc.

History of C



Why is C so popular?

- Small and concise (32 keywords - Afzal table 1.3)
- Portable (ANSI standard)
- Efficient (compiler produces efficient machine code)
- Structured
- Closely tied to Unix (Linux)
- System level control (drivers, etc.)
- Available (compilers for almost every platform)
- Large user and code base

A first program: *Hello World*

```
1:  #include <stdio.h>
2:  int main(void)
3:  {
4:    printf("Hello World\n"); /*one line of output*/
5:    return 0;
6:  }
```

- Note: No class or objects mentioned
- Line 1 includes standard IO library interface
- `main()` is a special function – with no arguments
- Print the string *Hello World* followed by a *new line*
- Return the functions result (to the operating system)

Compiling and running your C program

- If your program is in a single file called `hello.c`, create an executable program by using the following command:

```
gcc -Wall hello.c -o hello
```

  where

  `gcc` $\longrightarrow$ the GNU C compiler

  `-Wall` $\longrightarrow$ show all warnings

  `-o hello` $\longrightarrow$ name of executable

- Run your code by typing the name of the executable

```
partch:~/comp2300/C1> ./hello
Hello World
partch:~/comp2300/C1>
```

- With multiple files, separate compilation and linking is best - we will discuss this later.

Basic C Program Structure

```
/*    */                    Comment line
#include                    Directive
int main()                  Function main heading
{
  statements                Body of the main
  return 0;                 Return statement
}
```

- We could write

```
int main(void){ printf("Hello World\n"); return 0; }
```

$\cdots$ but it would be bad style!

Basic data-types

- Integers (signed or unsigned)
  - `char` (8bit integer)
  - `short int` (small integer)
  - `int` (standard integer)
  - `long int` (long integer)
- Floating-points:
  - `float` (standard precision float)
  - `double` (higher precision float)
- Typeless/valueless:
  - `void`
- No boolean data-type, instead: $0 = false$ and $not\ 0 = true$
- Sizes are not explicitly defined, but relative size is respected (Afzal Fig 2.4)

Literals

- Integer: Decimal (e.g. 42, -1), octal (leading zero e.g. 017, -01) or hexadecimal (leading zero x e.g. 0xF, -0x1)
- Floating point (e.g. 123.4, -0.789, -0.001, 1.234e-2)
- Characters:
  - By symbol (e.g. 'q', 'A', '\%')
  - By ASCII code (e.g. '\012', '\xA')
  - By some escape code (e.g. '\n' new line, '\t' tabulator)
  - As integer (e.g. '\n' == '\xA')
  - Note: '\000' (often only '\0') is not equal to '0'

String literals

- A *string* literal, or *string constant*, is a sequence of zero or more characters surrounded by double quotes (e.g. "COMP2300")
- Strings are automatically terminated with a null character (i.e. '\0'), so "Hello!" will require 7 bytes of storage
- There is no limit on string length!
- Different character representations are valid in one string (e.g. "\x57indows?\n")

Identifiers

- Variable names, function names, macro names:
  - Start with a letter or with '_'
  - Followed by letters, digits or '_'
  - Case sensitive
- By convention:
  - Starting with '_' is reserved for use by the compiler and software libraries
  - `#define` constants are in upper case, with '_' separators. Examples:

```
#define PI 3.14159
#define COURSE "comp2300"
```

Variables

- Must be **declared** at the beginning of the function they are used in
- Only exist within the function they are declared in (their *scope*)!
- Global variables do exist, but they should only be used if needed
- Variables may be qualified as `const`, `static` or `register`
- May be initialised at declaration time

  Examples:
  ```
  int year;
  float length;
  const int pi=3.14159;
  char month[]="March"
  ```
  (Note: Length 6!)

Outline of a C program

```
/*    */                            Comment line
#include                            Directive
int main()                          Function main heading
{
  declaration statements            Variables declared
  initialization statements         Variables initialized
  statements                        Body of the main
  return 0;                         Return statement
}
```

The Output Function: `printf()`

- A function from the `stdio` library
- Displays the characters between the double quotes (called a character string) to the screen
- Notice \n is not displayed, this is a "control" or "escape" character causing a new line

\a	Audible alert (bell)	\b	Back space	\f	Form feed
\n	New line	\r	Carriage return	\v	Vertical tab
\\	Backslash	\'	Single quote	\"	Double quote
\%	Percentage	\t	Tab		

Displaying Numbers and Strings

- `printf("control string", arg1, arg2, ...);`

- A variable number of arguments (parameters)
  - First a format string
  - Subsequent arguments are the values to be displayed

- The function inserts the values into the format string (in accordance with the specified format) and then displays it

  ```
  printf("Temperature:  %d\n", 24);
  ```

  will display

  Temperature: 24

`printf()` – **Format strings**

- The format string contains:
  - Ordinary characters, which are displayed without being changed
  - Conversion specifications, which are replaced by characters representing the corresponding value in the subsequent function parameters

`%d`	Signed integer as decimal (int)
`%u`	Unsigned integer as decimal (int)
`%f`	Floating point number as decimal (float or double)
`%c`	Character
`%s`	String (or array of characters) (char *)
`%%`	Display the character %

printf() – Example

```
 1:  #include <stdio.h>
 2:  #define COURSE "comp2300"
 3:  int main(void) {
 4:    int day=2, year=2006;
 5:    char month[]="March";
 6:
 7:    printf("Hello %s\n", COURSE);
 8:    printf("Today's date is %d, %s, %d\n", day, month, year);
 9:    printf("\n\n\t\t(it's Thursday)\n");
10:
11:    return 0;
12:  }
```

Fancy Formats

- Look at Afzal 2.2, 2.3 and 2.4 and consider...

```
 1:  #include <stdio.h>
 2:  int main(){
 3:    int i=0;
 4:    float fx=5.0/3.0;
 5:    char me[]="Peter Christen";
 6:    printf(" character %c integer %d \n",65,'A');
 7:    for (i=0;i<128;i++){
 8:      printf(" character %c %d %o %x \n",i,i,i,i);
 9:    }
10:    printf("%10.2f %-10.2f %+10.2f\n",fx,fx,fx);
11:    printf("%s%20s%20.4s%-20.4s\n",me,me,me,me);
12:    return 0;
13:  }
```

For Next Lecture!

- What's the value of the following C expression?

```
7+4*2%6*4
```

- Is the following true or false?

```
(16-3<4*3+4)||(7*2==28/2)&&!(16-3<12+6)
```

- Are the following all equal?

```
3*4/2
4*3/2
3/2*4
4/2*3
```