ENSF 337: Programming Fundamentals
Lab 10 - Solutions

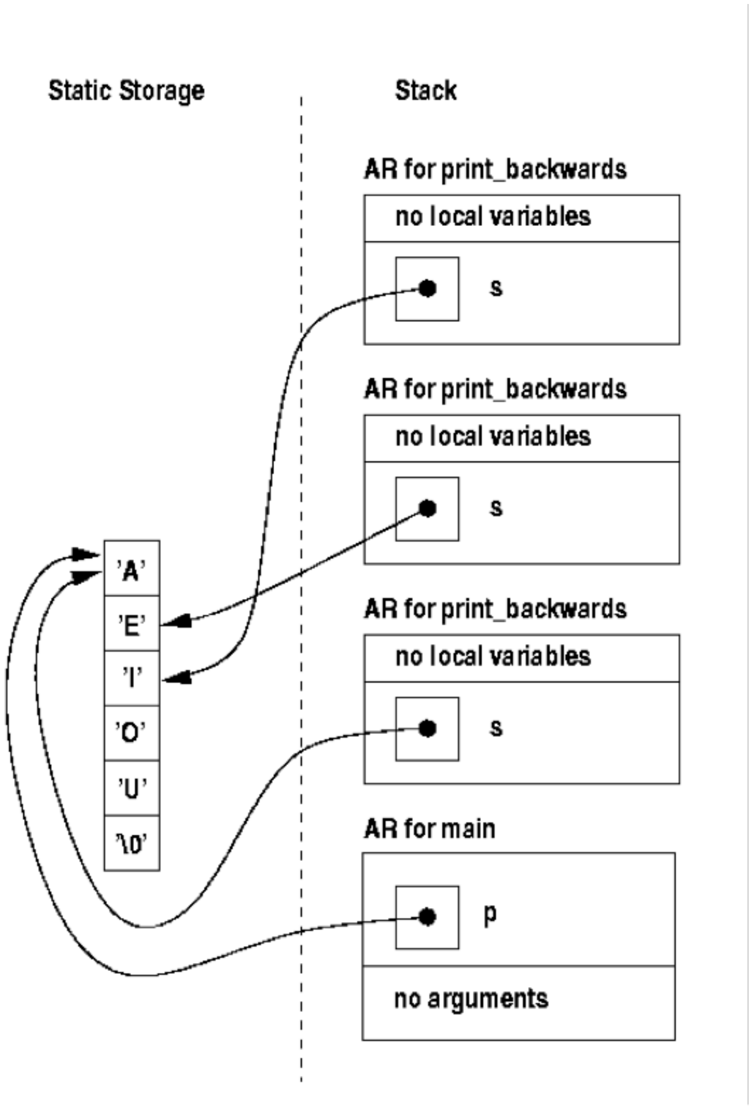
Exercise A: Array of Pointers and Command Line Arguments

```
int main(int argc, char** argv){
    int sort_order = 1; // 1 for ascending order and 2 for descending order
   if(argc > 2){
        cout << "\nUsage: Invalid input on the command line...";
    }
    else if(argc == 1)
        sort_order = 1;
    else {
        if(argv[1][0] == '-' && toupper(argv[1][1]) == 'A')
            sort_order = 1;
        else if(argv[1][0] == '-' && toupper(argv[1][1]) == 'D')
            sort_order= 2;
        else {
            cout << "Usage: Sort options can be only -a, -A, -d, or -D.\n";
            exit(1);
        }
    }

    int a[] = { 413, 282, 660, 171, 308, 537 };

    int n_elements = sizeof(a) / sizeof(int);

    cout << "Here is your array of integers before sorting: \n";
    for(int i = 0; i < n_elements; i++)
        cout <<  a[i] << endl;
    cout << endl;

    insertion_sort(a, n_elements, sort_order);

    if(sort_order == 1)
        cout << "Here is your array of integers after ascending sort:  \n" ;
    else if(sort_order == 2)
        cout << "Here is your array of integers after descending sorting:  \n" ;

    for(int i = 0; i < n_elements; i++)
        cout << a[i] << endl;


    return 0;
}
```

Exercise B: Using Pointer to Pointers

```
void append_strings (const char** string_array, int n, char** appended_string) {
int total_length = 0;
    for(int i=0; i < n; i++)
        total_length += strlen(string_array[i]);

    *appended_string = new char [total_length + 1];
    int k = 0;

    for(int i = 0; i < n; i++) {
        for(int j = 0; string_array [i][j] != '\0'; j++) {
            (*appended_string)[k] = string_array[i][j];
            k++;
        }
    }
    (*appended_string)[k] = '\0';
}
```

Exercise C: Understanding Recursion



Exercise D: A simple problem in writing recursive code

This is an in-lab exercise

There are lots of ways to solve this problem recursively. This one is perhaps the most obvious:

```
int sum_of_array(const int *a, int n)
{
  int result;
  assert(n > 0);
  if (n == 1)
    result = a[0];
  else
    result = a[0] + sum_of_array(a + 1, n - 1);
  return result;
}
```

Exercise E: A slightly harder example

Here is one way to solve the problem by first solving a simpler problem of the same kind:
If there is only one element, the result is 1. That's the base case.
Otherwise, the result is 1 if and only if two things are true: (1) a[0] < a[1]; and (2) elements a[1] to a[n-1] are all ordered properly. Notice that checking (2) is a simpler problem of the same kind.

This can be coded in C++ as:

```
int strictly_increasing(const int *a, int n)
{
    int result;

    assert(n > 0);

    if (n == 1)
        result = 1;
    else
        result = a[0] < a[1]
        && strictly_increasing(a + 1, n - 1);
    return result;
}
```

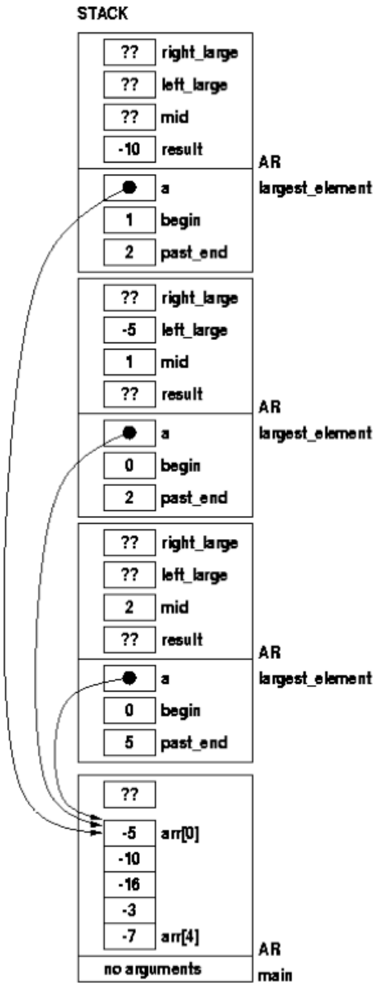
Here is a second solution. This version is based on the following ideas:

Again, if there is only one element, the result is 1. That's the base case.
Otherwise, let's split the array into two subarrays that are roughly equal in size. The result is 1 if and only if three things are true: (1) the last element in the front subarray is less than the first element in the back subarray; (2) the elements in the front subarray are ordered properly; and (3) the elements in the back subarray are ordered properly. (2) and (3) are both simpler problems of the same kind.

A C++ translation of this second solution is:

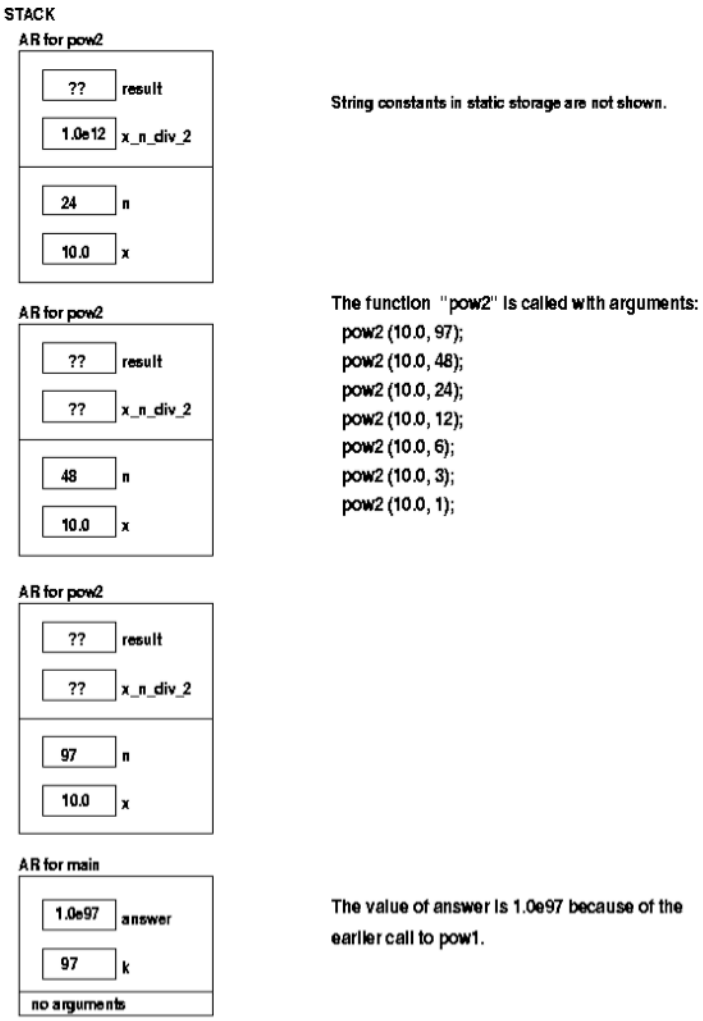
```
int strictly_increasing(const int *a, int n)
{
    int result;
    int half;    /* approximately half of n */

    assert(n > 0);

    if (n == 1)
        result = 1;
    else {
        half = n / 2;
        result = a[half - 1] < a[half]
        && strictly_increasing(a, half)
        && strictly_increasing(a + half, n - half);
    }
    return result;
}
```

Exercise F: Largest element in an array:



String constants in static storage are not shown

Exercise G: Raising a number to an integer power



String constants in static storage are not shown.

The function "pow2" is called with arguments:
pow2 (10.0, 97);
pow2 (10.0, 48);
pow2 (10.0, 24);
pow2 (10.0, 12);
pow2 (10.0, 6);
pow2 (10.0, 3);
pow2 (10.0, 1);

The value of answer is 1.0e97 because of the earlier call to pow1.

Exercise H: A class with recursive member functions

```
void OLList::copy(const OLList& source)
{
   headM = copy_sublist(source.headM);
}

Node * OLList::copy_sublist(const Node *source_sublist) {
  Node * result;
  if (source_sublist == 0)
    result = 0;
  else {
    result = new Node;
    result->item = source_sublist->item;
    result->next = copy_sublist(source_sublist->next);
  }
  return result;
}
```