15-213 Recitation: Learn to C

29 Sep 2014
Arjun Hans

Agenda

- C-Assessment
- Best Practices
- Debugging Tools
- Version Control
- Compilation
- Demo

C Assessment

C Assessment

- Can you solve *all* of these upcoming C-exercises *effortlessly*?
  - These problems test *fundamental* C-concepts.
- If not, *please* come to the C-bootcamp
  - Wednesday October 1st, 8-10 PM, Rashid Auditorium
- Syllabus Details:
  - Types: Pointers/Structs
  - Memory Management: Malloc/Free, Valgrind
  - Common library functions: string.h, stdlib.h, stdio.h
  - Grab-bag: macros, typedefs, function-pointers, header-guards
- Make the investment *now*

Exercise 1a): Can you find the bug?

```
#include <stdlib.h>

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;
}
```

```
#include <stdio.h>
#include <string.h>
int main() {
  char w[strlen("C programming")];
  strcpy(w,"C programming");
  printf("%s\n", w);
  return 0;
}
```

Exercise 1b): Can you find the bug?

```
#include <stdlib.h>
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 curr;
}
```

```
// called from other functions
// return value is used
char *strcdup(int n, char c) {
  char dup[n+1];
  for (int i = 0; i < n; i++)
    dup[i] = c;
  dup[i] = '\0';
  char *A = dup;
  return A;
}
```

Exercise 2a): Grab-bag

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

What is m1? What is m2?

```
#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);
```

What is b1? What is b2?

Exercise 2b): Grab-bag

```
char* nextChar(char* str) {
  for (int i = 0; i < strlen(str); i++) {
    str[i] = str[i] + 1;
  }
  return str;
}
```

Good code or bad code?

```
int A[40][30];
int **B = malloc(sizeof(int*) * 40);
for (size_t i = 0; i < 30; i++)
  B[i] = malloc(sizeof(int) * 30);
```

Is sizeof(A) == sizeof(B)?

Best Practices

Libraries, Robustness, Style Guide

Standard Libraries

- Get comfortable with commonly used libraries
  - Improve your efficiency
  - Avoid redundancy

- stdlib.h: malloc, calloc, free, exit, atoi, abs, etc
- string.h: strlen, strcpy, strcmp, strstr, memcpy, memset, etc
- stdio.h: printf, scanf, sscanf, etc

- Use man/online references to learn their usage

Keep it Robust

- We are writing code for the real world: errors will happen
  - system calls may fail
  - user may enter invalid arguments
  - connections may die
  - … but your code should NOT crash!
- Handle errors gracefully
  - Indicate when errors happen
  - May be recoverable, may have to terminate
  - Remember to free any resources in use
  - Else, suffer the wrath of a thousand unicorns
  - … and our sadistic style-grading.

Solution 1: Use CSAPP Wrappers!

- http://csapp.cs.cmu.edu/public/1e/ics/code/src/csapp.c
- Has wrapper methods for all core system calls
  - Explicitly checks for return values
  - Calls unix_error if something went wrong

```
void *Malloc(size_t size)
{
   void *p;
   if ((p  = malloc(size)) == NULL)
      unix_error("Malloc error");
   return p;
}
```

- Copy/paste required wrappers in source code, since we will accept only single files.
- Definitely include this file in your proxy lab submission!

Solution 2: Check the values yourself!

- Example: file IO functions
  - fopen: open a given file in a given mode (read/write/etc)
  - fclose: close file associated with given stream
  - fscanf: read data from the stream, store according to parameter format
- Error-codes:
  - fopen: return NULL
  - fclose: EOF indicated
  - fscanf: return fewer matched arguments, set error indicator
- May be useful for Cache lab!

```
FILE *pfile; // file-pointer
if (!(pfile = fopen("myfile.txt", "r"))) {
  printf("Could not find file");
  exit(EXIT_FAILURE);
}
char c;
int x;
while (fscanf(pfile, "%c %d\n", &c, &x) > 0) {
  if (ferror (pFile)) {
    printf ("Error reading from file\n");
    exit(EXIT_FAILURE);
  printf("%c %d\n", c, x);
}
fclose(pfile);
```

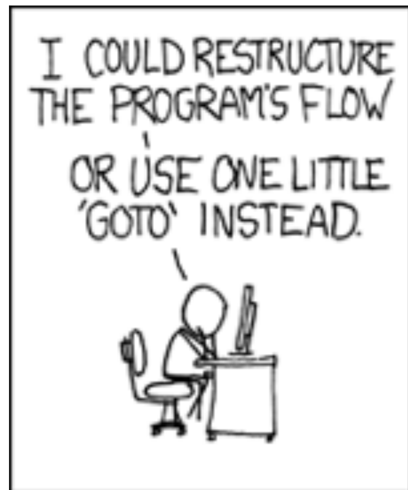
More Examples: getopt

- Used to parse command-line arguments.
- Typically called in a loop to retrieve arguments
- Switch statement used to handle options
  - colon indicates required argument
  - optarg is set to value of option argument
- Returns -1 when no more arguments present
- May be useful for Cache lab!

```
int main(int argc, char** argv){
  int opt, x;
  /* looping over arguments */
  while(-1 != (opt = getopt(argc, argv, “x:"))){
    switch(opt) {
      case 'x':
        x = atoi(optarg);
        break;
      default:
        printf(“wrong argument\n");
        break;
    }
  }
}
```

Style Guide

- We will *read* your code. We will *grade* your code.
- Read the style guide: http://www.cs.cmu.edu/~213/codeStyle.html
- … then read it again



Style Guide Principles

- Consistency:
  - Whitespace: use for all code-blocks, don’t mix tabs/single-spaces
  - Variable Names: camelCase, hyphens, etc
  - Curly-braces: Allman, K&R
  - Bottom-line: choose a set of conventions and *stick to them.*

- Clarity:
  - Documentation: approach, algorithmic thinking, etc
  - Naming: variable/function names should indicate their usage
  - Modularity: use helper functions profusely, avoid duplicated code
  - Line-length: 80 characters and no more
  - No magic numbers; use macros/cons

Version Control

Git Introduction

Version Control

- You should use it. Now.
- Avoid suffering during large labs (malloc, proxy)
- Basic ideas:
  - complete record of everything that happened in your code repository
  - ability to create branches to test new components of code
  - ease in sharing code with other.
- A skill that will pay you dividends in the future

Version Control Basics (Git)

- git init:
  - Create a new repository
  - Indicated by .git file
- git status:
  - Show working tree-status
  - Untracked files, staged files
- git add <file_name>
  - Stage a file to be committed (does *not* perform the commit)
  - git add . stages all files in current directory
- git commit
  - Make a commit from all the stage files
  - git commit -m “Commit message”

Distributing your Source

- Should probably also use a website for hosting a remote repository (github, bitbucket)
  - MUST ensure that your repository is PRIVATE
- git push:
  - Pushes the local repository to a remote repository
- git pull:
  - Pushes the local repository to a remote repository
- git clone:
  - Clone a repository into a new directory
  - git clone <online-repo-name>

Other Git stuff

- Git is complicated; be careful
- Run into a problem, look it up
  - StackOverflow
  - Github
  - http://git-scm.com/docs/
  - man pages
- Some online tutorials:
  - http://pcottle.github.io/learnGitBranching/
  - https://try.github.io/

	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
	AAAAAAAA	3 HOURS AGO
	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
	MY HANDS ARE TYPING WORDS	2 HOURS AGO
	HAAAAAAAANDS	2 HOURS AGO

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

Debugging

GDB, Valgrind

GDB

- No longer stepping through assembly!
  - Use the step/next commands
  - break on line numbers, functions
  - Use list to display code at line-numbers and functions
  - Use print with variables

- Use gdbtui
  - Nice display for viewing source/executing commands

Valgrind

- Find memory errors, detect memory leaks
- Common errors:
  - Illegal read/write errors
  - Use of uninitialized values
  - Illegal frees
  - Overlapping source/destination addresses
- Typical solutions
  - Did you allocate enough memory?
  - Did you accidentally free stack variables/something twice?
  - Did you initialize all your variables?
  - Did use something that you just free'd?
- --leak-check=full
  - Memcheck gives details for each definitely/possibly lost memory block (where it was allocated

Compilation

GCC, Make Files

GCC

- Used to compile C/C++ projects
  - List the files that will be compiled to form an executable
  - Specify options via flags
- Important Flags:
  - -g: produce debug information (**important**; used by GDB/valgrind)
  - -Werror: treat all warnings as errors (this is our **default**)
  - -Wall/-Wextra: enable all construction warnings
  - -pedantic: indicate all mandatory diagnostics listed in C-standard
  - -O1/-O2: optimization levels
  - -o <filename>: name output binary file 'filename'
- Example:
  - gcc -g -Werror -Wall -Wextra -pedantic foo.c bar.c -o baz

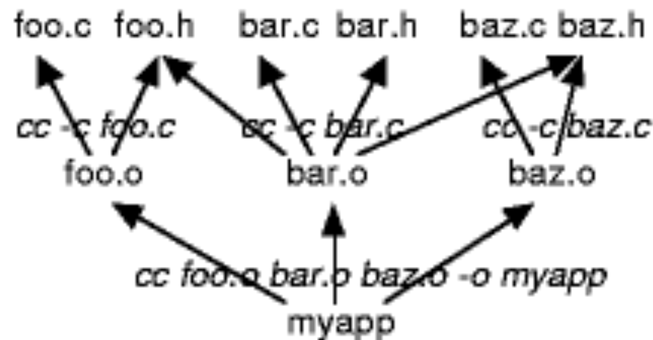
Make Files

- Command-line compilation becomes inefficient when compiling many files together
- Solution: use make-files
  - Single operation to compile files together
  - Only recompiles updated files

```
# Makefile for the malloc lab driver
#
CC = gcc
CFLAGS = -Wall -Wextra -Werror -O2 -g -DDRIVER -std=gnu99

OBJS = mdriver.o mm.o memlib.o fsecs.o fcyc.o clock.o ftimer.o

all: mdriver

mdriver: $(OBJS)
	$(CC) $(CFLAGS) -o mdriver $(OBJS)

mdriver.o: mdriver.c fsecs.h fcyc.h clock.h memlib.h config.h mm.h
memlib.o: memlib.c memlib.h
mm.o: mm.c mm.h memlib.h
fsecs.o: fsecs.c fsecs.h config.h
fcyc.o: fcyc.c fcyc.h
ftimer.o: ftimer.c ftimer.h config.h
clock.o: clock.c clock.h

clean:
	rm -f *~ *.o mdriver
```

Make File Rules

- Comments start with a ‘#’, Commands start with a TAB.
- Common Make File Format:
  - target: source(s)
    TAB: command
    TAB: command
- Macros: similar to C-macros, find and replace:
  - CC = gcc
    CCOPT = -g -DDEBUG -DPRINT
    foo.o: foo.c foo.h
        $(CC) $(CCOPT) -c foo.c
- See http://www.andrew.cmu.edu/course/15-123-kesden/index/lecture_index.html for more details



Demo Time!

Putting it all together

Questions?

Credits

- Inspired by slides from previous semesters, by Art Chang, Anita Zhang, Peyton Randolph, Brandom Lum
- C-assessment questions taken from 15-122 HW theory problems
- Library function descriptions taken from http://www.cplusplus.com/reference