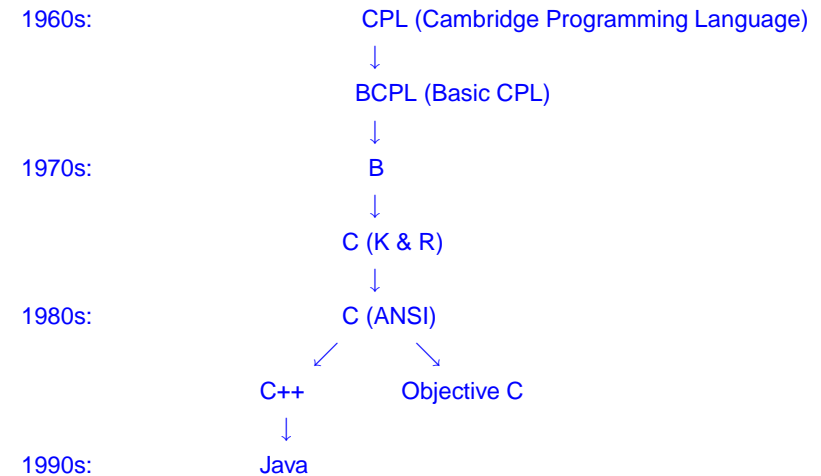


The C Programming Language

- refs: ref. books ([King,], K&R, Afzal), related web links
- what is C and why we learn it
- history of C
- running the `helloWorld` program
- language components: data types, literals, identifiers, variables
- generating output!

History of C



What is C?

- an imperative programming language
 - contains a list of instructions or commands (Latin *imperare* *to command*)
- emphasis is on saying *what* a program has to do, instead of objects (like Java, Eiffel or Smalltalk) or functional relationships (like Haskell or Lisp)
 - functional languages emphasise evaluation of expressions rather than execution of commands
- 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”; the universal assembly language!
- why do we learn C?
 - introduction to imperative programming
 - used later in COMP (and other courses); wide usage in the “Real World”
 - other languages (e.g. C++, Java, csh, ...) will be easier to learn
 - *the* language of computer systems; *the* fundamental compiled language helps understanding of how programs from other languages are executed

A First Program: `helloWorld.c`

```
#include <stdio.h>
int main(void)
{
    printf("Hello_World\n"); /*one line of output*/
    return 0;
}
```

- note: no class or objects mentioned
- line 1 includes standard IO library interface (header file)
- `main()` is a special function – in this case with no arguments
 - can use `int main(int argc, char *argv[])` to access the command line parameters
- prints the string `"Hello_World"` followed by a new line
- returns the function's result (to the operating system)
 - `'0'` signifies normal termination

Compiling and Running your C Program

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

```
gcc -Wall -o helloWorld helloWorld.c
```

where:

```
gcc           → the GNU C compiler
-Wall        → show all warnings
-o helloWorld → name of executable
```

- run your program by typing the name of the executable

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

- with programs made of multiple C files, separate compilation and linking is best - we will discuss this later

Basic C Program Structure

```
/*      */           comment line
#include <...>       directives
int main(void)      function main() heading
{
    declarations    variables declared

    statements      body of main()
    return 0;       return statement
}
```

- we could write

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

... but it would be bad style!

(and C allows you to do worse...)

e.g. a program producing this graphic)

Why is C so popular?

- small and concise (32 keywords - [King, table 2.1])
- portable (ANSI standard) and available (compilers for almost every platform)
- efficient (compiler produces efficient machine code)
 - programmer has more control of data layout and object code produced
 - examples: `optimizedMatrixMult.c`, `inlineAssemblerEx.c`
 - very convenient for low-level data manipulation e.g. `underflow.c`
- arguably *the* programming language for computer systems
 - closely tied to Unix (Linux)
 - system-level control (drivers, etc.)
- structured; modular
 - *can* support abstract data types and object-oriented design
- large user and code base

Basic Data Types

- integer (signed or unsigned):
 - `char` (8 bit integer)
 - `short int` (small integer)
 - `int` (standard integer)
 - `long int` (long integer)
- floating-point:
 - `float` (standard precision float - 32 bit)
 - `double` (higher precision float - 64 bit)
- typeless/valueless:
 - `void`
- no Boolean data type; instead: 0 is 'false' and non-0 is 'true'
- sizes are not explicitly defined, but relative size is respected

Literals

- integer: decimal (e.g. 42, -1), octal (leading 0, e.g. 017, -01) or hexadecimal (leading 0x, 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' tab)
 - as an integer (e.g. '\n' == '\x10')
 - note: '\000' (or '\0') is not equal to '0'
- strings
 - a string literal (constant) is a sequence of zero or more characters surrounded by double quotes (e.g. "COMP2300")
 - are automatically terminated with a null character ('\0'), so "Hello!" will require 7 bytes of storage
 - there is no limit on string length!
 - different character representations valid in one string (e.g. "\x57indows\n")

The Output Function: printf()

- a function from the `stdio` library
- displays the string (characters between the double quotes) to the screen
- special characters are displayed using escape sequences;

<code>\a</code>	audible alert (bell)	<code>\b</code>	back space	<code>\f</code>	form feed
<code>\n</code>	new line	<code>\r</code>	carriage return	<code>\v</code>	vertical tab
<code>\\</code>	backslash	<code>\'</code>	single quote	<code>\"</code>	double quote
<code>\%</code>	percentage	<code>\t</code>	tab		

- `printf("format_string", arg1, arg2, ...);`
- has 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, e.g.:

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

will display: Temperature: 24

Identifiers and Variables

- identifiers used for 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. e.g.

```
#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 can be declared, but should only be used with good reason
 - variables may be qualified as `const`, `static`, `register` or `volatile`
 - may be initialised at declaration e.g.

```
int year;
float length;
const double pi = 3.14159;
char month[] = "March"; // note: has length of 6!
```

printf() – Format strings

- the format string contains:
 - ordinary characters, which are displayed without being changed
 - format specifiers, which are replaced by characters representing the corresponding value in the subsequent parameters
 - `%d` signed integer as decimal (`int`);
 - `%u` unsigned integer as decimal (`int`)
 - `%x` unsigned integer as hexadecimal (`int`)
 - `%f` floating point number as decimal (`float` or `double`)
 - `%c` character (`char`)
 - `%s` string (or array of characters) (`char *`)
 - `%%` display the character %

