

15-213 Recitation: Cache Lab & C

Jack Biggs

16 Feb 2015

Agenda

- **Buffer Lab!**
- C Exercises!
- C Conventions!
- C Debugging!
- Version Control!
- Compilation!

Buffer Lab...

Is due soon. So maybe do it... soon

Agenda

- Buffer Lab!
- **C Exercises!**
- C Conventions!
- C Debugging!
- Version Control!
- Compilation!

C-1

```
int main() {
    int* a = malloc(100*sizeof(int));
    for (int i=0; i<100; i++) {
        if (a[i] == 0) a[i]=i;
        else a[i]=0;
    }
    free(a);
    return 0;
}
```

C-1

```
int main() {  
    int* a = malloc(100*sizeof(int));  
    for (int i=0; i<100; i++) {  
        if (a[i] == 0) a[i]=i;  
        else a[i]=0;  
    }  
    free(a);  
    return 0;  
}
```

No value in `a` was initialized. The behavior of `main` is undefined.

C-2

```
int main() {  
    char w[strlen("C programming")];  
    strcpy(w, "C programming");  
    printf("%s\n", w);  
    return 0;  
}
```

C-2

```
int main() {  
    char w[strlen("C programming")];  
    strcpy(w, "C programming");  
    printf("%s\n", w);  
    return 0;  
}
```

`strlen` returns the length of the string not including the null character, so we end up writing a null byte outside the bounds of `w`.

C-3

```
struct ht_node {
    int key;
    int data;
};
typedef struct ht_node* node;

node makeNnode(int k, int e) {
    node curr = malloc(sizeof(node));
    node->key = k;
    node->data = e;
    return node;
}
```

C-3

```
struct ht_node {
    int key;
    int data;
};
typedef struct ht_node* node;
```

```
node makeNnode(int k, int e) {
    node curr = malloc(sizeof(node));
    curr->key = k;
    curr->data = e;
    return curr;
}
```

node is a typedef to a struct ht_node pointer, not the actual struct. So malloc could return 4 or 8 depending on system word size.

C-4

BOOM!

The C-4 has blown up.

*Your instructor has
been notified. :(*

C-5

```
char *strcdup(int n, char c) {  
    char dup[n+1];  
    int i;  
    for (i = 0; i < n; i++)  
        dup[i] = c;  
    dup[i] = '\0';  
    char *A = dup;  
    return A;  
}
```

C-5

```
char *strcdup(int n, char c) {  
    char dup[n+1];  
    int i;  
    for (i = 0; i < n; i++)  
        dup[i] = c;  
    dup[i] = '\0';  
    char *A = dup;  
    return A;  
}
```

strcdup returns a stack-allocated pointer. The contents of A will be unpredictable once the function returns.

C-6

```
#define IS_GREATER(a, b) a > b
inline int isGreater(int a, int b) {
    return a > b ? 1 : 0;
}
int m1 = IS_GREATER(1, 0) + 1;
int m2 = isGreater(1, 0) + 1;
```

C-6

```
#define IS_GREATER(a, b) a > b
inline int isGreater(int a, int b) {
    return a > b ? 1 : 0;
}
int m1 = IS_GREATER(1, 0) + 1;
int m2 = isGreater(1, 0) + 1;
```

IS_GREATER is a macro that doesn't end in a semicolon. **m1** will actually evaluate to 0, since $1 > 0+1 = 0$.

C-7

```
#define NEXT_BYTE(a) ((char*) (a + 1));
```

```
long a1 = 54; // &a1 = 0x100
```

```
int a2 = 42; // &a2 = 0x200
```

```
void* b1 = NEXT_BYTE(&a1);
```

```
void* b2 = NEXT_BYTE(a2);
```


C-7

```
#define NEXT_BYTE(a) ((char*)(a + 1));
```

```
int a1 = 54; // &a1 = 0x100
```

```
long long a2 = 42; // &a2 = 0x200
```

```
void* b1 = NEXT_BYTE(&a1);
```

```
void* b2 = NEXT_BYTE(&a2);
```

b1 is a void pointer to the address 0x104.

b2 is a void pointer to the address 0x108.

C Workshop

- If you had trouble with the previous exercises, **go!!!**
- Location, Time TBA (Check your email!)
- Material:
 - Structs, pointers
 - Memory management
 - Standard library functions
 - Random stuff: macros, typedefs, function pointers, header guards... and anything else you have questions on!

Agenda

- Buffer Lab!
- C Exercises!
- **C Conventions!**
- C Debugging!
- Version Control!
- Compilation!

The C Standard Library

- Use it. It is your friend!
- Don't write code that's already been written!
 - Your work might have a bug or lack features
- All C Standard Library functions are documented.
 - Use the UNIX `man` command to look up usage

Robustness

- Code that crashes is bad.
 - Avoid making bad things!
 - Check for failed system calls and invalid input
- Some errors should be recoverable, others not
 - Proxy Lab is an excellent example of this
- Free memory that you allocate
 - Leaky code will crash (and code that crashes is bad!)
 - Memory leaks will cost you style points

Robustness: Continued

- CSAPP wrappers check return values of system calls
 - Terminate program when error is encountered
 - Malloc, Free, Open, Close, Fork, etc.
 - Super duper useful for Proxy & Shell Labs
- Alternatively, check for error codes yourself
 - Useful when you don't want program to terminate

```
FILE *pfile; // file pointer
if (!(pfile = fopen("myfile.txt", "r"))) {
    printf("Could not find file. Opening default!");
    pfile = fopen("default.txt", "r");
}
```

Quick C Tip: `getopt`

- Used for parsing command-line arguments
- **Don't write your own code for this.** Not worth it.
 - In fact, we *strongly* discourage it
 - Autograder randomizes argument order to make this task more difficult (seriously, just use `getopt`)
 - Try it: `man getopt`

getopt: a quick example

```
int main(int argc, char *argv[]) {  
    while ((opt = getopt(argc, argv, "nt:")) != -1) {  
        switch (opt) {  
            case 'n':  
                flags = 1;  
                break;  
            case 't':  
                nsecs = atoi(optarg);  
                tfnd = 1;  
                break;  
            case '?':  
                if (optopt == 'c') printf ("-c requires argument\n");  
                else printf (usage_string);  
                exit(1);  
        }  
    }  
}
```

arg string

Things without colons in the arg string can be used for setting flags

Things with colons in the arg string can specify options; the option is placed in a variable called 'optarg'

If the user does something wrong, we go to the '?' case

The case we're in is placed in a variable called 'optopt'

Style Points

- We read and grade your code for style
 - Style guide: <http://cs.cmu.edu/~213/codeStyle.html>
 - Vim macro to highlight lines longer than 80 cols:
 - `2mat ErrorMessage '\%80v.'`
 - Emacs users... this is why your editor sucks:

```
(setq whitespace-style '(trailing lines space-  
before-tab indentation space-after-tab)  
whitespace-line-column 80)
```
- View your annotated code on Autolab

Agenda

- Buffer Lab!
- C Exercises!
- C Conventions!
- **C Debugging!**
- Version Control!
- Compilation!

gdb

- Step through C code side-by-side with Assembly
 - Print variables, not just registers and addresses!
 - Break at lines, not just addresses and functions!
- `gdbtui <binary>` is gdb with a less-breakable user interface.
 - Nice for looking at your code during execution
 - Type `layout split` to view Assembly alongside
 - `layout cmd, layout asm, layout source`
 - **Change windows:** `focus <cmd, asm, source>`

gdbtui

```

test.c
1
2     int foo(int x, int y, char z) {
> 3     int i = x*y;
4         i ^= x - z;
5         i &= y;
6         return i*z-y;
7     }
8
9     int main() {
B+ 10    return foo(23, 583, 'x');

```

```

0x4004b3 <foo+7>      mov     %esi,-0x18(%rbp)
0x4004b6 <foo+10>     mov     %edx,%eax
0x4004b8 <foo+12>     mov     %al,-0x1c(%rbp)
> 0x4004bb <foo+15>     mov     -0x14(%rbp),%eax
0x4004be <foo+18>     imul   -0x18(%rbp),%eax
0x4004c2 <foo+22>     mov     %eax,-0x4(%rbp)
0x4004c5 <foo+25>     movsbl -0x1c(%rbp),%eax
0x4004c9 <foo+29>     mov     -0x14(%rbp),%edx
0x4004cc <foo+32>     mov     %edx,%ecx
0x4004ce <foo+34>     sub     %eax,%ecx

```

```

child process 25783 In: foo
Reading symbols from /home/jack/test...done.
(gdb) layout split

```

valgrind

- Best tool for finding...
 - Memory leaks
 - Other memory errors (like double frees)
 - Memory corruption
- Use `gcc` with `-g` to give you line numbers of leaks
- Use `valgrind --leak-check=full` for thoroughness

Using `printf` wisely

- Don't spend your time inserting and removing `printf`s. Disable/enable them all easily using `#define`
- Leave them in for when you want to play with it later

```
#define DEBUG
#ifdef DEBUG
#define dbg_printf(...) printf(__VA_ARGS__)
#else
#define dbg_printf(...)
#endif
```

Inserting line numbers into debug statements

- Check a condition repeatedly and know which line number it failed on

```
static inline check_list(ll_node_t *curr, int lineno) {
    if (curr == NULL
        || curr->next == NULL
        || curr->next != curr->next->prev)
        dbg_printf("Check failed on line %d\n", lineno);
}
...
check_list(node_ptr, __LINE__)
```

Agenda

- Buffer Lab!
- C Exercises!
- C Conventions!
- C Debugging!
- **Version Control!**
- Compilation!

Version Control: Your Friend

- Initialize and add files to version tracking program
 - “Commit” your files as you reach checkpoints
 - Rewind and fast-forward between commits
 - Keep different program versions in branches
- `malloc`: branches for implicit, explicit, seglist
- Can sometimes be tricky to use

For dummies: Folders

- Keep different versions in different folders
 - As one stage of project is finished, copy to folder
 - Useful for trying different implementations
- Easy to use! Gets cluttered *fast*.
 - No easy way to merge changes between folders

git

- `git init` initializes a new repository in present folder
- `git status` shows files being tracked by version control
- `git add <file or folder>` adds to version tracking
- `git commit -am "message"` commits with note "message"
- Look online for more examples
 - Create branches for different implementations
 - Merge branches, rollback to older changes

git pitfalls

- Be cautious when rewinding commits
- Follow online usage instructions carefully
- Stack Overflow, <http://try.github.io>, man git

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT
MESSAGES GET LESS AND LESS INFORMATIVE.

Agenda

- Buffer Lab!
- C Exercises!
- C Conventions!
- C Debugging!
- Version Control!
- **Compilation!**

gcc

- Open source C compiler
- We will give you instructions for compilation in handouts
- `man gcc`, [Stack Overflow](#) if you're having trouble

make

- Lab handouts come with a `Makefile`
 - Don't modify them
- You write your own for Proxy Lab
 - Examples for syntax found in previous labs
- `make` reads `Makefile` and compiles your project
- Easy way to automate tedious shell commands