9

Functions

Exercises

9.1 This program is in the file `ex91.cpp` in the *ch09code* directory. The answers are:

(a) When passed by value, `x` remains {5, 2, 4}.

(b) When passed by reference, `x` becomes {2, 5, 4}.

(c) When passed by value-result, `x` becomes {2, 5, 4}.

9.2 The interpreter is called `Cliteextended.jar` and is in the software distribution along with the program `functions.cpp`. A listing of its output, including the values of the globals `h` and `i` in this program when each of the functions main, A, and B begin executing, is in the file `functions.output`?

9.3 This question can be answered by running the `Cliteextended.jar` interpreter with the program `recFib.cpp` or viewing the file `recFib.output` in the software distribution.

9.4 This C++ program is in the file `ex94.cpp` in the *ch09code* directory. It is also approximated by the Clite extended program called `hanoi.cpp` in the *programs* subdirectory of the software distribution. Running the Clite interpreter with this program shows the topmost stack activation record for each call to the function `moveTower`. Running either program will answer this question.

9.5 Here is the modified program:

```
int h, i;
void A(int x, int y) {
   bool j;
   i = B(h);
}
int B(int w) {
   int locali, j, k;
   locali = 2*w;
   w = w + 1;
   return locali;
}
int main () {
   int a, b;
   h = 5; a = 3; b = 2;
   A(a, b);
}
```

(a) See the file `ex95.output` in the *ch09code* directory.

(b) See the file `ex95.output` in the *ch09code* directory.

(c) If we had not removed the declaration of the local variable `i` from `A`, a (mixed mode assignment) type error would have occurred, since the assignment `i = B(h)` would be attempting to assign an integer value (B(h)) to the bool variable i. (Try it!)

9.6 See the output of running the `Cliteextended.jar` interpreter with the program `gcd.cpp` as input. The output in the file `gcd.output` provides answers to this question.

9.7 Consider the Ada program in Figure 9.9.

(a) Show the contents of the new stack frame when the call `Quicksort(a)` is initiated, for the array `a` = {4, 2, 5, 1}. A copy of the value of `a` is placed in the stack frame, since List is passed by value-result (in out).

(b) Show the contents of each new stack frame added when each of the next four calls is initiated (i.e., to `Sort`, to `Swap`, and twice to `Sort` again).

```
The first call to Sort is:
Sort({4,2,5,1}, 0, 3)
    Prior to the first swap, Sort computes I = 0, J = 4,
        Key = List[0] = 4,  I = 2, and J = 3.
    So the first call, Swap(List, I, J), passes arguments:
        Swap({4,2,5,1}, 2, 3)
        which returns with List = {4,2,1,5}.
    So the next call passes arguments:
    Sort({4,2,1,5}, 0, 2)
     which leaves List = {1,2,4,5}
    and the last call:
    Sort({1,2,4,5}, 4, 3)
         leaves List = {1,2,4,5}
```

9.8 The functions must be split apart, and the parameter passing mechanism for the parameter `List` is call by reference. List must be explicitly passed between Quicksort, Sort, and Swap. Also, any call to Quicksort must explicitly pass the length of the argument, since that cannot otherwise be determined by Quicksort. An adjustment is also needed for C's zero-based indexing of arrays. Here is a rewrite in C:

```
void Swap(int[] List, int I, int J) {
        int Temp;
        Temp = List[I];
        List[I] = List[J];
        List[J] = List[I];
}
void Sort (int[] List, int M, int N) {
        int I, J, Key;
        if (M < N) {
            I = M; J = N + 1;
            Key = List[M];
            do
                I = I + 1;
            while (List[I] < Key);
            do
                J = J - 1;
            while (List[J] > Key);
            Swap(List, I, J);
            Sort(List, M, J-1);
            Sort(List, J+1, N);
        }
}
void Quicksort(int[] List, int Length) {
    Sort(List, 0, Length-1);
}
```