

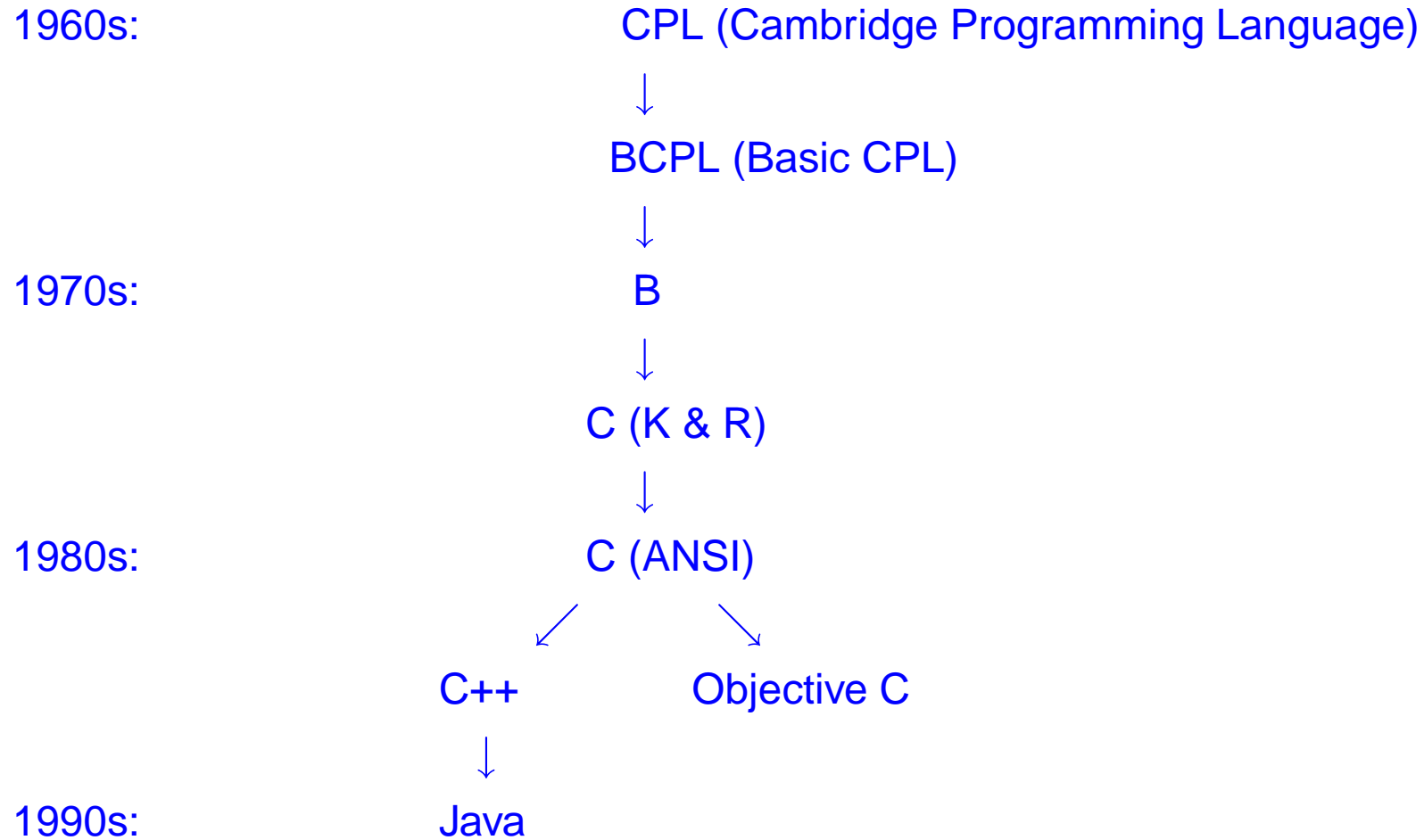
# The C Programming Language

- 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!

# 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 ML 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 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 - [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`
- 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



## 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 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"`)

## 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!
```

## 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

## `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`)
    - `%f` floating point number as decimal (`float` or `double`)
    - `%c` character (`char`)
    - `%s` string (or array of characters) (`char *`)
    - `%%` display the character `%`

## printf() Formats: Example printDate.c

```
#include <stdio.h>
#define COURSE "COMP2300"
int main(void) {
    int day = 6, year = 2008;
    char month[] = "March";

    printf("Hello_%s\n", COURSE);
    printf("Today's_date_is_%d,_%s,_%d\n", day, month, year);
    printf("\n\n\t\t(it's_Thursday)\n");

    return 0;
}
```

## Fancy Formats: fancyPrintf.c

- look at [King, ch 3] and consider...

```
#include <stdio.h>
int main() {
    int i = 0;
    float fx = 5.0/3.0;
    char me [] = "Peter Christen";
    printf( "character %c integer %d\n", 65, 'A' );
    for (i=0; i<128; i++) {
        printf( "character %c %d %o %x\n", i, i, i, i );
    }
    printf( " %10.2f % -10.2f % +10.2f\n", fx, fx, fx );
    printf( " %s %20s %20.4s % -20.4s\n", me, me, me, me );
    return 0;
}
```

## 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?

$3 * 4 / 2$

$4 * 3 / 2$

$3 / 2 * 4$

$4 / 2 * 3$

- although it might look similar so far, remember C is *not* Java!