CSE 413: Intro to C

Laura Effinger-Dean

10/12/07

(slides mostly copied from Dan Grossman)

Welcome to C

Compared to Java:

- low-level
- unsafe
- not object-oriented
- standard library is much smaller

Welcome to C

Compared to Java:

- similar programming style (if statements, for loops, …)
- syntactic similarities (types, variables, parameters, …)

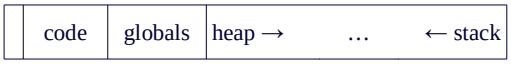
C has different view of the world; more for you to keep track of

Running process – low-level view

- One *address space*
  - huge array of bytes
  - usually 2^{32} bytes, but not guaranteed
  - that's more memory than you really have
  - "address" = position in array
- All code and data for program is in the array
  - program knows the difference
  - can also read/write files, print, take input, etc.

Address space layout

The following varies between systems, but is a good approximation:



- code instructions
- global variables
- heap for other data (like Java's `new`)
- unused portions – access causes "segfault"
- call stack for local variables and code addresses

The stack

- One *activation record* for each function (think method) call that has not yet returned
- Activation record holds:
  - local variables
  - *return address* – position of code to execute when method returns
- Also: function parameters

What could go wrong?

- C may exhibit weird behavior… like what?
  - Trying to access `arr[13]` when `arr` is an array with only 5 elements
  - Try to read an int as if it were a double
  - overwriting the return address
- Correct programs won't do this, but buggy programs may behave unpredictably
- But no array-bounds checks, type checks, etc.

Hello World

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

- Compile with: `gcc –o hello hello.c`
- Run with: ./hello
- Runs `main` with command-line args; program exits when `main` returns
- A lot going on even in this short program!

Hello World

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

- #include copies file stdio.h into hello.c
  - stdio.h defines fputs and stdout
- declaration of main function (very similar to Java methods):
  - return type `int`
  - takes 2 parameters (more on `char**` in a moment)
  - not part of a class, no "implicit parameter" `this`
- `main` is a special function; all executable programs have one

Pointers

- an index into the address space array
- If `x` is a pointer, `*x` is the value it points to
  - or `x[0]`
- If `a` is an array with 2 elements, the second element is `a[1]`, or `*(a+1)`
- "arrays are pointers in C" – not quite, but useful to think of them as the same

Pointers

- Type syntax: `t*` is a pointer to type `t`
  - e.g.: `int *`, `char **`
  - may be `NULL`, i.e. 0
- Array of type `t*` points to zero or more elements of type `t`
  - how many elements? no `arr.length` – must keep track somehow

Pointers

- `int **`: pointer to (zero or more) pointer(s) to (zero or more) int(s)
- So `argv` is a pointer to *j* pointers to (one or more) char(s), where *j* is held in `argc`
  - common idiom: pass array length with array
- one or more because the strings are NULL-terminated: the last character is always `'\0'`
  - common idiom for arrays of characters

Back to Hello World

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

- `fputs` is a function that takes a NULL-terminated string and a `FILE*` (`FILE` is a type defined in stdio.h)
- `"Hello, World\n"` is a global variable – a char array with ??? elements
- `stdout` is a global variable of type `FILE*` - defined in a library