Lecture 12:
What is a Data Structure? / How to Make Things Generic
Plan for the next few weeks

LEC 11
LEC 12
LEC 13
LEC 14
LEC 15

Project 2G (HARD)
Project 2H (NORMAL)
Project 3A (HARD, depends on 2H)
Project 3B (HARD)
Project 1C (NORMAL, LAB)

LEC
YouTube
Hank
OH
OH
Hank
OH
OH
Comments

• Overall workload is down a little
  – 5 projects in 3 weeks, instead of 6 in 3 weeks
• 1C should be achievable without additional support via Lec14 + Lab that will reinforce Unix
• For hard projects, trying to offer extra OH right as I arrive back (including Sunday)
  – Critical to attempt projects beforehand
• Different flow:
  – Projects 1A,1B, 2A-2F: between 4 and 7 days to complete
  – Projects 2G,2H,3A,3B,1C: 7 to 10 days to complete
Reading: Chapter 4

- All of Chapter 4, except:
  - 4.1.2 (although reading this is a bit necessary for later in the chapter). But we are not doing “make” or multi-file development
  - 4.5.3: skip (for multi-file development)
  - 4.5.4: statics, hope to lecture on by end of course
  - 4.7.7: unions, hope to lecture on by end of course
  - 4.8: interesting reading! (scanf, which is partner of printf)
  - 4.11: valgrind is great, may be a lab soon
Reading: Chapters 5 and 6

- I will be following the textbook more closely the next few weeks
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 IsStudentInClass(char *thisName, Student *students, int numStudents)
{
    /* etc */
    /* How should we implement this? */
}

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

gettimeofday, settimeofday -- get/set date and time

SYNOPSIS
#include <sys/time.h>

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

int settimeofday(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 platform.

The timezone structure indicates that Daylight Saving Time is observed, and a flag that, if nonzero, indicates that Daylight Saving time adjusts

Only the super-user may set the time and time zone. Attempting to set the time to a value greater than 1 (see init(8)), the time may only be advanced. This limitation is imposed to ensure consistent behavior over many systems, and especially to ensure consistency of time stamps on files. The system time can be still be adjusted backwards using the settimeofday() system call. The settimeofday() system call is best used to set system clocks to the current system time, and ensure that the system time is consistent with the clocks of other systems on the network.

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 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);
}
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
    
    seconds\n");
}
fawcett:330 child$ g++ -O2 timings.C
fawcett:330 child$ ./a.out
done executing, took 0.000000
fawcett:330 child$
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);
    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 child$ 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 %f\n", seconds);
}
fawcett:330 child$ g++ -o2 timings2.C
fawcett:330 child$ ./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 ~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)
• \( \to O(n^2) \)
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(n²)
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;
}
```  

- 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 TwoStudentsWithName(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: $O(n\log n) + O(n) \rightarrow O(n\log n)$
  – (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...
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
   The `memcpy()` function conforms to ISO/IEC 9899:1990 (``ISO C90'').
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");
}

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;
}

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
dehlllloorw
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
  – This about it from the perspective of the user
  – NOT the implementor
Data structure for specific data type
(example: array of Students)

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

Abstract data structure for *any* data type
(example: search for “void *”)

Data structure for *any* data type
(example: array of “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)
  - Abstract data structure for specific data type (example: search for student names)

- When I want to solve a problem for many, many people:
  - Data structure for *any* data type (example: array of “void *”) (if the context is programming, this might typically be an array, not a pointer)

- When writing something for my specific program:
  - Abstract data structure for *any* data type (example: search for ”void *”)

- Never?
Specific data structure for specific type

Specific data structure for generic type

Abstract data types

No management/organization of data
Project 2H

Assignment: You will implement 3 structs and 9 functions. All the files (.c, .h, Makefile, correct output, grader program) needed for this project are available on the course website.

The three structs are Rectangle, Circle, and Triangle, and are described below.

The 3 structs refer to 3 different shapes: Triangle, Circle, and Rectangle. For each shape, there are 3 functions: Initialize, GetArea, and GetBoundingBox. You must implement 9 functions total (3*3).

The prototypes for these 9 functions are available in the file prototypes.h

There is also a driver program (driver_2C.c) that calls your functions and prints the results to stdout. The correct ground-truth output for the driver program is contained in the driver_output file.

Note: this project will serve as input to 3A!