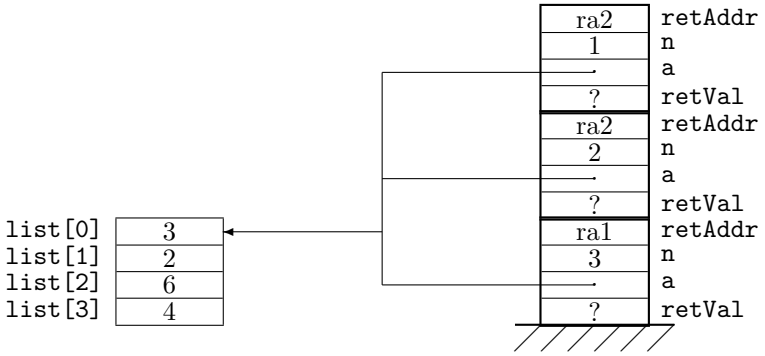
Homework 1 Solutions

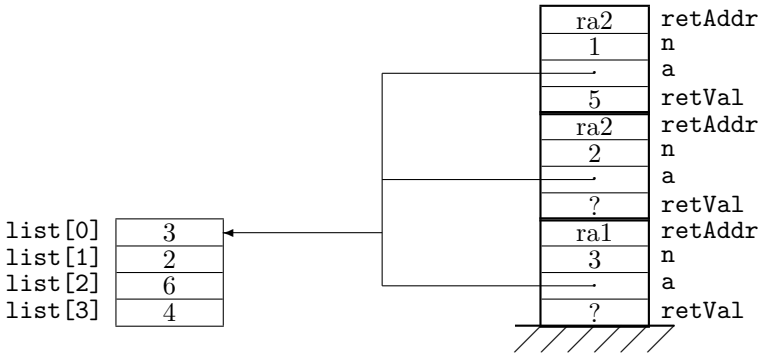
- **Exercise 2.1.** *The function* `sum` *in Program 2.12 is called for the first time by the main program. From the second time on it is called by itself.*

  a) *How many times is it called altogether?*
  It is called once each with argument `n` being 3, 2, 1, and 0, for a total of four times.

  b) *Draw a picture of the main program variables and the run-time stack just after the function is called for the third time. You should have three stack frames.*



  c) *Draw a picture of the main program variables and the run-time stack just before the return from the call of part (b). You should have three stack frames, but with different contents from part (b).*
  The only change is the return value in the top-most stack frame:



- **Exercise 2.5.** *Draw the call tree as in Figure 2.15 for procedure* `permute` *of Program 2.15 for the following call statements from the main program:*

  a) `permute("wxyz ", 0, 3)`

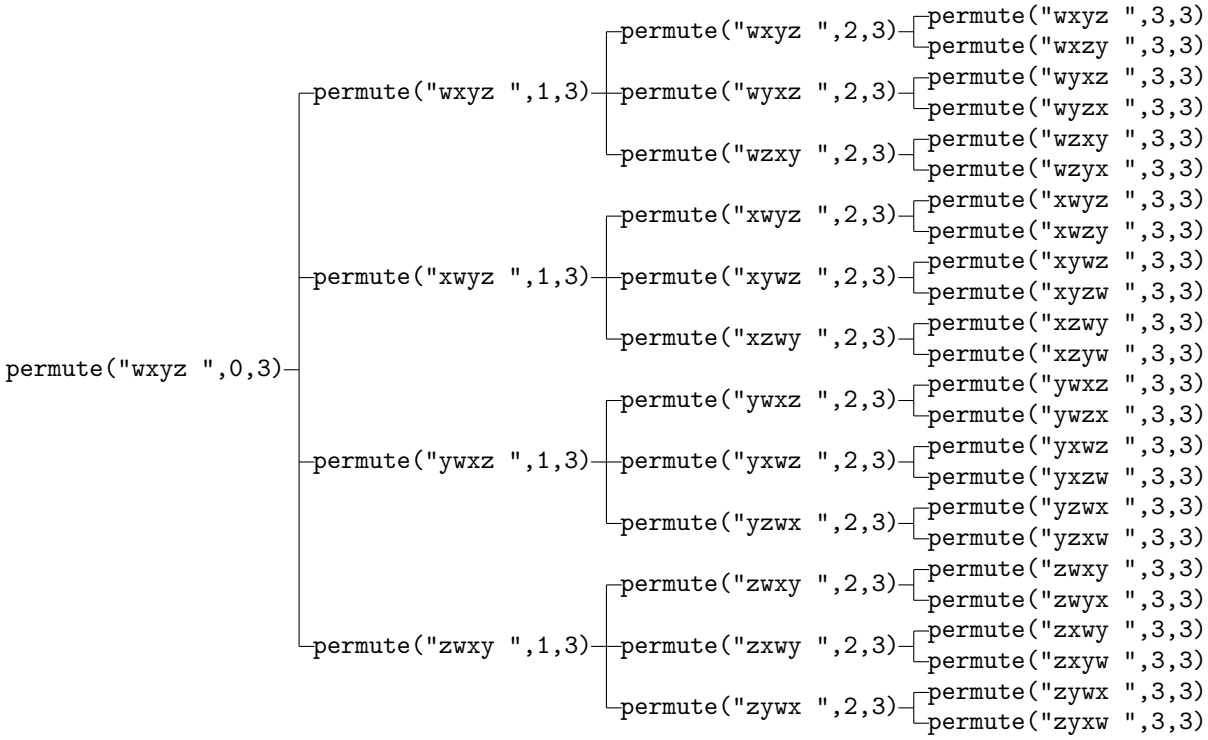
  b) `permute("wxyz ", 1, 3)`

  c) `permute("wxyz ", 1, 2)`

  d) `permute("wxyz ", 2, 2)`

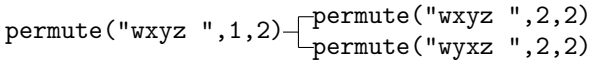
  How many times is the procedure called? What is the maximum number of stack frames on the run-time stack during the execution? In what order does the program make the calls and returns?

  a) The call tree below shows that the procedure is called a total of 41 times. There are a maximum of four stack frames (for calls to `permute`) on the run-time stack at any one time, as can be seen from the fact that the call tree is only four levels deep; that is, the stack only has to remember the sequence of parent nodes from the root to any particular call, since it finishes visiting one branch of the tree before moving on to the next (this is called a "depth-first tree traversal"). The order in which the calls are made can be read off of the call tree by tracing around the outer edge of the tree clockwise (similar to Figure 2.13).



b) The call tree in this case is the first child subtree of the root in part (a), containing 10 nodes. The maximum depth is three, so that is also the maximum number of stack frames.

c) The call tree in this case has only three nodes:

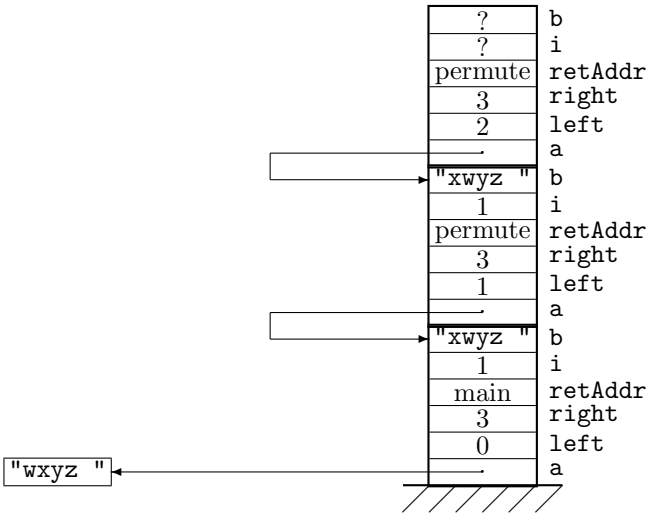


The depth is two, so there will be at most two stack frames on the run-time stack.

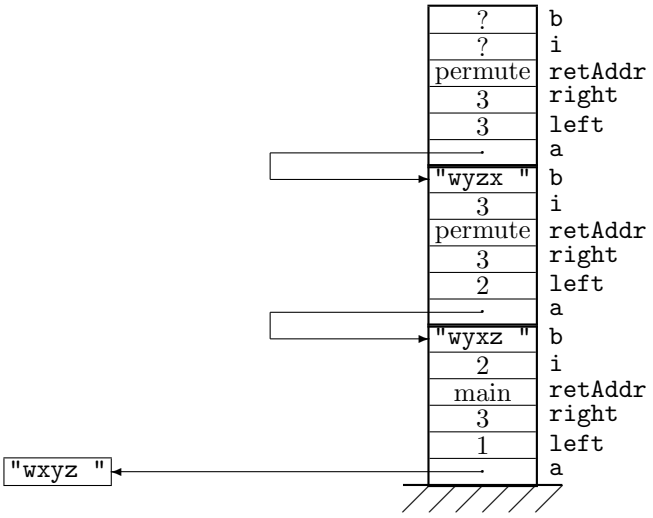
d) The call tree in this case is the first child subtree of the root in part (c), containing only one node. There is only one stack frame.

- **Exercise 2.6.** *For Exercise 5, draw the run-time stack just after the following function calls:*

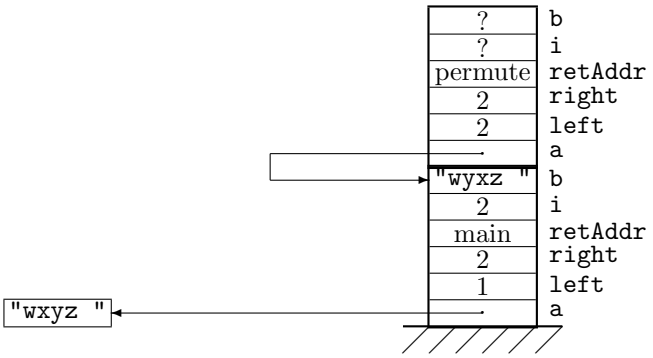
a) `permute("xwyz ", 2, 3)`



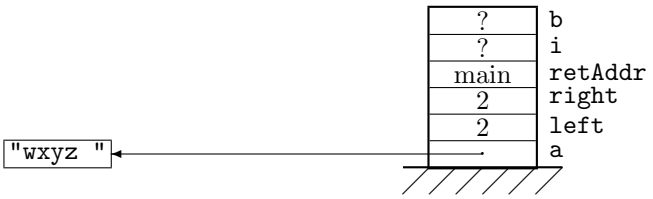
b) `permute("wyzx ", 3, 3)`



c) `permute("wyxz ", 2, 2)`



d) `permute("wxyz ", 2, 2)`



- **Exercise 2.12.** *Write a recursive procedure called* `rotateLeft`*, which should rotate the first n numbers in an array of numbers to the left. To rotate n items left, rotate the first* $n-1$ *items left recursively, and then exchange items* $n-1$ *and n. For example, to rotate the five items*

$$5.0 \quad -2.3 \quad 7.0 \quad 8.0 \quad 0.1$$

to the left, recursively rotate the first four items to the left:

$$-2.3 \quad 7.0 \quad 8.0 \quad 5.0 \quad 0.1$$

and then exchange items four and five.

$$-2.3 \quad 7.0 \quad 8.0 \quad 0.1 \quad 5.0$$

Do not use a loop.

Here is one solution:

```
void rotateLeft(double a[], int n)
{
    // Do nothing if n <= 1 -- base case
    if (n > 1)
        // First rotate n-1 items, then exchange the last two
        rotateLeft(a, n - 1);

        double temp = a[n - 2];
        a[n - 2] = a[n - 1];
        a[n - 1] = temp;
    }
}
```

- **Exercise 2.15.** *Write a program to print all combinations of n letters taken r at a time. As opposed to permutations, the order of the elements in combinations is irrelevant. For example, the combinations of six letters taken four at a time are the possible sets of four letters from abcdef as follows:*

abcd	*bcde*	*cdef*
abce	*bcdf*	
abcf	*bcef*	
abde	*bdef*	
abdf		
abef		
acde		
acdf		
acef		
adef		

The solution for selecting four letters from abcdef is to first output 'a' followed by the solution for selecting three letters from bcdef. Then output 'b' followed by the solution for selecting three letters from cdef. Then output 'c' followed by the solution for selecting three letters from def.

The following is the parameter list for a procedure to output the combinations:

```
void comb(char prefix[], int n, int r, char suffix[])
// Prints the prefix string followed by the combination
// of n characters taken r at a time from the suffix
// string.  Assumes suffix contains n characters.
```

To produce the previous list of combinations, the main program called

```
comb("", 6, 4, "abcdef")
```

The top-level recursive calls were

```
comb("a", 5, 3, "bcdef")
comb("b", 4, 3, "cdef")
comb("c", 3, 3, "def")
```

As in Program 2.15, you will need to make copies of `prefix` *and* `suffix` *using* `strcpy` *from* `string.h`*. Before each recursive call you will need to concatenate the first character from the suffix onto the last character of the prefix, and then strip the first character from the suffix. You can use the procedure*

```
strncat(char s[], const char c[], int n)
```

from `string.h` *to perform the concatenation. It concatenates* `n` *characters of string* `c` *to string* `s`*, so you can call it with an actual parameter of 1 for* `n`*.*

Draw the call tree for this data set. How many times is your procedure called? What is the maximum number of stack frames on the run-time stack?

```
#include <iostream>
#include <cstring>
using namespace std;

void comb(char prefix[], int n, int r, char suffix[])
// Prints the prefix string followed by the combination of n characters taken
// r at a time from the suffix string.  Assumes suffix contains n characters.
{
    if (r == 0) {
        // No more characters are needed from suffix
        cout << prefix << endl;
    } else {
        // Allocate space for a copy of prefix, with one extra character
        // (plus a terminating NUL, for a C-style string)
        char *newPrefix = new char[strlen(prefix) + 2];

        // Choose one letter at a time from suffix to append to prefix,
        // making sure that at least r-1 characters may be chosen from the
        // rest of suffix
        for (int i = 0; i <= n - r; i++) {
            strcpy(newPrefix, prefix);
            strncat(newPrefix, suffix + i, 1);
            comb(newPrefix, n - (i + 1), r - 1, suffix + (i + 1));
        }

        delete newPrefix;
    }
}

int main ()
{
    comb("", 6, 4, "abcdef");
}
```

Here is the call tree. The procedure is called 35 times, with a maximum depth of five stack frames.