Recitation 5
I/O & Dynamic Allocation

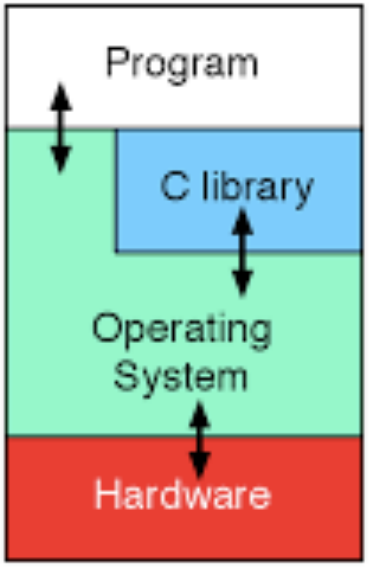
2017-09-29

Resources.

- Piazza. We will use Piazza for questions and announcements. Instructor announcements are required reading. Posts by fellow students an answers to their questions are strongly encouraged readings. Students are also encouraged to answer each other's questions.
- Office hours and their locations will be listed on Piazza for all instructors.
- The course syllabus.
- A gdb quick reference card.
- Old exams and practice exams:

Fall 2013 Practice Midterm Exam	Solutions
Fall 2013 Practice Final Exam	Solutions
Fall 2014 Practice Midterm Exam	Solutions
Fall 2014 Practice Final Exam	Solutions
Fall 2014 Midterm Exam	
Fall 2016 Midterm Exam v1 v2	

Operating System



- Program sent request through system call to let OS responsible for managing all resources
- If request not permissible, OS will return an error
- If request permissible, OS performs underlying hardware operations and return result to the program

Errors from System Call

- errno (error number) : When system call fail in C, they set a global variable called errno
- perror : print a descriptive error message based on cuurent value of errno

Command Line Arguments

```
int main(int argc, char ** argv) {}
```

- argc : the number of inputs (arguments)
- argv[] : an array of pointers to the arguments from 1 to argc-1
- argv[0] : name of the program
- check argc first!

Example – A small Calculator

- User provides + or –and two numbers
- 3 inputs in all

```
int main(int argc, char **argv){
    if (argc != 4) {
        printf("Usage: ./prog  <oper>  <num1>  <num2> \n");
        return EXIT_FAILURE;
    } else {
    …
    }
}
```

Open Files

FILE *fopen(const char* filename, const char * mode);

- Check the return value of fopen to see whether it is NULL or a valid stream

```
FILE *input;
input = fopen("./file.txt, "w+");
if (!input) {
    printf("Error opening file!\n");
}
```

Different options for different requirements

Modes for fopen()

Mode	Read and/or write	Does not exist?	Truncate?	Position
r	read only	fails	no	beginning
r+	read/write	fails	no	beginning
w	write only	created	yes	beginning
w+	read/write	created	yes	beginning
a	writing	created	no	end
a+	read/write	created	no	end

Write Files

FILE * f = fopen(“./myfile.txt”,);

char out[] = “Write that to the file\n”;

fwrite(const void * p, size_t size, size_t elements, FILE * binfile);

example: fwrite(out, 1, sizeof(out)-1, f);

fprintf(FILE * f, constchar * string, …);

example: fprintf(f, ”Write that to the file\n”);

Write Files

```
FILE * input = fopen("./file.txt", "w+" );
if (!input){
    printf("Error opening file!\n");
    return EXIT_FAILURE;
}
char *a = "ECE551\n";
for(int i = 0; i < 7; ++i) {
    fwrite(&a[i], sizeof(char),1,input);
}
fclose(input);
```

Read Files

Same as fwrite!
char input[10];

fread(void * p, size_t size, size_t elements, FILE * binfile);
example: fread(input, sizeof(char), sizeof(input), f);

fgets(char * in, int size, FILE * f);
example: fgets(input, sizeof(input), f);

intc = fgetc(f);

Read Files – EOF

Special “mark” that at the end of a file (EOF)

How do I detect EOF ?

One way is fgetc(input). Similar to fwrite(), but it only reads one byte, one by one.

Returns EOF at the end of file.

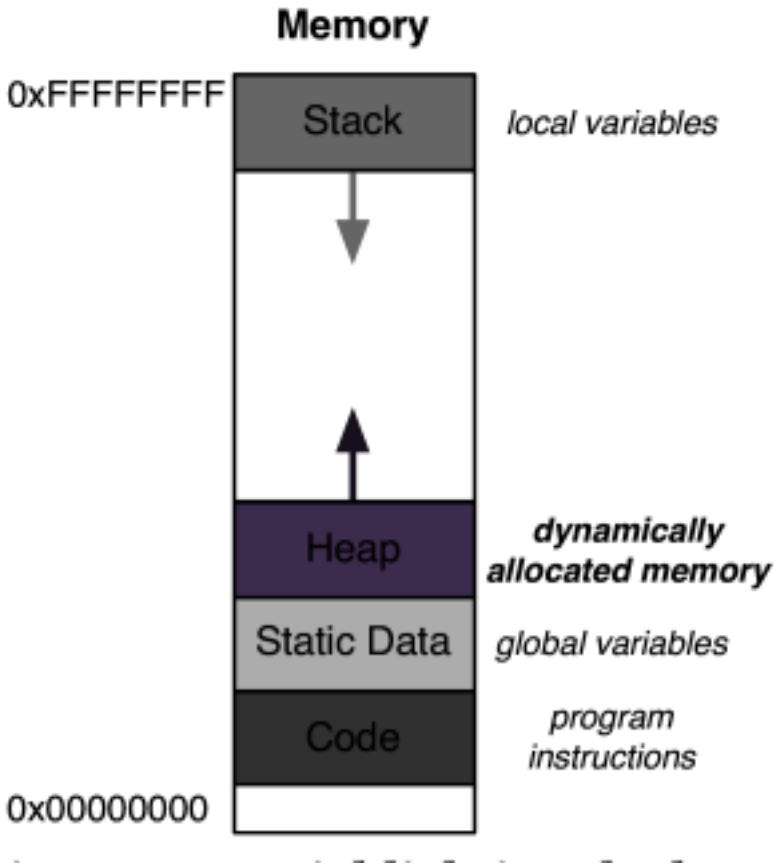
```
int x = fgetc(input)
if(x == EOF) {
   …
}
```

Close files

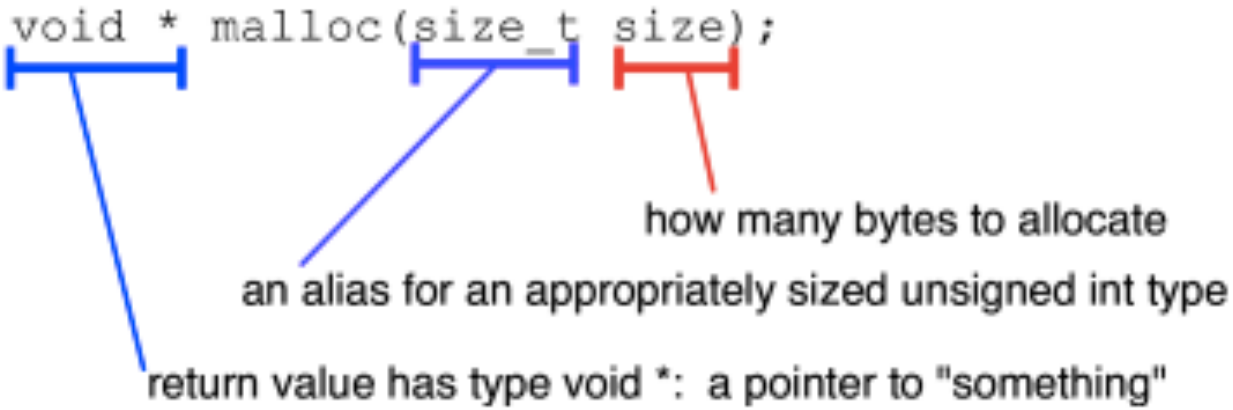
```
int fclose(FILE * stream)
int check = fclose(input);
if (!check) {
    printf("Error closing file!\n");
}
```

possible failure reason: Disk is full, etc.

Dynamic Allocation



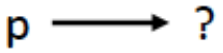
Malloc – memory allocation



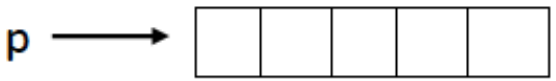
Don’t write a specific number in bytes:
(1) portability
(2) maintainability

Malloc – memory allocation

```
int * p;
```




```
p = malloc(5*sizeof(int));
```



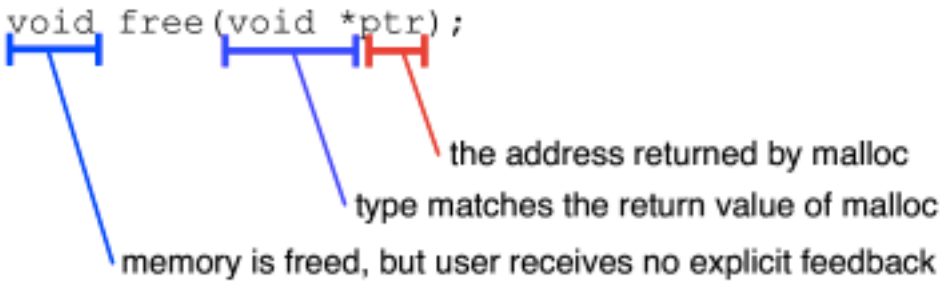
Free

Now we have: p ⟶ 

Assume we do not need the array anymore and we want to free that memory

free(p); p ⟶

Note that the pointer p is not deleted!



Free

- Memory Leak:

Lose all references to a block of memory but the memory is still allocated.

Common problems:

(1) Double freeing

(2) Free something not at the start of the block returned by malloc

(3) free memory not on the heap

Question 3 Debugging [8 pts]

A C programmer wrote the following buggy code to read in many lines of numbers, sort them, and print them out:

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3: int cmplong(const void * vp1, const void * vp2) {
 4:   const long * p1 = vp1;
 5:   const long * p2 = vp2;
 6:   return *p1 - *p2;
 7: }
 8: int main(void) {
 9:   char * line= NULL;
10:   size_t sz =0;
11:   long * array = NULL;
12:   size_t n = 0;
13:   while(getline(&line, &sz, stdin) > 0){
14:     n++;
15:     array=realloc(array, n * sizeof(*array));
16:     array[n] = strtol(line, NULL, 0);
17:   }
18:   free(line);
19:   qsort(&array, n, sizeof(*array), cmplong);
20:   for (size_t i = 0; i < n; i++) {
21:     printf("%ld\n", array[i]);
22:     free(&array[i]);
23:   }
24:   return EXIT_SUCCESS;
25: }
```

Error 1 The first problem is:

B

```
Invalid write of size 8
   at 0x400788: main (broken.c:16)
 Address 0x51fc108 is 0 bytes after a block of size 8 alloc'd
   at 0x4C2AB80: malloc (in ...)
   by 0x4C2CF1F: realloc (in ...)
   by 0x400759: main (broken.c:15)
```

This problem can be correctly fixed by:

A. Using `malloc` instead of `realloc` on line 15

B. Changing line 16 to have `array[n-1]` instead of `array[n]`

C. Changing line 13 to read `size_t n = 1;`

D. Moving line 14 after line 15.

E. None of the above—specify what to do instead:

Error 2 After correctly fixing the first problem, the next error is:

A

```
Conditional jump or move depends on uninitialised value(s)
   by 0x4E726CB: qsort_r (...)
   by 0x4007CF: main (broken.c:19)
```

This problem can be correctly fixed by:

A. Changing `qsort(&array,` to `qsort(array,` on line 19.

B. Changing `qsort(&array,` to `qsort(*array,` on line 19.

C. Changing `sizeof(*array)` to `sizeof(array)` on line 19.

D. Changing `sizeof(*array)` to `sizeof(&array)` on line 19.

E. None of the above—specify what to do instead:

Error 3 After correctly fixing the first two problems, the next error is:

E

```
Invalid read of size 8
   at 0x4007ED: main (broken.c:21)
Address 0x51fd408 is 8 bytes inside a block of size 224 free'd
   at 0x4C2BDEC: free (in ...)
   by 0x40081C: main (broken.c:22)

Invalid free() / delete / delete[] / realloc()
   at 0x4C2BDEC: free (in ...)
   by 0x40081C: main (broken.c:22)
Address 0x51fd408 is 8 bytes inside a block of size 224 free'd
   at 0x4C2BDEC: free (in ...)
   by 0x40081C: main (broken.c:22)
```

This problem can be correctly fixed by:

A. Changing line 22 from `free(&array[i]);` to `free(array[i]);`

B. Changing line 22 from `free(&array[i]);` to `free(array);`

C. Changing line 22 from `free(&array[i]);` to `free(*array[i]);`

D. Changing line 22 from `free(&array[i]);` to `free(array+i);`

E. None of the above—specify what to do instead: Remove line 22

B

Error 4 After correctly fixing the first three errors, the code runs, but leaks memory:

```
224 bytes in 1 blocks are definitely lost in loss record 1 of 1
   at 0x4C2CE8E: realloc (...)
   by 0x400759: main (broken.c:15)
```

This error can be corrected by:

A. Placing `free(array[i]);` between lines 22 and 23 (inside the for loop)

B. Placing `free(array)` between lines 23 and 24 (after the for loop)

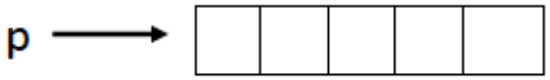
C. Placing `free(line);` between lines 16 and 17 (inside the while loop)

D. Changing `sizeof(*array)` to \verbsizeof(array)+ on line 19

E. None of the above—specify what to do instead:

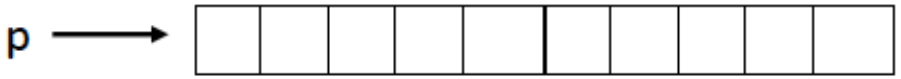
Realloc

From before we had:



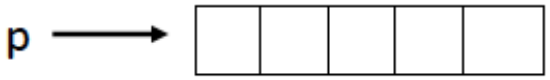
What if we decided we need more space?

```
p = realloc(p, 10*sizeof(*p));
```



Realloc

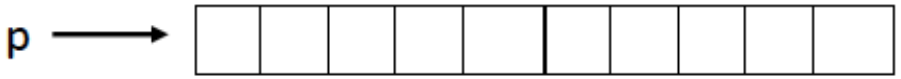
From before we had:



&p[0] == a

What if we decided we need more space?

p = realloc(p, 10*sizeof(*p));



&p[0] == b

address “a” and address “b” are different

Malloc and read at the same time

ssize_t getline(char **result, size_t* size, FILE * file);

example:

```
char * newline=NULL;
size_t sz= 0;
FILE * f = fopen("./myfile.txt","r");
getline(&newline, &size, f);
```

Q1.3 For each question, fill in the blank with the word or words which most correctly complete the sentence.

1. A program makes a system call if it needs to interact with the outside world.
2. The difference between perror and just using fprint to stderr is that perror also prints a description of error message based on errno.
3. If you need to resize a block allocated by malloc, you would use realloc.
4. Abstraction is the separation of interface from implementation.
5. When declaring a variable which points at a string literal, its most correct type is const char *.
6. If you need to find out how many bytes a type occupies in memory, you would use the sizeof() operator.
7. If you are debugging in `gdb`, and are currently at a line with a function call, you would use the next command to go past the function call, without going inside it.

Question 6 Coding 3: Dynamic Allocation [13 pts]

Write the function `char * int2Str(unsigned int x)` which takes an `unsigned int`, allocates memory for a string, and fills it in with the digits of the decimal representation of `x`. For example, if your function is passed `"123"`, then it should return 123. Note that `x` is unsigned, so you do not have to deal with negative numbers. You should not make assumptions about the maximum size of a number that is representable in an `unsigned int` (*e.g.*, if I run your code on a weird platform where `unsigned int` is 1024 bits, it should still work right).

You may use any of the following library functions (but only the following library functions): `malloc`, `realloc`, and `free`..

```
char * int2Str(unsigned int x) {
```

```
char *int2Str(unsigned int x) {
    char * result = NULL;
    size_t i = 0;
    while (x % 10 != x) {
        result = realloc(result, (i+1) * sizeof(*result);
        result[i] = '0' + x % 10;
        x = x / 10;
        i++;
    }
    result = realloc (result, (i+2) * sizeof(*result));
    result[i++] = '0' + x % 10;
    result[i] = '\0';
    for (size_t j = 0; j < i / 2; j++) {
        char tmp = result[j];
        result[j] = result[i-j-1];
        result[i-j-1] = tmp;
    }
    return result;
}
```

Question 7 Coding 4: File IO [18 pts]

Write a program which takes 2 command line arguments. The first argument is the name of an input file, and the second is an arbitrary string, which you will treat as a set of characters. Your program should read the file named by its first argument, and count how many times each character in its second argument appears in that file. After reading the entire file, it should print each character from its second argument, followed by a `:`, and the count of how many of that character it found in the input. For example, if the input file is

```
Hello World
I like to program!
```

and the second argument is `i!o`, your program should print:

```
i:1
!:1
o:4
```

(because `i` appears once, `!` appears once, and `o` appears 4 times).

For this problem, we will relax the "no assumptions" restriction by allowing you to assume that (1) two command line arguments are provided (2) the input file requested exists and is readable (fopen succeeds) (3) the characters in the second argument are distinct (none are repeated) (4) malloc and realloc always succeed and (5) fclose succeeds. You may use any library functions you wish. You may *NOT* assume anything about the length of the second argument, nor the contents of the input file.

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

int main(int argc, char  **argv) {
      char  *  content = argv[2];
      int len = strlen(argv[2]);
      for (int i = 0; i < len; ++i) {
            int count = 0;
            char * cur = NULL;
            size_t linecap;
            FILE *f = fopen(argv[1], “r”);
            while (getline(&cur, &linecap, f) >= 0) {
                  for (int j = 0; j < (int)linecap; ++j) {
                        if (cur[j] == content[i]) {
                              ++count;
                        }
                  }
            }
            free(cur);
            fclose(f);
            print(“%c:%d\n”, content[i], count);
      }
      return EXIT_SUCCESS;
}
```