Problem 5 [11 points] Recursion

Given a 10x10 2D array, `data`, with a number of 1s and a starting point (x, y). The number at (x, y) spreads out to its 4-neighbor (i.e. right, bottom, left, top) which their values will be assigned by the number of starting point + 1. For example, suppose `data` is

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0 0 0
0 0 1 1 1 1 1 1 0 0
0 0 1 1 1 1 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 1 1 1 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

and starting point (x, y) is (3, 3).

- Starting from the point data[3][3] with value 1. Its neighbor's value will be 2.

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 2 1 0 0 0 0 0
0 0 2 1 2 1 1 1 0 0
0 0 1 2 1 1 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 1 1 1 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

- Next, the 4 neighbors of the point having value of 2 will be 3.

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 3 2 3 0 0 0 0 0
0 0 2 1 2 3 1 1 0 0
0 0 3 2 3 1 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 1 1 1 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

- The same process is repeated until all number 1s in the 2D array are processed.

```
0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0
0 0 3 2 3 0 0 0 0 0    0 0 3 2 3 0 0 0 0 0    0 0 3 2 3 0 0 0 0 0
0 0 2 1 2 3 4 1 0 0    0 0 2 1 2 3 4 5 0 0    0 0 2 1 2 3 4 5 0 0
0 0 3 2 3 4 0 0 0 0 -> 0 0 3 2 3 4 0 0 0 0 -> 0 0 3 2 3 4 0 0 0 0 ->
0 0 0 0 4 1 0 0 0 0    0 0 0 0 4 5 0 0 0 0    0 0 0 0 4 5 0 0 0 0
0 0 0 0 1 1 0 0 0 0    0 0 0 0 5 1 0 0 0 0    0 0 0 0 5 6 0 0 0 0
0 0 1 1 1 0 0 0 0 0    0 0 1 1 1 0 0 0 0 0    0 0 1 1 6 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0    0 0 1 1 0 0 0 0 0 0    0 0 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0
0 0 3 2 3 0 0 0 0 0    0 0 3 2 3 0 0 0 0 0    0 0 3 2 3 0 0 0 0 0
0 0 2 1 2 3 4 5 0 0    0 0 2 1 2 3 4 5 0 0    0 0 2 1 2 3 4 5 0 0
0 0 3 2 3 4 0 0 0 0 -> 0 0 3 2 3 4 0 0 0 0 -> 0 0 3 2 3 4 0 0 0 0
0 0 0 0 4 5 0 0 0 0    0 0 0 0 4 5 0 0 0 0    0 0 0 0 4 5 0 0 0 0
0 0 0 0 5 6 0 0 0 0    0 0 0 0 5 6 0 0 0 0    0 0 0 0 5 6 0 0 0 0
0 0 1 7 6 0 0 0 0 0    0 0 8 7 6 0 0 0 0 0    0 0 8 7 6 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0    0 0 1 8 0 0 0 0 0 0    0 0 9 8 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0 0 0
```

Your task is to implement the global function `spreadout` according to the process described above so that the function will work with the testing program below to produce the expected output.

```
void spreadout(int data[10][10], int x, int y, int v, int mark[10][10]) {
  // ASSUME YOUR CODE WILL BE HERE.
}

int main() {
  int data[10][10] = {
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 1, 1, 1, 0, 0, 0, 0, 0},
    {0, 0, 1, 1, 1, 1, 1, 1, 0, 0},
    {0, 0, 1, 1, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
    {0, 0, 1, 1, 1, 0, 0, 0, 0, 0},
    {0, 0, 1, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  };

  int mark[10][10] = {}; // All elements are zero-initialized.
  spreadout(data, 3, 3, 1, mark);

  for(int i = 0; i < 10; ++i) {
    for(int j = 0; j < 10; ++j) {
      cout << data[i][j] << " ";
    }
    cout << endl;
  }
  return 0;
}
```

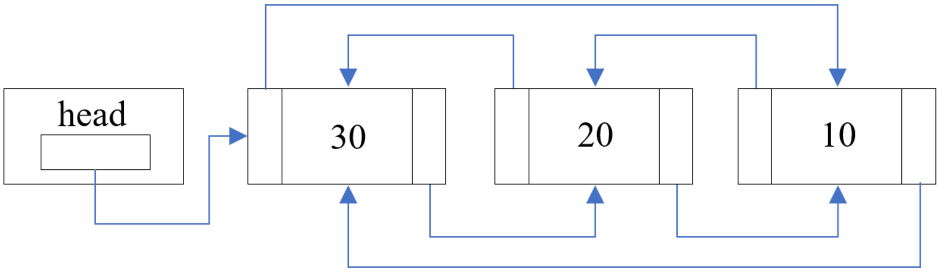
Expected output of the testing program

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 3 2 3 0 0 0 0 0
0 0 2 1 2 3 4 5 0 0
0 0 3 2 3 4 0 0 0 0
0 0 0 0 4 5 0 0 0 0
0 0 0 0 5 6 0 0 0 0
0 0 8 7 6 0 0 0 0 0
0 0 9 8 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Note: You must implement the function as a recursive function.

Problem 6 [24 points] Circular Doubly Linked List

A circular doubly linked list (CDLL) is shown below.



Given the following definition of CDLL_Node, CDLL and a number of global function prototypes:

```
struct CDLL_Node {   /* Filename: CDLL.h */
  int data;
  CDLL_Node* prev;
  CDLL_Node* next;
};

struct CDLL {
  CDLL_Node* head = nullptr;
};

// It creates a CDLL object with a CDLL_Node that stores the specified int value,
// and it returns created CDLL object by value.
CDLL create(int value);

// It checks whether a CDLL is empty. If it is empty, return true.
// Otherwise, return false.
bool isEmpty(const CDLL& cdll);

// It inserts one CDLL_Node with the specified value at the start of cdll.
void insertAtFront(CDLL& cdll, int value);

// It removes one CDLL_Node from the end of cdll.
// It returns the item value of the removed node.
// In case the CDLL is empty, returns -999.
int removeFromBack(CDLL& cdll);

// --------------------------------------
//  ASSUME YOUR IMPLEMENTATIONS ARE HERE
// --------------------------------------
```

Implement all the global functions such that the following test program produces the expected output.

```
#include "CDLL.h" /* Filename: test-CDLL.cpp */

void printCDLL(const CDLL& cdll) {
  if(isEmpty(cdll)) {
    cout << "Empty" << endl;
    return;
  }
  CDLL_Node* cur = cdll.head;
  do {
    cout << cur->data << " ";
    cur = cur->next;
  }while(cur != cdll.head);
  cout << endl;
}

/* Filename: test-CDLL.cpp */
int main() {
  CDLL cdll = create(10);
  insertAtFront(cdll, 20);
  insertAtFront(cdll, 30);
  cout << "After insertAtFront 10, 20, 30" << endl;
  cout << "Current CDLL: ";
  printCDLL(cdll);
  cout << endl;

  int value = removeFromBack(cdll);
  cout << "After removeFromBack" << endl;
  cout << "Current CDLL: ";
  printCDLL(cdll);
  cout << "The node at the back of CDLL is " << value << endl << endl;

  value = removeFromBack(cdll);
  cout << "After removeFromBack" << endl;
  cout << "Current CDLL: ";
  printCDLL(cdll);
  cout << "The node at the back of CDLL is " << value << endl << endl;

  value = removeFromBack(cdll);
  cout << "After removeFromBack" << endl;
  cout << "Current CDLL: ";
  printCDLL(cdll);
  cout << "The node at the back of CDLL is " << value << endl << endl;
}
```

Expected output of the test program

```
After insertAtFront 10, 20, 30
Current CDLL: 30 20 10

After removeFromBack
Current CDLL: 30 20
The node at the back of CDLL is 10

After removeFromBack
Current CDLL: 30
The node at the back of CDLL is 20

After removeFromBack
Current CDLL: Empty
The node at the back of CDLL is 30
```

Problem 7 [35 points] Pointer and Dynamic Array

Write an application program for implementing a vocabulary dictionary that can store an arbitrarily number of words using a dynamic array. The following shows what you need to implement for this question.

- A structure named `Dictionary`, which has two members:
  - a pointer to pointer named `wordList`, which will point to an array of string pointers, and each string pointer will be pointing at a string object representing a word.
  - an int variable named `len` that records the number of words stored in the dictionary.

- A global function, `void init(Dictionary& dict)`, which initializes `wordList` and `len` of dictionary to `nullptr` and 0 respectively.

- A global function, `int findWord(const Dictionary& dict, const string& word)`, which returns the index for the memory location of the array `wordList` of `dict` that contains the specified `word` in the dictionary. It returns -1 if the word is not in the dictionary.

- A global function, `bool insertWord(Dictionary& dict, const string& word)`, which inserts the specified `word` into dictionary. The `word` should be inserted into the dictionary and sorted according to lexicographic order (i.e. ascending alphabetical order). Before insertion, the function should check whether the specified word, `word`, has already been stored in the dictionary. If so, it returns false. Otherwise, it returns true.

  Note:
  As the size of original array is fixed once it is created, you need to do the following in order to store a new word:
  - Allocate a new array of size `len + 1`.
  - Copy all the pointers in the original array to the new array.
  - Insert the new word to the array in lexicographic order.
  - Make `wordList` point at the new array.
  - Make sure there is **NO** memory leak problem after performing all the above operations.

- A global function, `bool removeWord(Dictionary& dict, const string& word)`, removes the specified `word` from the dictionary. The function should check whether the specified word, `word`, is in the dictionary. If not, it returns false. Otherwise it returns true. Similar to `insertWord`, you need to allocate a new array of size `len - 1`, which makes it just fit to keep all the remaining words. The process of this should be similar to the one for `insertWord`.

- A global function, `void displayDict(const Dictionary& dict)`, which prints all the words stored in the dictionary.

- A global function, `void destroy(Dictionary& dict)`, which de-allocates ALL the dynamically allocated memory for the dictionary.

Your task is to implement the structure Dictionary and the 6 required global functions in `Dictionary.h`. Your implementation is supposed to work with the test program "`test-dictionary.cpp`" shown below:

```cpp
#include "Dictionary.h" /* Filename: test-dictionary.cpp */

string inputWord() {
  cout << "Enter a word: ";
  string word;
  cin >> word;
  return word;
}

int main() {
  Dictionary dict;
  init(dict);

  char option = ' ';
  int index;
  while(option != 'Q' && option != 'q') {
    cout << "(F) Find a word, (I) Insert a word, (R) Remove a word, ";
    cout << "(D) Display dictionary, (Q) Quit\n";
    cout << "Option: ";
    cin >> option;
    switch(option) {
      case 'F': case 'f':
        index = findWord(dict, inputWord());
        if(index == -1) cout << "Not in the dictionary\n";
        else cout << "It is at location " << index << "\n";
        break;
      case 'I': case 'i':
        cout << ((insertWord(dict, inputWord())) ? "Success\n" : "Failure\n");
        break;
      case 'R': case 'r':
        cout << ((removeWord(dict, inputWord())) ? "Success\n" : "Failure\n");
        break;
      case 'D': case 'd':
        displayDict(dict);
        break;
    }
    cout << "\n";
  }
  destroy(dict);
}
```

A sample run of the test program is given as follows:

```
(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: F
Enter a word: University
Not in the dictionary

(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: I
Enter a word: University
Success

(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: I
Enter a word: Science
Success

(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: D
Science
University

(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: R
Enter a word: Technology
Failure

(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: R
Enter a word: Science
Success

(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: D
University

(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: R
Enter a word: University
Success

(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: R
Enter a word: University
Failure

(F) Find a word, (I) Insert a word, (R) Remove a word, (D) Display dictionary, (Q) Quit
Option: Q
```

/* Rough work */

/* Rough work */

/* Rough work */

/* Rough work */