C Memory

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

UNIVERSITY of VIRGINIA | ENGINEERING

Announcements

- Homework 8 releasing soon, due next Monday on Gradescope

C Reference Guide

Functions Definition

```
int f(int x) {
    return 2130 * x;
}
```

- Definition of the function

We only want this in **one** `.c` file

- Do not want 2 definitions
- Which one should the linker choose?

Header Files

C header files: `.h` files

- Written in C, so look like C
- Only put header information in them
  - Function headers
  - Macros
  - `typedef`s
  - `struct` definitions
- Essentially: information for the **type checker** that does not produce any actual binary
- `#include` the header files in our `.c` files

Big Picture

Header files

- Things that tell the type checker how to work
- Do not generate any actual binary

C files

- Function definitions and implementation
- Include the header files

Including Headers

`#include "myfile.h"`

- Quotes: look for a file where I'm writing code
- Our header files

`#include <string.h>`

- Angle brackets: look in the standard place for includes
- Code that came with the compiler
- Likely in `/usr/include`

Macros

```
#define NAME something else
```

- Object-like macro
- Replaces `NAME` in source with `something else`

```
#define NAME(a,b) something b and a
```

- Function-like macro
- Replaces `NAME(X,Y)` with `something Y and X`

Lexical replacement, *not* semantic

Interesting Example

```
#define TIMES2(x)  x * 2          /* bad practice */
#define TIMES2b(x) ((x) * 2)      /* good practice */

int x = ! TIMES2(2 + 3);


int y = ! TIMES2b(2 + 3);
```

Examples and More

- header example
- `string.h`
- variadic functions

Memory

An Interesting Stack Example

```
int *makeArray() {
    int answer[5];
    return answer;
}

void setTo(int *array, int length, int value) {
    for(int i=0; i<length; i+=1)
        array[i] = value;
}

int main(int argc, const char *argv[]) {
    int *a1 = makeArray();
    setTo(a1, 5, -2);
    return 0;
}
```

The Heap

The heap: unorganized memory for our data

- Most code we write will use the heap
- *Not a heap data structure…*

The Heap: Requesting Memory

```
void *malloc(size_t size);
```

- Ask for `size` bytes of memory
- Returns a `(void *)` pointer to the first byte
- It does not know what we will use the space for!
- Does not erase (or zero) the memory it returns

Java

What is the closest thing to malloc in Java?

malloc **man page**

malloc **Example**

```
typedef struct student_s {
    const char *name;
    int credits;
} student;

student *enroll(const char *name, int transfer_credits) {
    student *ans = (student *)malloc(sizeof(student));
    ans->name = name;
    ans->credits = transfer_credits;
    return ans;
}
```

The Heap: Freeing Memory

Freeing memory: `free`

```
void free(void *ptr);
```

- Accepts a pointer returned by `malloc`
- Marks that memory as no longer in use, available to use later
- You should `free()` memory to avoid *memory leaks*