Recursion: function calls itself

- if we've provided a prototype, a function can call itself
- such functions are called recursive
- recursion can be used for simple repetition, or for gradual repeated calls with simpler and simpler data to process
- relies on the use of an if/else statement to check whether or not a recursive call is necessary, to guarantee that eventually the function stops calling itself
- logic errors can lead to runaway/infinite recursion: runs until the program crashes or the user hits control-C to kill it

Base/stop vs general case

- the base case (or cases) dictate circumstances under which a function should NOT call itself again
- the general case (or cases) refers to all the cases where the function does need to call itself again
- recursive functions need to check if they've encountered a base case *before* they make a recursive call
- usually have one set of actions to do in the base case, and a different set of actions (including the recursive call) in the general case

Example: print vals 1..N

```
// program to print N
// and count down to 1

#include <iostream>
using namespace std;

void printVals(int N);

int main()
{
   int start = 10;
   printVals(10);
}
```

```
void printVals(int N)
{
   // base case, just print N and end
   if (N <= 1) {
      cout << N << endl;
   }

   // general case, print N then call recursively
   //        on N-1 to print the smaller values of N
   else {
      cout << N << endl;
      printVals(N-1);
   }
}
```

Calls complete before returning...

- why did that count down from 10 to 1?
- main calls printVals(10)
  - printVals(10) prints 10 then calls printVals(9)
    - printVals(9) prints 9 then calls printVals(8)
      - ...
        - printVals(2) prints 2 then calls printVals(1)
          - printVals(1) prints 1 then ends, returning to the printVals(2)
        - printVals(2) ends, returning to printVals(3)
      - ...
    - printVals(9) ends, returning to printVals(10)
  - printVals(10) ends, returning to main

Example: repeat until input valid

```
// program to get positive int
// from user, keep trying until
// valid input is obtained

#include <iostream>
using namespace std;

int getPosInt();

int main()
{
   int userVal;
   userVal = getPosInt();
   cout << “You chose “ << userVal;
}
```

```
int getPosInt()
{
   cout << “Enter a positive int” << endl;
   int val;
   cin >> val;
   if (cin.fail()) {
      cout << “That was not an int, ”;
      cout << “please try again” << endl;
      cin.clear();
      cin.ignore(80, '\n');
      val = getPosInt();
   } else if (val < 1) {
      cout << “That was not positive, ”;
      cout << “please try again” << endl;
      val = getPosInt();
   } else {
      cout << “Valid int obtained” << endl;
   }
   return val;
}
```

Efficiency issues with recursion

- each call to a function sets up space in memory (the system stack, or call stack) for that function call's local variables, parameters etc
- in previous example, while printVals(1) is running the program has memory space set aside to remember variables and params for main, printVals(10) call, printVals(9) call, ... , printVals(1) call
- if main called printVals(100000) then by the time printVals(1) was called the program would be storing 100,000 sets of local vars/parameters
- a different form of repetition (loops) is thus often preferable