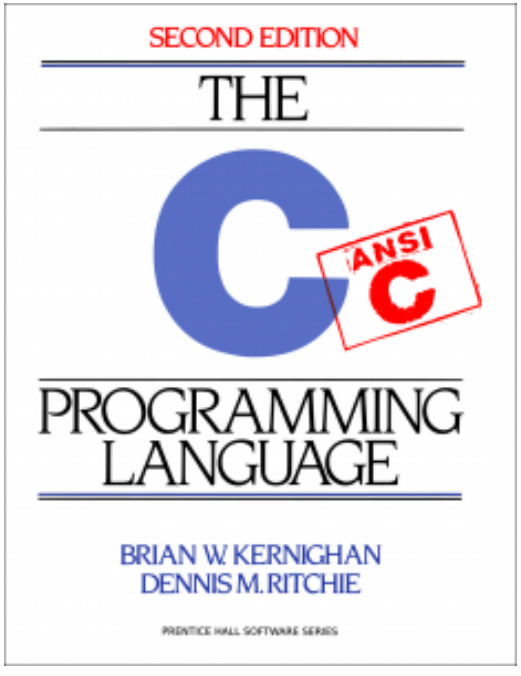
CS 241
Data Organization
Recursion and Quicksort

Brooke Chenoweth

University of New Mexico

Fall 2014

Read Kernighan & Richie



5 Pointers and Arrays

What is wrong with this program?

```
 1  #include <stdio.h>
 2
 3  void intToStr(int n)
 4  {
 5    if (n / 10)
 6    {
 7      intToStr(n);
 8    }
 9    putchar(n % 10 + '0');
10  }
11
12  void main(void)
13  {
14    intToStr(342);
15  }
```

This program causes a segmentation fault. Why?

intToStr

```
 1 #include <stdio.h>
 2
 3 void intToStr(int n)
 4 {
 5   if (n / 10)
 6   {
 7     intToStr(n / 10);
 8   }
 9   putchar(n % 10 + '0');
10 }
11
12 void main(void)
13 {
14   intToStr(342);
15 }
```

```
intToStr(342)
 intToStr(34)
  intToStr(3)
  put '3'
 put '4'
put '2'
```

Fibonacci Sequence

Fibonacci Sequence:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

Recursive definition:

$$F_n = F_{n-1} + F_{n-2}$$

Fibonacci Sequence by Loop

```
int fibonacci(int n)
{ int f0 = 1;
  int f1 = 1;
  int i;
  for (i=0; i<n; i++)
  { printf("%d, ", f1);
    int f2 = f0 + f1;
    f0 = f1;
    f1 = f2;
  }
  return f1;
}
int main()
{ printf("%d\n",fibonacci(20));
}
```

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

Fibonacci Sequence by Recursion

```
int fibonacci(int n)
{
  /* termination condition */
  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

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

64 characters including spaces.

Quicksort Algorithm

- Quicksort is a *divide and conquer* algorithm for sorting the elements of an array.
- Given an array, one element (the *pivot*) is chosen and the others are partitioned into two subsets:
  1. Those less than the pivot.
  2. Those greater than or equal to it.
- The same process is then 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.

Quicksort: main()

```
#include <stdio.h>

/* Used for display code, not part of algorithm. */
int arraySize;
int level;

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

  arraySize = sizeof(v)/sizeof(int);
  level = 0;

  quicksort(v, 0, arraySize-1);
}
```

Quicksort: Helper Function swap

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

Quicksort: Helper Function printArray

```
/* Fancy output, not part of sort */
void printArray(int levelCode, int v[],
                int left, int right)
{ int i=0;
  if (levelCode < 0)
  { printf("  Done%2d [", -levelCode);
  }
  else
  { printf("Level=%2d [",levelCode);
  }

  for(i=0; i<arraySize; i++)
  { if (i<left || i>right) printf("    ");
    else printf("%2d ", v[i]);
  }
  printf("]\n");
}
```

Quicksort

```
void quicksort(int v[], int left, int right)
{ int i, last;
  level++;
  printArray(level, v, left, right);

  /* nothing to sort if fewer than two elements */
  if (left < right)
  {
    /* Partition array - shown on next slide */

    quicksort(v, left, last-1);
    quicksort(v, last+1, right);
    printArray(-level, v, left, right);
  }
  level--;
}
```

Quicksort: partition

```
/* Using middle element for partitioning */
/* Move partition element out partition range */
swap(v, left, (left+right)/2);
last = left;

for (i=left+1; i <= right; i++)
{ if (v[i] < v[left])
  { last++;
    swap(v, last, i);
  }
}

/* restore partition element */
swap(v, left, last);
```

Quicksort Output Trace

```
Level= 1 [23 13 82 33 51 17 45 75 11 27 ]
Level= 2 [27 13 33 23 17 45 11          ]
Level= 3 [11 13 17                      ]
Level= 4 [11                            ]
Level= 4 [    17                        ]
  Done 3 [11 13 17                      ]
Level= 3 [          33 45 27            ]
Level= 4 [          27 33               ]
Level= 5 [                              ]
Level= 5 [             33               ]
  Done 4 [          27 33               ]
Level= 4 [                              ]
  Done 3 [          27 33 45            ]
  Done 2 [11 13 17 23 27 33 45          ]
Level= 2 [                      82 75   ]
Level= 3 [                      75      ]
Level= 3 [                              ]
  Done 2 [                      75 82   ]
  Done 1 [11 13 17 23 27 33 45 51 75 82 ]
```

Analysis

- Quicksort has average performance of $O(n \log n)$, but worst case is $O(n^2)$. Why?
- We were using the middle item as our pivot for partitioning. What happens if we made a different choice? First? Last?