Lecture 13:
Pointers to Functions
Complexity
Talk about Q3-5 from midterm
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
Review
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
Call by Value

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

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

```bash
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
```
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

128-223-223-72-wireless_CLI hank$ cat function_ptr.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
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;
    /* etc */
} 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;
    /* ... */
}
```
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

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

fawcett:330 childsl$ 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);
}
gettimeofday example

fawcett:330 child$ 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 child$ g++ -O2 timings.C
fawcett:330 child$ ./a.out
done executing, took 0.000000
fawcett:330 child$
gettimeofday example

```c
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);
    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);
}
```
```
#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 \%f\n", seconds);
}
```
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 ~80*50 = 4000 operation
Big O Notation (1/3):
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;
}
```

- 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

```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;
}
```

- 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 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;
}
```

• 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

```
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;
}
```

```
int TwoStudentsWithName(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 \( O(\log n) \)
• \( \rightarrow \) 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
(SLIDE REPEAT)

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”

  – Can you access data in $O(n)$ time? $O(\log n)$? Etc.
  – If the answer is “fast,” then it is a good data structure
  – And: the right data structure varies by task
Making It Generic
Making it generic

• We just looked at an example: sorted arrays of students

• We could make this more generic:
  – Sorted arrays for *anything*
    • Doesn’t have to be students
  – Don’t even need to know we are using sorted arrays
    • Could just use “Search”
Making it generic

• We just looked at an example: sorted arrays of students

• We could make this more generic:
  – Sorted arrays for *anything*
    • Doesn’t have to be students
  – Don’t even need to know we are using sorted arrays
    • Could just use “Search”
Need some background material before exploring this
Important: void *

- void *: pointer to memory
- Pointer arithmetic: 1 byte
- Just a location in memory
- Useless without a “cast”
  - “Cast” change to a different type
  - void *p = 0x7ffff;
  - int *x = (int *) p; /* now we can treat it as an int */
memcpy

- Function in C standard library
- Copies data
- We could write it ourselves, but this is easier...
memcpy

**NAME**
memcpy -- copy memory area

**LIBRARY**
Standard C Library (libc, -lc)

**SYNOPSIS**
```
#include <string.h>

void *
memcpy(void *restrict dst, const void *restrict src, size_t n);
```

**DESCRIPTION**
The `memcpy()` function copies `n` bytes from memory area `src` to memory area `dst`. If `dst` and `src` overlap, behavior is undefined. Applications in which `dst` and `src` might overlap should use `memmove(3)` instead.

**RETURN VALUES**
The `memcpy()` function returns the original value of `dst`.

**SEE ALSO**
`bcopy(3), memccpy(3), memmove(3), strcpy(3), wmemcpy(3)`

**STANDARDS**
void my_memcpy(void *dest, void *src, int nelems)
{
    int i;
    unsigned char *dest_as uc = (unsigned char *) dest;
    unsigned char *src_as uc = (unsigned char *) src;
    for (i = 0 ; i < nelems ; i++)
        dest_as uc[i] = src_as uc[i];
}
Making it generic

• We just looked at an example: sorted arrays of students

• We could make this more generic:
  – Sorted arrays for *anything*
    • Doesn’t have to be students
  – Don’t even need to know we are using sorted arrays
    • Could just use “Search”
void Sort(void *array, int element_size, int num_elements, int (*Compare)(void *, void *))
{
    int i, j;
    void *tmp_location = malloc(element_size);
    for (i = 0 ; i < num_elements ; i++)
    {
        int smallest = i;
        for (j = i+1 ; j < num_elements ; j++)
        {
            int comp_value = Compare(array+j*element_size, array+smallest*element_size);
            if (comp_value < 0)
                smallest = j;
        }
        memcpy(tmp_location, array+i*element_size, element_size);
        memcpy(array+i*element_size, array+smallest*element_size, element_size);
        memcpy(array+smallest*element_size, tmp_location, element_size);
    }
    free(tmp_location);
}
Using Generic Sort

```c
int main()
{
    int i;
    char str[12];
    strcpy(str, "hello world");
    int A[10] = { 5, 3, 1, 8, 9, 0, 2, 4, 6, 7 };  
    Sort(str, 1, 11, CharCompare);
    Sort(A, sizeof(int), 10, IntCompare);
    printf("%s\n", str);
    for (i = 0; i < 10; i++) { printf("%d ", A[i]); };
    printf("\n");
}
```

```c
int CharCompare(void *a, void *b)
{
    char *a_as_char = (char *) a;
    char *b_as_char = (char *) b;
    if (*a_as_char < *b_as_char)
        return -1;
    return 1;
}
```

```c
int IntCompare(void *a, void *b)
{
    int *a_as_int = (int *) a;
    int *b_as_int = (int *) b;
    if (*a_as_int < *b_as_int)
        return -1;
    return 1;
}
```

```
Hanks-iMac:212 hank$ ./a.out
dehllloorw
0 1 2 3 4 5 6 7 8 9
```
Making it generic

• We just looked at an example: sorted arrays of students

• We could make this more generic:
  – Sorted arrays for *anything*
    • Doesn’t have to be students
  – Don’t even need to know we are using sorted arrays
    • Could just use “Search”
This will be Thursday’s lecture

• Abstract data types
  – Think about it from the perspective of the user
  – NOT the implementor
Data structure for specific data type (example: array of Students)

Data structure for *any* data type (example: array of “void *”)

Abstract data structure for specific data type (example: search for student names)

Abstract data structure for *any* data type (example: search for “void *”)
When Do I Use What?

When I have developed a data structure I want others to use:

- Data structure for specific data type (example: array of Students)
- Data structure for *any* data type (example: array of “void *”)

When I want to solve a problem for many, many people:

- Abstract data structure for specific data type (example: search for student names)

When writing something for my specific program:

- Abstract data structure for *any* data type (example: search for "void *")

Never?
<table>
<thead>
<tr>
<th>No management/organization of data</th>
<th>Specific data structure for specific type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specific data structure for generic type</td>
</tr>
<tr>
<td></td>
<td>Abstract data types</td>
</tr>
</tbody>
</table>