Lab 13 - Recursion

Below are several exercises for you to practice writing your own recursive functions. Within your functions, make sure you comment your code to specify the base case and the recursive function call.

1. Write a recursive function that computes the sum of all numbers from 1 to n, where n is given as parameter.

```
//return the sum 1 + 2 + 3 + … + n
int sum(int n)
{
if(n<=1)
      return 1;
else
      return n + sum(n-1);
}
```

2. Write a recursive function that finds and returns the minimum element in an array, where the array and its size are given as parameters.

```
//return the minimum element in a[]
int findmin(const int a[], int n)
```

```
int findmin(const int a[], int n)
{
      if (n == 1)
          return a[0];

      int result;

      //can replace "result" with findmin(a, n - 1)
      //every single place "result" appears, and
      //although it makes the total recursive call
      //use significantly more memory, if not used,
      //have to do a recursive call twice, within
      //every call of the function.
      //This is unnecessary since I'm only comparing
```

```
        //the value and then returning said value.
        result = findmin(a, n - 1);
        if (result < a[n - 1])
            return result;
        else
            return a[n - 1];
}
```

3. Write a recursive function that computes and returns the sum of all elements in an array, where the array and its size are given as parameters.

```
//return the sum of all elements in a[]
int findsum(const int a[], int n)
```

```
int findsum(const int a[], int n)
{
   if (n==0)
   {
       return a[0];
   }
   else
       return findsum(a,n-1)+a[n-1];
}
```

4. Write a recursive function that determines whether an array is a palindrome, where the array and its size are given as parameters.

```
//returns true if a[] is a palindrome, false otherwise
bool ispalindrome(const char a[], int n)
```

```
bool ispalindrome(const char a[], int n)
{
   if (n <= 0)
       return true;

   if (a[n-1] == a[0])
```

```
        return ispalindrome(a+1,  n-2);
    else
        return false;
}
```

5. Implement a function that generates all substrings of a string. For example, the substrings of the string "rum" are the seven strings "r", "ru", "rum", "u", "um", "m", "" **Hint:** First enumerate all substrings that start with the first character. There are n of them if the string has length n. Then enumerate the substrings of the string that you obtain by removing the first character.

```
//returns a string vector containing all substrings of the the
//string s
vector<string> generate_substrings(string s)
```

```
void recursive_substring(string s, vector<string>& substrings)
{
   if (s.size() == 0)
       return;

   substrings.push_back(s);
   s.pop_back();
   recursive_substring(s, substrings);
}

vector<string> generate_substrings(string s)
{
   vector<string> substrings;
   for (int i = 0; i < s.size(); i++)
       recursive_substring(s.substr(i, s.size()), substrings);

   substrings.push_back("");
   return substrings;
}
```

6. Write a recursive function that takes as a parameter a nonnegative integer and generates the following pattern of stars. If the nonnegative integer is 4, then the pattern generated is:

```
****
***
**
*
*
**
***
****
//draws the above pattern based on the input
void drawPattern(int n)
```

```
void drawPattern(int n)
{
   if (n == 0)
       return;

   for (int i = 0; i < n; i++)
       cout << '*';
   cout << "\n";
   drawPattern(n - 1);
   for (int i = 0; i < n; i++)
       cout << '*';
   cout << "\n";
}
```