CS 101: Computer Programming and Utilization

13-Recursion

Instructor: Sridhar Iyer
IIT Bombay
void g(double *x, double *y) {
    if (*x > *y) {
        double z = *x;
        *x = *y; *y = z;
    }
}

int main() {
    double x, y;
    cin >> x >> y;
    g(&y, &x);
    cout << x << ' ' << y << endl;
}
void g(double &x, double &y) {
    if(x > y) {
        double z = x;
        x = y; y = z;
    }
}

int main() {
    double x, y, z;
    cin >> x >> y >> z;
    g(x, y); g(y, z); g(x, y)
    cout << x << ' ' << y << ' ' << z << endl;
    What is the output when [x, y, z] input is [23, 32, 12]
Recall: Passing parameters to functions

- Default is “pass by value”
  - Value of variable is copied from caller to callee
  - Callee has local copy of variable, even if the variable has the same name as the caller
  - Modification in the callee has no effect on the caller

- For modification by callee to have effect in caller, use “pass by reference”
  - Address of variable is passed, either explicitly through the use of a pointer variable, or implicitly through the use of “&” in the callee declaration

Example: void fun(int &x);
Be careful with pointers!

• Swap function without temporary variables
  
  ```c
  void swap(int& a, int& b) {
    a = a - b; b = a + b; a = b - a;
  }
  ```

• Consider the call
  
  ```
  int x = 3, y = 5; swap(x, y);
  ```

• Compare with the call
  
  ```
  int x = 3; swap(x, x);
  ```

• Can’t happen in pass by value
  
  ```
  a ≡ x  b ≡ x
  ```
  
  ```
  3   3
  ```
  
  ```
  0   0
  ```
  
  ```
  0   0
  ```
  
  ```
  0   0
  ```
Recursion – Function calling itself

• Function is called from its own body.

• Is it ok to do so?
  – OK if we eventually get to a call which does not call itself. => similar to base case of induction.
  – Then that call will return.
  – Previous call will return … and so on.

• What exactly happens during execution?
  – Stack of Activation Records builds up like any other series of function calls. → Should eventually stop!

• Is it useful?
Recursion examples

```c
int fac(int n) {
    if (n == 0) return 1;
    else return n * fac(n-1);
}

int fib(int n) {
    if (n < 2) return 1;
    else return fib(n-1) + fib(n-2);
}

int gcd(int m, int n) {
    if (n == 0) return m;
    else return gcd(n, m % n);
}
```
fac(n)

- fac(5) calls fac(4) calls fac(3) calls fac(2) calls fac(1) calls fac(0)
- Returns 1 returns 1 returns 2 returns 6 returns 24 returns 120
- Easy way to visualize recursive calls
  - Pass additional recursion level parameter
  - Indent all messages by level blank spaces
- Up to n activation records on the stack

- Draw the recursion stack for:
  - demo13-recursion-gcd.cpp
  - demo13-recursion-fibonacci.cpp
Recursion - usefulness

• Applicable whenever you can divide a problem into sub-problems of the same type as the original, solve those sub-problems, and combine the results

• Examples
  – Towers of Hanoi
  – Binary Search
  – Sorting Algorithms
  – Traversing in a file system
Recursion – predict the output

void printnum (int a) {
    cout << a;
    if ( a < 9 ) printnum (a + 1);
    cout << a;
}

int main () {
    int a; cin >> a;
    printnum (a);
}

What is the output when a is 2?