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)

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)

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)

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)

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)

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)

```
#include <iostream>
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include <cstring>
#include <vector>
/*********************************************
*   Author        : Jonathan Sum
*   Assignment    : Lab 13 - Recursion
*   SECTION       : CS 003A
*   Due Date      : 4/23/19
*   ____________________________________________
*   Goal: practice writing your own recursive functions
*********************************************/

using namespace std;

/************************************************
*
*   FUNCTION int sum(int n)
_____________________________________________
*   This method computes the sum of all numbers
*   from 1 to n
_____________________________________________
*   PRE-CONDITIONS
*   input the number n in integer
*   POST-CONDITIONS
*   computes the sum of all numbers
************************************************/
int sum(int n          //IN - input number
);

/************************************************
*
*   FUNCTION findmin
_____________________________________________
*   A recursive function that finds the minimum element
```

```
*   and returns the minimum element in an array
_____________________________________________________
*   PRE-CONDITIONS
*   input an array and its size
*   POST-CONDITIONS
*   returns the minimum element in an array
*****************************************************/
int findmin(const int a[]          //IN - array to find the smallest number
            , int n);          //IN - size of array

/*****************************************************
*
*   FUNCTION findsum
_____________________________________________________
*   A recursive function that computes and returns
*   the sum of all elements in an array
_____________________________________________________
*   PRE-CONDITIONS
*   input an array and its size
*   POST-CONDITIONS
*   returns the sum of each element in the array
*****************************************************/
int findsum(const int a[]          //IN - array to find the sum of the array
        , int n);



/*****************************************************
*
*   FUNCTION ispalindrome
_____________________________________________________
*   A recursive function that determines whether
*   an array is a palindrome array
_____________________________________________________
*   PRE-CONDITIONS
*   input the array and its size
*   POST-CONDITIONS
*   returns a boolean that if the array is
*   palindrome it will return true.
```

```
**************************************************/
bool ispalindrome(const char a[]          //IN - array to check ispalindrome
                , int n);             //IN - size of the array

/**************************************************
*
*   FUNCTION generate_substrings
_____________________________________________________
*   A function that generates all
*   substrings of a string
_____________________________________________________
*   PRE-CONDITIONS
*   input a string
*   POST-CONDITIONS
*   return a vector array in string that has
*   all the substrings
**************************************************/
vector<string> generate_substrings(string s);

/**************************************************
*
*   FUNCTION drawPattern
_____________________________________________________
*   A recursive function that takes as a parameter
*   a nonnegative integer and generates
*   the following pattern of stars.
_____________________________________________________
*   PRE-CONDITIONS
*   input a positive number
*   POST-CONDITIONS
*   print stars on screen
**************************************************/
void drawPattern(int n          //IN - positive number for drawing a pattern
                );
int main() {
```

```
}

int sum(int n           //IN - input number
) {
  if (n == 1) {
     return 1;
  } else {
     return n + sum(n - 1);
  }
}

int findmin(const int a[]          //IN - array to find the smallest number
     , int n                       //IN - size of the array
) {
  //PROCESSING -  base cause to return the first element
  if (n == 0) {
     return a[0];
  }
  //PROCESSING -  if it is small, return the small number at last
  if (a[n - 1] < findmin(a, n - 1))
     return a[n - 1];
  //PROCESSING -  if it is small, return the small number in the array
  if (findmin(a, n - 1) < a[n - 1])
     return findmin(a, n - 1);
  else
     return a[n - 1];
}

int findsum(const int a[]          //IN - array to find the sum of the array
     , int n) {
  //PROCESSING -  Keep recursiving to calc the sum
  if (n == 0) {
     return 0;
  } else {
     return a[n - 1] + findsum(a, n - 1);

  }
}
```

```
bool ispalindrome(const char a[]           //IN - array to check ispalindrome
      , int n)              //IN - size of the array
 {
    //PROCESSING - base cause to return true if it is ispalindrome
  if (n == 0 || n == 1) {
      return true;

  }
    //PROCESSING - return true if two elements are same
  if (a[0] == a[n - 1]) {
      string str(a);
      str = str.substr(1, static_cast<unsigned long>(n - 1));
      char a2[n - 2];
      strcpy(a2, str.c_str());
      return ispalindrome(a2, n - 2);

      //PROCESSING - return false if two elements are not same
  } else {

      return false;
  }

}

vector<string> generate_substrings(string s         //IN - input string
) {
  vector<string> list;
  //PROCESSING - loop thru each element
  for (int i = 0; i < s.size(); i++) {
      string head = string(1, s[i]);
      cout <<"starting: "<< head << endl;
      list.push_back(head);

      //PROCESSING - add each element for substring
      for (int j = i+1; j < s.size(); j++) {
          list.push_back(list[(list.size()-1)]+s[j]);
          cout<<"j is: "<<j<<": ";
          cout<<s[j];
          cout<<endl;
```

```
      }
   }
   list.emplace_back("");
   //OUT - return the list
   return list;

}

void drawPattern(int n      //IN - input positive number for drawing the pattern
){

   //PROCESSING - base case if n is 1, then print nonthing
   if (n < 1)
      cout << "";
   else {
      //PROCESSING - print the upper part of pattern
      for(int i=0;i<n;i++){
         cout<<"*";
      }
      cout<<endl;

      //PROCESSING - recursive  the function
      drawPattern(n-1);

      //PROCESSING - print the lower part of pattern
      for(int i=0;i<n;i++){
         cout<<"*";
      }
      cout<<endl;
   }
}
```

Lab 14

```
#include <iostream>
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include <cstring>
#include <vector>
#include <string>
/*******************************************
*   Author         : Jonathan Sum
*   Assignment     : Lab 14 - Recursion
*   SECTION        : CS 003A
*   Due Date       : 4/28/19
*   ____________________________________________________
*   Goal: write a maze
*******************************************/


using namespace std;
```

```
/*************************************************
*
*   FUNCTION void drawCross(int minX, int maxX, int minY, int maxY,vector<vector<string>
> &maze);
_________________________________________________
*   This function builds a maze recursively.
_________________________________________________
*   PRE-CONDITIONS
*   empty maze with only borders of walls
*   POST-CONDITIONS
*   a complete maze
*************************************************/
const int ROW = 10,         //IN - row of the maze
      COL = 12;         //IN - column of the maze

void drawCross(int minX      //IN - left edge of the maze
      , int maxX      //IN - right edge of the maze
      , int minY      //IN - upper edge of the maze
      , int maxY      //IN - lower edge of the maze
      ,vector<vector<string> > &maze      //IN - maze
      );


int main(){
  const string b = "\u25A0";      //IN shape of the wall

  //set up the randam seed
  srand(static_cast<unsigned int>(time(nullptr)));



  //IN & PROCESSING build a default maze
  vector<vector<string> > mazeMap
        {{b, b,   b,   b,   b,   b,   b,   b,   b,   b,   b,   b},  //0
         {b, " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", b},  //1
         {b, " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", b},  //2
         {b, " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", b},  //3
         {b, " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", b},  //4
         {b, " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", b},  //5
```

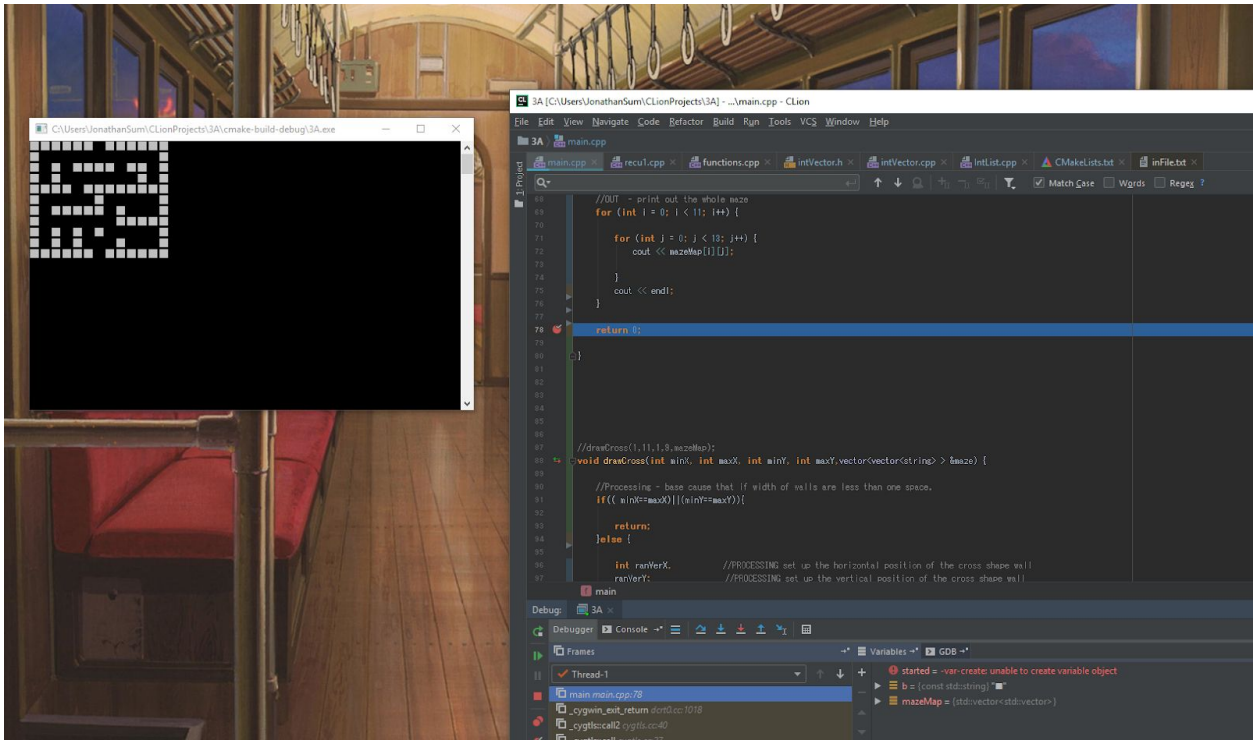
```
        {b," ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ", b},  //6
        {b," ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ", b},  //7
        {b," ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ", b},  //8
        {b," ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ","  ", b},  //9
        {b, b,   b,   b,   b,   b,   b,   b,   b,   b,   b,   b,   b,   b},};//10
      //  0  1   2    3    4    5    6    7    8    9    10   11   12

  //Processing build all the walls in the maze
  drawCross(1,COL-1,1,ROW-1,mazeMap);

  //OUT  - print out the whole maze
  for (int i = 0; i < 11; i++) {

     for (int j = 0; j < 13; j++) {
        cout << mazeMap[i][j];

     }
     cout << endl;
  }

  return 0;

}
```

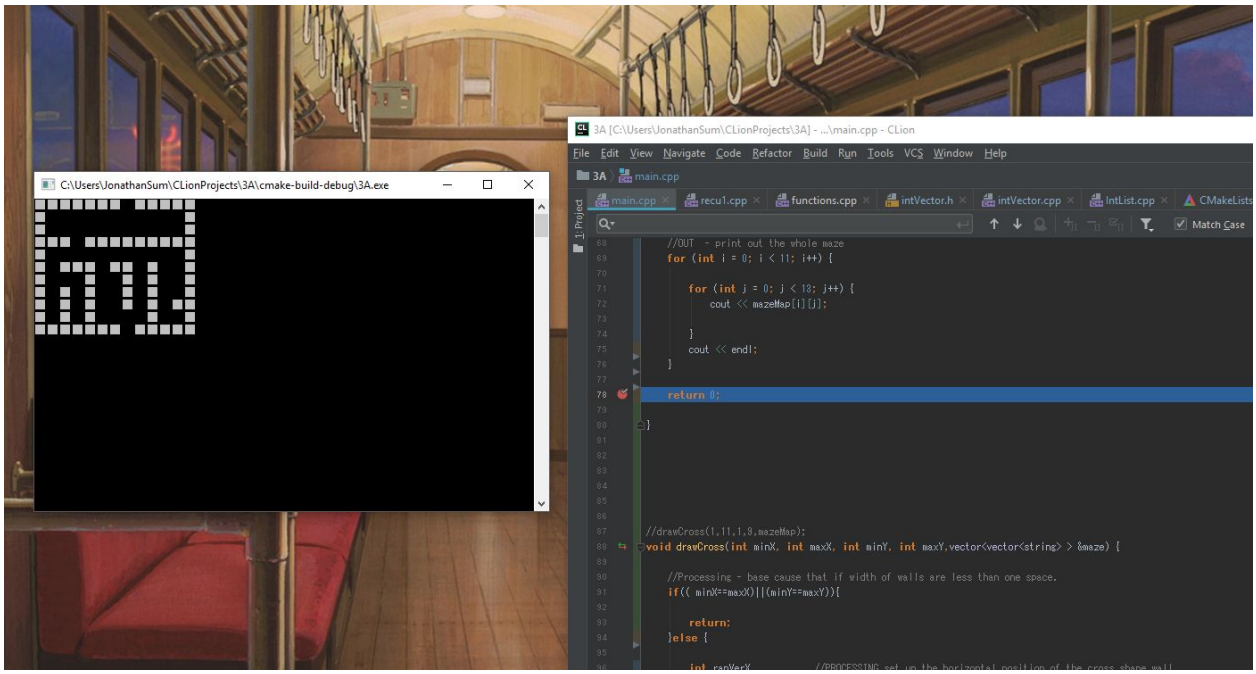
```
//drawCross(1,11,1,9,mazeMap);
void drawCross(int minX, int maxX, int minY, int maxY,vector<vector<string> > &maze) {

  //Processing - base cause that if width of walls are less than one space.
  if(( minX==maxX)||(minY==maxY)){

     return;
  }else {

     int ranVerX,          //PROCESSING set up the horizontal position of the cross shape wall
```

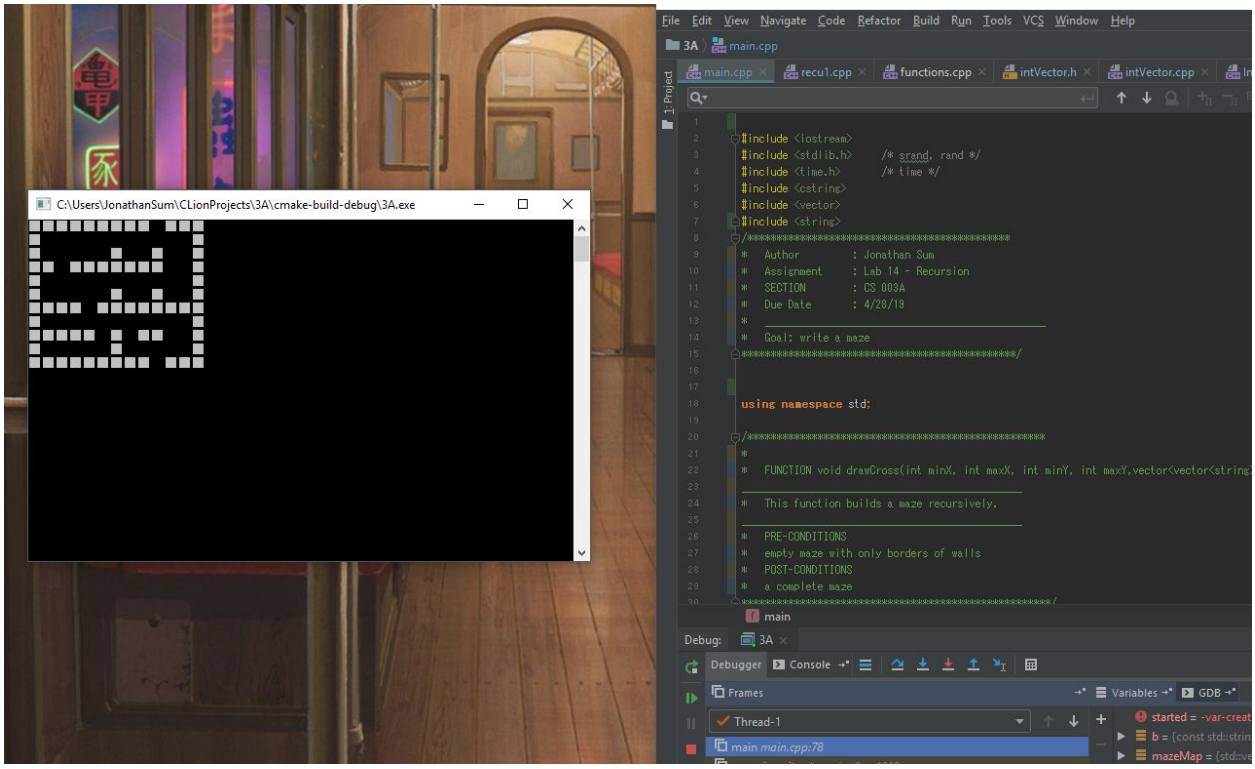
```
ranVerY;              //PROCESSING set up the vertical position of the cross shape wall
const string b = "\u25A0";          //IN - the shape of wall

//PROCESSING set up the horizontal and vertical position
// of the cross shape wall

//Processing - base cause that if width of walls are than one space.
if((minX+1)==maxX){
   return;
}else{
  ranVerX = (minX + 1) + rand() % (((maxX - 1) + 1) - (minX + 1));



}
if((minY+1)==maxY){

   return;
}else{
    ranVerY = (minY + 1) + rand() % (((maxY - 1) + 1) - (minY + 1));
}



int left          //PROCESSING - hole at the left of the cross shape wall
, right           //PROCESSING - hole right of the cross shape wall
, up           //PROCESSING - hole of right the cross shape wall
, down;          //PROCESSING - holeof right the cross shape wall

//PROCESSING & IN set up the location of three holes of two walls
if ((maxX - 1) == (ranVerX + 1)) {
   right =( maxX - 1);
} else {
   right = (ranVerX + 1) + rand() % ((maxX - 1) - (ranVerX + 1));
}
if ((maxY - 1) == (ranVerY + 1)) {
   down = maxY - 1;
} else {
   down = (ranVerY + 1) + rand() % (maxY - 1 - (ranVerY + 1));
}
```

```
if((ranVerX  == minY)==0){

}
left = minX + rand() % (((ranVerX + 1) - 1) - minX);

up = minY + rand() % ((ranVerY + 1) - 1 - minY);




//PROCESSING - build a horizontal line
for (int j = minX; j <= maxX; j++) {

   maze[ranVerY][j] = b;
}
//PROCESSING - build a vertical line
for (int i = minY; i <= maxY; i++) {

   maze[i][ranVerX] = b;
}

//randomly choice holes are left wall, right wall and upper wall
//or down wall, left wall and upper wall
int half = 1 + rand() % ((2 + 1) - 1);

//PROCESSING - build three holes
if (half == 1) {

   maze[ranVerY][left] = "  ";
   maze[ranVerY][right] = "  ";
   maze[up][ranVerX] = "  ";
} else {

   maze[down][ranVerX] = "  ";
   maze[ranVerY][left] = "  ";
   maze[up][ranVerX] = "  ";
}
//PROCESSING - build an exit and entry
if(maxX==11&&maxY==9&&minX==1&&minY==1){
```

```
        maze[0][ranVerX]="  ";
        maze[1][ranVerX]="  ";
        maze[9][ranVerX]="  ";
        maze[10][ranVerX]="  ";
    }
  //PROCESSING - Recursion any upper right corners of four corners in the square maze

  drawCross(minX, ranVerX - 1, minY, ranVerY - 1, maze);
    //PROCESSING - Recursion any lower right corners of four corners in the square maze
  drawCross( minX, ranVerX-1, ranVerY+1,  maxY,maze);
    //PROCESSING - Recursion any upper left corners of four corners in the square maze
  drawCross(ranVerX+1,  maxX,  minY, ranVerY-1,maze);
    //PROCESSING - Recursion any lower left corners of four corners in the square maze
  drawCross(ranVerX+1,  maxX, ranVerY+1,  maxY,maze);
  }

}
```




C:\Users\JonathanSum\CLionProjects\3A\cmake-build-debug\3A.exe
3A [C:\Users\JonathanSum\CLionProjects\3A] - ...\main.cpp - CLion
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
3A main.cpp
main.cpp recu1.cpp functions.cpp intVector.h intVector.cpp intList.cpp CMakeLists
Match Case
//OUT - print out the whole maze
for (int i = 0; i < 11; i++) {
for (int j = 0; j < 18; j++) {
cout << mazeMap[i][j];
}
cout << endl;
}
return 0;
}
//drawCross(1,11,1,8,mazeMap);
void drawCross(int minX, int maxX, int minY, int maxY,vector<vector<string> > &maze) {
//Processing - base cause that if width of walls are less than one space.
if((minX==maxX)||(minY==maxY)){
return;
}else {


C:\Users\JonathanSum\CLionProjects\3A\cmake-build-debug\3A.exe
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
3A main.cpp
main.cpp recu1.cpp functions.cpp intVector.h intVector.cpp
1: Project
#include <iostream>
#include <stdlib.h> /* srand, rand */
#include <time.h> /* time */
#include <cstring>
#include <vector>
#include <string>
* Author : Jonathan Sum
* Assignment : Lab 14 - Recursion
* SECTION : CS 003A
* Due Date : 4/28/19
* Goal: write a maze
using namespace std;
* FUNCTION void drawCross(int minX, int maxX, int minY, int maxY,vector<vector<string
* This function builds a maze recursively.
* PRE-CONDITIONS
* empty maze with only borders of walls
* POST-CONDITIONS
* a complete maze
main
Debug: 3A
Debugger Console
Frames Variables GDB
Thread-1
main main.cpp:78
started = -var-creat
b = {const std::strin
mazeMap = {std::ve