C Programming – Quick Reference Sheet

1) Write C code (=instructions) into a text file with extension .c   
 For example “sample.c”, created here with the nano text editor in linux:

***nano sample.c***

2) Compile the code. This typically takes care of pre-processing the code, creating intermediary object code in object files (\*.o or \*.obj) and then linking the object files and libraries to an executable application.

***gcc –c sample.c*** 🡨 compile

***gcc –o myApp sample.o gb\_common.o*** 🡨  *link*

3) Execute the executable file (in linux from a terminal window):

./myApp

or sudo ./myApp if superuser rights are required to execute myApp (*super user do*)

**Statements**

C programmes contain statements that are executed in sequential order (generally).

Each statement must be terminated by a semicolon (;).

**Functions**

The main execution entry point for the application is the function main.

For repetitive tasks and to structure the code, other functions (containing statements) can be declared and have to be defined for use in the code:

*returnType functionName( functionParameters );* Syntax of a function declaration.

int mySum ( int a, int b ); Example.

*returnType functionName( functionParameters )*  Syntax of a function definition.  
{ … } Function body in curly braces {.,,}

int mySum ( int a, int b ) Example of a function definition.  
 { return (a + b); } Call with: s = mySum( 4, 3 );

**Hello World in C**

// This code displays a message on the command line.

#include <stdio.h>

void main(void) {

printf(“Hello World - I am alive!\n”);

}

**Commenting code**

// This is a **single-line comment** which starts at // and ends at the end of the line

/\*…\*/ This is a **multi-line comment** which can span several lines. Anything between  
 /\* and \*/ is ignored by the compiler.

**C Keywords**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Types** | **Qualifiers** | **Storage** | **Flow** | **Flow** | **Other** |
| char | const | auto | break | goto | typedef |
| double | long | extern | case | if | sizeof |
| enum | short | register | continue | return | void |
| float | signed | static | default | switch |  |
| int | unsigned | volatile | do | while |  |
| struct |  |  | else |  |  |
| union |  |  | for |  |  |

char single character literal (‘a’, ‘b’, etc)

double double precision number (0.11223, -1.0, 3.1415)

int integer number (-1, 0, 1, 2, 3, etc)

if … else … conditional branching

do … while … loop for(<start>;<while-condition>;<increment>)

void “nothing” (e.g. as return type)

**Escape Sequences (for printf)**

\n newline \r carriage return \? ? \o.. octal number

\t horizontal tab \f formfeed \’ ‘ \x.. hexadecimal number

\v vertical tab \a bell (beep) \” “

\b backspace \\ \

**Common printf type conversions**

%[+][-][*w*]d 🡪 int in signed decimal notation; +: with sign; -: left adjusted; *w*: field width

%[+][-][*w*][.*p*]f 🡪 double precision in decimal notation; *p*: precision

%c 🡪 single character (int is converted to unsigned char)

%s 🡪 string (char\*) until ‘\0’ is reached in string; e.g. printf(“%s”, “hi there\0”)

Width or precision is usually an integer literal, however, it can be a \* in which case the value of the next function argument is used. This must be of type int.

**Mathematical and Boolean operators**

! not ++ increment by 1 -- decrement by 1

\* multiplication / division % remainder

+ addition - subtraction

<< left bit shift >> right bit shift

< <= >= > comparison operators

!= not equal to == equal to comparison

&& logical AND || logical OR

& bit-wise AND | bit-wise OR

?: short form of if…then…else, e.g. (a==1) ? (b=2) : (b=3); if a is 1 then b=2 else b=3.