Miscellaneous

CS143: lecture 20

Byron Zhong, July 29

Final Exam

Thursday, August 1, 6:00-8:00pm

- Same format as the quiz
- 1 letter-size, double-sided, note
- Topics:
  - C Basics: types, variables, functions, struct, enum, union
  - Bits: operators, signed/unsigned numbers, bit-packing
  - Pointers: syntax, allocations, arrays, strings
  - Lists: ArrayList, Linked List
  - Sorting: Insertion, Selection, Bubble, Quick, Merge, Heap
  - Dict: BST, Hash table
  - Threads (general concept)
- Review tomorrow (July 30), bring questions

Miscellaneous

- C Preprocessor Macros
- `typedef`
- Python under the hood

Macros

#define -- Define text replacement

```c
#define N_ALPHABET 26

int main(void)
{
    int c;
    int counts[N_ALPHABET] = { 0 };
    int total = 0;

    ...
}
```

clang -E freq.c

```
# 1 "freq.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 445 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "freq.c" 2
# 16 "freq.c"
int main(void)
{
    int c;
    int counts[26] = { 0 };
    int total = 0;

    ...
}
```

Macros

#define

```c
#define STUDNET_MASK 0x70
#define IS_STUDENT(x) x & STUDNET_MASK

int main(void)
{
    uint32_t student = /* student record */;
    if (IS_STUDENT(student)) {
        ...
    }

    return EXIT_SUCCESS;
}
```

clang -E macro.c

```
# 1 "macro.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 445 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "macro.c" 2



int main(void)
{
    uint32_t student = ;
    if (student & 0x70) {
        ...
    }

    return EXIT_SUCCESS;
}
```

Macros

#ifdef #ifndef -- Check if a macro is defined

```c
#define DEBUG

int student_read(FILE *file, stru
{
#ifdef DEBUG
    printf("[DEBUG] reading stude
#endif

    /* READING */

#ifdef DEBUG
    printf("[DEBUG] DONE\n");
#endif

    return 1;
}
```

```c
int student_read(FILE *file, struct student *student_p)
{

    printf("[DEBUG] reading student record ...\n");




    printf("[DEBUG] DONE\n");


    return 1;
}
```

Macros

#ifdef #ifndef

```c
#define DEBUG
#undef DEBUG

int student_read(FILE *file, struct
{
#ifdef DEBUG
    printf("[DEBUG] reading student
#endif

    /* READING */

#ifdef DEBUG
    printf("[DEBUG] DONE\n");
#endif

    return 1;
}
```

```
# 1 "macro.c"
# 1 "<built-in>" 1
# 1 "<built-in>" 3
# 445 "<built-in>" 3
# 1 "<command line>" 1
# 1 "<built-in>" 2
# 1 "macro.c" 2



int student_read(FILE *file, struct student *student_p)
{
# 16 "macro.c"
    return 1;
}
```

Macros

#include -- Copy/paste a file

```
const char *major_name(enum major major)
{
    switch (major) {
        case ANTHROPOLOGY:
            return "Anthropology";
        case ARCHITECTURAL_STUDIES:
            return "Architectural Studies";
        case ART_HISTORY:
            return "Art History";
        case ASTRONOMY_ASTROPHYSICS:
            return "Astronomy and Astrophysics";
        case BIG_PROBLEMS:
            return "Big Problems";
        case BIOLOGICAL_CHEMISTRY:
            return "Biological Chemistry";
        case BIOLOGICAL_SCIENCES:
            return "Biological Sciences";
        ...
    }
}
```

compress.c

Macros

#include -- Copy/paste a file

```
const char *major_name(enum major major)
{
    switch (major) {
        case ANTHROPOLOGY:
            return "Anthropology";
        case ARCHITECTURAL_STUDIES:
            return "Architectural Studies";
        case ART_HISTORY:
            return "Art History";
        case ASTRONOMY_ASTROPHYSICS:
            return "Astronomy and Astrophysics";
        case BIG_PROBLEMS:
            return "Big Problems";
        case BIOLOGICAL_CHEMISTRY:
            return "Biological Chemistry";
        case BIOLOGICAL_SCIENCES:
            return "Biological Sciences";
        ...
    }
}
```

compress.c

```
case ANTHROPOLOGY:
    return "Anthropology";
case ARCHITECTURAL_STUDIES:
    return "Architectural Studies";
case ART_HISTORY:
    return "Art History";
case ASTRONOMY_ASTROPHYSICS:
    return "Astronomy and Astrophysics";
case BIG_PROBLEMS:
    return "Big Problems";
case BIOLOGICAL_CHEMISTRY:
    return "Biological Chemistry";
case BIOLOGICAL_SCIENCES:
    return "Biological Sciences";
...
```

cases.c

Macros

#include -- Copy/paste a file

```
const char *major_name(enum major major)
{
    switch (major) {
#include "cases.c"
    }
}
```

compress.c

```
case ANTHROPOLOGY:
    return "Anthropology";
case ARCHITECTURAL_STUDIES:
    return "Architectural Studies";
case ART_HISTORY:
    return "Art History";
case ASTRONOMY_ASTROPHYSICS:
    return "Astronomy and Astrophysics";
case BIG_PROBLEMS:
    return "Big Problems";
case BIOLOGICAL_CHEMISTRY:
    return "Biological Chemistry";
case BIOLOGICAL_SCIENCES:
    return "Biological Sciences";
...
```

cases.c

Macros

#include -- Copy/paste a file

- This is how `#include <stdio.h>` works!
- Demo

Macros

#include guard

```
#ifndef DICT_H_
#define DICT_H_

struct dict;

...

#endif
```

Why do we do this?

- If `DICT_H_` is not defined, do the following:
  - Define `DICT_H_`
  - Declare types/functions
- This prevents us from `#include`ing the same header more than once.
  - Avoids double-declaring the same types/functions

Macros

__FILE__ __LINE__

```c
#include <stdio.h>

int main(void)
{
    printf("This is %s:%d\n", __FILE__, __LINE__);
    return 0;
}
```

```
% clang -o macro macro.c
% ./macro
This is macro.c:5
%
```

Macros

- Macros are one of the simplest forms of *meta-programming.*
  - Manipulate the text of a program (i.e., *programming* a program)
- A powerful tool for encapsulation:
  - Hiding uninteresting details from readers
- Use them with care:
  - Hard to trace what is going on, hard to debug

Typedef

Give a type another name

```
typedef uint32_t student;
typedef uint32_t record;
typedef uint32_t number;

typedef old_type new_name;

int main(void)
{
    student x = ...;
    record x = ...;
    number x = ...;
}
```

- Self documentation:
  - Distinguishes different interpretations of the same underlying representation
  - `student` and `number` should be treated differently.

Typedef

Give a type another name

```
typedef struct student_record student_record;
typedef struct student_record sr;
```

- Self documentation:
  - Distinguishes different interpretations of the same underlying representation
  - `student` and `number` should be treated differently.
- Simplification:
  - Common pattern to avoid writing the `struct` keyword
  - Give a shortened name for some repetitive use

Python under the hood

- Python is implemented in C.
  - Python, the language, is a dynamic scripting language.
  - `python3`, the program that reads and executes Python code, is written in C.
  - There are other not-so-popular implementations of Python as well.
- In `python3`, every Python object is represented by a C structure.
  - modules, types, functions, variables, ...
  - are all `structs`

Python under the hood

- Internally, Python uses hash tables for variables:
  - Keys: names (`char *`)
  - Values: `PyObject *`
  - `x = 3`
  - `insert(vartable, "x", PyLong_FromLong(3));`

Python under the hood

Demo: Calling C from Python

- We can implement a function in C and use it in Python
- Setup: Image saturation adjustment
  - Given an image, adjust its color saturation.
  - Colorspace conversion

Python under the hood

Background: color-space conversion

- RGB colors: Use three integers to represent different intensities of red, green and blue to represent a color.
- HSV colors: Use Hue, Saturation, and Lightness to represent a color