COMP 2012H Midterm Exam - Fall 2018 - HKUST

Date: October 20, 2018 (Saturday)
Time Allowed: 2 hours, 10am–12nn
Instructions:

1. This is a closed-book, closed-notes examination.
2. There are **7** questions on **28** pages (including this cover page and 4 blank pages at the end).
3. Write your answers in the space provided.
4. All programming codes in your answers must be written in the ANSI C++ version as taught in the class.
5. For programming questions, unless otherwise stated, you are **NOT** allowed to define additional structures, classes, helper functions and use global variables, **auto**, nor any library functions not mentioned in the questions.
6. Approved calculators are allowed for this exam.

Student Name	SOLUTIONS AND MARKING SCHEME
Student ID	
Email Address	
Venue & Seat Number	

For T.A. Use Only

Problem	Topic	**Score**
1	True or False	/ 10
2	Function Parameter Passing Mechanism I	/ 7
3	Function Parameter Passing Mechanism II	/ 5
4	Structure and Object	/ 8
5	Recursion	/ 11
6	Circular Doubly Linked List	/ 24
7	Pointer and Dynamic Array	/ 35
Total		/ 100

Problem 1 [10 points] True or false

Indicate whether the following statements are *true* or *false* by circling **T** or **F**. You get 1.0 point for each correct answer, −0.5 for each wrong answer, and 0.0 if you do not answer.

T **F** (a) The following program:

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

int main() {
  int a = 10, b = 20, c = 30;
  if(a < 10)
    if(b > 15)
      cout << "Point A" << endl;
  else if(c < 30)
    cout << "Point B" << endl;
  else
    cout << "Point C" << endl;
}
```

can compile with NO errors and run, and it prints `Point C`.

T **F** (b) Given the following program:

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

enum classification { FIRST, SECOND_UPPER = -2, SECOND_LOWER, THIRD, PASS };

int main() {
  cout << "FIRST: " << FIRST << endl;
  cout << "SECOND_UPPER: " << SECOND_UPPER << endl;
  cout << "SECOND_LOWER: " << SECOND_LOWER << endl;
  cout << "THIRD: " << THIRD << endl;
  cout << "PASS: " << PASS << endl;
}
```

The output of the program is

```
FIRST: -1
SECOND_UPPER: -2
SECOND_LOWER: -1
THIRD: 0
PASS: 1
```

T **F** (c) The following program CANNOT be compiled successfully.

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

void print(int num);

int main() { print(); }

void print(int num = 10) {
  for(int i=0; i<num; i++)
    cout << "Printing..." << endl;
}
```

T F (d) The following code can be compiled WITHOUT errors.

```
int arr1[] = { 1, 2, 3, 4, 5 };
int arr2[] = { 6, 7, 8, 9, 10 };
arr1 = arr2;
```

T **F** (e) Assume an array is declared as follows:

```
int arr[] = { 1, 2, 3 };
```

The following statements all print the same value on screen.

```
cout << arr << endl;
cout << &arr << endl;
cout << &arr[0] << endl;
```

T F (f) The following program prints the address of `str` on screen.

```
#include <iostream>
using namespace std;
int main() {
  char str[] = "Peter";
  cout << str << endl;
}
```

T F (g) The following program CANNOT be compiled successfully.

```
int main() {
  int* const p = new int;
  const int* q = p;
  delete p;
}
```

T **F** (h) The following program will result in memory leak.

```
int main() {
  int* p;
  for(int i=0; i<10; i++) {
    p = new int[10];
  }
  delete [] p;
}
```

T **F** (i) The following program can be compiled WITHOUT errors.

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

struct Person {
  string name;
  char gender;
  int age;
  int marks[5];
};

int main() {
  Person bingyen, guangneng, jingyang;
  bingyen = guangneng = jingyang;
}
```

T **F** (j) The following program can be compiled WITHOUT errors.

```
struct MyArray {
  const int size;
  int* arr;
};

int main() {
  MyArray myArr { 10, new int[10] };
  delete [] myArr.arr;
}
```

Problem 2 [7 points] Function Parameter Passing Mechanism I

```
1  #include <iostream>
2  using namespace std;
3
4  int mystery(int x, int& y, int z) {
5    x = z += 5;
6    y = x + y + z;
7    return y;
8  }
9
10 int main() {
11   int a = 1, b = 2, c = 3;
12
13   /* This comment line is to be replaced by the statement in the question.  */
14
15   return 0;
16 }
```

If line #13 of the above program is replaced by each of the following statements, give the output if the resulting program can be compiled; otherwise, explain the compilation error.

(a) [2 points] `cout << mystery(a, b, c) << endl;`

Answer: 18

(b) [2 points] `cout << mystery(mystery(b, b, b), b, c) << endl;`

Answer: 32

(c) [3 points] `cout << mystery(a, mystery(a, b, c), c) << endl;`

Answer: Compilation error: expects a variable, not a value, for the 2nd parameter.

Problem 3 [5 points] Function Parameter Passing Mechanism II

C++ provides plus equal operator, +=, for integers. Implement this operator by a function `plus_equal` so that the following program

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

/* The definition of function plus_equal will be put here */

int main() {
  int a = 10;
  int b = 20;

  plus_equal(a, plus_equal(b, 5)); // Equivalent to a += b += 5;

  cout << "a: " << a << ", b: " <<  b << endl;
}
```

will give the output below:

```
a: 35, b: 25
```

Remark:

i. You are not allowed to use C++'s built in (plus equal) += operator in your answer.

ii. You have to decide the exact function header of the `plus_equal` function yourselves.

Answer:

```
// 1 point for giving correct return type
// 1 point for giving the correct type of the first parameter
// 1 point for giving the correct type of the second parameter

int plus_equal(int& x, int val) {
  x = x + val;  // 1 point
  return x;     // 1 point
}
```

Problem 4 [8 points] Structure and Object

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  struct EnglishWord {
6    char* str = nullptr; // You are NOT allowed to modify this line
7  };
8
9  void print(EnglishWord& ew);
10
11 // Initialize the EnglishWord ew with the given C string s
12 void init(EnglishWord& ew, char* s) {
13   ew.str = s;
14   print(ew);
15 }
16
17 // You are NOT allowed to modify this function
18 void remove(EnglishWord& ew) {
19   print(ew);
20   delete [] ew.str;
21 }
22
23 void print(EnglishWord& ew) {
24   cout << ew.str << endl;
25 }
26
27 // You are NOT allowed to modify the main function
28 int main() {
29   char word[] = "Tricky";
30   EnglishWord w1, w2, w3;
31   init(w3, word);
32   w1 = w3;
33   remove(w3);
34   remove(w2);
35   remove(w1);
36 }
```

The program above will compile but it runs with 3 run-time errors during the remove function calls for the 3 `EnglishWord` objects, `w3`, `w2` and `w1` in that order.
Some remarks:

- *You are NOT allowed to change the meaning of any given global function.*
- *You are NOT allowed to modify the main function nor the remove function in your answers to this question.*
- *You don't get ANY marks if you only give the line numbers without correction for parts (a) and (b), or explanation for part (c).*

(a) [3 points] Identify the statement, by giving its line number in the program, that causes the run-time error when `w3` is destructed. Re-write the concerned statement to eliminate this error.

Answer:
Line #13. When `w3` is destructed, the remove function tries to delete the memory for `w3.str` which is pointing to a non-dynamic array on the stack.
Fix: rewrite the `init` function to dynamically allocate memory and copy the content from `s` to `ew.str`.

```
// 1 point for giving correct line number
// 2 points for demonstrating how to eliminate the error
```

(b) [3 points] However, even after you fix the error in part (a), it still will run into another run-time error when `w2` is destructed. Again, identify the statement, by giving its line number, that causes the error, and then re-write the concerned statement to eliminate this 2nd error.

Answer:
Line #24. Word `w2` is created and its `str` is a `nullptr`. When `w2` is destructed, `print(EnglishWord& ew)` tries to print a `nullptr` and gives a run-time error.
Fix: modify the print function, check `str` and print it only if it is not a `nullptr`.

```
// 1 point for giving correct line number
// 2 points for demonstrating how to eliminate the error
```

(c) [2 points] To your dismay, even after you fix the 2 errors in part (a) and (b), your program still will run into the 3rd run-time error when `w1` is destructed. Identify, for the last time, the statement, by giving its line number, that causes this last error. This time, you only need to explain how the error is produced and you don't need to fix it.

Answer:
Line #32. With the assignment `w1 = w3`, the remove function call of `w1` will delete `w1.str` again which has been returned to the heap during `w3`'s call of remove function.
Fix (not required): rewrite the `EnglishWord::operator=` function to do deep copy.

```
// 1 point for giving correct line number
// 1 point for explaining how the error is produced
```

Here is the complete program after fixing the 3 errors.

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

struct EnglishWord {
  char* str = nullptr; // You are NOT allowed to modify this line

  const EnglishWord& operator=(const EnglishWord& ew) { // Part (c)
    if(this != &ew) {
      delete [] str;
      str = new char[strlen(ew.str)+1]; strcpy(str, ew.str);
    }
    return *this;
  }
};

void print(EnglishWord& ew);

// Initialize the EnglishWord ew with the given C string s
void init(EnglishWord& ew, char* s) {
  ew.str = new char[strlen(s)+1]; strcpy(ew.str, s); // Part (a)
  print(ew);
}

// You are NOT allowed to modify this function
void remove(EnglishWord& ew) {
  print(ew);
  delete [] ew.str;
}

void print(EnglishWord& ew) {
  cout << (ew.str ? ew.str : "") << endl; // Part (b)
}

// You are NOT allowed to modify the main function
int main() {
  char word[] = "Tricky";
  EnglishWord w1, w2, w3;
  init(w3, word);
  w1 = w3;
  remove(w3);
  remove(w2);
  remove(w1);
}
```

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.

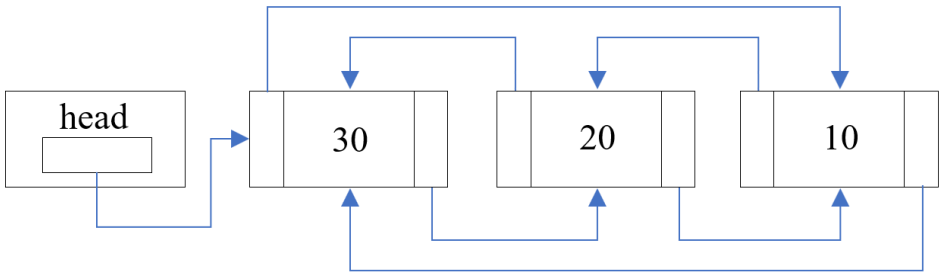
```
/* - int data[10][10]: A 10x10 2D array with certain number of 1s.
   - int x, int y: Starting point (x, y).
   - int v: The value to be processed in the current round.
   - int mark[10][10]: A 10x10 2D array that records the processed locations
                       and values so far.

   Note: You can assume the 2D array, data, is always valid, only contains
   1s and 0s, and there will only be one distinct group of connected 1s.
*/

void spreadout(int data[10][10], int x, int y, int v, int mark[10][10]) {
  if(x < 0 || x >= 10) return;
  if(y < 0 || y >= 10) return;

  // WRITE YOUR CODE HERE









}
```

Answer:

```
void spreadout(int data[10][10], int x, int y, int v, int mark[10][10]) {
  // WRITE YOUR CODE HERE

  if(x < 0 || x >= 10) return;
  if(y < 0 || y >= 10) return;

  if(data[x][y] == 0)                          // 1 point
    return;                                    // 0.5 point
  if(mark[x][y] != 0 && mark[x][y] <= v)       // 3 points
    return;                                    // 0.5 point
  data[x][y] = v;                              // 1 point
  mark[x][y] = v;                              // 1 point

  spreadout(data, x + 1, y, v + 1, mark);      // 1 point
  spreadout(data, x, y + 1, v + 1, mark);      // 1 point
  spreadout(data, x - 1, y, v + 1, mark);      // 1 point
  spreadout(data, x, y - 1, v + 1, mark);      // 1 point
}

// Remark: If using mark as bool array, but otherwise correct, flat -3 penalty.
```

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
```

(a) [4 points] Implement `CDLL create(int value)`.

Answer:

```
CDLL create(int value) {
  CDLL cdll;                                          // 0.5 point
  cdll.head = new CDLL_Node;                          // 1 point
  cdll.head->data = value;                            // 1 point
  cdll.head->prev = cdll.head->next = cdll.head;      // 1 point
  return cdll;                                        // 0.5 point
}
```

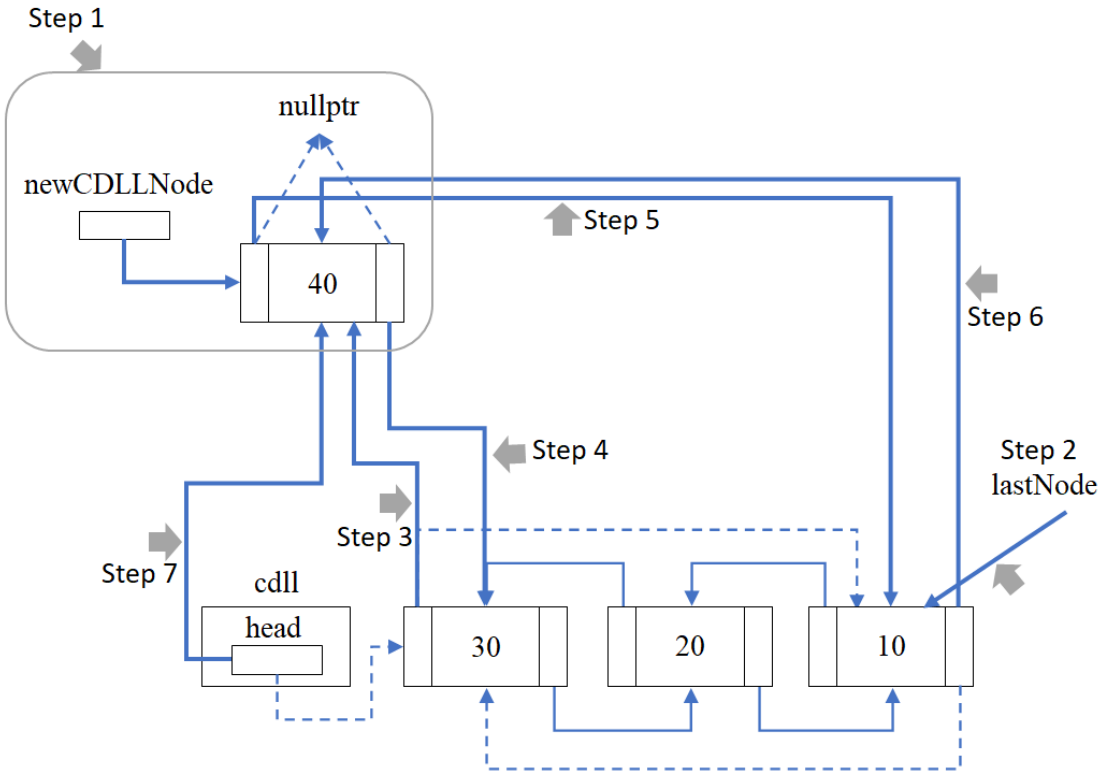
(b) [1 point] Implement `bool isEmpty(const CDLL& cdll)`.

Answer:

```
bool isEmpty(const CDLL& cdll) {
  return (cdll.head == nullptr);                      // 1 point
}
```

(c) [7 points] Implement `void insertAtFront(CDLL& cdll, int value)`.

Hint: You may want to implement the function according to the example given below.



Answer:

```
bool insertAtFront(CDLL& cdll, int value) {
    if (isEmpty(cdll)) {                    // 1 point
        cdll = create(value);
        return;
    }

    // 3 points (1 point for each member)
    CDLL_Node* newCDLLNode = new CDLL_Node{value, cdll->head->prev, cdll.head};
    cdll.head->prev->next = newCDLLNode;    // 1 point
    cdll.head->prev = newCDLLNode;          // 1 point
    cdll.head = newCDLLNode;                // 1 point
}
```

(d) [12 points] Implement `int removeFromBack(CDLL& cdll)`.

Remark: Make sure to handle all the special cases.

Answer:

```
// Concise combination of all cases.
int removeFromBack(CDLL& cdll) {
    if (isEmpty(cdll)) return -999;          // 1 point

    CDLL_Node* last = cdll.head->prev;       // 2 points
    last->prev->next = cdll.head;            // 1 point
    cdll.head->prev = last->prev;            // 1 point
    int value = last->data;                  // 2 points
    if (last == cdll.head)                   // 1 point
        cdll.head = nullptr;                 // 1 point
    delete last;                             // 2 points
    return value;                            // 1 point
}

// Split version.
int removeFromBack(CDLL& cdll) {
    // Special Case: If the CDLL is empty
    if (isEmpty(cdll)) return -999;          // 1 point

    // Special Case: Only one CDLL_Node
    if (cdll.head == cdll.head->next) {      // 1 point
        int value = cdll.head->data;         // 1 point
        delete cdll.head;                    // 1 point
        cdll.head = nullptr;                 // 1 point
        return value;                        // 0.5 point
    }

    // General Case: More than one CDLL_Node
    CDLL_Node* last = cdll.head->prev;       // 2 points
    last->prev->next = cdll.head;            // 1 point
    cdll.head->prev = last->prev;            // 1 point
    int value = last->data;                  // 1 point
    delete last;                             // 1 point
    return value;                            // 0.5 point
}
```

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:

```
#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
```

(a) [3 points] Implement the `Dictionary` structure.

Answer:

```
struct Dictionary {                                    // 1 point
  string** wordList;                                   // 1 point
  int len;                                             // 1 point
};
```

(b) [2 points] Implement `void init(Dictionary& dict)`.

Answer:

```
void init(Dictionary& dict) {
  dict.wordList = nullptr;                             // 1 point
  dict.len = 0;                                        // 1 point
}
```

(c) [3 points] Implement `int findWord(const Dictionary& dict, const string& word)`.

Hint: Two string objects can be compared using equality operator (i.e. ==). It returns true if the two strings are the same. Otherwise, it returns false.

Answer:

```
int findWord(const Dictionary& dict, const string& word) {
  for(int i=0; i<dict.len; i++) {                      // 1 point
    if(*(dict.wordList[i]) == word)                    // 1 point
      return i;                                        // 0.5 point
  }
  return -1;                                           // 0.5 point
}
```

(d) [12 points] Implement `bool insertWord(Dictionary& dict, const string& word)`.

Note: You can use the relational operators, <, <=, ==, >=, > for lexicographic string comparison.

Answer:

```
bool insertWord(Dictionary& dict, const string& word) {
    if (findWord(dict, word) != -1) {                          // 1 point
        return false;                                          // 0.5 point
    }

    string** newWordList = new string*[dict.len + 1];          // 1 point

    int index;
    for (index = 0; index < dict.len; index++) {               // 1 point
        if (*(dict.wordList[index]) > word) {                  // 1 point
            break;
        }
        newWordList[index] = dict.wordList[index];             // 1 point
    }

    newWordList[index] = new string(word);                     // 1 point

    for (; index < dict.len; index++) {                        // 1 point
        newWordList[index + 1] = dict.wordList[index];         // 1 point
    }

    delete[] dict.wordList;                                    // 1 point
    dict.wordList = newWordList;                               // 1 point
    dict.len++;                                                // 1 point

    return true;                                               // 0.5 point
}
```

(e) [10 points] Implement `bool removeWord(Dictionary& dict, const string& word)`.

Answer:

```
bool removeWord(Dictionary& dict, const string& word) {
    int index = findWord(dict, word);                          // 1 point
    if (index == -1)                                           // 1 point
        return false;                                          // 0.5 point

    string** newWordList = new string*[dict.len - 1];          // 1 point
    for (int i = 0; i < index; i++)                            // 0.5 point
        newWordList[i] = dict.wordList[i];                     // 0.5 point

    delete dict.wordList[index];                               // 1 point

    for (int i = index+1; i < dict.len; i++)                   // 0.5 point
        newWordList[i-1] = dict.wordList[i];                   // 0.5 point

    delete[] dict.wordList;                                    // 1 point
    dict.wordList = newWordList;                               // 1 point
    dict.len--;                                                // 1 point
    return true;                                               // 0.5 point
}
```

(f) [2 points] Implement `void displayDict(const Dictionary& dict)`.

Answer:

```
void displayDict(const Dictionary& dict) {
  for(int i=0; i<dict.len; i++)                          // 1 point
    cout << *(dict.wordList[i]) << endl;                 // 1 point
}
```

(g) [3 points] Implement `void destroy(Dictionary& dict)`.

Answer:

```
void destroy(Dictionary& dict) {
  for(int i=0; i<dict.len; i++)                          // 1 point
    delete dict.wordList[i];                             // 1 point
  delete [] dict.wordList;                               // 1 point
}
```

---------- END OF PAPER ----------

/* Rough work */

/* Rough work */

/* Rough work */

/* Rough work */