CS 111

recursion

Model for recursive function

```
TITLE LINE{
    if (DETECT BASE VALUE OF x) return BASE ANSWER;
    return CALCULATED ANSWER FOR LARGER x;
     // applies easier version of function
}
```

Example 1

```
int f(int x){
    if(x <= 0) return 0;
    return x + f(x – 1);
}
```

1. What value of x is used for the base case?
2. What is the answer for the base case?
3. What easier version of the function is called to calculate f(x)?
4. How do we change the answer from f(x – 1) to f(x)?
5. What would f(4) return?

Recursion as a loop

- Any simple recursion can be recoded as a loop
- Example of loop working from base case through x:

```
int f(int x){
    int sum = 0;
    for(int i = 0; i < x; i++){
        sum = sum + i;
    }
}
```

```
int f(int x){
    if(x <= 0) return 0;
    return x + f(x – 1);
}
```

Recursion as a loop

- Example of a loop working from x to the base case:

```
int f(int x){
    int sum = x;
    while(x > 0){
        x--;
        sum = sum + x;
}
```

```
int f(int x){
    if(x <= 0) return 0;
    return x + f(x – 1);
}
```

Recursion rewritten as a loop

- Keep the same title line and replace the whole code block with a new one that calls a loop instead
- There will now be no base case and no recursive step
- Whenever you suspect a problem needs recursion it will be possible to write a loop instead but the code might be much longer

Example 2

- The function first2 returns the first two digits of a positive integer x
- So first2(3456) returns 34 and first2(7) returns 7

```
The function first2(int x) {
    if(TEST FOR BASE) return EASY ANSWER;
    return first2(x / 10);
}
```

Example 2

```
The function first2(int x) {
    if(TEST FOR BASE) return EASY ANSWER;
    return first2(x / 10);
}
```

1. What easier version of x is used here in the recursive step?
2. What are base values for x? What answer would they give?
3. To find first2(3456) what easier versions of x are called on by the recursion?

Notes

- For recursive functions, you don't need to figure out the whole solution
- Focus on figuring out how you could solve the current problem if you KNEW how to solve a slightly smaller version of the problem
  - In Example 1, we solved the problem for x by combining it with the solution for x – 1

```
int f(int x){
    if(x <= 0) return 0;
    return x + f(x – 1);
}
```

Questions to ask

1. What smaller value of x is useful because it takes care of most of our work?
2. How do we adjust the result of the smaller task to get our answer?
3. Translate the answers of these questions into C++ code

Example 3

- The plan below is to examine all digits except the last and also the last digit on its own
- If either gives a yes, we know there's a 3 in there
- Look at the outline below and ask the questions from the previous slide

```
// return true if any digit of x is a 3
bool has3(int x){
    if(x <= 0) return false;
    return RECURSIVE CALL || LOOK AT LAST DIGIT;
}
```

Example 4

- This computes the sum of cubes of numbers from 1 to x. Since it is a sum we expect to adjust the recursive call by adding something

```
int sumCubes(int x){
    if(x <= 0) return 0;
    return RECURSIVE CALL + WHAT;
}
```