CS100 Introduction to Programming

Recitation 2

llk89

NO PLAGIARISM!!!

- The most likely cause for failing this course.
- You WILL be caught!
- We WILL punish!
- They WILL know!
  - Parents
  - University
  - School
  - Fellows
- <u>We have already found a dozen of you cheating.</u>

Code Smell

Every recitation will have this section

Code smell

- You already know this won't work every time

```
int times,a[100],b[100],c[100],d[100];
float mood=100,gap[100];
char y;
```

- We don't grade on how small your source file is

```
int n,i,a[100],b[100],c[100]={0},d[100]={0},e[100],f[100];
float mood=100.0;
```

- This is causing san check every now and then

```
  a=(q+r+u)/3;
  b=(w+t+i)/3;
  m=sqrt(     (q-a)*(q-a)   +  (w-b)*(w-b)               );
  n=sqrt(     (q-a)*(q-a)   +  (w-b)*(w-b)               );
  m=max(m,  sqrt(     (r-a)*(r-a)   +  (t-b)*(t-b)               )   );
  n=min(n,  sqrt(     (r-a)*(r-a)   +  (t-b)*(t-b)               )      );
m=max(m,  sqrt(     (u-a)*(u-a)   +  (i-b)*(i-b)               )   );
  n=min(n,  sqrt(     (u-a)*(u-a)   +  (i-b)*(i-b)               )   );
```

Code smell

- No OJ oriented programming. Just NO. And don’t indent with 3 spaces.

```
max>a?max=max:max=a;
min<a?min=min:min=a;
```

```
sacnf("%f",&save);
```

- Do not use home made abbreviations. Plus, we are still using C.

```
inline void ckmx(double &a, double b) {
    if (a < b) a = b;
}
inline void ckmi(double& a, double b) {
    if (a > b) a = b;
}
```

Code smell

- Don’t repeat yourself. (a.k.a. DRY)

```
buffer[0]=q;
buffer[1]=0;
buffer[2]=0;
buffer[3]=0;
buffer[4]=0;
buffer[5]=0;
buffer[6]=0;
buffer[7]=0;
```

Code smell

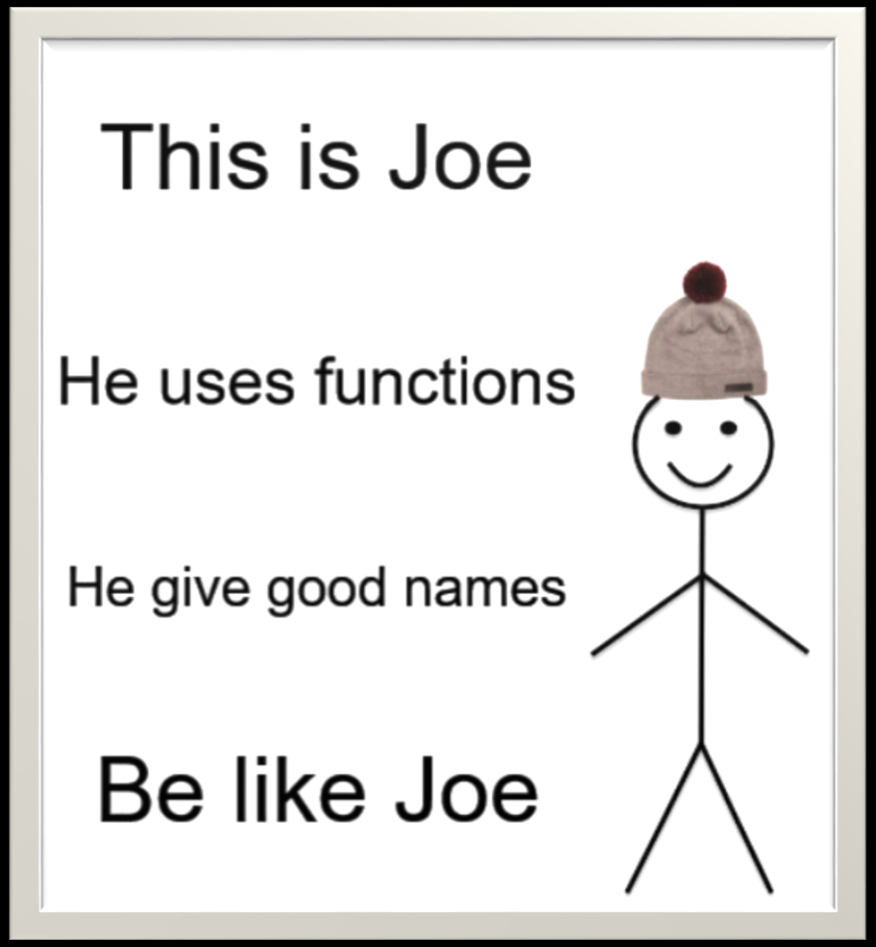
- Global variables are global. Keep everything to local unless absolutely necessary.

```
#include<memory.h>
int SUM1 = 1, SUM2 = 1, ZAN = 0, SHI = 0, k = 1
, CHA, i = 1, j, TOT = 0, NE = 0, XT = 0, ok, i
2, n = 1, p, PRI;

int main(void)
{
    int* A = NULL;
    int* B = NULL;
    int NUMBER = 5;
    // ......
```

Be like Joe

- Some of you deserve a big thumbs up.
- His code quality is not perfect, but is definitely among the top.
- Some other submissions were considered, we choose this with the help of RNG.
- We decided not to disclose his name, nor alert him beforehand.
  - This is a honor. Do not be ashamed nor worried.



Be like Joe

```c
float drop(float a,int b)
{
    if(b<=60)
        a=a-b*0.4;
    else
        a=a-24.0-(b-60.0)*0.8;
    return a;
};
float recover(float a,int b)
{
    a=a+b*0.5;
    if(a>100)
        a=100.0;
    return a;
};
```

```c
for(n;n>0;n--)
{
    scanf("%d:%d-%d:%d",&h1,&m1,&h2,&m2);
    time=TimeInterval(temph1,tempm1,h1,m1);
    mood=recover(mood,time);
    time=TimeInterval(h1,m1,h2,m2);
    mood=drop(mood,time);
    temph1=h2;
    tempm1=m2;
    if(mood<=0)
    {
        printf("…… hospital.");
        goto loop;
    }
}
time=TimeInterval(h2,m2,22,00);
mood=recover(mood,time);
if(mood>0)
    printf("…… is %.1f at the……",mood);
```

*Slightly edited to fit in the slides
Variable declarations and some unimportant details omitted

Overview

- Pointers
- Memory allocation
- Arrays
- Functions
- String functions
- Basic file IO
- ed--

Pointers

What it is?

```
#include <stdio.h>
int main(void) {
    int num1 = 3, num2 = 5;
    int *ptr1, *ptr2;
    ptr1 = &num1;
    (*ptr1)++;
    ptr2 = &num2;
```

Name	Address	Value
num1	0xffff0000	3
num2	0xffff0004	5
	0xffff0008	
	0xffff000c	
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

What it is?

```
#include <stdio.h>
int main(void) {
    int num1 = 3, num2 = 5;
    int *ptr1, *ptr2;
    ptr1 = &num1;
    (*ptr1)++;
    ptr2 = &num2;
```

Name	Address	Value
num1	0xffff0000	3
num2	0xffff0004	5
ptr1	0xffff0008	
ptr2	0xffff000c	
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

What it is?

```c
#include <stdio.h>
int main(void) {
    int num1 = 3, num2 = 5;
    int *ptr1, *ptr2;
    ptr1 = &num1;
    (*ptr1)++;
    ptr2 = &num2;
```

Name	Address	Value
num1	0xffff0000	3
num2	0xffff0004	5
ptr1	0xffff0008	0xffff0000
ptr2	0xffff000c	
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

What it is?

```c
#include <stdio.h>
int main(void) {
    int num1 = 3, num2 = 5;
    int *ptr1, *ptr2;
    ptr1 = &num1;
    (*ptr1)++;
    ptr2 = &num2;
```

Name	Address	Value
num1	0xffff0000	4
num2	0xffff0004	5
ptr1	0xffff0008	0xffff0000
ptr2	0xffff000c	
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

What it is?

```c
#include <stdio.h>
int main(void) {
    int num1 = 3, num2 = 5;
    int *ptr1, *ptr2;
    ptr1 = &num1;
    (*ptr1)++;
    ptr2 = &num2;
```

Name	Address	Value
num1	0xffff0000	4
num2	0xffff0004	5
ptr1	0xffff0008	0xffff0000
ptr2	0xffff000c	0xffff0004
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

What it is?

```
    *ptr2 = *ptr1;
    *ptr2 = 10;
    num1 = *ptr2;
    *ptr1 = *ptr1 * 5;
    ptr2 = ptr1;
    return 0;
}
```

Name	Address	Value
num1	0xffff0000	4
num2	0xffff0004	4
ptr1	0xffff0008	0xffff0000
ptr2	0xffff000c	0xffff0000
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

What it is?

```
    *ptr2 = *ptr1;
    *ptr2 = 10;
    num1 = *ptr2;
    *ptr1 = *ptr1 * 5;
    ptr2 = ptr1;
    return 0;
}
```

Name	Address	Value
num1	0xffff0000	4
num2	0xffff0004	10
ptr1	0xffff0008	0xffff0000
ptr2	0xffff000c	0xffff0004
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

What it is?

```
    *ptr2 = *ptr1;
    *ptr2 = 10;
    num1 = *ptr2;
    *ptr1 = *ptr1 * 5;
    ptr2 = ptr1;
    return 0;
}
```

Name	Address	Value
num1	0xffff0000	10
num2	0xffff0004	10
ptr1	0xffff0008	0xffff0000
ptr2	0xffff000c	0xffff0004
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

What it is?

```
    *ptr2 = *ptr1;
    *ptr2 = 10;
    num1 = *ptr2;
    *ptr1 = *ptr1 * 5;
    ptr2 = ptr1;
    return 0;
}
```

Name	Address	Value
num1	0xffff0000	50
num2	0xffff0004	10
ptr1	0xffff0008	0xffff0000
ptr2	0xffff000c	0xffff0004
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

What it is?

```
    *ptr2 = *ptr1;
    *ptr2 = 10;
    num1 = *ptr2;
    *ptr1 = *ptr1 * 5;
    ptr2 = ptr1;
    return 0;
}
```

Name	Address	Value
num1	0xffff0000	50
num2	0xffff0004	10
ptr1	0xffff0008	0xffff0000
ptr2	0xffff000c	0xffff0000
	0xffff0010	
	0xffff0014	
	…	

Assume 32 bit machine

More pointers!

- Pointer of pointer

```
int value = 3, value2 = 2;
int *ptr = &value;
int **ptr_to_ptr = &ptr;
```

- Question: How to update `value` if you only have `ptr_to_ptr`?

```
**ptr_to_ptr = 5;
```

- Question: How to update `ptr` to point to `value2` if you only have `ptr_to_ptr`?

```
*ptr_to_ptr = &value2;
```

Still puzzled?

Two altitudes of SIST students

1. I know what it is, what it means, how it works and how to use.
   - Best.
   - Eventually you will get to this state.
2. I don’t know what it is, what it means or how it works, but I know how to use.
   - Better than knowing nothing.
   - You won’t stay at this point forever.
   - Eureka moment
3. I don’t know how to use, <u>and I don’t want to learn</u>.
   - File a request to quit ShanghaiTech (no joke)

NULL

- A preprocessor macro.
- Has a value of `0`.
- Used as placeholder, default value or a sign of “not found”
- e.g. return NULL in a search function to signal “not found”
- Billion dollar mistake
  - What may happen if you dereference NULL?
  - Never use NULL?
- `nullptr`?

Memory management

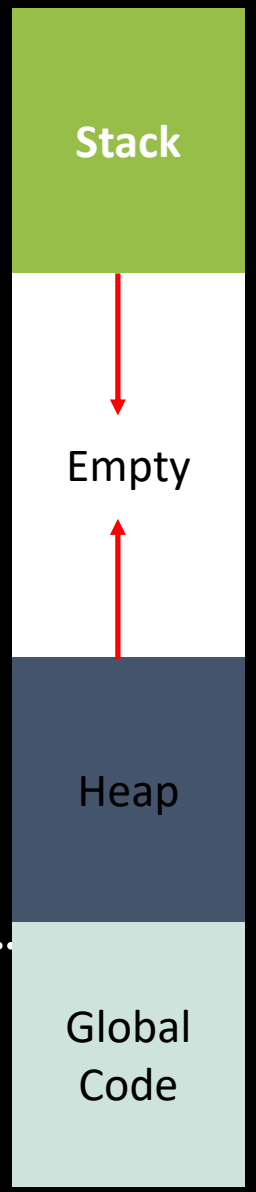
Stack and heap

- Stack: Store local variables
  - Arrangement fixed at compile time. (mostly)
  - Managed allocation and deallocation
  - Small size
  - Does not persist. (?)
- Heap: Store dynamic allocated
  - Dynamic
  - Manual allocation and deallocation
  - Huge size
  - Persist until program exit

Stack and heap: Address conventions

- Stack:
  - Start near max
  - Grow downwards
  - e.g. `0xff9f77ac 0xff9f77b0 0xff9f77b4 0xff9f77b8, ……`
- Heap:
  - Start near 0
  - Grow upwards
  - e.g. `0x01605010, 0x01605440, 0x01605460, 0x01605480, …`

    1st malloc(1) 2nd malloc(1) 3rd malloc(1) 4th malloc(1)



malloc()/free()

- The way to allocate something on the heap

- `void *malloc(size_t size);`
- `void free(void *ptr);`
- `void *calloc(size_t nmemb, size_t size);`
- `void *realloc(void *ptr, size_t size);`

- `void *`: The generic pointer. No type info.
- `size_t`: An unsigned integer large enough to hold a pointer

`realloc()` & `calloc()`

- `realloc()`:
  - Adjust the size of an allocated array
  - Truncate last elements when size reduces
  - Extend when size increases. New regions contains random value.
  - The old pointer becomes dangling pointer after successful `realloc()`
- `calloc()`:
  - Can be considered convenience wrapper for malloc.
  - Initialize allocated memory to zero.

Typical usage

```
int *to_heap(int stack_variable) {
    /* allocate space */
    int *space = malloc(sizeof(int));
    /* use indirection assignment to transfer it to heap */
    *space = stack_variable;
    /* return the pointer to this space */
    return space;
}
```

- Notice there is no explicit cast for `void *`
- `sizeof`: operator to get the size of a type <u>at compile time</u>.

Typical usage

```
int add(int *lhs, int *rhs) {
    return *lhs + *rhs;
}
void print_three() {
    int *a = to_heap(1),
        *b = to_heap(2);
    printf("%d\n", add(a, b));
    free(a); free(b);  <- BAD CODING STYLE.
}
```

- What’s wrong?

The careless

```
void print_three() {
    int *a = to_heap(3),
    free(a);
    printf("%d\n", *a);
}
```

- What’s wrong?

The paranoid

```
void print_three() {
    int *a = to_heap(3);
    printf("%d\n", *a);
    free(a);
    free(a);
}
```

- What’s wrong?

free() is clever

```
void print_three() {
    int *a = to_heap(3),
    free(a + 1);
    printf("%d\n", *a);
}
```

- What’s wrong?

The cleverest person

```
int *add(int lhs, int lhs) {
    int result = lhs + rhs;
    return &result;
}
```

- What’s wrong?

Functions

A small part covered in previous recitation is repeated

Function prototype

- Function without its body.
- `int main(int argc, char **argv);`
- `size_t read_cmds(char **buf, size_t *n);`

Scoping

```c
#include <stdio.h>
int foo = 1;

int test(int foo) {
    return -foo;
}

int main() {
    int foo = 3, bar = 2;
    bar = test(bar);
    printf("Foo is: %d\n", foo);
    printf("Bar is: %d\n", bar);
    return 0;
}
```

What is the output? Why?

Call by pointer

```
void swap(int a, int b) {
    int c = a;
    a = b;
    b = c;
}
```

- Does it works?
- How to fix?

Call by pointer: Exercise

- Write a swap function to swap two generic pointers.

```
void swap(void **a, void **b) {
    void *c = *a;
    *a = *b;
    *b = c;
}
```

Arrays

Declaration

- What does these mean?

```
float sales[365];
char name[12];
int states[50];
int *pointers[5];
```

Initialization

- How to initialize an array to make it look like this?

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
days	31	28	31	30	31	30	31	0	0	0	0	0

  - `int days[12]={31,28,31,30,31,30,0,0,0,0,0,0};`
  - `int days[12]={31,28,31,30,31,30};`
  - `int days[]={31,28,31,30,31,30,0,0,0,0,0,0};`
- Combine last two to make it even shorter?

Array traversal

```
int days[]={31, 28, 31, 30, 31, 30, 0, 0, 0, 0, 0, 0};
for (int i = 0; i < sizeof(days) / sizeof(int) ; i++) {
    printf("%d\n", days[i]);
}
```

- `sizeof(days)`: Get the byte size of this array.
  - Does NOT work on arrays on its pointer form.
  - Divide by element size to get actual element count.

Array traversal

```
char **argv = {"/bin/rush", "script"};
for (int i = 0; i < argc; i++) {
    printf("%s\n", argv[i]);
}
```

- < or <=?
- What if print one char per line?
  - Nested for
  - Extract function called `print_chars()`

Array or pointer?

- This piece of code won't compile. Why?

```
void foo() {
    int days[] = {31,28,31,30,31,30,31,31,30,31,30,31};
    days = &days[6];
}
```

Strings

Common String Ops.

- Get basic info
  - strlen
  - str[index]
- Search
  - strstr
  - strchr/strrchr
- Compare
  - strcmp/strncmp
- Copying
  - strcat/strncat
  - strcpy/strncpy
- Tokenizing
  - strtok

Under the hood?

- Show you how you would typically manipulate string
- Some are naïve version.
- See glibc for actual implementations
  - https://sourceware.org/git/?p=glibc.git;a=tree;f=string;hb=HEAD
  - Highly optimized

strchr: under the hood

```
char * strchr(const char *s, int c_in) {
    while (*s) {
        if (*s == c_in) {
            return s;
        }
        s++;
    }
    return NULL;
}
```

- Imagine how would you implement strrchr
- Template for traversal of one string

strcmp: under the hood

```
int strcmp(const char *p1, const char *p2) {
    const unsigned char *s1 = (const unsigned char *) p1;
    const unsigned char *s2 = (const unsigned char *) p2;
    unsigned char c1, c2;
    do {
        c1 = (unsigned char) *s1++;
        c2 = (unsigned char) *s2++;
        if (c1 == '\0')
            return c1 - c2;
    } while (c1 == c2);
    return c1 - c2;
}
```

- Dual strings manipulation. Easily extended to more.

Actual glibc implementation

strcat: under the hood

```
char *strcat(char *dest, const char *src) {
    strcpy(dest + strlen(dest), src);
    return dest;
}
```

- Use standard library
  - Do not reinvent the wheels
  - Even if you are the wheel makers…

Actual glibc implementation

Learn by mistakes

```
char digits[10] = "0123456789";

char *string = "Hello world!";
printf("string: %s", &string);

char *strdup(const char *src) {
    char *duplicate = malloc(sizeof(char) * strlen(src));
    strcpy(duplicate, src);
    return duplicate;
}
```

- What’s wrong with all these?

Basic File I/O

fopen()

- fopen(“foo.txt”, “a”)
- Flags:
  - Access: r, r+, w, w+, a, a+
  - Content: b
- Remember fclose()

fread()/fwrite()

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- Be aware of overflows

Read/write a very large file?

- Scan for ROI first. Read/write in smaller blocks
- fseek()
  - Go to a particular location in file
  - Much like array subscript
- Memory mapped IO
  - Not covered right now
  - Consult `man 2 mmap` if interested

fprintf()/fscanf()

```
int fscanf(FILE *stream, const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

- printf/scanf for files.
- Difference?
  - No.

Trivia: streams?

- STDIN/STDOUT: Console stream
- fopen: File stream
- Similar design make for good code reuse
- Anything operating on file stream can operate on console stream.

ed-- (Part A)

An ed-like editor

A mini project

- Guided programming. Designed to help you make a project.
- Last indefinitely many recitations.
- <u>This will not be graded.</u>
- You are encouraged to extend this project beyond our spec.
- You are encouraged to try other ways to do this.
- Sharing code with others is <u>allowed</u>.

- Only program on recitation sessions.
  - We don’t want to unreasonably take too much of your time

Overview

- A line based text editor
- Basic line manipulation
- Search/replace
- Does not display whole file on screen

- Not POSIX compatible
  - We only have a reduced command set to fit in recitation time.
  - Feel free to extend it.

Unix ed

- Commands: (start_line,end_line)action
- Operate on selected lines
- Action P turns on prompt
- Action a append several line to selected
- Action p print out selected lines
- Action w write the buffer to disk
- Action q quit the editor

```
root@athena:~/x# ed a.c
242
P
*1,5p
#include<stdio.h>
#include<math.h>
char * strchr(const char *s, int c_in) {
    while (*s) {
        if (*s == c_in) {
*3a
        if (s == NULL) return NULL;
.
*1,5p
#include<stdio.h>
#include<math.h>
char * strchr(const char *s, int c_in) {
        if (s == NULL) return NULL;
    while (*s) {
*w
271
*q
root@athena:~/x#
```

Our spec

- Line selection:
  - Absolute selection: (3,5)
  - Relative selection: (+3,+5)
- Commands to implement
  - a <arg>: append one line <arg>
  - w: write back
  - p: print selected lines
  - pp: print every line
  - s <arg>: search for <arg> (plain text) in selection
  - r <arg1> <arg2>: replace every <arg1> with <arg2> in selection

Design considerations

- One large main()?
  - Not going to work
  - We will condemn you for such behavior
  - Your code will be in “Code Smell” section next recitation
- One large .c file?
  - Split it into pieces?
  - Use headers?
  - Decide these as you go, not from the start

Design considerations

- Single huge buffer vs multiple smaller line buffers?
  - Ease of programming?
    - Are you skilled enough?
    - Do you have like 40 hours to spend on this thing?
  - Efficiency?
    - Will this be invoked a billion time?
    - Will this take a huge amount of time to complete?

Design considerations

- How to organize line buffers?
  - Fixed length array?
    - Wasteful on small file
    - Crash on large file
  - Dynamic array
    - Line insertion?
    - Line removal?
    - The only choice so far……

Today’s menu

- Implement search and replace
  - Search for first occurrence in a few selected lines
  - Replace all occurrence in a few selected lines
- Write a simple main to test it out

Design considerations

- How should I search/replace?
  - Write by hand?
    - Show of skill
    - Again, 40 hours for this?
  - string.h?
    - Which to use?
    - Side effects?
      - Barely any

Design considerations

- How should other part of the program use this search/replace?
  - What this part need?
    - Lines to search in
    - String to search for
    - Replacement (only for replace)
  - What other part need?
    - The line number
    - The resulting lines
  - Function prototype?
    - Think for a while

Search/replace prototype

```
int line_search(const char ** haystack, int len, const char *needle);
char **line_replace(char ** haystack, int len, const char *needle,
    const char* replacement);
```

- Puzzled by the pointers?
  - The more you practice, the more likely you understand
  - Ask your questions

Search/replace prototype

```
int line_search(const char ** haystack, int len, const char *needle);
char **line_replace(char ** haystack, int len, const char *needle,
    const char* replacement);
```

- Notice the similarity
  - Both need haystack & needle
- Can we reuse some code?
  - Let `line_replace` call `line_search` instead of reinvent searching.

Design considerations

- What should I test?
  - Randomly generated input?
    - Could be.
    - You (should) don’t know how to do that though.
    - Not exhaustive.
  - Crafted input trying to break everything, i.e. hackers?
    - We won’t consider it.
    - Yet, of-by-one errors will still go to “code smell” section

Reference Implementation

- <u>Will NOT be available for your download</u>
- Contains an obvious bug and a subtle problem
  - The first is likely to be found in your HW2 as well
  - The second is more related to designing

Interesting Questions

- VLA: Variable length arrary
  - `int vla[argc];`
  - Length not determined at compile time
  - Pros
    - Convenient
    - Fast
    - Clear
  - Cons
    - Not portable, e.g. MSVC
    - Stack overflow
  - Use at your own peril