Announcements 2/28

New 1-page handout today

HW5 due next Sunday night/Monday in class

Reminder: Final exam: **Wed 3/14,** Kane 120

CSE 142A lecture 10:30 am

CSE 142B lecture 12:30 pm

Old: if you have a conflict with your scheduled time, you may attend the other exam. If you have a conflict with both times, you should *already* **have contected Melissa Albin (course administrator, malbin@cs). If you haven't,** *do it now.*

New: You must bring your UW ID and a picture ID to the exam.

Review sessions: Sunday 3/11 @3:00; Wed 3/13 @4:40; locations TBA

CSE 142 Computer Programming I

Recursion: *Programming for the Lazy*

Overview

Review
- **Function calls in C**

Concepts
- **Recursive definitions and functions**
- **Base and recursive cases**

Reading
- **Read textbook sec. 10.1-10.3 & 10.7**
- **Optional: sec. 10.6 (Towers of Hanoi, a classic example)**
- **Skip sec. 10.4-10.5**

Recursion

Basic Idea:
- **You've got a problem "of size N"**
- **DON'T try to write a program that can solve problems of size N**
- **INSTEAD**
  - **Write a program that can solve problems of size 1 (easy!)**
  - **Write a program that can convert a problem of size N into a problem of size N-1**
  - **VOILA! Done!**
- **The hard part: Understanding "size", and finding a way to get from N to N-1**

The Example We Hate But Can't Avoid: Factorial

Definition: ***n! = n * (n-1) * (n-2) * … * 2 * 1***

Example: 6! = 6*5*4*3*2*1 = 720

(Can you write this as a loop?)

To think of a recursive solution, we need three things:

1. **What does "problem size" mean?** ***n***
2. **A base case: n=1 ⇒result is 1**
3. **A way convert a problem of size n into one of size less than n: n! = n * (n-1)!**

Factorial: Recursive Solution in C

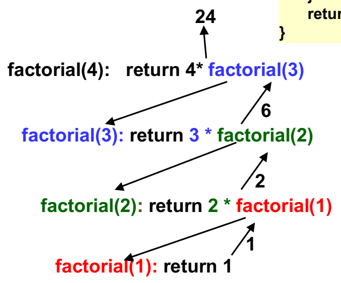
```
int factorial (int n){

    if (n <= 1) {
        return 1;
    }

    return n * factorial(n - 1);
}
```

Trace

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



```
                  24
factorial(4):   return 4* factorial(3)
                                  6
  factorial(3): return 3 * factorial(2)
                                  2
     factorial(2): return 2 * factorial(1)
                                  1
        factorial(1): return 1
```

W-7

Recursive & Base Cases

A recursive definition has two parts
- One or more recursive cases where the function calls itself
- One or more *base cases* that return a result without a recursive call

There *must* be at least one base case
Every recursive case *must* make progress towards a base case

Forgetting one of these rules is a frequent cause of errors with recursion

W-8

Recursive & Base Cases

Base case

Recursive case

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

W-9

Example 2: Array Sum

Problem: Write a function that returns the sum of the first N elements of an integer:
A[0] + A[1] + A[2] + A[3] + … + A[N-1]

```
int ArraySum(int A[ ], int N) {
    // base case: N ==1

    // recursive case:
    // ArraySum(N) = (A[0]+A[1]+…+A[N-2])+A[N-1]
    //             = ArraySum(N-1) + A[N-1]
}
```

W-10

Example 2: Array Sum

```
int ArraySum(int A[ ], int N) {
    // base case: N ==1
    if (N==1) {
        return A[0];
    }
    // recursive case:
    return ArraySum(N-1) + A[N-1];
}
```

W-11

Iteration *vs.* Recursion

Technically, *any* iterative algorithm can be reworked to use recursion instead (and vice versa).

There are programming languages where recursion is the only choice(!)

Some algorithms are more naturally written with recursion

The following examples are "easy" recursively, not easy iteratively…

W-12

Example 3: Print Keyboard Input in Reverse

User Types:	My name is Harry.
Program Types:	.yrraH si eman yM
User Types:	A man, a plan, a canal, Panama.
Program Types:	.amanaP,lanac a ,nalp a ,nam A

W-13

Example 3: Print Keyboard Input in Reverse

Doing this recursively:

1. Base Case: There are no more characters -> done (return)
2. Recursive Case:
   Reverse(<next single char> <all following chars>)
   = Reverse(<all following chars>) <next char>

```
Reverse("abcd") = Reverse("bcd") "a"
                = Reverse ("cd") "b" "a"
                = Reverse ("d") "c" "b" "a"
                = Reverse("") "d" "c" "b" "a"
                = "d" "c" "b" "a"
```

W-14

Example 3: In C

```
void ReverseInput()
{
    char        nextChar;

    scanf("%c", &nextChar);
    if (nextChar=='\n') {
        return;
    }

    ReverseInput();
    printf ("%c", nextChar);
}
```

W-15

Example 4: N Balls, J Jars

Want to print out all possible ways to put N balls in J jars.

Example: N = 3, J = 3

```
3 0 0
2 1 0
2 0 1
1 2 0
1 1 1
1 0 2
0 3 0
0 2 1
0 1 2
0 0 3
```

W-16

Example 4: N Balls, J Jars

Base Case: J=1 (Doesn't matter what N is)
There's only 1 way to put N balls in 1 jar

Recursive Case:

N in jar 0,	0 in jars 1…J-1
N-1 in jar 0,	1 in jars 1…J -1
N-2 in jar 0,	2 in jars 1…J -1
…	
1 in jar 0,	N-1 in jars 1…J –1
0 in jar 0,	N in jars 1…J-1

W-17

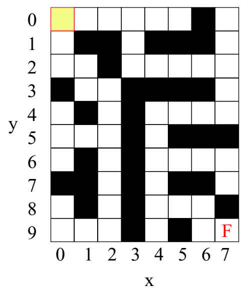
Example 4: N Balls, J Jars

```
void PrintAllocations(int N, int J) {
    if (J==1) {
        printf (" %d", N);
        return;
    }

    for (n=0; n<=N; n++) {
        printf (" %d", n);
        PrintAllocations(N-n, J-1);
    }
}
```

W-18

Example: Path planning



```
/* 'F' means finished!
   'X' means blocked
   '  ' means ok to move */
char maze[MAXX][MAXY];
int x =0, y=0;    /* start in yellow */
```

Unless blocked, can move up, down, left, right

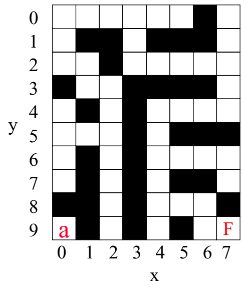
Objective: determine if there is a path?

Simple Cases

Suppose at (x,y)

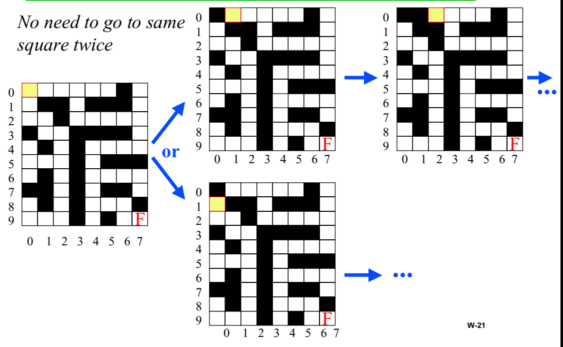
If maze[x][y]=='F'
Then "yes!"

If no place to go
Then "no!"



Redefining a hard problem as several simpler ones

No need to go to same square twice



Helper function

```
/* Return true if <x,y> is a legal move
   given the maze, otherwise returns false */
int legal_mv (char m[MAXX ][MAXY],
                                int x, int y) {
  return(x>=0 && x<MAXX &&
             y>=0 && y<MAXY &&
              m [x][y] != 'X');
}
```

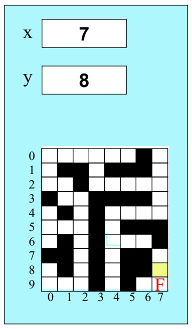
Elegant Solution

```
/* Return true if there is a path from <x,y> to an element of maze
   containing 'F' otherwise returns false */
int is_path(char m[MAXX][MAXY ], int x, int y) {
   if (m [x][y] == 'F')
      return(TRUE);
   else {
      m[x][y] = 'X';
      return((legal_mv(m,x+1,y) && is_path(m,x+1,y)) ||
                 (legal_mv(m,x-1,y) && is_path(m,x-1,y)) ||
                 (legal_mv(m,x,y-1) && is_path(m,x,y-1)) ||
                 (legal_mv(m,x,y+1) && is_path(m,x,y+1)))
   }
}
```

Example

is_path(maze, 7, 8)

```
int is_path(char m[MAXX][MAXY ], int x, int y) {
  if (m [x][y] == 'F')
    return(TRUE);
  else {
    m[x][y] = 'X';
    return((legal_mv(m,x+1,y) && is_path(m,x+1,y)) ||
              (legal_mv(m,x-1,y) && is_path(m,x-1,y)) ||
              (legal_mv(m,x,y-1) && is_path(m,x,y-1)) ||
              (legal_mv(m,x,y+1) && is_path(m,x,y+1)))
```



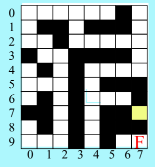
Example Cont

is_path(maze, 7, 7)

```
int is_path(char m[MAXX][MAXY ], int x, int y) {
  if (m [x][y] == ‘F’)
    return(TRUE);
  else {
    m[x][y] = ‘X’;
    return((legal_mv(m,x+1,y) && is_path(m,x+1,y)) ||
              (legal_mv(m,x-1,y) && is_path(m,x-1,y)) ||
              (legal_mv(m,x,y-1) && is_path(m,x,y-1)) ||
              (legal_mv(m,x,y+1) && is_path(m,x,y+1)))
```

x 7

y 7



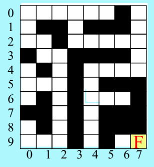
Example Cont

is_path(maze, 7, 9)

```
int is_path(char m[MAXX][MAXY ], int x, int y) {
  if (m [x][y] == ‘F’)
    return(TRUE);
  else {
    m[x][y] = ‘X’;
    return((legal_mv(m,x+1,y) && is_path(m,x+1,y)) ||
              (legal_mv(m,x-1,y) && is_path(m,x-1,y)) ||
              (legal_mv(m,x,y-1) && is_path(m,x,y-1)) ||
              (legal_mv(m,x,y+1) && is_path(m,x,y+1)))
```

x 7

y 9



Recursion Wrap-up

Recursion is:

(a) Incredibly useful
(b) Under-used in practice
(c) All of the above