Course Notes: CPSC 4100 Spring 2020

Languages

C

- originally 1973
- Dennis Ritchie (The R in K&R)
- ANSI (American National Standards Institute) C standard since 1989
- imperative: statements affect program state
- structured: formal control structures / blocks
- procedural: code organized into called procedures (subroutines)
- static typing: data type property assigned at compile time
- weakly typed (`void*`) : implicit type casting under some conditions
- compiles all the way to the hardware (executables not portable)
- allows for raw memory management and manipulation
- modeled naturally on the standard *von Neumann machine* architecture
  - CPU with registers, ALU, control unit
  - memory containing both instructions and data

hello world in C (the parts of)

```
#include <stdio.h>               /* preprocessor directive */

int main(int argc, char *argv[]) /* program entry point with command line arguments
{
    printf("Hello World\n");     /* a subroutine that does IO -- declared in stdio.h
    return 0;                    /* return code from main -> exit code for program */
}
```

General Concepts

Binding Times

The act of associating *names* with properties (data type, address, value) is called *binding*, and different properties are bound at different times.

```
#include <math.h>


  void main()
  {

  int i;
  double sum=0;

      for (i=1; i<100; i++)
           sum += sqrt(i);
  }
```

- **language definition time**
  meaning of keywords is bound – all implementations must behave the same way (void, for)
- **language implementation time**
  e.g. the range of values for `int` is implementation dependent. (not the same in java)
- **compile time**
  - data type for `i` is bound here. (static typing)
  - details of `sqrt` interface (declaration in math.h)
- **link time**
  definition of `sqrt`
- **load time**
  memory address for all of these symbols
- **runtime**
  `i` takes on a sequence of values

- early binding : before runtime / late binding == runtime binding
- not all language systems use all times (interpreters are not compiled)

Parameter Passing Semantics

Definitions

- formal parameters (specified in subroutine)
- actual parameters (passed to subroutine)
- the call stack

parameter *correspondence*

- java and C use positional parameters
- other languages may have keyword parameters
- default parameters (C++ has this)

- variable arguments in C processed with system calls

Call-by-value

- formal parameters are local variables in the stack frame (aka *activation record*) of the called method
- initialized with the value of the corresponding actual parameter
- variables used in calling function cannot be directly modified since only the values are passed (pointers & references complicate this)