

Recursion and the Maze

CS 241

Data Organization using C

Instructor: **Joel Castellanos**

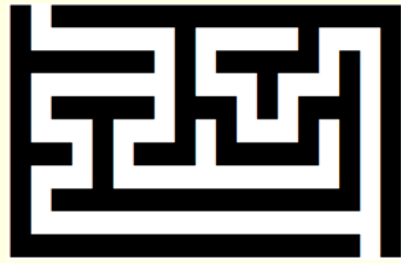
e-mail: joel@unm.edu

Web: <http://cs.unm.edu/~joel/>

Office: Farris Engineering
Center (FEC) room 321

Lab Instructor: **Dongye Meng**

e-mail: dymeng@cs.unm.edu



4/9/2009

Lab April 3: Recursion, Section 4.10

- The lab assignment this week will be similar to:

```
void printd(int n)
```

on page 87.



2

Lab April 10: malloc(), and 2D arrays

This week's lab makes use of:

- malloc(): *a single block accessed as a 2D array.*
- Command line arguments.
- Formatted output: "%n.m£".

Please read:

- 5.7: Multi-dimensional Arrays
- 5.8: Initialization of Pointer Arrays
- 5.9: Pointers vs. Multi-dimensional Arrays
- 5.10: Command-line Arguments



3

What is the Real Problem?

- During lab, Dongye, says there are very few questions.
- After lab, Dongye has office hours: no one ever goes.
- Dongye says he checks his e-mail:
 - Very few Lab questions
 - No Project questions
- What about Torin? – Look for the ray of sunlight.
- Tutors say they rarely see anyone from CS-241.

4

<stdlib.h>: malloc of 2D Array

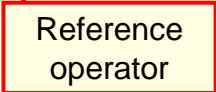
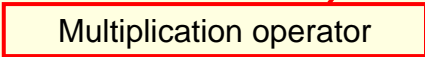

```
int rows = 10;
int columns = 20;
char **strArray;

//allocate space for (char *)'s
strArray = malloc(rows * sizeof *strArray);

int i;
for (i = 0; i < rows; i++)
{ strArray[i] = malloc(columns);
}
strArray[3][5] = 'a';
```

Multiplication operator

Reference operator



5

Fibonacci Sequence by Loop

```
int fibonacci(int n)
{ int f0 = 0;
  int f1 = 1;
  int i;
  for (i=0; i<n; i++)
  { printf("%d, ", f1);
    int f2 = f0 + f1;
    f0 = f1;
    f1 = f2;
  }
  return f1;
}
```

0,	1,	1,
2,	3,	5,
8,	13,	21,
34,	55,	89,
144,	233,	377,
610,	987,	1597,
2584,	4181,	6765,
==>10946		

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

6

Fibonacci Sequence by Recursion

```
int fibonacci(int n)
{ if (n==0) return 0;
  if (n==1) return 1;
  return fibonacci(n-1) + fibonacci(n-2);
}

main()
{
  printf("%d\n", fibonacci(7));
}
```

7

What Some C Coders Find Beautiful

```
int fib(int x) {if (x<=1) return x; return
  fib(x-1) + fib(x-2);}
```

64 characters including spaces.



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

8

-- Matthew Chambers

Recursive Fibonacci Walk-Through

```
int level = 0;   int id = 0;

int fibonacci(int n)
{ printf("%2d %sfib(%d)\n",
        id, dot(level), n);

  id++; level++;

  if (n==0) return 0;
  if (n==1) return 1;

  int a = fibonacci(n-1); level--;
  int b = fibonacci(n-2); level--;
  return a + b;
}

main()
{ printf("fib(5)=%d\n", fibonacci(5));
}
```

Output:

```
0 fib(5)
1 .fib(4)
2 ..fib(3)
3 ...fib(2)
4 ....fib(1)
5 ....fib(0)
6 ...fib(1)
7 ..fib(2)
8 ...fib(1)
9 ...fib(0)
10 .fib(3)
11 ..fib(2)
12 ...fib(1)
13 ...fib(0)
14 ..fib(1)
fib(5)=5
```

9

Quiz: Recursive Fibonacci

```
int level = 0;   int id = 0;

int fibonacci(int n)
{ printf("%2d %sfib(%d)\n",
        id, dot(level), n);

  id++; level++;

  if (n==0) return 0;
  if (n==1) return 1;

  int a = fibonacci(n-1); level--;
  int b = fibonacci(n-2); level--;
  return a + b;
}

main()
{ fibonacci(3));
}
```

- a) 0 fib(3)
1 .fib(2)
2 ..fib(1)
3 ...fib(0)
4fib(1)
- b) 0 fib(3)
1 .fib(2)
2 ..fib(1)
3 .fib(0)
4 fib(1)
- c) 0 fib(3)
1 .fib(2)
2 ..fib(1)
3 ..fib(0)
4 .fib(1)

10

Fibonacci: Closed Form Expression

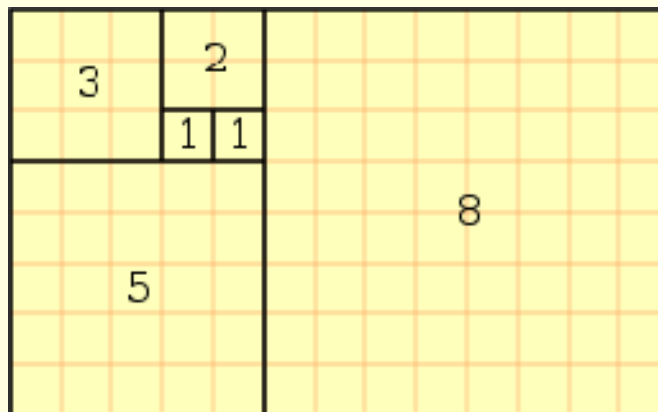
- Every sequence defined by linear recurrence, has a closed-form solution. (although it is not always easy to find).
- The Closed Form Expression of the Fibonacci sequence is:

$$F(n) = \frac{\varphi^n - (1 - \varphi)^n}{\sqrt{5}}$$

- Where φ is the golden ratio: $\varphi = \frac{1 + \sqrt{5}}{2}$

11

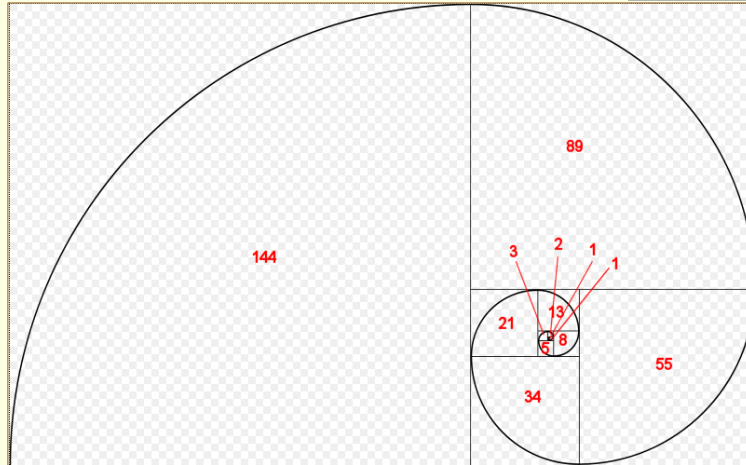
Fibonacci Tiling



A tiling with squares whose sides are successive Fibonacci numbers in length.

12

Fibonacci Spiral



Created by drawing arcs connecting the opposite corners of squares in the Fibonacci tiling.

13

Fibonacci in a Nautilus Shell



14

Fibonacci in the Sunflower



The head of a flower is made up of small seeds which are produced at the center, and migrate towards the outside.

Each new seed appears at a certain angle in relation to the preceding one.

15

Koch Curve

In its limit, the Koch Curve is:

- Everywhere continuous
- Nowhere Differentiable
- Topological dimension = 1
- Box dimension > 1

Generation 0 

Generation 1 

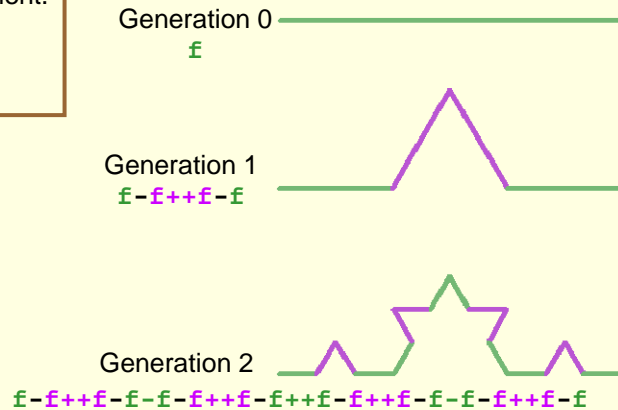
Generation 2 

Generation 3 

16

Koch Curve: A String Representation

f: Draw Unit line Segment.
-: 60° Left Turn
+: 60° Right Turn



17

substitute() - Part 1 of 2

```
char* substitute(char* source, char c, char* sub)
{ int i;
  int n=0;
  int sourceLen = strlen(source);
  int subLen    = strlen(sub);
  for (i=0; i<sourceLen; i++)
  { if (source[i] == c) n++;
  }
  char* outStr = malloc(sourceLen + n*(subLen-1));

  ....

  return outStr;
}
```

What is going on here?

18

substitute() - Part 2 of 2

```
char* substitute(char* source, char c, char* sub)
{
    .....
    char* outStr = malloc(sourceLen + n*(subLen-1));
    int k=0;
    for (i=0; i<sourceLen; i++)
    { if (source[i] == c)
      { sprintf(&outStr[k], sub);
        k += subLen;
      }
      else
      { outStr[k] = source[i];
        k++;
      }
    }
    outStr[k] = '\0';
    return outStr;
}
19 }
```

And what here?

kochCurve()

```
char* kochCurve(int n)
{ if (n==0)
  { char* baseGen = malloc(2);
    baseGen[0] = 'f';
    baseGen[1] = '\0';
    return baseGen;
  }

  return substitute(kochCurve(n-1),
                   'f', "f-f++f-f");
}
```

20

valgrind kochCurve

ERROR SUMMARY: 18 errors from 7 contexts (suppressed: 8 from 1)

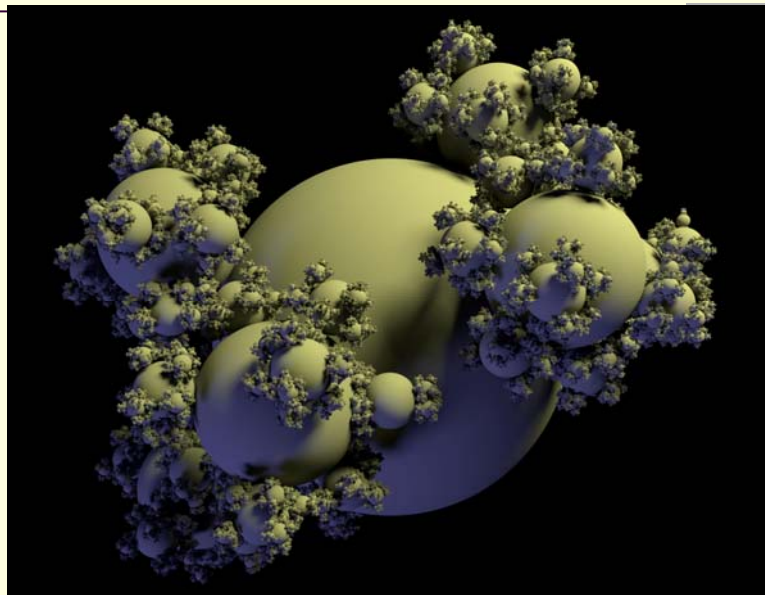
malloc/free: in use at exit: 252 bytes in 10 blocks.
malloc/free: 10 allocs, 0 frees, 252 bytes allocated.
For counts of detected errors, rerun with: -v
searching for pointers to 10 not-freed blocks.
checked 71,384 bytes.

LEAK SUMMARY:

definitely lost: 252 bytes in 10 blocks.
possibly lost: 0 bytes in 0 blocks.
still reachable: 0 bytes in 0 blocks.
suppressed: 0 bytes in 0 blocks.
Rerun with --leak-check=full to see details of leaked memory.

21

Koch Curve : Spheres Replacing Lines



22

Header and Library

- **mazegen.h** Header file defining functions that must be implemented.
- **mazetest.c** Test program that calls your implementation of the functions defined in `mazegen.h`. This file contains `main()`.
 - To get started, ***comment out*** all of the tests in this file except the first 2 or 3.
- **mazegen.c** This is a set of functions that ***you*** write.

23

Maze Requirements: Part 1 of 2

1. Your generator must create mazes of any specified width and height (within the computer's memory limitations) where each of these dimensions are odd integers greater than or equal to 3.
2. Each cell in a $(2n+1) \times (2m+1)$ maze is either open space or a wall.
3. The mazes must be random: A pair of mazes of the same, sufficiently large size, must have a low probability of being identical.
4. Each maze must have exactly two openings along its outer edge: one in the leftmost location of the top edge, the other in the rightmost location of the bottom edge.

24

Maze Requirements: Part 2 of 2

5. Each maze must have exactly one path that connects the two outer edge openings.
6. Every open cell in each maze must be reachable from either opening.
7. Every open cell must be adjacent to at least one wall cell.
8. Every wall cell must be adjacent to at least one open cell.
9. Each maze must contain dead ends which, on average, increase in number, get longer, have more turns, and have more branches as the maze size increases.

25

Possible Implementation of the Maze

1. Always start carving in an odd row and odd column (with the first row and column being 0).
2. Always look exactly two steps in each direction – but never anywhere else. If two steps in a direction is off the board or not a wall, then do not move in that direction.
3. Always move two space in the chosen direction. Then look again: call `maze_carve()` recursively.
4. `maze_carve(int x, int y)` returns (the recursion bottoms out) when there does not exist a legal move from (x,y) .
5. When the first call to `maze_carve(int x, int y)` returns, there are no legal moves in the maze. At this point, carve out the entrance and exit.

26

Maze Output Format

- Print the maze size before the maze
- Print a blank line after the maze.