COMP 2012H Midterm Exam - Fall 2019 - HKUST

Date:	October 19, 2019 (Saturday)
Time Allowed:	2 hours, 10:00–12:00nn

Instructions:

1. This is a closed-book, closed-notes examination.
2. There are <u>**7**</u> questions on **27** pages (including this cover page and 2 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 <u>**NOT**</u> allowed to define additional structures, classes, helper functions and use global variables, **auto**, nor any library functions not mentioned in the questions.

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

For T.A. Use Only

Problem	**Topic**	**Score**
1	True / False Questions	/ 10
2	Operators	/ 22
3	Function Parameter Passing	/ 10
4	Arrays and Loops	/ 10
5	Structure and Objects	/ 17
6	Recursion	/ 13
7	Circular Doubly Linked List	/ 18
	Total	/ 100

Problem 1 [10 points] True / False Questions

Indicate whether the following statements are true or false buy 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 is the ONLY possible main function prototpye of a C++ program according to C++11.

```
int main();
```

T [✓ **F**] (b) Defining an int variable without assigning an initial value contains garbage value no matter where it is defined.

[✓ **T**] **F** (c) The following boolean expression evaluates to false.

```
17 % 5 == -17 % 5
```

[✓ **T**] **F** (d) The following program has NO compilation error.

```
int main() {
  const int& c = 10;
  int cc = c;
}
```

T [✓ **F**] (e) Given the following program:

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

enum Prize { GOLD, SILVER, BRONZE = 0 };

int main() {
  cout << "GOLD: " << GOLD << endl;
  cout << "SILVER: " << SILVER << endl;
  cout << "BRONZE: " << BRONZE << endl;
  return 0;
}
```

The output of the program is

```
GOLD: -2
SILVER: -1
BRONZE: 0
```

T [✓ F] (f) The following program will NOT cause compilation error NOR runtime error.

```
int& func() {
  int x = 20;
  return x;
}
int main() {
  int& x = func();
  x = 30;
}
```

[✓ T] F (g) The following code can be compiled WITHOUT errors.

```
struct Point {
  double x, y;
} p1, p2;

int main() {
  p1 = p2;
  return 0;
}
```

[✓ T] F (h) The following program can be compiled successfully.

```
int main() {
  const int* const p = new int const;
  delete p;
  return 0;
}
```

T [✓ F] (i) Assume an array is defined as follows:

```
int* arr = new int[3];
```

The following statements all print the same value on screen.

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

[✓ T] F (j) The following program has NO compilation error but WITH runtime error.

```
int main() {
  int arr[] = { 1, 2, 3, 4, 5 };
  int* p = arr;
  delete [] p;
}
```

Problem 2 [22 points] Operators

C++ has `+=` and `=` operator for integers. Let's implement these 2 operators by 2 separate function `plus_equal` and `assign` for `Point` so that the following program

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

struct Point {
  int x, y;
};

void print(const Point& p, const Point& q, const Point& r) {
  cout << "a = (" << p.x << ", " << p.y << "), "
       << "b = (" << q.x << ", " << q.y << "), "
       << "c = (" << r.x << ", " << r.y << ")" << endl;
}

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

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

int main() {
  Point a = { 10, 20 }, b = { 20, 30 }, c = { 30, 40 };

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

  // Your plus_equal function should cause compilation error
  // for the following statement, if it is uncommented.
  // plus_equal(plus_equal(a, b), c);

  // -------------------------------------------------------------------------

  // Equivalent to a = b = c;
  assign(a, assign(b, c));
  print(a, b, c);

  // Your assign function should cause compilation error
  // for the following statement, if it is uncommented.
  // assign(assign(a, b), c);

  // Your assign function shouldn't support self-assignment, i.e. it does not
  // perform assignment of data members for object a for the following.
  assign(a, a);
}
```

will give the output below:

```
a = (60, 90), b = (50, 70), c = (30, 40)
a = (30, 40), b = (30, 40), c = (30, 40)
```

Remarks:

i. You are not allowed to use the C++'s built-in struct-struct assignment operator in your answer. But you are allowed to use assignment operators for primitive data types.

ii. Both functions will take two `Point` type variables as their arguments.

iii. You have to decide the exact function header of the 2 functions yourselves.

iv. You should make sure that your functions are implemented according to the requirements as stated in the comment lines of the program. Also, you have to mark those variables that will not be changed with const keyword.

Answer:

(a) [10 points] Implement the function `plus_equal` in the space provided.

(b) [12 points] Implement the function `assign` in the space provided.

Solution:

```
const Point& plus_equal(Point& p1, const Point& p2) {
  p1.x += p2.x;
  p1.y += p2.y;
  return p1;
}

// 3 points for return type
// 2 points for the 1st parameter, 3 points for the 2nd parameter (5 points in total)
// 1 point for 2 increment statements
// 1 point for returning p1

const Point& assign(Point& p1, const Point& p2) {
  if(&p1 != &p2) {
    p1.x = p2.x;
    p1.y = p2.y;
  }
  return p1;
}

// 3 points for return type
// 2 points for the 1st parameter, 3 points for the 2nd parameter (5 points in total)
// 2 points for self-assignment check
// 0.5 point for each data member assignment (1 point in total)
// 1 point for returning p1
```

Problem 3 [10 points] Function Parameter Passing

Fill in the blanks with appropriate parameter-passing mechanism, so that the output of the following program is as follows.

```
10 20 30
5000 6000 4000
```

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

void mysterious(_____________ a, _____________ b, _____________ c) {
  int temp = a;
  a = b;
  b = *c;
  delete c;
  c = new int; // Assume c stores address 4000, 5000, 6000 when executing
               // mysterious the 1st time, 2nd time, 3rd time respectively.
  *c = temp;
}

int main() {
  int* a = new int; // Assume a stores address 1000
  *a = 10;
  int* b = new int; // Assume b stores address 2000
  *b = 20;
  int* c = new int; // Assume c stores address 3000
  *c = 30;

  mysterious(*a, *b, c);
  mysterious(*b, *c, a);
  mysterious(*c, *a, b);

  cout << *a << " " << *b << " " << *c << endl;
  cout << a << " " << b << " " << c << endl;

  delete a;
  delete b;
  delete c;

  return 0;
}
```

Solution:

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

void mysterious(______int&______ a, ______int&______ b, ______int*&______ c) {
  int temp = a;
  a = b;
  b = *c;
  delete c;
  c = new int; // Assume c stores address 4000, 5000, 6000 when executing
               // mysterious the 1st time, 2nd time, 3rd time respectively.
  *c = temp;
}

int main() {
  int* a = new int; // Assume a stores address 1000
  *a = 10;
  int* b = new int; // Assume b stores address 2000
  *b = 20;
  int* c = new int; // Assume c stores address 3000
  *c = 30;

  mysterious(*a, *b, c);
  mysterious(*b, *c, a);
  mysterious(*c, *a, b);

  cout << *a << " " << *b << " " << *c << endl;
  cout << a << " " << b << " " << c << endl;

  delete a;
  delete b;
  delete c;

  return 0;
}
```

Marking Scheme

- 3 points for the type of the 1st parameter
- 3 points for the type of the 2nd parameter
- 4 points for the type of the 3rd paramerer

Remark: 3 points for putting int, int for the 1st and 2nd parameter.

Problem 4 [10 points] Array and Loops

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

int mysterious1(int a[], int n) {
  int m1 = 0, m2 = 0;

  for(int i = 0; i < n; ++i) {
    m2 += a[i];
    if(m2 < 0) {
      m2 = 0;
    }
    if(m1 < m2) {
      m1 = m2;
    }
  }

  return m1;
}

int mysterious2(int a[], int n) {
  int m3 = mysterious1(a, n);
  int m4 = 0;
  int i = 0;

  while(i < n) {
    m4 += a[i];
    a[i] = -a[i];
    ++i;
  }

  m4 += mysterious1(a, n);

  return (m4 > m3) ? m4 : m3;
}

int main() {
  int a[] = { 20, 5, -20, 25, -13, -1, 10, -28, 17 };
  int n = sizeof(a) / sizeof(int);
  cout << mysterious2(a, n) << endl;
  return 0;
}
```

(a) [7 points] What is the output of the above program?

Answer:

47

Marking scheme:

- If the output is 47, 7 points are given.
- If the output is 15, 1 point is given.
- If the output is 62 or 77, 2 points are given.
- If the output is 25 or 28, 3 points are given.
- If the output is 30, 4 points are given.
- If the output is 42, 5 points are given.

(b) [3 points] Briefly describe what it does.

Answer:

It finds the maximum contiguous sum of a given circular array.

Marking scheme:

- If both contiguous and circular are mentioned, 3 points are given.
- If only contiguous is mentioned, 1 point is given.
- If only circular is mentioned, 2 points are given.

Problem 5 [17 points] Structure and Object

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

// You are NOT allowed to modify this struct
struct MyArray {
  int* p = nullptr;
  int size = 0;
};

// You are NOT allowed to modify this function
void init(MyArray& arr, int size) {
  arr.p = new int[size];
  arr.size = size;
}

MyArray& clone(const MyArray& arr) {
  MyArray* pArr = new MyArray;
  pArr->p = arr.p;
  pArr->size = arr.size;
  return *pArr;
}

// You are NOT allowed to modify this function
void print(const MyArray& arr) {
  for(int i=0; i<arr.size; ++i) {
    cout << arr.p[i] << " ";
  }
  cout << endl;
}

// You are NOT allowed to modify this function
void remove(MyArray& arr) {
  print(arr);
  delete [] arr.p;
}

int main() {
  MyArray myArr;                 // --
  init(myArr, 3);                //   |
  myArr.p[0] = 1;                //   | You are not allowed
  myArr.p[1] = 2;                //   | to modify all these
  myArr.p[2] = 3;                //   | lines.
  MyArray& copy = clone(myArr); //   |
  remove(myArr);                 //   |
  remove(copy);                  // --

  return 0;
}
```

The program above will compile but it runs with **runtime error(s)** and **memory leak**. Identify the statement(s) by giving the line number(s) in the program that causes the run-time error(s) and memory leak. Propose how to eliminate the error(s).

Note:

- You are NOT allowed to set `copy.arr` to `nullptr` in `main`.
- Also, you are NOT allowed to modify all the functions, except `clone` and `main`. Also, you are NOT allowed to modify line 38 to line 45.

Answer:

1. Line 34: Double de-allocation of dynamically allocated memory. To fix the problem, the clone function needs to perform deep copy as follows:

   ```
   MyArray& clone(const MyArray& arr) {
     MyArray* pArr = new MyArray;
     pArr->p = new int[arr.size];       // 1 point

     for(int i=0; i<arr.size; ++i) {    // 2 points
       pArr->p[i] = arr.p[i];
     }

     pArr->size = arr.size;
     return *pArr;
   }
   ```

   Marking scheme:

   - 2 points for the line number.
   - 3 points for explaining what's the problem.
   - 4 points for proposing how to fix the problem.

2. Line 43: Dynamic memory reference by `copy` needs to be de-allocated. To do so, insert the following line in Line 46.

   ```
   delete &copy;
   ```

   Marking scheme:

   - 2 points for the line number.
   - 3 points for explaining what's the problem.
   - 3 points for proposing how to fix the problem.

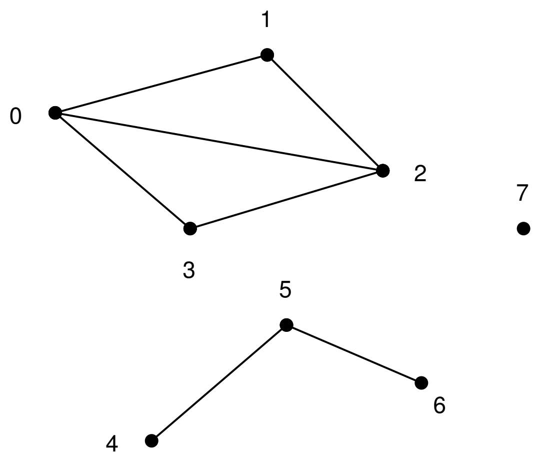
Remarks:

- Stating runtime error happens at line 26 $\rightarrow$ okay.
- Do not explain the problem but showing how to make modification $\rightarrow$ full mark.
- `delete copy` instead of `delete &copy`, -1 point.

Problem 6 [13 points] Recursion

Given a graph with `N` vertices and a list of edges that connecting the vertices, we want to find the number of connected components.

Example: Suppose there are 8 vertices and a set of edges connecting these 8 vertices as follows.



The set of edges is specified by the following adjacency matrix `bool adj[8][8]`.

```
  | 0 1 2 3 4 5 6 7
--+----------------
0 | 0 1 1 1 0 0 0 0
1 | 1 0 1 0 0 0 0 0
2 | 1 1 0 1 0 0 0 0
3 | 1 0 1 0 0 0 0 0
4 | 0 0 0 0 0 1 0 0
5 | 0 0 0 0 1 0 1 0
6 | 0 0 0 0 0 1 0 0
7 | 0 0 0 0 0 0 0 0
```

where `adj[i][j]` is true if and only if there is an edge connecting `i` and `j`.

To find the number of connected components in a given graph, we first mark the first vertex as visited. Find all the vertices connected to the first one and also mark them as visited. Then we check if there are still unmarked vertices. If yes, that means we find a new connected component. Similarly, we mark this vertex and all the vertices connected to it. Repeat this until all the vertices are marked.

A skeleton code of the program, which finds the number of connected components in a given graph of `n` vertices, is given below.

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

void mark(int v, int n, bool visit[], bool adj[][100]) {
    // TODO: Part (a)
}

int main() {
    int n = 8;

    // All the other variables that are not assigned with initial values are 0.
    bool adj[100][100] = {
        {0, 1, 1, 1, 0, 0, 0, 0},
        {1, 0, 1, 0, 0, 0, 0, 0},
        {1, 1, 0, 1, 0, 0, 0, 0},
        {1, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 1, 0, 0},
        {0, 0, 0, 0, 1, 0, 1, 0},
        {0, 0, 0, 0, 0, 1, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0}
    };

    bool visit[100] = {0}; // Initialize all variables to 0
    int cnt = 0;

    // TODO: Part (b)

    cout << "The number of connected components is " << cnt << endl;
    return 0;
}
```

When the program runs, it produces the following out.

```
The number of connected components is 3
```

(a) [7 points] Implement `mark` as a recursive function that marks all the vertices connected to `v` by completing the missing part in `TODO: Part (a)`. Use `n` instead of hard coding 8 in your code.

Answer:

```
void mark(int v, int n, bool visit[], bool adj[][100]) {
    visit[v] = true;                        // 1.5 points
    for (int i=0; i<n; ++i) {               // 1.5 points
        if (adj[v][i] && !visit[i]) {       // 2.5 points
            mark(i, n, visit, adj);         // 1.5 point
        }
    }
}
```

(b) [6 points] Complete the missing part in `TODO: Part (b)` of `main()` so that it finds the number of the connected components specified by `adj` and stores the number in `cnt`. Use `n` instead of hard coding 8 in your code.

Answer:

```
for (int i=0; i<n; ++i) {                    // 1.5 points
    if (!visit[i]) {                         // 1.5 points
        ++cnt;                               // 1.5 points
        mark(i, n, visit, adj);              // 1.5 points
    }
}
```

Problem 7 [18 points] **Circular Doubly Linked List**

This question is about an implementation of a Circular Doubly Linked List, named CDLL.
The following shows the header file of the CDLL program.

```
struct CDLL_Node {
    int data;
    CDLL_Node* prev;
    CDLL_Node* next;
};

// create() and destroy() are like class Constructor and Destructor.
// CDLL will never be used before create() nor after destroy().
struct CDLL {
    CDLL_Node* sentinel = nullptr;
};

// Create and initialize an empty CDLL, by setting sentinel to circular
// doubly point to itself.
CDLL create();

// Destroy and deallocate ALL nodes in the CDLL, INCLUDING the sentinel.
void destroy(CDLL& cdll);

// Print the contents of each node in the CDLL, from front to back.
void print_forward(const CDLL& cdll);

// Print the contents of each node in the CDLL, from back to front.
void print_reverse(const CDLL& cdll);

// CDLL is empty if there are no other nodes with valid data except for the sentinel.
bool is_empty(const CDLL& cdll);

// Return the node at the specified index.
// Index 0 refers to the first valid node, NOT the sentinel.
// Return sentinel if index is out-of-range.
CDLL_Node* get_node_at_index(const CDLL& cdll, unsigned int index);

// Create and insert a node with the specified data at the specified index.
// Index 0 means the inserted node will become the first valid node, NOT the sentinel.
// Index 1 means the inserted node will become the second valid node, etc.
// Index out-of-range, do nothing. But insert at the end is still OK.
void insert(CDLL& cdll, unsigned int index, int data);

// Remove and destory the node at the specified index.
// Index 0 refers to the first valid node, NOT the sentinel.
// DON'T destroy the sentinel.
// Index out-of-range, do nothing.
void remove(CDLL& cdll, unsigned int index);
```

Your task is to implement all the functions as specified in 'CDLL.h' such that your solution works with the testing program "test-cdll.cpp" and produces the following outputs.

Note: Assume you implement all the functions in "CDLL.cpp".

```
Create empty CDLL:
Is CDLL empty? true

Inserting 10, 20, 30, in numerical order:
Current CDLL:
Forward: 10 20 30
Reverse: 30 20 10

Is CDLL empty? false

Inserting 5, 15, 25, 35, in numerical order:
Current CDLL:
Forward: 5 10 15 20 25 30 35
Reverse: 35 30 25 20 15 10 5

Removing 35, 5, 20, in that order:
Current CDLL:
Forward: 10 15 25 30
Reverse: 30 25 15 10

CDLL index 0: 10
CDLL index 1: 15
CDLL index 2: 25
CDLL index 3: 30
CDLL index out-of-range == sentinel? true

Destroy CDLL:
Is CDLL destroyed? true
```

```cpp
#include "CDLL.h"

#include <iostream>
using namespace std;

void print_cdll_test(const CDLL& cdll) {
    cout << "Current CDLL: " << endl;
    cout << "Forward: ";
    print_forward(cdll);
    cout << endl;
    cout << "Reverse: ";
    print_reverse(cdll);
    cout << endl << endl;
}

int main() {
    cout << "Create empty CDLL:" << endl;
    CDLL cdll = create();
    cout << "Is CDLL empty? " << boolalpha << is_empty(cdll) << endl << endl;

    cout << "Inserting 10, 20, 30, in numerical order:" << endl;
    insert(cdll, 0, 10);
    insert(cdll, 1, 20);
    insert(cdll, 2, 30);
    print_cdll_test(cdll);
    cout << "Is CDLL empty? " << is_empty(cdll) << endl << endl;

    cout << "Inserting 5, 15, 25, 35, in numerical order:" << endl;
    insert(cdll, 0, 5);
    insert(cdll, 2, 15);
    insert(cdll, 4, 25);
    insert(cdll, 6, 35);
    print_cdll_test(cdll);

    cout << "Removing 35, 5, 20, in that order:" << endl;
    remove(cdll, 6);
    remove(cdll, 0);
    remove(cdll, 2);
    print_cdll_test(cdll);

    for (unsigned int index = 0; index < 4; ++index) {
        cout << "CDLL index " << index << ": "
             << get_node_at_index(cdll, index)->data << endl;
    }
    cout << "CDLL index out-of-range == sentinel? "
         << (get_node_at_index(cdll, 99) == cdll.sentinel) << endl << endl;

    cout << "Destroy CDLL:" << endl;
    destroy(cdll);
    cout << "Is CDLL destroyed? " << (cdll.sentinel == nullptr) << endl;

    return 0;
}
```

Answer: `/* File: CDLL.cpp */`

Answer: `/* File: CDLL.cpp */`

Answer: `/* File: CDLL.cpp */`

```cpp
#include "CDLL.h"

#include <iostream>
using std::cout;
using std::endl;

// 2.5 point
CDLL create() {
    CDLL cdll;                                              // 0.5 point
    cdll.sentinel = new CDLL_Node;                          // 0.5 point
    cdll.sentinel->prev = cdll.sentinel->next = cdll.sentinel;  // 1 point
    return cdll;                                            // 0.5 point

    // CDLL cdll;                                           // 0.5 point
    // cdll.sentinel = new CDLL_Node;                       // 0.5 point
    // cdll.sentinel->prev = cdll.sentinel;                 // 0.5 point
    // cdll.sentinel->next = cdll.sentinel;                 // 0.5 point
    // return cdll;                                         // 0.5 point

    // Minus 0.5 point if doing such a one-liner,
    // as sentinel hasn't properly been initialized yet,
    // and can't point to itself correctly.
    // CDLL cdll = {new CDLL_Node{cdll.sentinel, cdll.sentinel}};
}

// 3 point
void destroy(CDLL& cdll) {
    // 1 point for correct looping to get all nodes.
    // 1 point for correct delete call on all nodes.
    for (CDLL_Node* current = cdll.sentinel->next;
         current != cdll.sentinel;
         current = current->next, delete current->prev) {}
    delete cdll.sentinel;                                   // 0.5 point
    cdll.sentinel = nullptr;                                // 0.5 point

    // Minus 0.5 point for this implementation due to inefficiency.
    // We don't care about correct pointer links anymore when destroying cdll.
    // while (!is_empty(cdll)) {                            // 1 point
    //     remove(cdll, 0);                                   // 0.5 point
    // }
    // delete cdll.sentinel;                                // 0.5 point
    // cdll.sentinel = nullptr;                             // 0.5 point

    // Minus 1 point for forgetting to utilize the sentinel,
    // and having multiple redundant if-else checking,
    // e.g. only 1 valid node, or only sentinel remaining.
}
```

```
// 1 point
void print_forward(const CDLL& cdll) {
    for (CDLL_Node* current = cdll.sentinel->next;
         current != cdll.sentinel;
         current = current->next) {                        // 0.5 point
        cout << current->data << " ";                      // 0.5 point
    }
}

// 1 point
void print_reverse(const CDLL& cdll) {
    for (CDLL_Node* current = cdll.sentinel->prev;
         current != cdll.sentinel;
         current = current->prev) {                        // 0.5 point
        cout << current->data << " ";                      // 0.5 point
    }
}

// 1 point
bool is_empty(const CDLL& cdll) {
    return (cdll.sentinel->next == cdll.sentinel);         // 1 point
}

// 2.5 point
CDLL_Node* get_node_at_index(const CDLL& cdll, unsigned int index) {
    CDLL_Node* current = cdll.sentinel->next;              // 0.5 point
    for (unsigned int i = 0;                               // 0.5 point
         (current != cdll.sentinel) && (i < index);        // 0.5 point
         current = current->next, ++i) {}                  // 0.5 point
    return current;                                        // 0.5 point

    // CDLL_Node* current = cdll.sentinel->next;           // 0.5 point
    // unsigned int i = 0;                                 // 0.5 point
    // while ((current != cdll.sentinel) &&                // 0.25 point
    //        (i < index)) {                               // 0.25 point
    //   current = current->next;                          // 0.25 point
    //   i += 1;                                           // 0.25 point
    // }
    // return current;                                     // 0.5 point

    // Minus 0.5 point if not returning sentinel correctly
    // when index is out-of-range.

    // Minus 1 point if forget to utilize circular doubly linked and sentinel,
    // and having redundant if-else checking, e.g. is_empty(),
    // counting valid nodes to check if index out-of-range, etc.
}
```

```
// 4 point
void insert(CDLL& cdll, unsigned int index, int data) {
    CDLL_Node* current = get_node_at_index(cdll, index);        // 1 point
    if ((index != 0) && (current == cdll.sentinel)) { return; }

    // 0.5 point for new keyword; 0.5 point each for data, prev, and next.
    CDLL_Node* insert_node = new CDLL_Node{data, current->prev, current};
    insert_node->prev->next = insert_node;                      // 0.5 point
    insert_node->next->prev = insert_node;                      // 0.5 point

    // CDLL_Node* current = get_node_at_index(cdll, index);     // 1 point
    // CDLL_Node* insert_node = new CDLL_Node;                  // 0.5 point
    // insert_node->data = data;                                // 0.5 point
    // insert_node->prev = current->prev;                       // 0.5 point
    // insert_node->next = current;                             // 0.5 point
    // insert_node->prev->next = insert_node;                   // 0.5 point
    // insert_node->next->prev = insert_node;                   // 0.5 point

    // Minus 1 point if forget to utilize circular doubly linked and sentinel,
    // and have redundant if-else checking, e.g. inserting first node, etc.
}

// 3 point
void remove(CDLL& cdll, unsigned int index) {
    CDLL_Node* current = get_node_at_index(cdll, index);        // 1 point
    if (current == cdll.sentinel) { return; }                   // 0.5 point

    current->prev->next = current->next;                        // 0.5 point
    current->next->prev = current->prev;                        // 0.5 point
    delete current;                                             // 0.5 point

    // Minus 1 point if forget to utilize circular doubly linked and sentinel,
    // and having redundant if-else checking, e.g. deleting last node, etc.
}

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

/* Rough work */

/* Rough work */