

# 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 */
}
```

## C data types

```
void
char
int
float
double
```

```
// these assume int
short
long
signed
unsigned
```

```
// may be optimized by using read only memory
const
```

```
// no implementation-independent semantics
volatile
```

- typedef syntactic renaming of a type

```
typedef unsigned int uint
typedef const double CONSTANT;
```

- arrays

```
int values[10]; --> int* values
```

array lookups as math problems

```
int values[10];

for (int i=0; i<10; i++)
{
    printf("%d: %d\n", i, values[i]);
}

printf("set values[3]\n");
/* type of values tells it how to interpret the 3 */
*(values + 3) = 999;

/* raw addresses */
printf("set values[8]\n");
*((int*)((void*)values + sizeof(int)*8))=1234;
```

## C++

- Bjarne Stroustrup
- first appeared 1985; standardized in 1998
- adds object oriented features, namespaces, generics, exceptions

### hello world in C++

```
#include <iostream>

int main(int argc, char *argv[])
{
    std::cout << "hello world" << std::endl;
    return 0;
}
```

### Null-terminated strings

String data is stored as an array of characters with the NULL character (ascii 0) indicating the end of the string. Many useful functions for dealing with data of this type are available in `string.h`.

## 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)

## Call-by-reference

The lvalue of the actual parameter is computed before the method executes. Formal parameters are replaced with actual parameter's lvalue. Effectively, the formal parameters become aliases for the actual parameters.

## Call-by-macro-expansion

- formal parameters replaced with text of actual parameters
- macro call replaced with expanded macro
- variable capture For a given code snippet
  - **free variables** have no binding (are not associated with a specific memory location)
  - **bound variables** do
  - Macro expansion can cause free variables to become bound inside the macro expansion – this has undefined semantics and will result in errors (see the SWAP example)

## Typing Systems

- rules surrounding the binding of data type to variables and expressions
- statically typed – data types bound at compile time
- type safety: how aggressively does the language apply typing rules to force you to write safe, good code
  - strongly typed  $\longleftrightarrow$  weakly typed
  - C is (relatively) weakly typed because we can throw away type information by casting to `void*`

## Definitions

### statements vs. expressions

- a statement is an executable step in the algorithm
  - the building block of an algorithm
- an expression is anything with a value (can be evaluated)

## **lvalues and rvalues**

Every expression is either an lvalue or rvalue

- rvalue's are temporary and have the lifetime of the corresponding expression
- lvalue's persist beyond the expression
  - variables (anything with a name)
- C has the ability to convert between the two in a manner