Lectures 18 & 19: Deques and Miscellaneous C Topics
## Plan For The Rest Of The Term

<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tues</th>
<th>Weds</th>
<th>Thurs</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3A assigned</td>
<td></td>
<td>3B, 3C assigned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hank OH</td>
<td></td>
<td>3A due</td>
<td>3B due</td>
<td></td>
<td></td>
<td>3C “due”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3D assigned</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/1</td>
<td>12/2</td>
<td>12/3</td>
<td>12/4</td>
<td>12/5</td>
<td>12/6</td>
<td>12/7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3C due</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>extension to this day</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12/8</td>
<td>12/9</td>
<td>12/10</td>
<td>12/11</td>
<td>12/12</td>
<td>12/13</td>
<td>12/14</td>
</tr>
<tr>
<td></td>
<td>Final, 8am-10am</td>
<td></td>
<td>Last day to turn in work, 3D due</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
midterm curve

Hi Everyone,

Thank you all for your thoughtful feedback today about the midterm grades. I read all of your comments this afternoon. Here are some (paraphrased) thoughts that stuck out for me:
- "We need to settle this and move on."
- "It is stressing me out to not have this resolved and know where I stand in the class."
- "I liked the proposed curve." (strong majority of responses)
- "I like the idea of the final having extra weight." (mentioned by ~10-15% of people)

I also want to thank those that sent positive comments. Further, there were many selfless people saying "this won't benefit me in any way, but I am supportive."

I believe there are generally three groups of people:
- G1: folks who did well on the midterm
- G2: folks who did less well on the midterm and would benefit from the curve
- G3: folks who did less well on the midterm and would not benefit from the curve

I think G2 wants the curve and G3 wants the option to have the final have extra weight. After consideration, I plan on the following, which I believe will benefit G1, G2, and G3.

Your midterm score will be:  
MAX(  
Q1+Q2+Q3+Q4+Q5 .... +1,  
2*Q1+Q2+0.2*Q3+0.9*Q4+0.9*Q5 
)
3A/3B grades

• Some of you were very unhappy
  – ... But I appreciated the conscientious and diplomatic messages
• I do strive to have fair grades
• When there is a checker and you decline to run the checker, then I am not sympathetic
• When there is no checker and it is worth a lot of points ... I am sympathetic
Midterm – Monday Dec 9th @ 8am

• No practice exam – midterm difficulty was representative
• There will be coding
  – I will not deduct too much for things like missing semi-colons
  – But I will grade quite lowly if it looks like you don’t know how to code in C
  – Since we have more time, I might do a multi-page question
• I also have emphasized memory and want you to know about memory
• I also expect you to feel comfortable answering questions about stacks, linked lists, queues, and hash tables, as well as reasoning about computational complexity
• There may also be:
  – True / false
  – Unix
Oversleeping for the final

• From class syllabus:
• “If you miss the Final exam period, you may take the Final afterwards for half-credit. Exceptions for full credit will be granted in appropriate situations. These situations include medical emergencies, etc, and do not include oversleeping, forgetting the day of the final, etc.”
Late Pass Decisions

• There will be a spot on the final

Name: ________________________________

Answer the questions in the spaces provided below the questions. If you run out of room for an answer, continue on the back of the page.

Please write your name on every page!

This is a closed-book test. No notes, no calculators, no computers.

Apply Late Pass #1 To:
Apply Late Pass #2 To:

List Projects That You Feel Have Been Improperly Graded:
Notes

- No notes
- Closed book
Will the Final Be Hard?

• I used to say “no”...
  but now I say “yes”...

I want to understand what you have learned about C/Unix/data structures

Everything in lecture is fair game
But: almost all parts of lecture are reinforced in the homeworks
No Lab This Week

• Lab this week is canceled
• Instead, extra OH time
• You can attend any of the OHs (not just your assigned lab time)
YouTube Video Ideas?

• (no one sent me email reminders last time)
HashTable

• Idea:
  – Create a big array with keys and values
  – But: don’t insert starting from the beginning
  – Instead: insert into “random” places in the array
  – Not truly random, as it needs to be reproducible
  – Typical: take key and perform some math operation on it

This will be a whirlwind intro to hash tables. We will return to this idea later (NOW).
# HashTable

<table>
<thead>
<tr>
<th>Table index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Student</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
</tr>
</tbody>
</table>
HashTable

<table>
<thead>
<tr>
<th>Table index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>...34</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Student</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>xFF</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
</tr>
</tbody>
</table>

Store:
UO_ID: 951001234
Student: “xFF”

Idea:
Turn UO_ID into an index.
In this case, %10.
(much more complex ideas)
## HashTable

<table>
<thead>
<tr>
<th>Table index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>...34</td>
<td>-1</td>
<td>..66</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Student</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>xFF</td>
<td>NUL</td>
<td>xAF</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
</tr>
</tbody>
</table>

Store:
UO_ID: 951003266
Student: “xAF”
# HashTable

<table>
<thead>
<tr>
<th>Table index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>...34</td>
<td>...44</td>
<td>..66</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Student</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>xFF</td>
<td>xAA</td>
<td>xAF</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
</tr>
</tbody>
</table>

- **Store:**
  - UO_ID: 951012344
  - Student: “xAA”

- **Idea:**
  - Slot 4 is full ... just use the next slot (slot 5).
  - Keep going until you find one, including wraparounds
# HashTable

<table>
<thead>
<tr>
<th>Table index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>...34</td>
<td>...44</td>
<td>...66</td>
<td>...45</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Student</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>xFF</td>
<td>xAA</td>
<td>xAF</td>
<td>xB8</td>
<td>NUL</td>
<td>NUL</td>
</tr>
</tbody>
</table>

Store:
UO_ID: 951012345
Student: “xB8”
Fetch:
UO_ID: 951045323
EASY: NULL
# HashTable

<table>
<thead>
<tr>
<th>Table index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>...34</td>
<td>...44</td>
<td>...66</td>
<td>...45</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Student</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>xFF</td>
<td>xAA</td>
<td>xAF</td>
<td>xB8</td>
<td>NUL</td>
<td>NUL</td>
</tr>
</tbody>
</table>

Fetch:
UO_ID: 951012345
More work ... walk from index 5 to 6 to 7 ... found it!
### HashTable

<table>
<thead>
<tr>
<th>Table index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>...34</td>
<td>...44</td>
<td>..66</td>
<td>...45</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Student</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>NUL</td>
<td>xFF</td>
<td>xAA</td>
<td>xAF</td>
<td>xB8</td>
<td>NUL</td>
<td>NUL</td>
</tr>
</tbody>
</table>

**Fetch:**

UO_ID: 951012355
More work ... walk from index 5 to 6 to 7 to 8... not there!
Another Data Structure for Store/Fetch: Hash Table

```c
typedef struct
{
    int     keys[MAX_STUDENTS];
    Student *values[MAX_STUDENTS];
} HashTable;

void Initialize(HashTable *arr)
{
    for (int i = 0; i < MAX_STUDENTS; i++)
    {
        arr->keys[i] = -1;
        arr->values[i] = NULL;
    }
}

void HashTableStore(void *a, int key, Student *v)
{
    HashTable *ht = (HashTable *) a;
    int iter = 0;
    for (int i = 0; i < MAX_STUDENTS; i++)
    {
        int idx = (i+key)%MAX_STUDENTS;
        if (ht->keys[idx] == -1)
        {
            ht->keys[idx] = key;
            ht->values[idx] = v;
            return
        }
    }
    exit(EXIT_FAILURE);
}

Student *HashTableFetch(void *a, int key)
{
    HashTable *ht = (HashTable *) a;
    int iter = 0;
    for (int i = 0; i < MAX_STUDENTS; i++)
    {
        int idx = (i+key)%MAX_STUDENTS;
        if (ht->keys[idx] == key)
            return ht->values[idx];
        if (ht->keys[idx] == -1)
            return NULL;
    }
    return NULL;
}
```

- Observation:
- Not very generic (int key, Student value)
Complexity for Hash Table

• Store: it depends
  – Things go well: $O(1)$
  – Things go poorly: $O(n)$

• Fetch: it depends
  – Things go well: $O(1)$
  – Things go poorly: $O(N)$

• Gets into new topic ... expected performance.
New Slides: Hash Tables
Hash Tables: Dealing With Collisions

• Collision:
  – HASH(1004) := 4
  – HASH(1014) := 4
  – ... both get mapped to index 4, called “collision”

• How to deal with collision?
  – Open addressing: What we did just now
  – Another idea: “chaining”
  – And more ideas beyond that

Will explore open addressing next, then discuss chaining
Hash Table Taxonomy

Open Addressing
- Linear Probing
- Quadratic Probing
- Double Hashing
- Other Techniques

Chaining
- Linked Lists
- Cuckoo Hashing
- Other Techniques

More...

+ choice of hash function
Probing

• When there’s a collision, and you have to find an entry to store it
• “Probing”: how to look for the next open entry
• “Linear probing”: look at the next entry, then the next one, etc.
  – (Our example code)
Clustering

• Clustering happens with bad hash functions

• This makes store and fetch slow

• Ideas:
  – Probe more intelligently (next slides)
  – Better hash function (in 10 slides or so)
Quadratic Probing

• Approach for avoiding clusters in hash table
• $H := \text{HASH}(\text{key})$
• Linear probing looks at
  – $H, H+1, H+2, H+3, H+4, \text{ etc.}$
• Quadratic probing looks at
  – $H, H+1, H+4, H+9, H+16, \text{ etc.}$
Double Hashing

• Idea: two hash functions! ... $H_1$ and $H_2$
• For key $K$, apply $H_1$ to get index $H_1(K)$
• If $H_1(K)$ is open, then use that index.
• If $H_1(K)$ is not open (collision), then:
  – Apply $H_2$ to get index $H_2(K)$
  – Do probing with
    • $H_1(K) + H_2(K)$, $H_1(K) + 2* H_2(K)$, $H_1(K) + 3*H_2(K)$, etc.
  – need to be careful you cover all entries in the table
Hash Table Taxonomy

Open Addressing:
- Linear Probing
- Quadratic Probing
- Double Hashing
- Other Techniques

Chaining:
- Linked Lists
- Cuckoo Hashing
- Other Techniques

More...

+ choice of hash function
Chaining: Linked Lists

- (called “separate chaining” on wikipedia)
- Idea: every element in the array is the head of a linked list
- When there is a collision, you put the new item at the front of the linked list
- Some linked lists may get long
- Requires linked list stuff (malloc, extra pointers, etc)
Deletions

• Chaining
  – Easy! Go to the linked list and find the item to remove

• Open Addressing
  – “Lazy Deletion”: don’t delete it, but indicate that it is removed

```
typedef struct
{
    int keys[MAX_STUDENTS];
    Student *values[MAX_STUDENTS];
    int isValid[MAX_STUDENTS];
} HashTable;
```
The simplest approach to a hash function for character strings is to simply sum the values of the ASCII characters in the string.

```c
long hash1(char *key, long N) {
    unsigned long sum = 0L;
    char *p;

    for (p = key; *p != '\0'; p++)
        sum += (unsigned long)(*p);
    return (long)(sum % N);
}
```
Why Previous Hash Function Was Bad

• Most words are about the same length
• ASCII characters have similar values (‘A’ == 65, ‘Z’ == 90)
• Almost all words will get hash values between 65 and 900
• If we hash a dictionary of 125000 words into a hash table of size 250000, we will get a lost of clustering
Reducing Clustering
(Polynomial Hash Function)

```c
#define A 31L
long hash2(char *key, long N) {
    unsigned long sum = 0L;
    char *p;

    for (p = key; *p != '\0'; p++)
        sum = A * sum + (unsigned long)(*p);
    return (long)(sum % N);
}
```
## Statistics

<table>
<thead>
<tr>
<th>Function</th>
<th>Total</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Median</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash1</td>
<td>123,237</td>
<td>0</td>
<td>639</td>
<td>78.8</td>
<td>18</td>
<td>128.3</td>
</tr>
<tr>
<td>hash2(A=1)</td>
<td>123,237</td>
<td>0</td>
<td>639</td>
<td>78.8</td>
<td>18</td>
<td>128.3</td>
</tr>
<tr>
<td>hash2(A=7)</td>
<td>6,072</td>
<td>0</td>
<td>6</td>
<td>0.0522</td>
<td>0</td>
<td>0.261</td>
</tr>
<tr>
<td>hash2(A=19)</td>
<td>806</td>
<td>0</td>
<td>2</td>
<td>0.00650</td>
<td>0</td>
<td>0.0812</td>
</tr>
<tr>
<td>hash2(A=29)</td>
<td>642</td>
<td>0</td>
<td>1</td>
<td>0.00517</td>
<td>0</td>
<td>0.0717</td>
</tr>
<tr>
<td>hash2(A=31)</td>
<td>642</td>
<td>0</td>
<td>1</td>
<td>0.00517</td>
<td>0</td>
<td>0.0717</td>
</tr>
<tr>
<td>hash2(A=37)</td>
<td>642</td>
<td>0</td>
<td>1</td>
<td>0.00517</td>
<td>0</td>
<td>0.0717</td>
</tr>
<tr>
<td>hash2(A=73)</td>
<td>642</td>
<td>0</td>
<td>1</td>
<td>0.00517</td>
<td>0</td>
<td>0.0717</td>
</tr>
<tr>
<td>hash2(A=129)</td>
<td>642</td>
<td>0</td>
<td>1</td>
<td>0.00517</td>
<td>0</td>
<td>0.0717</td>
</tr>
<tr>
<td>hash2(A=257)</td>
<td>642</td>
<td>0</td>
<td>1</td>
<td>0.00517</td>
<td>0</td>
<td>0.0717</td>
</tr>
<tr>
<td>hash2(A=511)</td>
<td>642</td>
<td>0</td>
<td>1</td>
<td>0.00517</td>
<td>0</td>
<td>0.0717</td>
</tr>
</tbody>
</table>
3D

• Implement two types of maps
  – Linked list
  – Hash table

• Analyze performance and write very short report (one paragraph)
Deques
Deque – Double Ended Queue

- New data structure
- Pronounced “deck”
- Queue:
  - $O(1)$ insert
  - $O(1)$ to remove head
  - $O(n)$ to remove tail
- Deque:
  - $O(1)$ insert
  - $O(1)$ to remove head
  - $O(1)$ to remove tail
Deque: key methods

- Insert first
- Insert last
- Remove first
- Remove last

- Real world examples?
From Sventek Book

- Head’s prev is NULL
- Tail’s next is NULL
Example exercise

• Palindrome checker
Misc. Topics
const

• const:
  – is a keyword in C
  – qualifies variables
  – is a mechanism for preventing write access to variables
const example

The compiler enforces const ... just like public/private access controls
Efficiency

Are any of the three for loops faster than the others? Why or why not?

Answer: NumIterations is slowest … overhead for function calls.

Answer: X is probably faster than Y … compiler can do optimizations where it doesn’t have to do “i < X“ comparisons (loop unrolling)
const arguments to functions

- Functions can use const to guarantee to the calling function that they won’t modify the arguments passed in.

```c
struct Image
{
    int width, height;
    unsigned char *buffer;
};
ReadImage(char *filename, Image *);
WriteImage(char *filename, const Image *);
```

read function can’t make the same guarantee

 guarantees function won’t modify the Image
const pointers

• Assume a pointer named “P”

• Two distinct ideas:
  – P points to something that is constant
    • P may change, but you cannot modify what it points to via P
  – P must always point to the same thing, but the thing P points to may change.
const pointer

• Assume a pointer named “P”

• Two distinct ideas:
  – P points to something that is constant
    • P may change, but you cannot modify what it points to via P
  – P must always point to the same thing, but the thing P points to may change.
**Idea #1:** violates `const`:

```
    *P = 3;
```

**OK:**

```
    int Y = 5; P = &Y;
```

**pointer can change, but you can’t modify the thing it points to**

**Idea #2:** violates `const`:

```
    int Y = 5; P = &Y;
    *P = 3;
```

**OK:**

```
    int Y = 5; P = &Y;
    *P = 3;
```

**pointer can’t change, but you can modify the thing it points to**
const pointers

int X = 4;
int *P = &X;

Idea #3: violates const:

"*P = 3;"
"int Y = 5; P = &Y;"

OK:
none

pointer can’t change, and you can’t modify the thing it points to
const pointers

```
int X = 4;
int *P = &X;
```

Idea #1:
violates const:
```
*P = 3;
```
OK:
```
int Y = 5; P = &Y;
```

pointer * can change, but you can’t modify the thing it points to
const pointers

\[
\begin{align*}
\text{int } X &= 4; \\
\text{int } *P &= &\&X;
\end{align*}
\]

Idea #2: violates const:
“int Y = 5; P = &Y;”

OK:
“*P = 3;”

pointer can’t change, but you can modify the thing it points to
const pointers

Idea #3:
violates const:
  "*P = 3;"
  "int Y = 5; P = &Y;"
OK:
  none

pointer can’t change, and you can’t modify the thing it points to
const usage

- class Image;
- const Image *ptr;
  - Used a lot: offering the guarantee that the function won’t change the Image ptr points to
- Image * const ptr;
  - Helps with efficiency. Rarely need to worry about this.
- const Image * const ptr;
  - Interview question!!
globals
globals

- You can create global variables that exist outside functions.

```c
#include <stdio.h>
int X = 5;

int main()
{
    printf("X is %d\n", X);
}
```

```bash
fawcett:Documents child$ cat global1.C
#include <stdio.h>
int X = 5;

int main()
{
    printf("X is %d\n", X);
}
```

```
$ g++ global1.C
$ ./a.out
X is 5
```

global variables

- global variables are initialized before you enter main

```c
#include <stdio.h>

intInitializer()
{
    printf("In initializer\n");
    return 6;
}

int X = Initializer();

int main()
{
    printf("In main\n");
    printf("X is %d\n", X);
}
```

```
fawcett:Documents child$ cat global2.C
#include <stdio.h>
int Initializer()
{
    printf("In initializer\n");
    return 6;
}

int X = Initializer();

int main()
{
    printf("In main\n");
    printf("X is %d\n", X);
}
fawcett:Documents child$ g++ global2.C
fawcett:Documents child$ ./a.out
In initializer
In main
X is 6
```
Storage of global variables...

- global variables are stored in a special part of memory
  - “data segment” (not heap, not stack)
- If you re-use global names, you can have collisions

```c
fawcett:Documents child$ cat file1.C
int X = 6;
int main()
{
}
fawcett:Documents child$ g++ -c file1.C
fawcett:Documents child$ cat file2.C
int X = 7;
int doubler(int Y)
{
    return 2*Y;
}
fawcett:Documents child$ g++ -c file2.C
fawcett:Documents child$ g++ file1.o file2.o
ld: duplicate symbol _X in file2.o and file1.o
collect2: ld returned 1 exit status
```
Externs: mechanism for unifying global variables across multiple files

```c
extern int count;

int doubler(int Y)
{
    count++;
    return 2*Y;
}
```

`extern`: there’s a global variable, and it lives in a different file.
There are three distinct usages of statics

- static memory: third kind of memory allocation
  - reserved at compile time
- contrasts with dynamic (heap) and automatic (stack) memory allocations
- accomplished via keyword that modifies variables
static usage #1: persistency within a function

fawcett:330 child$ cat static1.C
#include <stdio.h>

int fibonacci()
{
    static int last2 = 0;
    static int last1 = 1;
    int rv = last1+last2;
    last2 = last1;
    last1 = rv;
    return rv;
}

int main()
{
    int i;
    for (int i = 0 ; i < 10 ; i++)
        printf("%d\n", fibonacci());
}
static usage #2: making global variables be local to a file

I have no idea why the static keyword is used in this way.
Miscellaneous C Topics
## Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>++ -- ()</td>
<td>Suffix/postfix increment and decrement, Function call, Array subscripting, Structure and union member access, Structure and union member access through pointer, Compound literal(C99)</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>2</td>
<td>+ - ! ~(type) + &amp; sizeof _Alignof</td>
<td>Prefix increment and decrement, Unary plus and minus, Logical NOT and bitwise NOT, Type cast, Indirection (dereference), Address-of, Size-of, Alignment requirement(C11)</td>
<td>Right-to-left</td>
</tr>
<tr>
<td>3</td>
<td>* / %</td>
<td>Multiplication, division, and remainder</td>
<td>Left-to-right</td>
</tr>
<tr>
<td>4</td>
<td>+ -</td>
<td>Addition and subtraction</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>&lt;&lt; &gt;&gt;</td>
<td>Bitwise left shift and right shift</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>&lt;= &gt;=</td>
<td>For relational operators &lt;= and &gt;= respectively, For relational operators &gt; and &gt;= respectively</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>== !=</td>
<td>For relational = and ≠ respectively</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>&amp;</td>
<td>Bitwise AND</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>^</td>
<td>Bitwise XOR (exclusive or)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>Bitwise OR (inclusive or)</td>
</tr>
<tr>
<td>11</td>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>[:]</td>
<td>Ternary conditional[note 2]</td>
<td>Right-to-Left</td>
</tr>
<tr>
<td>14</td>
<td>= += -= *= /= %= &lt;&lt;= &gt;&gt;= &amp;= ^=</td>
<td>=</td>
<td>Simple assignment, Assignment by sum and difference, Assignment by product, quotient, and remainder, Assignment by bitwise left shift and right shift, Assignment by bitwise AND, XOR, and OR</td>
</tr>
<tr>
<td>15</td>
<td>,</td>
<td>Comma</td>
<td></td>
</tr>
</tbody>
</table>

DRAM vs NV-RAM

• DRAM: Dynamic Random Access Memory
  – stores data
  – each bit in separate capacitor within integrated circuit
  – loses charge over time and must be refreshed
  – → volatile memory

• NV-RAM: Non-Volatile Random Access Memory
  – stores data
  – information unaffected by power cycle
  – examples: Read-Only Memory (ROM), flash, hard drive, floppy drive, ...
Relationship to File Systems

• File Systems could be implemented in DRAM.
• However, almost exclusively on NV-RAM
  – Most often hard drives
• Therefore, properties and benefits of file systems are often associated with properties and benefits of NV-RAM.
## DRAM vs NV-RAM properties

<table>
<thead>
<tr>
<th>Property</th>
<th>DRAM</th>
<th>NV-RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>~10GB</td>
<td>~10TB</td>
</tr>
<tr>
<td>Cost</td>
<td>$5/GB</td>
<td>$0.03/GB</td>
</tr>
<tr>
<td>Latency</td>
<td>&lt;100 nanoseconds</td>
<td>~10 milliseconds</td>
</tr>
</tbody>
</table>

### Distance:
- A 20” map of Oregon is 1:100,000 scale.

### Time:
- 1 second to 27 hours is 1:100,000 scale.
- 1 minute to 69 days is 1:100,000 scale.
- 1 hour to 11 years is 1:100,000 scale.
- 1 day to 273 years is 1:100,000 scale.
Hank’s Unsolicited Advice
Internships

• Great opportunity
  – Learn what you like and don’t like
  – Networking
  – Helps bolster your resume
Things you should be doing during your undergrad

• Personal projects?
• Good grades
• Investing in yourself, esp. coding
  – It is very likely you will have an interview where you are asked to code. Practice early and often.
Grad School?

- **MS**: possibly better job than BS
  - Possibly not

- **PhD**: opens the door to research (being at the leading edge of computing)
  - Closes other doors
Big Analogy:
Training for a Marathon
My mental model about getting behind in this class

If you get too far behind, then it will not be possible to catch up.
This class

• This class was challenging

• Many of you stayed on the “Pacific Crest Trail”
  – You should be very proud ... you have done something very hard

• If you keep “running,” the “marathon” is well within your reach
How a Computer is Not Like Monopoly

- Actions during cycle/turn:
  - Monopoly: roll, buy, build, trade
  - Computer: other

- Who’s turn?
  - Monopoly: passes between players
  - Computer: always the computer’s turn

- Time spent per cycle/turn:
  - Monopoly: variable
  - Computer: fixed

- Duration of turn:
  - Monopoly: sometimes over a minute
  - Computer: much faster!!!
“First” computer: ENIAC

- Year: 1946
- Location: Pennsylvania
- Purpose: military
- Cost: $487K
  - ($6.9M today)
- Technology:
  - very different than today
  - ... but still the same
Vacuum Tubes

• Vacuum tubes:
  – Glass tubes with no gas
  – Used to control electron flow in early computers

• Occasionally, a bug would get stuck in the tube and cause the program to malfunction

• We no longer have vacuum tubes, but the term bug has remained with us...
An ENIAC Computation

• Used for military calculations:
  – A-bomb design
  – Missile delivery

• ENIAC could do ~5000 calculations in one minute

• In one case:
  – ENIAC did a calculation in 30 seconds
  – Human being took 20 hours
  – 2400x increase in speed
Hertz (Hz) = unit of measurement for how fast you do something

- 1 Hertz = do something once per second
- KHz = 1024 Hz
- MHz = 1024 KHz
- GHz = 1024 MHz

- The ENIAC machine ran at 5000 Hertz, or about 5 KHz.
  - Vocab term: “clock speed” → the number of cycles per second
    - the clock speed of the ENIAC was 5 KHz
Today’s Desktop Computers Are Fast!

- Most computers run at ~1-3 GHz
- i.e., operates billions of instructions each second

- This is about one million times faster than the ENIAC
  - ... and the ENIAC was 2400X faster than humans
  - (at least at tasks computers are good at)
What does a million-fold increase mean?

Distance: a 2” map of Oregon is 1:1,000,000 scale

Time: 1 second to 277 hours is 1:1,000,000 scale

Time: 1 minute to 694 days is 1:1,000,000 scale

Time: 1 hour to 114 years is 1:1,000,000 scale

Time: 1 day to 2738 years is 1:1,000,000 scale
1 million-fold increase! How does this happen?

- Moore’s Law (old timer’s version)
  - Clock speed doubles every 18 months
- Moore’s Law (newer version but still for old timers)
  - Clock speed doubles every 24 months
Moore’s Law

- Moore’s Law (actual version)
  - Number of transistors doubles every 24 months
  - And clock speed is a reflection of number of transistors

- So what is a transistor?
  - Semiconductor device for amplifying or switching electronic signals/power
  - Fundamental building block of modern electronics
  - Replacement for vacuum tube
Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.

The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic. Licensed under CC-BY-SA by the author Max Roser.
But actually…

Source: maximumpc.com
The reason is power

• Desktop computer takes ~200W
  – There are multiple components that consume the power:
    • CPU
    • Monitor
    • Disk
    • Memory

• 200W * 1 year → ~$70
Relationship Between Power and Clock Speed

• Clock goes twice as fast $\rightarrow$ Power goes up by factor of 8
  – (Increase of $X$ in clock speed $\rightarrow$ Increase of $X^3$ in power)

• Clock speeds haven’t changed in 12 years

• What if they had doubled every 2 years?
  • Then 64X faster
    – $\rightarrow$ 262144X more power (for the CPU)
    – $\rightarrow$ power bill now $18M$
New vocab term: "core"

What Changed?

• We are getting double the transistors every two years
• … but clock speed is the same
• … so what is changing?

### CHOOSE YOUR OPTIMIZATION POINT

<table>
<thead>
<tr>
<th></th>
<th>CORES</th>
<th>GHZ</th>
<th>MEMORY</th>
<th>FABRIC</th>
<th>DDR4</th>
<th>POWER²</th>
<th>RECOMMENDED CUSTOMER PRICING³</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>7290¹</strong></td>
<td>72</td>
<td>1.5</td>
<td>16GB</td>
<td>Yes</td>
<td>384GB</td>
<td>245W</td>
<td>$6254</td>
</tr>
<tr>
<td>Best Performance/Node</td>
<td></td>
<td></td>
<td>7.2 GT/s</td>
<td></td>
<td>2400 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7250</strong></td>
<td>68</td>
<td>1.4</td>
<td>16GB</td>
<td>Yes</td>
<td>384GB</td>
<td>215W</td>
<td>$4876</td>
</tr>
<tr>
<td>Best Performance/Watt</td>
<td></td>
<td></td>
<td>7.2 GT/s</td>
<td></td>
<td>2400 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7230</strong></td>
<td>64</td>
<td>1.3</td>
<td>16GB</td>
<td>Yes</td>
<td>384GB</td>
<td>215W</td>
<td>$3710</td>
</tr>
<tr>
<td>Best Memory Bandwidth/Core</td>
<td></td>
<td></td>
<td>7.2 GT/s</td>
<td></td>
<td>2400 MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>7210</strong></td>
<td>64</td>
<td>1.3</td>
<td>16GB</td>
<td>Yes</td>
<td>384GB</td>
<td>215W</td>
<td>$2438</td>
</tr>
<tr>
<td>Best Value</td>
<td></td>
<td></td>
<td>6.4 GT/s</td>
<td></td>
<td>2133 MHz</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

¹Available beginning in September ² Plus 15W for integrated fabric
³Pricing shown is for parts without integrated fabric. Add additional $278 for integrated fabric versions of these parts. Integrated fabric parts available in October.
How To Use Multiple Cores?

• Answer: parallel programming
  – Write computer programs that use all the cores
  – Ideally the coordination between the cores is minimal
Parallel Programming Concepts

• Usual goal:
  – if program takes $N$ seconds to run with one core
  – then take $N/2$ seconds to run with two cores
  – and $N/M$ seconds to run with $M$ cores

Let’s consider an example outside of computers
Example: paint a house

• One person: 6 days (1 day = 10 hours)
• Two people: 3 days
• Three people: 2 days
• Six people: 1 day

• Sixty people: 1 hour?
• Six hundred people: 6 minutes?
Example: paint a house, plan #2

- One person: paint one house in 6 days
- Two people: paint two houses in 6 days
- Three people: paint three houses in 6 days
- One thousand people: paint 1000 houses in 6 days?

Parallel programming is hard, and smart people spend their whole careers figuring out how to make parallel programs be efficient
GPUs: Graphical Processing Units (graphics cards)

• Historical:
  – Introduced to accelerate graphics (gaming!)
  – Boom with desktop PCs in the late 90s onward
  – Mid-2000’s: people start hacking the interface to program a GPU to make it do things besides graphics
  – Late 2000’s: GPU makers jump on board and start encouraging folks to program GPUs directly

• GPGPU: General-purpose GPU programming
  – Mid 2010’s: GPUs used for *lots* of computing problems.
  – Machine learning workhorse!
Why Are GPUs So Good?

Market summary > NVIDIA Corporation
NASDAQ: NVDA - Mar 5, 7:59 PM EST

235.65 USD ↓0.89 (0.38%)
After-hours: 236.50 ↑0.36%

1 day | 5 day | 1 month | 3 month | 1 year | 5 year | max
12.82 Mar 8, 2013

• NVIDIA: company that makes GPUs
• NVIDIA Volta: latest type of NVIDIA GPU
• Volta facts:
  – 5120 cores
  – 1200MHz clock speed
  – can do 2000X more operations than my laptop
  – Suggested MSRP: $2,999.00

This level of increase in computation is not just a quantitative change, it is a qualitative one too.