COMP2129

Week 1

C-Language

- Mainly used for systems software and software that needs hardware interaction.
- Does not have objectives & classes, templates, operator/function overloading.
    - C++ overcomes this & is a successor of C
- C-Programs consist of two language components:
    - Preprocessing language
        - Text-macro language
        - Definition of macros
        - Include files
        - Conditional compilation
    - C-language
- The hello world output:

```
#include <stdio.h>
int main (int argc, char **argv)
{
        printf("Hello World!\n");
        return 0;
}
```

- To run a c program, execute the following command:
    - clang hello.c –o hello
    - ./hello

Java to C

- Differences:
    - Control flow structures are the same
    - References are called pointers in C
    - No garbage collection → programmer is responsible for allocating & freeing memory
    - A C-Program consists of a set of files containing: global variables, function definitions (main is the first function invoked), functions have local variables.
- Philosophical differences:
    - C closer to underlying machine
    - C has simple memory modal:
        - Pointers, bit-level operators
        - Arrays very close to memory model
    - C assume programmer knows best
    - Java object-oriented, C is procedural (no object, no polymorphism, no inheritance).
- Strong similarities:
    - Block structured
    - Most control structures
    - Arrays
    - Operators

    - Basic data types
- Preprocessor differences:
    - C macros (#define)
    - Call-by-name
    - C has declaration for variables & functions, often in header files that are included
    - Conditional compilation

Arrays and memory

- Arrays can be handled with pointers
- Arrays can be created and initialized in declaration
- C strings are just arrays (with termination character)
- sizeof operator
- Create dynamic data structures with malloc()
- C allows declarations only at block start

Functions in C

- A function consists of:
    - A function declaration: name of function, return type of function, parameter list & their types
      int foo(float f1, char c2)
    - Followed by a function body: local variables & control flow
- External or forward function declarations do not have a function body, just a semicolon
    - Parameter types are specified without variable names
      int foo(float, char);
      extern int foo(float, char);
- A function with a given name can only be defined once
- If no return value exists for a function, use the type void
  void foo(…) { … }
- If no parameters exist, use type void
  void foo(void) {…}
- Functions with arbitrary numbers of parameters are possible
  int printf(const char *format,..)
    - In this case, a special interface is required for querying values of parameters → at least one fixed parameter in the function is necessary

  printf(“%d, %f”, 10, 10.5);

C modules

- Programs consist of “modules” → a module is a file
- Modules consist of: function declarations; function definitions; global variables
- Modules are translated to object files
- Object files are linked by linker with other object files & standard libraries
- A module can refer to global variables and functions of other modules
    - Use the extern qualifier for global variables
- Symbols can only be defined in one module
- Data structures definitions & declarations, macro definitions & external function declarations are found in modules.

Input/output functions

- Basic input: int getchar(void);
    - Reads from standard input next character
    - Returns -1 (defined as the symbols EOF) if end of input reached
- Basic output: void putchar(int c);
    - Write a character (represented as integer) to standard output

printf() function

- printf() – function writes to standard output:
    - Strings
    - Variables of primitive data-type

  int printf(const cahr *format, ..);
- Return value: number of printed characters → hence why the return type is int
- Arguments: first argument is a format string; followed by an arbitrary number of parameters depending on format string.
- Example: printf("%d %f\n", 10, 10.5); → output: 10 10.5

Code	**Description**
%c	Character
%d	Integer
%u	Unsigned integer
%f, %g, %e	Double floating point number
%x	Hexadecimal
%ld	Long
%.2f	Print floating point numbers with two decimal points
%s	String
%p	Pointer
%%	Print %

scanf() function

- scanf() function reads from standard input: values of primitive data-type & strings
  int scanf(const char *format,…);
- Return value: number of successfully read items
- Argument: first argument is a format string, followed by an arbitrary number of parameters depending on format string, parameters must be pointers, not values.