UNIX, C, and Data Structures

Lecture 12: Pointers to Functions
Complexity
Talk about Q1 from midterm
CIS 212: Project #2G
Assigned: November 7, 2019
Due: November 12, 2019
(which means submitted by 6am on November 13, 2019)
Worth 10% of your projects grade

Assignment:
1. Write a C program that reads integers from a file, and produces a file with specific integers, as detailed here:
   1.1. The input file contains 144 integers, in binary, and they represent a 12×12 matrix.
   1.2. Your program must read the top-left 4×4 sub-matrix, which consists of entries (inclusive) 0–3, 12–15, 24–27, and 36–39, from the binary file.
   1.3. With these specific integers (representing the 4×4 sub-matrix), your program should produce a text file containing them, one integer per line.
   1.4. Important: you must read exactly 16 integers. Do not read all 144.
2. Your program must take two file name arguments (the input binary file, and the output text file), as demonstrated below.
3. Important: you may use C file stream functions fopen, fseek, fread, fprintf, fclose, (and ftell for debugging as necessary).

Your program should operate as follows:
./project2G binaryfile textfile

For example, if I run:
./project2G binary-file-2G resultfile

And binary-file-2G contains a binary integer representation of this matrix:
3 1 4 1 5 9 2 6 5 3 5 8
9 7 9 3 2 3 8 4 6 2 6 4
3 3 8 3 2 7 9 5 0 2 8 8
4 1 9 7 1 6 9 3 9 9 3 7
5 1 0 5 8 2 0 9 7 4 9 4
(... remaining rows omitted)

Then resultfile should contain:
3
1
4
1
9
7
(... skipping seven lines for brevity... )
Quiz on Tuesday, Nov 12

- There will be a quiz Tuesday Nov 12.
- It will be worth 3 points.
- It will be on file I/O.
- No notes.
- Expecting you to know fopen/fread/fwrite/etc.
Reading: Chapters 5 and 6

- I will be following the textbook more closely the next few weeks
Call by value / call by reference

• Refers to how parameters are passed to a function.
  – Call by value: send the value of the variable as a function parameter
    • Side effects in that function don’t affect the variable in the calling function
  – Call by reference: send a reference (pointer) as a function parameter
    • Side effects in that function affect the variable in the calling function
C does not care that foo has an argument variable called “x” and main has an automatic variable called “x”. They have NO relation.
Call by reference

```c
#include <stdio.h>

void foo(int *x)
{
    *x = *x + 1;
}

int main()
{
    int x = 2;
    foo(&x);
    printf("X is %d\n", x);
}
```

```
C02LN00GFD58:330 hank$ cat cbr.c
#include <stdio.h>

void foo(int *x)
{
    *x = *x + 1;
}

int main()
{
    int x = 2;
    foo(&x);
    printf("X is %d\n", x);
}
```

```
C02LN00GFD58:330 hank$ gcc cbr.c
C02LN00GFD58:330 hank$ ./a.out
X is 3
```
Preprocessor

• Preprocessor:
  – takes an input program
  – produces another program (which is then compiled)

• C has a separate language for preprocessing
  – Different syntax than C
  – Uses macros ("#")

macro ("macroinstruction"): rule for replacing input characters with output characters
#include

- compiler can only compile one file at a file
- takes another file and includes it in the current file
- the file is a “header” file
  - it contains function prototypes
  - a function prototype declares a function exists, but not how it is implemented
Demonstrate #include

```c
#include <stdio.h>
int main()
{
    printf("Hello world\n");
}

% gcc -E printf.c
```
Function Prototype

```c
int doubler(int); 
int main() 
{
    return doubler(2); 
}
int doubler(int X) { return 2*X; }; 

C02LN00GFD58:212 hank$ gcc main.c  
C02LN00GFD58:212 hank$ vi main.c  
C02LN00GFD58:212 hank$ cat main.c  
/* int doubler(int); */
int main() 
{
    return doubler(2); 
}
int doubler(int X) { return 2*X; }; 

C02LN00GFD58:212 hank$ gcc main.c  
main.c:4:12: warning: implicit declaration of function 'doubler' is invalid in C99 [-Wimplicit-function-declaration]
    return doubler(2); 

1 warning generated.
C02LN00GFD58:212 hank$
```
Preprocessor Phases

- Resolve #includes
- Conditional compilation (#ifdef)
- Macro replacement
- Special macros
This is an example of macro replacement.
#define via gcc command-line option

```c
C02LN00GFD58:330 hank$ cat defines.c
int main()
{
    return RV;
}
C02LN00GFD58:330 hank$ gcc -DRV=4 defines.c
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ echo $?
4
```
Conflicting –D and #define

C02LN00GFD58:330 hank$ cat defines.c
#define RV 2
int main()
{
    return RV;
}
C02LN00GFD58:330 hank$ gcc -DRV=4 defines.c
defines.c:1:9: warning: 'RV' macro redefined
#define RV 2
^<command line>:1:9: note: previous definition is here
#define RV 4
^1 warning generated.
C02LN00GFD58:330 hank$ ./a.out
C02LN00GFD58:330 hank$ echo $?2
Conditional compilation

```c
C02LN00GFD58:330 hank$ cat conditional.c
#define USE_OPTION 1

int main()
{
    DoMainCode();
    ifdef USE_OPTION
        UseOption();
    endif
    DoCleanupCode();
}
```
Conditional compilation controlled via compiler flags

```c
#include <stdio.h>

int main()
{
    #ifdef DO_PRINTF
        printf("I am doing PRINTF!!\n");
    #endif
}
```

```
$ cat conditional_printf.c
#include <stdio.h>

int main()
{
    #ifdef DO_PRINTF
        printf("I am doing PRINTF!!\n");
    #endif
}
```

```
$ gcc conditional_printf.c
$ gcc -DDO_PRINTF conditional_printf.c
```

```
$ ./a.out
```

```
I am doing PRINTF!!
```

Function Pointers

• Idea:
  – You have a pointer to a function
  – This pointer can change based on circumstance
  – When you call the function pointer, it is like calling a known function
Function Pointer Example

```c
#include <stdio.h>

int doubler(int x) { return 2*x; }
int tripler(int x) { return 3*x; }
int main()
{
    int (*multiplier)(int);
    multiplier = doubler;
    printf("Multiplier of 3 = %d\n", multiplier(3));
    multiplier = tripler;
    printf("Multiplier of 3 = %d\n", multiplier(3));
}
```

```
128-223-223-72-wireless:cli hank$ gcc function_ptr.c
128-223-223-72-wireless:cli hank$ ./a.out
Multiplier of 3 = 6
Multiplier of 3 = 9
```
Function Pointer Example #2

```
#include <stdio.h>
void doubler(int *X) { X[0] *= 2; X[1] *= 2; }
void tripler(int *X) { X[0] *= 3; X[1] *= 3; }
int main()
{
    void (*multiplier)(int *);
    multiplier = doubler;
    multiplier(A);
    printf("Multiplier of 3 = %d, %d\n", A[0], A[1]);
    multiplier = tripler;
    multiplier(A);
    printf("Multiplier of 3 = %d, %d\n", A[0], A[1]);
}
```

Don’t be scared of extra ‘*’s ... they just come about because of pointers in the arguments or return values.
Simple-to-Exotic Function Pointer Declarations

void (*foo)(void);
void (*foo)(int **, char ***);
char ** (*foo)(int **, void (*)(int));

These sometimes come up on interviews.
What are the pros and cons of each approach?
Callbacks

- Callbacks: function that is called when a condition is met
  - Commonly used when interfacing between modules that were developed separately.
  - ... libraries use callbacks and developers who use the libraries “register” callbacks.
Callback example

128-223-223-72-wireless:callback hank$ cat mylog.h
void RegisterErrorHandler(void (*eh)(char *));
double mylogarithm(double x);

128-223-223-72-wireless:callback hank$ cat mylog.c
#include <mylog.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* NULL is an invalid memory location. */
/* Useful for setting to something known, rather than leaving uninitialized */
void (*error_handler)(char *) = NULL;

void RegisterErrorHandler(void (*eh)(char *))
{
    error_handler = eh;
}

void Error(char *msg)
{
    if (error_handler != NULL)
        error_handler(msg);
}

double mylogarithm(double x)
{
    if (x <= 0)
    {
        char msg[1024];
        sprintf(msg, "Logarithm of a negative number: %f !!", x);
        Error(msg);
        return 0;
    }

    return log(x);
}
Callback example

```c
#include <mylog.h>
#include <stdio.h>

FILE *F1 = NULL;
void HanksErrorHandler(char *msg)
{
    if (F1 == NULL)
    {
        F1 = fopen("error", "w");
    }
    fprintf(F1, "Error: %s\n", msg);
}

int main()
{
    RegisterErrorHandler(HanksErrorHandler);

    mylogarithm(3);
    mylogarithm(0);
    mylogarithm(-2);
    mylogarithm(5);
    if (F1 != NULL)
        fclose(F1);
}
```

```
128-223-223-72-wireless:callback hank$
cat program.c
```

```
128-223-223-72-wireless:callback hank$
#include <mylog.h>
#include <stdio.h>

FILE *F1 = NULL;
void HanksErrorHandler(char *msg)
{
    if (F1 == NULL)
    {
        F1 = fopen("error", "w");
    }
    fprintf(F1, "Error: %s\n", msg);
}

int main()
{
    RegisterErrorHandler(HanksErrorHandler);

    mylogarithm(3);
    mylogarithm(0);
    mylogarithm(-2);
    mylogarithm(5);
    if (F1 != NULL)
        fclose(F1);
}
```

```
128-223-223-72-wireless:callback hank$
128-223-223-72-wireless:callback hank$
128-223-223-72-wireless:callback hank$
128-223-223-72-wireless:callback hank$
128-223-223-72-wireless:callback hank$
```

```
128-223-223-72-wireless:callback hank$
128-223-223-72-wireless:callback hank$
128-223-223-72-wireless:callback hank$
128-223-223-72-wireless:callback hank$
```

```
Error: Logarithm of a negative number: 0.000000 !!!
Error: Logarithm of a negative number: -2.000000 !!!
```
What is a Data Structure?
What is a Data Structure?

• Data structure definitions
  – Textbook: “a systematic way to organize data”
  – Wikipedia: “data organization, management and storage format that enables efficient access and modification”
We Are Already Familiar With Some Data Structures

- Arrays
- From Python
  - List
  - Tuple
  - Dictionary
  - Set
Key Concept

• It organizes data
• It enables efficient access
  – What does efficient mean??
Example:

Data Structure: Arrays
Operation: Search

```c
typedef struct {
    float score_1A;
    float score_1B;
} Grades;

typedef struct {
    char *name;
    int UO_id;
    Grades grade;
} Student;

int main()
{
    Student cis212_F18[80];
    cis212_F18[0].name = "Henry Shields";
    cis212_F18[0].grades.score_1A = 0;
    cis212_F18[0].grades.score_1B = 0;
    /* ... */
}
```

```c
int IsStudentInClass(char *thisName, Student *students, int numStudents)
{
    /* etc */
    /* How should we implement this? */
}
```
Is It Efficient?:
Two Sub-questions

• 1) How long does this take to run?
  – (how do we measure this?)

• 2) Could we do it with less operations?
Is It Efficient?:
Two Sub-questions

• 1) How long does this take to run?
  – (how do we measure this?)

• 2) Could we do it with less operations?
How Long Does This Take To Run?

• One answer: time it!
Unix command: time

Linux supplies a program, /usr/bin/time, which will execute a program and report various measurements concerning the resources consumed by the program. Let’s assume that we have a file named verylargefile, that it has 12,480,100 lines, each line has a single word, and the total number of characters is 111,148,500. Let’s use wc on the file to count the lines, words, and characters, and use /usr/bin/time to determine the resource utilization of wc.

```
$ /usr/bin/time wc verylargefile
   12480100  12480100  111148500 verylargefile
1.59user 0.03system  0:01.63elapsed 99%CPU (0avgtext+0avgdata 429468
maxresident)k 0inputs+0output (1760major+0minor)pagefaults 0 swaps
```
Another option: add timings to your program!

• (note: now taking a few slide aside)
gettimeofday

gettimeofday(2) BSD System Calls Manual GETTIMEOFDAY(2)

NAME
gmtime, localtime -- get/set date and time
SYNOPSIS
#include <sys/time.h>

int
gmtime(struct timeval *restrict tp, void *restrict tzp);

int
localtime(const struct timeval *tp, const struct timezone *tzp);

DESCRIPTION
The system's notion of the current Greenwich time and the current time zone is obtained with the gettimeofday() call, and set with the settimeofday() call. The time is expressed in seconds and microseconds since midnight (0 hour), January 1, 1970. The resolution of the system clock is hardware dependent, and the time may be updated continuously or in "ticks." If tp is NULL and tzp is non-NULL, gettimeofday() will populate the timezone struct in tzp. If tp is non-NULL and tzp is NULL, then only the timeval struct in tp is populated. If both tp and tzp are NULL, nothing is returned.

The structures pointed to by tp and tzp are defined in <sys/time.h> as:

struct timeval {
    time_t   tv_sec;  /* seconds since Jan. 1, 1970 */
    suseconds_t tv_usec;  /* and microseconds */
};

struct timezone {
    int         tz_minuteswest;  /* of Greenwich */
    int         tz_dsttime;    /* type of dst correction to apply */
};

The timeval structure specifies a time value in seconds and microseconds. The values in timeval are opaque types whose length may vary on different machines; depending on the implementation, the meaning of the time values may also vary.

The timezone structure indicates the current time zone as the number of minutes west of Greenwich, and a flag that, if nonzero, indicates that Daylight Saving time is currently in effect.

Only the super-user may set the time. This is a limitation imposed by the need to modify the system clock at the time that, for example, the system time is used to build time stamps on files. The system time can still be adjusted backwards using clock_settime() for system processes that need to record, or replay, time stamps.

RETURN
A 0 return value indicates that the call succeeded. A -1 return value indicates an error occurred, and in this case an error code is stored into the global variable errno.

(there are lots of Unix system calls, which do lots of different things)
gettimeofday example

fawcett:330 childs$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    gettimeofday(&endTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        count += i;
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.0;
    printf("done executing, took %f\n", seconds);
}
gettimeofday example

fawcett:330 childs$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took \%f\n", seconds);
}

fawcett:330 childs$ g++ -02 timings.C
fawcett:330 childs$ ./a.out
done executing, took 0.000000
fawcett:330 childs$
gettimeofday example

fawcett:330 childds$ cat timings.C
#include <sys/time.h>
#include <stdio.h>

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    gettimeofday(&startTime, 0);
    for (int i = 0; i < num_iterations; i++)
    {
        count += i;
        printf("Count was %d\n", count); /* NEW LINE OF CODE */
        struct timeval endTime;
        gettimeofday(&endTime, 0);
        double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                        double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
        printf("done executing, took %f\n", seconds);
    }
}
gettimeofday example

fawcett:330 childsd$ cat timings2.C
#include <sys/time.h>
#include <stdio.h>

int LoopFunction(int iteration, int &count)
{
    count += iteration;
}

int main()
{
    int num_iterations = 100000000;
    int count = 0;
    struct timeval startTime;
    gettimeofday(&startTime, 0);
    for (int i = 0 ; i < num_iterations ; i++)
        LoopFunction(i, count);
    /* No longer need this: printf("Count was %d\n", count); */
    struct timeval endTime;
    gettimeofday(&endTime, 0);
    double seconds = double(endTime.tv_sec - startTime.tv_sec) +
                     double(endTime.tv_usec - startTime.tv_usec) / 1000000.;
    printf("done executing, took %.2f\n", seconds);
}

fawcett:330 childsd$ g++ -O2 timings2.C
fawcett:330 childsd$ ./a.out
done executing, took 0.213101
How Long Does This Take To Run?

- One answer: time it!
- Question: what if I change the data?
  - What if we have 8000 students instead of 80 students?
- New idea: analyze the program
Analyze the Program

```c
int IsStudentInClass(char *thisName, Student *students, int numStudents) {
    int i;
    for (i = 0 ; i < numStudents ; i++)
        if (strcmp(thisName, students[i].name) == 0)
            return 1;
    return 0;
}
```

• How many operations does this perform?
• Assume `strcmp` takes 50 operations
• And `numStudents` is 80
• Then \(\sim 80 \times 50 = 4000\) operation
Big O Notation (1/3):
Important Computer Science Concept

- If input data size is “N”, then you can describe how many operations occur using N
- If “numStudents” is N, and strcmp takes 50 operations, then 50N
Big O Notation (2/3): Important Computer Science Concept

• Idea #1 behind Big O: don’t worry about constants
• Idea #2 behind Big O: just say how many operations with respect to N (number of data elements)
• Answer: O(N)
  – This is so simple, almost not useful ... need new example!
Big O Notation (3/3): Important Computer Science Concept

```c
int IsStudentInClass(char *thisName, Student *students, int numStudents) {
    int i;
    for (i = 0; i < numStudents; i++)
        if (strcmp(thisName, students[i].name) == 0)
            return 1;
    return 0;
}
```

- This is called “Asymptotic Analysis”
  - Why? ... what happens when N gets “large”?
More complex example

```c
int TwoStudentsWithName(Student *students, int numStudents) {
    int i, j;
    for (i = 0 ; i < numStudents ; i++)
        for (j = 0 ; j < numStudents ; j++)
            if (i == j)
                continue;
            if (strcmp(students[i].name, students[j].name) == 0)
                return 1;
    return 0;
}
```

- How many operations now?
- `numStudents*numStudents*50` (about)
- \( \Rightarrow O(n^2) \)
Is This Better?

```c
int TwoStudentsWithSameName(Student *students, int numStudents) {
    int i, j;
    for (i = 0 ; i < numStudents ; i++)
        for (j = i+1 ; j < numStudents ; j++)
            if (strcmp(students[i].name, students[j].name) == 0)
                return 1;
    return 0;
}
```

- How many operations now?
- numStudents*numStudents*25 (about)
- STILL $\Rightarrow O(n^2)$
Comparing Our Two Implementations

```c
int TwoStudentsWithSameName(Student *students, int numStudents)
{
    int i, j;
    for (i = 0 ; i < numStudents ; i++)
        for (j = 0 ; j < numStudents ; j++)
        {
            if (i == j)
                continue;
            if (strcmp(students[i].name, students[j].name) == 0)
                return 1;
        }
    return 0;
}
```

```c
int TwoStudentsWithSameName(Student *students, int numStudents)
{
    int i, j;
    for (i = 0 ; i < numStudents ; i++)
        for (j = i+1 ; j < numStudents ; j++)
        {
            if (strcmp(students[i].name, students[j].name) == 0)
                return 1;
        }
    return 0;
}
```

- Answer: second version is about twice as fast
  - But both O(n^2), so maybe not a huge difference
New data structure: sorted array

• Like an array, but everything is sorted
• Unsorted array
  – int X[6] = { 4, 1, 3, 7, 9, 2 };
• Sorted array
  – int X[6] = { 1, 2, 3, 4, 7, 9 };
Can We Improve On This One With An Unsorted Array?

```c
int IsStudentInClass(char *thisName, Student *students, int numStudents) {
    int i;
    for (i = 0 ; i < numStudents ; i++)
        if (strcmp(thisName, students[i].name) == 0)
            return 1;
    return 0;
}
```

- Probably not, but...
But We Can Improve With a Sorted Array...

```c
int IsStudentInClass(char *thisName, Student *sorted_students, int numStudents) {
    int idx = numStudents/2;
    if (strcmp(thisName, sorted_students[idx].name) == 0) {
        return 1;
    }
    if (numStudents <= 1) {
        return 0;
    }
    if (IsLessThan(thisName, sorted_students[idx])) {
        return IsStudentInClass(thisName, sorted_students, idx);
    } else {
        return IsStudentInClass(thisName, sorted_students+idx+1, numStudents-(idx+1));
    }
}
```
How many operations did this take?

- Start with $N$ elements
- After 1 iteration, $N/2$ elements
- After 2 iterations, $N/4$ elements
- After 3 iterations, $N/8$ elements
- After 4 iterations, $N/16$ elements
- Etc...
- After $\log_2 N$ iterations, 1 element
- $\Rightarrow O(\log n)$
But there was work to take an array and make a sorted array

• How long does it take to sort?
• Sort: complexity varies: $O(n \log n)$ to $O(n^2)$
  – Project 2B example: $O(n^2)$
  – Most real world: $O(n \log n)$
• So: we could do IsStudentInClass on unsorted array for $O(n)$
• Or: sort array (cost $O(n \log n)$) and do IsStudentInClass for $O(\log n)$
• → we only want to do a sort if we plan to call IsStudentInClass a bunch of times
Can we get a better Big O for this one?

```c
int TwoStudentsWithSameName(Student *students, int numStudents)
{
    SortNames(students, numStudents); /* how long does this take? */
    int i;
    for (i = 0; i < numStudents-1; i++)
    {
        if (strcmp(students[i].name, students[i+1].name) == 0)
            return 1;
    }
    return 0;
}
```

• ‘for’ loop over numStudents: O(n)
• Combined: O(nlogn) + O(n) → O(nlogn)
  – (asymptotic analysis)
<table>
<thead>
<tr>
<th>$n$</th>
<th>$\log n$</th>
<th>$n$</th>
<th>$n \log n$</th>
<th>$n^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>8</td>
<td>24</td>
<td>64</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>32</td>
<td>160</td>
<td>1,024</td>
</tr>
<tr>
<td>64</td>
<td>6</td>
<td>64</td>
<td>384</td>
<td>4,096</td>
</tr>
<tr>
<td>128</td>
<td>7</td>
<td>128</td>
<td>896</td>
<td>16,384</td>
</tr>
<tr>
<td>256</td>
<td>8</td>
<td>256</td>
<td>2,048</td>
<td>65,536</td>
</tr>
<tr>
<td>512</td>
<td>9</td>
<td>512</td>
<td>4,608</td>
<td>262,144</td>
</tr>
<tr>
<td>1,024</td>
<td>10</td>
<td>1,024</td>
<td>10,240</td>
<td>1,048,576</td>
</tr>
</tbody>
</table>
Is It Efficient?:

Two Sub-questions

• 1) How long does this take to run?
  – (how do we measure this?)

• 2) Could we do it with less operations?

• Unsorted arrays were more efficient for
  search, but there was work in creating them
  – Different data structures are efficient for different
    operations
  – Hence, we have a lot of them