Recursion

The Information Company

Operators know

- The first of the month is a Tuesday
- How to subtract one (e.g., 5-1=4)
- How calculate the next day of the week (e.g., given Sunday return Monday)

Q: what day of the week is X?

A: If X == "first" Then Reply "Tuesday"

Else Call the Information Company

Ask "What day of the week is X-1?"

Take response, reply with next day of week

Outline

Recursion

- Recursive call of function
- Base (non-recursive, degenerate) case
- Recursive case
- Tracing
  - program stack, activation record
- Recursive order effects
- Math Induction and Recursion
- Backtracking and multiple solution paths
- Examples
  - Factorial, Power, Maze search

Recursion

Solving a problem P by solving problem P', where P' is simpler than P AND identical in nature to P

versus Iteration - iteration generally involves loops and builds UP a solution

Recursion in a program - where a function calls itself as part of its solution (simple recursion) OR

where a function P calls a function or functions that will eventually call P

Binary Search: A Dictionary

IF the "dictionary" contains one page THEN

- scan the page for the word

ELSE

- open the dictionary to the middle
- determine which half of dictionary the word appears in
- IF word appears in first half THEN
  - search a dictionary consisting of first half of dictionary
- ELSE
  - search a dictionary consisting of second half of dictionary

Divide-and-Conquer Approach

Solve problem by "dividing" problem into one (or more) easier to solve pieces

- Base (degenerate) case - version of problem that is so easy to solve no further recursion needed
  - The first day of the month is Tuesday
  - Finding a word in a one-page dictionary
- Recursive case - other than the Base case where problem must be "divided"
  - Day of the month for any day but the first
  - Finding a word in a dictionary with more than one page

Recursive Binary Search

```
search(Dictionary,word)
  if (length of Dictionary == 1) then
      search page for word
  else
      open dictionary to middle
      determine which half contains word
      if (word in first half) then
            search(first half of dictionary,word)
      else
            search(second half of dictionary,word)
```

Recursively Defined Function

Factorial of n

fact(n) returns n!

- fact(0) - 0! = 1
- fact(1) - 1! = 1
- fact(2) - 2! = 2
- fact(3) - 3! = 6
- fact(4) - 4! = 24

fact(n) =

- n <= 1, return 1
- n > 1, return n * fact(n-1)

Factorial code:

```
int fact(int n) {
  if (n <= 1)
    return 1;
  else
    return n * fact(n-1);
}
```

How does this work?

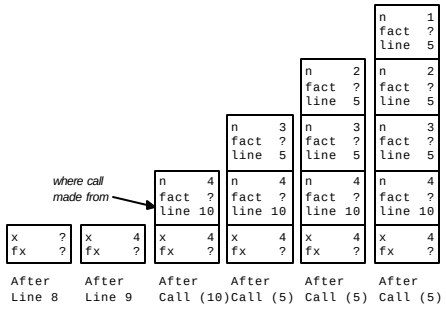
Tracing Recursive Functions (Box Method)

- Computer allocates space for variables on a "stack"
- When function called, computer puts "activation record" (AR) on stack with space for variables in function
- Activation record contains:
  - indication of where function called from (used at function end)
  - space for value parameters
  - space for reference parameters
  - space for local variables
  - space for the value returned by the function (if any)
- When function ends, its AR is removed from top of stack and computer returns to point function called from

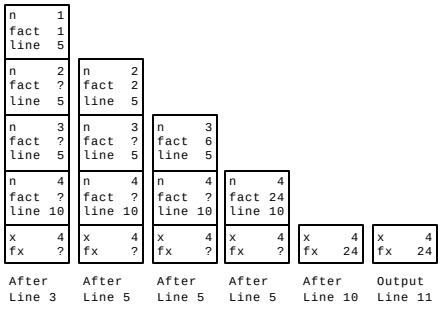
A Recursive Function

```
 1: int fact(int n) {
 2:  if (n <= 1)
 3:    return 1;
 4:  else
 5:    return n * fact(n-1);
 6: }
 7: void main() {
 8:  int x, fx;
 9:  x = 4;
10:  fx = fact(x);
11:  printf("%d! = %d\n",x,fx);
12: }
```

Tracing Factorial



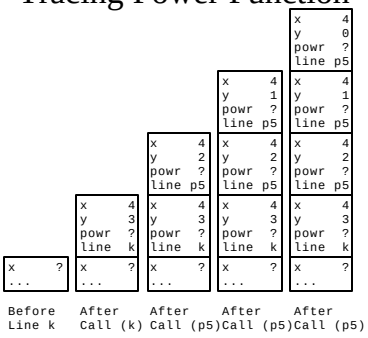
Tracing Factorial (cont)



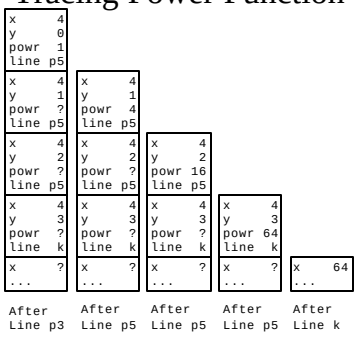
Recursive Power Function

```
                                y
Function powr(x,y) returns x    where x,y ints

p1: int powr(int x, int y) {
p2:  if (y == 0)
p3:    return 1;
p4:  else
p5:    return x * powr(x,y-1);
p6: }

call to power:
 k: x = powr(4,3);
```

Tracing Power Function



Tracing Power Function



Mathematical Induction and Recursion

- Two are closely related
- Math induction often used to prove the correctness of recursive algorithms

```
1: int fact(int n) {
2:  if (n <= 1)
3:    return 1;
4:  else
5:    return n * fact(n-1);
6: }
```

- Prove call to fact(n) returns n!

Proof of Fact Function

Base case: n is 1

since n is 1, line 3 executed, fact returns 1 (1!)

Inductive case: n > 1

Assume: fact(j-1) returns (j-1)! for j > 1

Prove: fact(j) returns j!

Call of fact(j) sets n to value of j, since j > 1, therefore n > 1, thus line 5 is executed, fact returns n * fact(n-1).

Since n is j, fact returns j * fact(j-1).

We know from inductive assumption that fact(j-1) returns (j-1)!

Therefore fact returns j * (j-1)! which equals j!

Calculation and Recursive Order

Some recursive calculations done on the "way in," some on the "way out"

```
int fct(int n, int a) {
  if (n <= 1)
    return a;
  else
    return
      fct(n - 1, n * a);
}
```

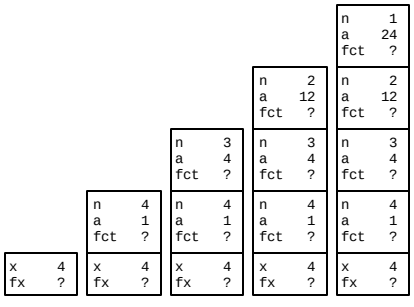
Call to fct:

```
x = 4;
fx = fct(x,1);
```

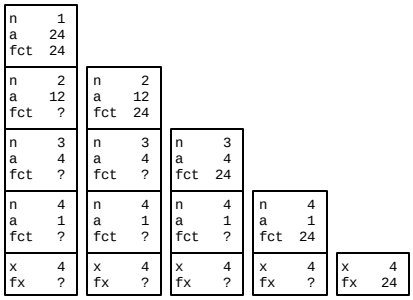
Or add second function

```
int fact(int n) {
  return fct(n,1);
}
```

Tracing Fct



Tracing Fct (cont)



Recursive Order: Writing Array

A | 2 | 3 | 5 | 7 | 8 |

```
void do_print_arrayF(int A[], int J, int N) {
  if (J < N) {
    printf(" %d",A[J]);
    do_print_arrayF(A,J+1,N);
  }
}
void print_arrayF(int A[], int N) {
  do_print_arrayF(A,0,N);
  printf("\n");
}
Call: print_arrayF(A,5);
Output: 2 3 5 7 8
```

Recursive Order: Writing Array

A | 2 | 3 | 5 | 7 | 8 |

```
void do_print_arrayR(int A[], int J, int N) {
  if (J < N) {
    do_print_arrayR(A,J+1,N);
    printf(" %d",A[J]);
  }
}
void print_arrayR(int A[], int N) {
  do_print_arrayR(A,0,N);
  printf("\n");
}
Call: print_arrayR(A,5);
Output: 8 7 5 3 2
```

Recursive Order: Writing Array

A | 2 | 3 | 5 | 7 | 8 |

```
void do_print_arrayB(int A[], int J, int N) {
  if (J < N) {
    printf(" %d",A[J]);
    do_print_arrayB(A,J+1,N);
    printf(" %d",A[J]);
  }
}
void print_arrayB(int A[], int N) {
  do_print_arrayB(A,0,N);
  printf("\n");
}
Call: print_arrayB(A,5);
Output: 2 3 5 7 8 8 7 5 3 2
```

Recursion: Backtracking

- A recursive method may have more than one alternative method for solving a problem
- One approach: try one method and see how far you get; if that fails, backtrack to the point where you made the choice and try a different method
- Built into recursive solution

Navigating a Maze

Maze in 2D Character Array

	0	1	2	3	4	5
0					X	
1		X	X	X		
2			X			X
3	X				X	
4	X	X		X	X	
5	X	X				

- Allowed to move from an open space to an adjacent open space (N,E,S,W)
- Question: is there a path from <0,0> to <5,5>?

A Search Algorithm

```
Visit(position)
   mark Position as Visited
   if (Position == <5,5>) then Found It!
   else
        if Position to East Open and Unvisited then
                Visit(Position to East)
        if Position to South Open and Unvisited then
                Visit(Position to South)
        if Position to North Open and Unvisited then
                Visit(Position to North)
        if Position to West Open and Unvisited then
                Visit(Position to West)
```

Tracing Visit

```
                                        Visit<0,3>
                              Visit<0,2>  Visit<0,2>
                    Visit<0,1>  Visit<0,1>  Visit<0,1>
Visit<0,0>  Visit<0,0>  Visit<0,0>  Visit<0,0>
----------  ----------  ----------  ----------

Visit<0,2>
Visit<0,1>  Visit<0,1>              Visit<1,0>
Visit<0,0>  Visit<0,0>  Visit<0,0>  Visit<0,0>
----------  ----------  ----------  ----------
Backtrack!  Backtrack!  Backtrack!  New direction!
```

Tracing Visit (cont)

```
                                    Visit<2,1>
                        Visit<2,0>  Visit<2,0>
            Visit<1,0>  Visit<1,0>  Visit<1,0>
Visit<0,0>  Visit<0,0>  Visit<0,0>  Visit<0,0>
----------  ----------  ----------  ----------

                        Visit<3,3>
            Visit<3,2>  Visit<3,2>
Visit<3,1>  Visit<3,1>  Visit<3,1>
Visit<2,1>  Visit<2,1>  Visit<2,1>
Visit<2,0>  Visit<2,0>  Visit<2,0>
Visit<1,0>  Visit<1,0>  Visit<1,0>
Visit<0,0>  Visit<0,0>  Visit<0,0>  and so on
----------  ----------  ----------  ----------
```

Issues in Visit

Order points visited: <0,0> <0,1> <0,2> <0,3> <1,0> <2,0> <2,1> <3,1> <3,2> <3,3> <2,3> <2,4> <1,4> <1,5> <0,5> <4,2> <5,2> <5,3> <5,4> <5,5> …

Representation issues:

- maze as an array of characters
- space characters for open positions
- X character for block

How to represent Visited?

- drop period (.) (bread-crumb) when position Visited

Code for Visit

```
int LegalPos(int r, int c) {
  if ((r < 0)||(c < 0)||(r > 5)||(c > 5)) return 0;
  return 1;
}
void Visit(char Maze[][6], int r, int c) {
  Maze[r,c] = '.';
  printf("Visiting <%d,%d>\n",r,c);
  if ((r == 5) && (c == 5)) printf("Found It!\n");
  else {
    if (LegalPos(r,c+1) && (Maze[r,c+1] == ' '))
      Visit(Maze,r,c+1);
    if (LegalPos(r+1,c) && (Maze[r+1,c] == ' '))
      Visit(Maze,r+1,c);
    if (LegalPos(r-1,c) && (Maze[r-1,c] == ' '))
      Visit(Maze,r-1,c);
    if (LegalPos(r,c-1) && (Maze[r,c-1] == ' '))
      Visit(Maze,r,c-1);
  }
}
```