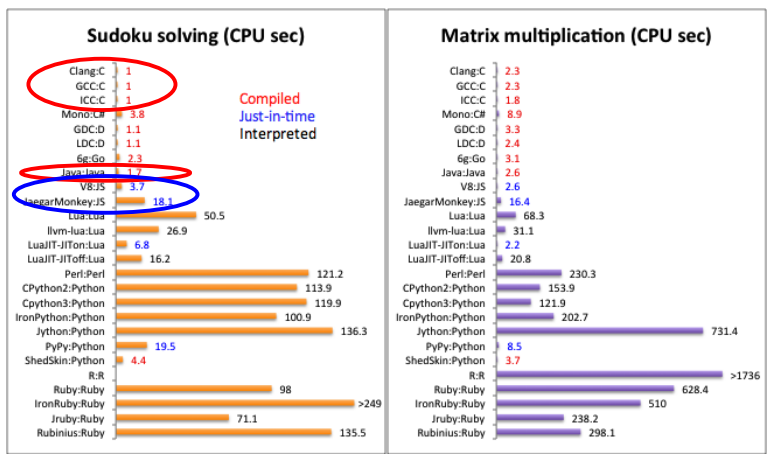
Computer Systems Lecture 7

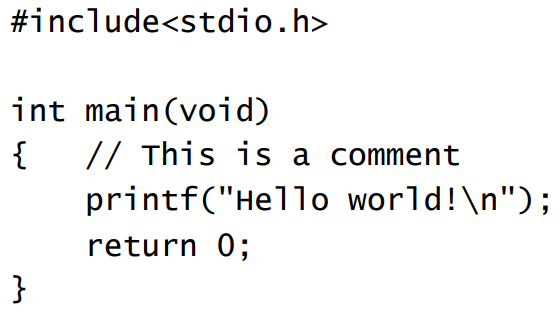
Intro To C

C is both a high and low-level language, it’s very useful for system programming and is very fast.

Performance: C vs. The Rest



The Hello World Program



Major Differences From Java

C is not object oriented, C programs are collections of functions and there is no inheritance, subtyping ect…

C is not interpreted, C is compiled into an executable machine ode program that runs directly on the processor where as java is compiled into byte code, which is read and executed by the java interpreter.

Run-time errors are not ‘caught’ in C, the java interpreter catches these errors before they are executed, but in C the program will just run until the error occurs and the program crashes.

The C compiler trusts the programmer not to be an idiot (not the best idea).

Memory Management in C

In java, all objects are dynamically allocated and unusable objects are recycled automatically by garbage collection.

In C on the other hand, there are no objects, only data structures. Some of these data structures are statically allocated, others dynamically through programmer-inserted directives and dynamically-allocated storage must be **free**d once the data structures there are no longer needed. Not **free**ing memory is a major source of errors.

C Has Pointers

Pointers are special variables that reference (or point to) data in memory (you can think of these as similar to java references).

We’ve already seen pointers in assembly:

Lw $t1,0($s2)

Here $s2 is a pointer, C pointers are basically the same thing.

Built-in Data Types

The normal basic data types are all included:

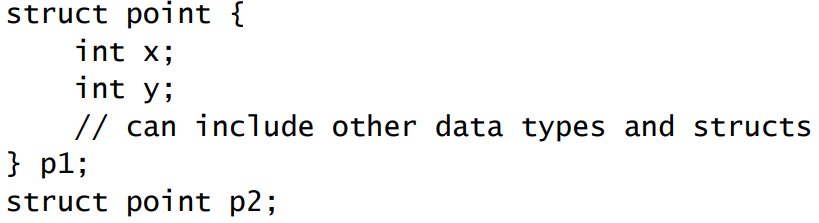
* Char 8 bits
* Short 16 bits
* Int 16, 32, 64 bits (same as machine word size)
* Long 32, 64 bits
* Float 32 bits
* Double 64 bits

Data type sizes are machine dependent (this is not the case in java) and are normally signed, though unsigned types are also available.

Note that there is no boolean type, for any numeric type, 0 means false, anything else means true.

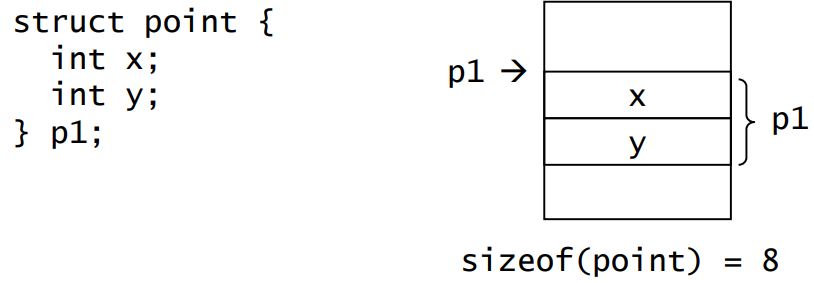
Composite Data Structures

Structures are like objects, but their types have no methods:



Components can be accessed using the ‘.’ Operator: p1.x = 2;

Structures in Memory



(8 because each int takes a word (4 bits), in this case)

p1.y in MIPS would translate to:

addi $t0, $s0, 4 // $s0 points to the start address of p1

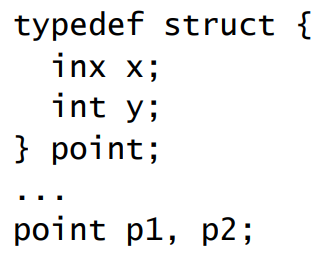
lw $t0, 0($t0) // load p1.y into $t0

User-Defined Types

You can define names for new (or built-int) types using:

Typedef <type> <name>;

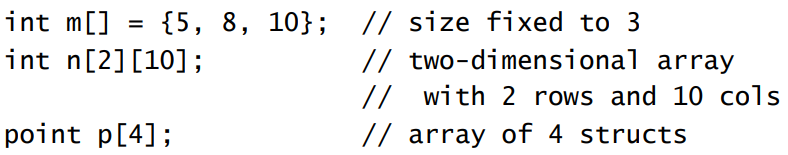
For example:



Arrays

Syntax of C arrays similar to Java, like in Java C arrays have fixed size.

Here is an example of declaring arrays in C



C arrays have no knowledge of their length, and there is a close relationship between arrays and pointers in C, pointers are commonly used to pass arrays between functions.

Strings

C strings are simply arrays of type char that must always end with ‘\0’ (the null character) meaning **char s[10];** can only actually be 9 characters long.

C arrays (for strings) cannot store more chars than reserved at its declaration, but its bounds are not checked (so if you accidentally miss the null character expect a segmentation fault).

Common String Operations

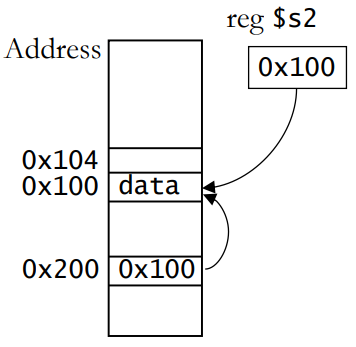
Assignment: **strcpy(s, “string”);** will copy the string starting at the second operand into the first;

Length: **strlen(s);** will return the number of characters starting at s until the next null character.

Comparison **strcmp(s1,s2);** will return 0 when equal, a negative number when s1<s2 and positive when s1 > s2.

To use these functions you must **#include <string.h>**

Pointers

We have seen pointers in assembly: **lw $t1, 0($s2**).$s2 points to the location in memory where the actual data is kept. $s2 is a register but there nothing stopping us from having pointers stored in memory like normal variables.

C Pointers

A C pointer is a variable that holds the address of a piece of data.

Pointers are declared using **\***:

**int \*p;**  creates a pointer to an integer, the compiler needs to know what data type the pointer points to.

Basic pointer usage:

**p = &I; // p now points to the location of I in memory**

**\*p = 5; // This takes p (now pointing to i) and tells it to access the memory and sets it to 5**