Quiz: Section 4.10

1) #include <stdio.h>
2) void intToStr(int n)
3) { if (n / 10)
4) { intToStr(n);
5) }
6) putchar(n % 10 + '0');
7) }
8)
9) void main(void)
10) { intToStr(342);
11) }
12)

This program will cause a segmentation fault because:

a) Line 5 is a function call to the function that line 5 is inside.
b) Line 5 uses recursion.
c) Line 5 uses recursion and intToStr, is not declared recursive.
d) Line 5 is a recursive call to a function that returns void.
e) The function intToStr(n), can call intToStr(n) without changing the value of n.
Recursion Example: `intToStr`

```c
#include <stdio.h>

void intToStr(int n)
{
    if (n / 10)
    {
        intToStr(n / 10);
    }
    putchar(n%10 + '0');
}

void main(void)
{ intToStr(342);
}
```

Recursion
Fibonacci Numbers: Recursive Definition

Recursive Definition:

Recurrence relation: \( f_{n+2} = f_{n+1} + f_n \)
Base case: \( f_1 = 1, \quad f_0 = 1 \)

Example:

\[
\begin{align*}
    f_7 &= f_6 + f_5 \\
    f_6 &= f_5 + f_4 \\
    f_5 &= f_4 + f_3 \\
    f_4 &= f_3 + f_2 \\
    f_3 &= f_2 + f_1 \\
    f_2 &= f_1 + f_0
\end{align*}
\]

Fibonacci Numbers and Tiling

Fibonacci Number Sequence:

\[
1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21 \quad 34 \quad 55 \quad 89 \quad 144 \\
233 \quad 377 \quad 610 \quad 987 \quad 1597 \quad 2584 \quad 4181 \quad 
\]

A Fibonacci Tiling is a tiling of the plane with squares whose sides have length equal to successive Fibonacci numbers.
Fibonacci Spiral

Fibonacci Spiral in Romanesco Broccoli
Fibonacci Spiral in Nature

Fibonacci Spiral in Physics
Fibonacci Spiral in Art

Fibonacci Series and the Golden Ratio

```c
void main(void)
{
    long f0 = 1;
    long f1 = 1;
    int i;
    for (i=2; i<26; i++)
    {
        double ratio = (double)f1 / (double)f0;
        printf("%6ld / %ld\t=%.9f\n", f0,f1,ratio);
        long f2 = f1 + f0;
        f0 = f1;
        f1 = f2;
    }
}
```
Fibonacci Series and the Golden Ratio

<table>
<thead>
<tr>
<th>Fraction</th>
<th>Ratio Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 / 1</td>
<td>1.0000000000</td>
</tr>
<tr>
<td>1 / 2</td>
<td>1.0000000000</td>
</tr>
<tr>
<td>2 / 3</td>
<td>1.0000000000</td>
</tr>
<tr>
<td>3 / 5</td>
<td>1.6666666667</td>
</tr>
<tr>
<td>5 / 8</td>
<td>1.6000000000</td>
</tr>
<tr>
<td>8 / 13</td>
<td>1.6250000000</td>
</tr>
<tr>
<td>13 / 21</td>
<td>1.615384615</td>
</tr>
<tr>
<td>21 / 34</td>
<td>1.619047619</td>
</tr>
<tr>
<td>34 / 55</td>
<td>1.617647059</td>
</tr>
<tr>
<td>55 / 89</td>
<td>1.618181818</td>
</tr>
<tr>
<td>89 / 144</td>
<td>1.61777528</td>
</tr>
<tr>
<td>144 / 233</td>
<td>1.618055556</td>
</tr>
<tr>
<td>233 / 377</td>
<td>1.618025751</td>
</tr>
<tr>
<td>377 / 610</td>
<td>1.618037135</td>
</tr>
<tr>
<td>610 / 987</td>
<td>1.618032787</td>
</tr>
<tr>
<td>987 / 1597</td>
<td>1.618034448</td>
</tr>
<tr>
<td>1597 / 2584</td>
<td>1.618033813</td>
</tr>
<tr>
<td>2584 / 4181</td>
<td>1.618034056</td>
</tr>
<tr>
<td>4181 / 6765</td>
<td>1.618033963</td>
</tr>
<tr>
<td>6765 / 10946</td>
<td>1.618033999</td>
</tr>
<tr>
<td>10946 / 17711</td>
<td>1.618033985</td>
</tr>
<tr>
<td>17711 / 28657</td>
<td>1.618033990</td>
</tr>
<tr>
<td>28657 / 46368</td>
<td>1.618033988</td>
</tr>
</tbody>
</table>

46368 / 75025 = 1.618033989

Golden Ratio in Architecture

- Acropolis
- Taj Mahal
- Eiffel Tower
Fibonacci Sequence by Recursion

```c
int fibonacci(int n)
{
    if (n==1 || n==2) return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}

void main(void)
{
    printf("%d\n", fibonacci(20));
}
```

When a function calls itself recursively, each invocation gets a separate copy of all automatic variables.

---

What Some C Coders Find Beautiful

```c
int f(int x){if(x<2)return 1;return f(x-1)+f(x-2);}
```

51 characters including spaces.

Minimalist yet complex: built from layering many circular ceramic sections within a single form.

-- Matthew Chambers
What C Program Reproduces this?

Extra Credit: Write, demo and explain a C program to draw this.

Quicksort Algorithm

- Quicksort is a divide and conquer algorithm for sorting the elements of an array.
- Performance: Average Case: O(n log n), Worst Case: O(n^2).
- Given an array, one element is chosen and the others are partitioned into two subsets:
  1) Those less than the partition element and
  2) Those greater than or equal to it.
- The same process is then recursively applied to each of the two subsets.
- When a subset has fewer than two elements, it doesn't need any sorting: this stops the recursion.
The Sound Sorting Video ($n=500$)

Bubble Sort

Quicksort

Quicksort: main()

```c
#include <stdio.h>
int arraySize;
int level = 0;

void main(void)
{
    int v[] = {23, 13, 82, 33, 51, 17, 45, 75, 11, 27};

    int arraySize = sizeof(v)/sizeof(int);
    printf("arraySize=%d\n", arraySize);

    quicksort(v, 0, arraySize-1);
}
```
Quicksort: Helper Function swap

```c
void swap(int v[], int i, int j)
{
    int c = v[i];
    v[i] = v[j];
    v[j] = c;
}
```

Quicksort: Helper Function printArray

```c
void printArray(int code, int v[], int left, int right)
{
    int i=0;
    if (code < 0) printf("  Done%2d ", -level);
    else printf("Level=%2d ", level);

    for(i=0; i<arraySize; i++)
    {
        if (i<left || i>right)
        { printf(" ");
        }
        else
        { printf("%2d ", v[i]);
        }
    }
    printf("]\n");
}
```
void quicksort(int v[], int left, int right)
{
    level++;
    printArray(level, v, left, right);
    
    int i, last;
    if (left < right) {
        swap(v, left, (left+right)/2);
        last = left;
        int num2 = v[left];
        for (i=left+1; i <= right; i++) {
            if (v[i] < v[left]) {
                last++;
                swap(v, last, i);
            }
        }
        swap(v, left, last);
        quicksort(v, left, last-1);
        quicksort(v, last+1, right);
    }
    printArray(-level, v, left, right);
    level--;
}

Nothing to sort if segment is fewer than two elements.
Move the partition item out of the partition range, then restore.

---

**Quicksort Output Trace**

<table>
<thead>
<tr>
<th>Level</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[23 13 82 33 51 17 45 75 11 27 ]</td>
</tr>
<tr>
<td>2</td>
<td>[27 13 33 23 17 45 11 ]</td>
</tr>
<tr>
<td>3</td>
<td>[11 13 17 ]</td>
</tr>
<tr>
<td>4</td>
<td>[11 ]</td>
</tr>
<tr>
<td>5</td>
<td>[17 ]</td>
</tr>
<tr>
<td>Done</td>
<td>3 [11 13 17 ]</td>
</tr>
<tr>
<td>Level</td>
<td>Array</td>
</tr>
<tr>
<td>3</td>
<td>[33 45 27 ]</td>
</tr>
<tr>
<td>4</td>
<td>[27 33 ]</td>
</tr>
<tr>
<td>5</td>
<td>[33 ]</td>
</tr>
<tr>
<td>Done</td>
<td>4 [27 33 ]</td>
</tr>
<tr>
<td>Level</td>
<td>Array</td>
</tr>
<tr>
<td>4</td>
<td>[33 ]</td>
</tr>
<tr>
<td>Done</td>
<td>3 [27 33 45 ]</td>
</tr>
<tr>
<td>2</td>
<td>[11 13 17 23 27 33 45 ]</td>
</tr>
<tr>
<td>Level</td>
<td>Array</td>
</tr>
<tr>
<td>2</td>
<td>[82 75 ]</td>
</tr>
<tr>
<td>3</td>
<td>[75 ]</td>
</tr>
<tr>
<td>3</td>
<td>[ ]</td>
</tr>
<tr>
<td>Done</td>
<td>2 [75 82 ]</td>
</tr>
<tr>
<td>Level</td>
<td>Array</td>
</tr>
<tr>
<td>1</td>
<td>[11 13 17 23 27 33 45 51 75 82 ]</td>
</tr>
</tbody>
</table>
QuickSort: Quiz

1) void quicksort(int v[], int left, int right)
2) { int i, last;
3) printArray(v, left, right);
4) if (left >= right) return;
5) swap(v, left, (left+right)/2);
6) last = left;
7) for (i=left+1; i <= right; i++)
8) { if (v[i] < v[left])
9) { last++;
10) swap(v, last, i);
11) printArray(v, left, right);
12) }
13) }
14) swap(v, left, last);
15) quicksort(v, left, last-1);
16) quicksort(v, last+1, right);
17) }

If the output from line 3 is: [75 62 33 41 24]
Then what would be the output the next time line 13 is reached?

a) [33 62 75 24 41]
b) [33 24 75 41 62]
c) [33 62 24 41 75]
d) [33 41 24 63 75]
e) [33 24 41 62 75]

QuickSort Runtime Performance

- Average Case: O(n log n)
- Worst Case: O(n^2) (same as bubble sort).
- What starting arrangement results in the worst case?
- How likely is it that uniformly distributed random data will hit or come near to hitting the worst case?
- Can the algorithm be improved by avoiding the worst case?