What do you think?

Work with a partner(s):

What terms do you think of when you think of C programming?

CSE 374 Lecture 8

Introduction to C

C v. Java

C

- Lower level (closer to assembly)
- No guaranteed memory safety
- Procedural
- Compiled (not interpreted like bash)
- Conditional controls (if, while)
- Modern syntax (human readable)
- Small standard library

Java

- Higher level (lots of compilation)
- Safe (sand-boxed in jvm, compiled limits)
- Object Oriented
- Compiled
- Conditional controls (if, while)
- Modern syntax (human readable)
- Large standard library, huge extended libraries

C v. Scripting

C

- Compiled
- Highly structured, data-typed
- Strings have library processing
- Data structures and libraries
- Good for large complex programs
    - Java, with object-oriented programming, is even better for complex programs

Scripting

- Interpreted
- Esoteric variable access
- Everything is a string
- Easy access to files and program
- Good for quick & interactive programs
    - Do one thing and do it well

C v. Rust

C

- Relies on user management for memory safety
- C++ Exceptions for error management
- C++ extensive std::lib
- Extensive legacy code

Rust

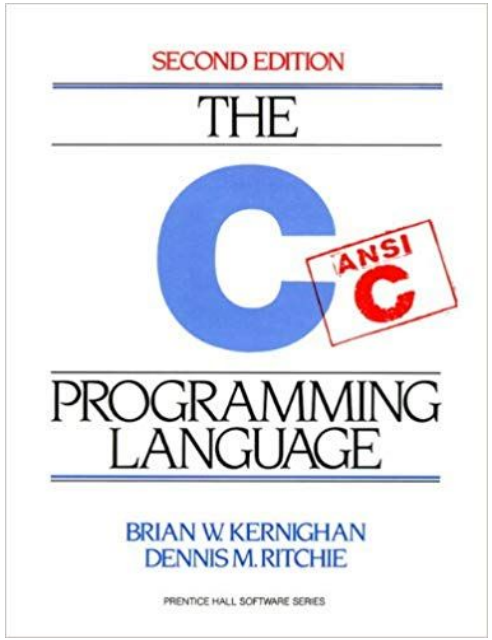
- Built in memory-safety
  - Garbage collection
- Designed for thread safetey
- Uses Result-type
- 30 years younger than C++
  - Growing ecosystem

Why C?

- ➔ C is a fairly compact language - fewer features than Java, but easier to implement efficiently
- ➔ Provides lower level (closer to assembly) language
- ➔ Understanding C can give insight into how computers (and memory) work
- ➔ Still used for
  - ◆ Embedded programming
  - ◆ Systems programming
  - ◆ High-performance code
  - ◆ GPU Programming

C reference books

The standard reference. Available on Kindle and in the UW library.



Essential C - Stanford pdf
http://cslibrary.stanford.edu/101/EssentialC.pdf

http://www.cplusplus.com/

- O'Reilly books (C in a Nutshell, etc.), also through UW library

Hello World in C

```
#include <stdio.h>

/**
 * Compile this file with:
 *      gcc -o hello hello.c
 */
int main(int argc, char **argv)
{
  printf("Hello, World!\n");
  return 0;
}
```

- ➔ Compile: `gcc hello.c`
  - ◆ creates executable `a.out`
- ➔ Or: `gcc -Wall -std=c11 -o hello hello.c`
  - ◆ Wall - turns all warnings on
  - ◆ C11 - specifies using C11 standard libraries
  - ◆ Creates executable `hello`
- ➔ Run: `./a.out` or `./hello`
  - ◆ Exits with ‘0’ (`return 0;`)

```
// includes for functions & types
defined elsewhere
#include <stdio.h>
#include "localstuff.h"
// symbolic constants
#define MAGIC 42
// global variables (if any)
static int days_per_month[ ] = { 31,
28, 31, 30, …};
// function prototypes
// (to handle "declare before use")
 void some_later_function(char, int);
// function definitions
void do_this( ) { … }
char *return_that(char s[ ], int n)
{ … }
int main(int argc, char ** argv) { … }
```

Source File Structures

Hello World in C

```
#include <stdio.h>
#define REPS 5
/**
 * Compile this file with:
 *      gcc -o hello hello.c
 */
int main(int argc, char **argv)
{
  printf("Hello, World!\n");
  return 0;
}
```

- ➔ Include the stdio library (printf, stdout, etc)
- ➔ Other standard libraries
  - ◆ Stdlib, math, assert, etc
- ➔ Also include developer files
  - ◆ #include “myFile.h”

Hello World in C

```
#include <stdio.h>
#define REPS 5
/**
 * Compile this file with:
 *      gcc -o hello hello.c
 */
int main(int argc, char **argv)
{
  printf("Hello, World!\n");
  return 0;
}
```

- ➔ Include the stdio library (printf, stdout, etc)
- ➔ Other standard libraries
  - ◆ Stdlib, math, assert, etc
- ➔ Also include developer files
  - ◆ #include “myFile.h”
- ➔ Preprocessor also defines macros

Hello World in C

```
#include <stdio.h>

/**
 * Compile this file with:
 *     gcc -o hello hello.c
 */
int main(int argc, char **argv)
{
  printf("Hello, World!\n");
  return 0;
}
```

➔ Comment block

- /* long form comments */
- // shorter comments

Hello World in C

```
#include <stdio.h>

/**
 * Compile this file with:
 *      gcc -o hello hello.c
 */
int main(int argc, char **argv)
{
  printf("Hello, World!\n");
  return 0;
}
```

- ➔ C functions look a lot like Java methods.
  - ◆ Have return type, arguments
  - ◆ Code block set off with ‘{‘ and ‘}’
- ➔ Program runs through ‘main’
  - ◆ But not part of class!!
- ➔ Return value - program exit
  - ◆ >> echo “$?”

“Hello, World!\n”

Is a string of length 15 (\n is one character, but contains \0)

In this case, is a ‘string literal’ - evaluates to a global, immutable array.

“printf”

Prints to stdout, which is defined in stdio.h

I/O : Printf, scanf

Printf and scanf are two I/O functions, prototyped in stdio.h

- Printf (print-format)
- int printf(const char *format, ...)
- ‘Format’ is a string that can contain format tags
- + additional arguments to match tags
- Number of arguments better match number of %
- Corresponding arguments better have the right types (%d, int; %f, float; %e, float (prints scientific); %s, \0- terminated char*; … Compiler might check, but not guaranteed
  - best case scenario: you crash
- `printf("%s: %d %g\n", p, y+9, 3.0)`

- scanf (gets input, formatted)
- int scanf(const char *format, ...)
- ‘Format’ is a string that can contain format tags
- + additional arguments to match tags - should be pointers to the right data type so input can be stored in them
- `scanf(“%d %s”, &n, str);`
- `scanf("%*s %d", &a);`
  - %*s ignores string until space, then reads in an integer

Hello World in C

```
#include <stdio.h>
#define REPS 5
/**
 * Compile this file with:
 *      gcc -o hello hello.c
 */
int main(int argc, char **argv)
{
   for (int i=0;i<REPS;i++) {
      printf("Hello, World!\n");
   }
   return 0;
}
```

- ➔ C functions look a lot like Java methods.
  - ◆ Have return type, arguments
  - ◆ Code block set off with ‘{‘ and ‘}’
- ➔ Program runs through ‘main’
  - ◆ But not part of class!!
- ➔ Return value - program exit
  - ◆ >> echo “$?”

Control constructs

Similar to Java: if, while, switch

Break, continue, etc.

https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#Statements

No Boolean type!

Use integers, can declare constants.

Generally, 0/NULL => False

Anything else => True

Or #include <stdbool.h>

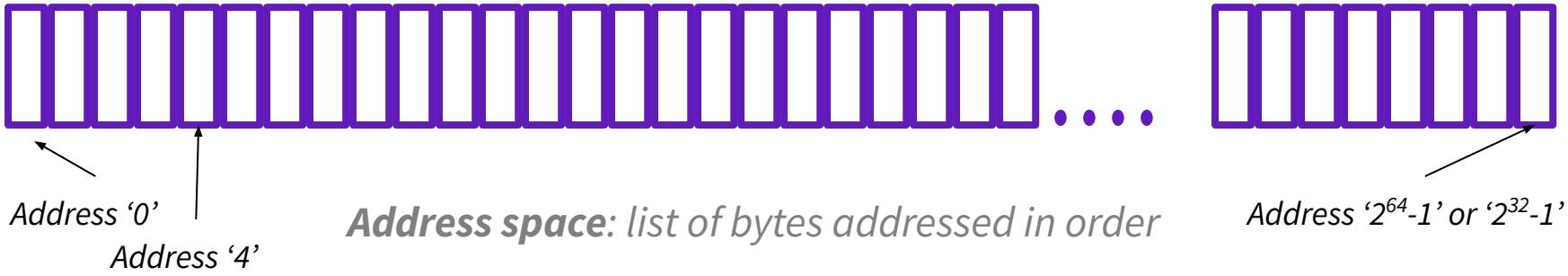
Computers & Memory

CPU - the ‘central processing unit’: computer circuitry that follows computer instructions with simple logic, arithmetic, and I/O

Hard disc storage (modernly often solid state memory instead of traditional drive): holds long-term memory which can persist across re-starts

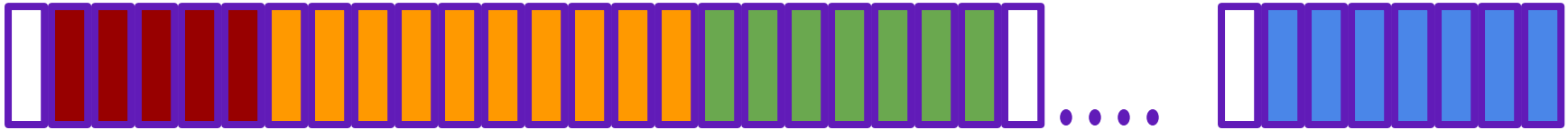
RAM (memory) : where data is stored during operation - short term memory

Working memory.



- Programs are said to have access to this 2^{64} byte space
  - '64 bit' system refers to needing 64 bits to index the space
  - But really don't - many other things are also using this space
- Location in array is the 'address' of a byte
- Programs keep track of addresses of each of their pieces of memory
- Accessing unused address causes a 'segmentation fault'

Working memory, cont.



code *globals* *heap ->* *<- stack*

- Lowest memory stores program instructions, then global variables (static constants, string literals)
- ‘Heap’ holds dynamically allocated variables (‘new’ or ‘malloc’ variables)
- ‘Stack’ holds current instructions, each function in a frame
  - ‘Stack’ memory implies that a frame is added, and then the last frame added is removed first
- The heap and stack grow dynamically. Meet in the middle ?= ‘out of memory’ error

Program address space

Pointers

“Point to memory location”

```
int x = 4;
```

Variable called ‘x’ of type ‘int’ given value of ‘4’

```
int *xPtr = &x;
```

Variable called ‘xPtr’ of type ‘pointer to an integer’, given value of the location of ‘x’

```
int xCopy = *xPtr;
```

Variable called xCopy given the value stored at the location pointed to by xPtr

```
int* noPtr = NULL;
```

Variable ‘noPtr’ correctly set when location is not yet known

Arrays

Contiguous blocks in memory

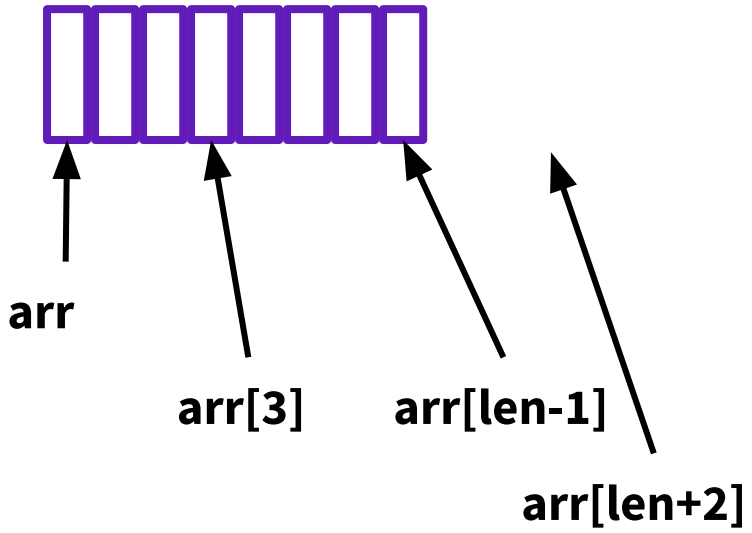
Declare as

```
Datatype arr[len]
```

Has type

```
Datatype*
```

Stores the location in memory of the first value; when arrays are passed passes this memory location



Danger, Will Robinson!!

Strings

No real strings - just arrays of characters.

```
[ "h", "e", "l", "l", "o", " ", "w", "o", "r", "l", "d", "!", \0 ]
```

Strings terminate with \0 so their length can be determined

```
char str[] = "hello";  // array syntax
char *str2 = "hello";  // pointer syntax
char *arrStr[] = {"ant", "bee"}; // array containing char*'s
char **arrStrPtr = arrStr;  // pointer to an array containing char*'s
arrStr[0] = "cat";
```