**Reference parameters**

* C/Java only support **call-by-value**
  + Value of parameter is copied onto the **call stack frame/activation record**
  + Only return value (if any) is copied back
  + Parameter values can be changed inside the function, but changes do not propagate back to the calling environment
  + Avoid using ptr parameters
* C++/C# support **reference parameters**
  + Parameter is not copied onto the call stack
  + Any changes made in the function will propagate back to the caller
  + The reference param is an **alias** for the variable in the calling environment – **call-by-reference**
  + Param doesn’t need to be derefenced by \*
  + E.g.

void swap(int& x, int& y) {

const int temp = x;

x = y;

y = temp;

}

* + A **const&** reference param is not allowed to be changed inside the procedure
    - Preferred over value params in C++ – just a reference to the object instead of copying it
  + Many cases:
    - Void function1 (Object x) {}
      * A copy of the caller’s instance of x is created on the call stack
      * Changes do not propagate back to caller; instance is deleted at the end of call
    - Void function2 (const Object x) {}
      * A copy of x is created on call stack, and may not be changed (same as 1)
    - Void function3 (Object & x) {}
      * Any mention of x inside the method refers to the variable in the caller’s environment
      * Changes propagate back immediately
    - Void function4 (const Object & x) {}
      * x refers to variable in caller’s environment, which can’t be changed
      * Similar to 2
  + Normal identifiers can be references too
    - E.g. int & x = list[i].get(j);
* ADTs can now be implemented using ref params:
  + bool isEmpty (const Stack& s) {…}
  + void push (Stack& s, string e) {…}
  + void pop (Stack& s) {…}
  + string peek (const Stack& s) {…}