CS101 Algorithms and Data Structures
Fall 2023
Homework 1

Panxin Tao
ID: 2022533112

Due date: October 15, 2023, at 23:59

1. Please write your solutions in English.
2. Submit your solutions to gradescope.com.
3. Set your FULL name to your Chinese name and your STUDENT ID correctly in Account Settings.
4. If you want to submit a handwritten version, scan it clearly. `CamScanner` is recommended.
5. When submitting, match your solutions to the problems correctly.
6. No late submission will be accepted.
7. Violations to any of the above may result in zero points.

1. (7 points) Academic Integrity

Please determine who violated academic integrity or committed plagiarism in the following situations. Tick ($\surd$) the correct answers.

(a) (1') Alice posted one of the homework questions on Stack Overflow and copied the answer from others to her homework with some trivial modifications.

$\surd$ **Alice** ◯ Bob ◯ Both Alice and Bob ◯ Neither Alice nor Bob

(b) (1') Alice entered the question description into ChatGPT, then slightly modified ChatGPT's response and submitted it as her own answer.

$\surd$ **Alice** ◯ Bob ◯ Both Alice and Bob ◯ Neither Alice nor Bob

(c) (1') To boost efficiency and save precious time, Bob used Copilot as an assistant to write code in his programming assignments.

◯ Alice $\surd$ **Bob** ◯ Both Alice and Bob ◯ Neither Alice nor Bob

(d) (1') Bob lent Alice his laptop to play Genshin. Alice found Bob's solution to Homework 1 on the desktop and copied it using USB drive sneakily.

◯ Alice ◯ Bob $\surd$ **Both Alice and Bob** ◯ Neither Alice nor Bob

(e) (1') Alice and Bob are good friends. Alice asked if Bob could send her some code snippets so that she could have some ideas of what is going on in this programming assignment (promising, of course, not copying). Bob agreed and shared his code repository with Alice. Alice copied Bob's code, changed some variable names, and submitted the code to CS101 Online Judge.

◯ Alice ◯ Bob $\surd$ **Both Alice and Bob** ◯ Neither Alice nor Bob

(f) (1') Alice completed the programming assignment herself on Bob's computer and accidentally submitted her code to CS101 Online Judge using Bob's account. After that, Alice and Bob resubmitted their assignments using their own accounts respectively.

◯ Alice ◯ Bob $\surd$ **Both Alice and Bob** ◯ Neither Alice nor Bob

(g) (1') Bob is Alice's boyfriend. In order to enhance their romantic relationship, Alice and Bob studied together in ShanghaiTech Library and collaborated on one homework question. They discussed about some algorithm using a whiteboard, without giving any exact solution.

◯ Alice ◯ Bob ◯ Both Alice and Bob $\surd$ **Neither Alice nor Bob**

2. (8 points) Multiple Choices

Each question has **one or more** correct answer(s). Select all the correct answer(s). For each question, you will get 0 points if you select one or more wrong answers, but you will get 1 point if you select a non-empty subset of the correct answers.

Write your answers in the following table.

(a)	(b)	(c)	(d)

(a) (2') Which of the following operations on a **Linked List** take constant time?

A. Given a pointer h which points to the head node of a linked list, we want to erase the head node.

B. Given a pointer h which points to the head node of a linked list, we want to gain access to the last element of the linked list.

C. Given a pointer p which points to a node in a linked list, we want to gain access the previous node of p.

D. Given a pointer p which points to a node in a linked list, we want to insert an element after p.

(b) (2') Which of the following statements about arrays and linked-lists are true?

A. Inserting an element into the middle of an array takes constant time.

B. A doubly linked list consumes more memory than a (singly) linked list of the same length.

C. Given a pointer to some node in a doubly linked list, we are able to gain access to every node of it.

D. Given a pointer to any node in a linked list, we are able to gain access to the last node.

(c) (2') Please evaluate the following reverse-Polish expressions. Which of them are legal reverse-Polish expressions and gives a result greater than 0?

A. `2 3 2 * + 1 /`

B. `1 2 4 - - 3 *`

C. `1 * 2 - 1 + 5`

D. `2 4 3 1 + * -`

(d) (2') Assume we implement a queue with a circular array indexed from 0 to $n-1$. Now the `front` pointer is at index a, and the `back` pointer is at index b. Which of the following best describes the number of the elements in the queue?

A. $b-a+1$

B. $|b-a+1|$

C. $b-a+1+n$

D. $(b-a+1+n) \mod n$

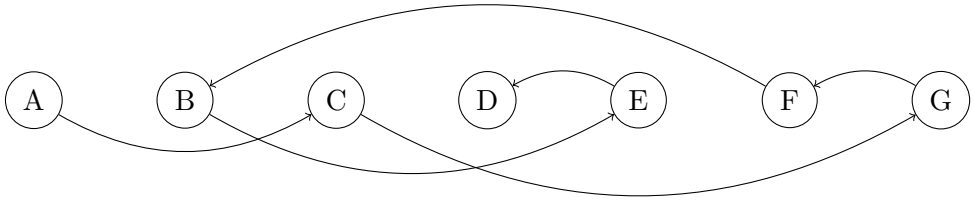
3. (10 points) Array Representation of Linked-List

Recall that we stored a linked-list in an array to avoid memory allocation problems in the lecture. In this question, we use two arrays: `next` and `value` to implement a singly-linked-list.

The elements at each index in `next` and `value` are together to represent a node in the linked-list, where `next` represents the array index where the next node is located (-1 for already reaching the tail), and `value` represents the data stored in the node.

(a) (5') **Store a Linked-List in an Array**

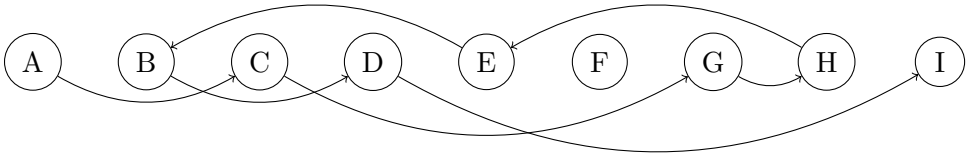
Fill in the table below to finish the array representation of the linked-list shown below. Fill in -1 to represent the end of the linked-list.



index	0	1	2	3	4	5	6
`value`	A	B	C	D	E	F	G
`next`	2	4	6	-1	3	1	5

(b) (5') **From Array to Linked-List**

Here are the arrays `next` and `value`. Please draw the linked-list below represented by these two arrays.



index	0	1	2	3	4	5	6	7	8
`value`	A	B	C	D	E	F	G	H	I
`next`	2	3	6	8	1	0	7	4	-1

4. (10 points) Vocabulary List

Alice is memorizing English words to enlarge her vocabulary in preparation for her approaching CET-4 test and she needs your help! Alice uses a `Node` to define a word and a linked list which contains the words that she is trying to memorize now. The structure of `Node` is as follows.

```
struct Node {
  string word;
  Node *next;
  Node(const string& _word, Node *_next)
      : word(_word), next(_next) {}
};
Node *head;
```

Each time she reviews her vocabulary list, she steps through the linked list starting from `head`. However, sometimes there is a word that she doesn't know, so she wants to grab the corresponding `Node` out and put it at the start of the linked list, so next time she can review it first.

For example, if the vocabulary list is

`algorithm` → `breath` → `capacity` → `degree`

and she doesn't know the word `capacity`, then she wants the list be modified to

`capacity` → `algorithm` → `breath` → `degree`

(a) (6') Alice needs your help to finish her function `move_to_first(Node *head, Node *p)` where `head` represents the first element of her non-empty vocabulary list and she wants to move the **next** word of `p` to the first. The function returns the **new** head pointer after this process. She can guarantee that `p` points to a `Node` in the linked list, but not the last one.

```
Node* move_to_first(Node *head, Node *p) {
    Node *to_move = p->next;
    // Write something below
    // ...
    // Write something above
}
```

Please help her complete the function.

Solution:

```
Node* move_to_first(Node *head, Node *p) {
    Node *to_move = p->next;
    p->next = to_move->next;
    to_move->next = head;
    return to_move;
}
```

(b) (4') Now Alice wants to reverse the entire list using the `move_to_first` function. Please help her complete the reversion function `reverse_list(Node *head)` where `head` is the head pointer of the vocabulary list to be reversed. The function returns the **new** head pointer after this process. She can guarantee that the list is non-empty.

```
Node* reverse_list(Node *head){
    Node *p = head;
    while(/* (1) */){
        /* (2) */ = move_to_first(/* (3) */, /* (4) */);
    }
    return head;
}
```

Please help her complete this function by filling your code into blank (1) to (4). Each blank should be just an expression or a variable.

Solution:

```
Node* reverse_list(Node *head){
    Node *p = head;
    while(p->next != nullptr){
        head = move_to_first(head, p);
    }
    return head;
}
```

5. (10 points) Car Park Scheduling

In a car park with a parking scheduling area, cars arrive at the entrance in a certain order, and each car has a unique identifier c_i.

Cars can choose to drive out directly or enter the parking scheduling area to wait. Due to the scheduling nature of the area, cars must follow the Last In First Out (LIFO) rule, meaning the car that entered the scheduling area last must be the first to exit.

Now you need to solve the following question.

Given a sequence of cars entering the park, we need to judge if a sequence of cars exiting is feasible or not.

Example

Suppose the sequence of cars entering is $\{c_1, c_2, c_3\}$, all possible exit sequences include:

1. $\{c_1, c_2, c_3\}$: All cars drive out in the order they arrived.
2. $\{c_1, c_3, c_2\}$: Car c_1 enters and drives out directly. Car c_2 enters the scheduling area to wait. Car c_3 enters and drives out, followed by Car c_2 exiting from the scheduling area.
3. $\{c_2, c_3, c_1\}$: Car c_1 enters the scheduling area to wait. Car c_2 enters and drives out directly. Car c_3 enters and drives out, followed by Car c_1 exiting from the scheduling area.
4. $\{c_2, c_1, c_3\}$: Car c_1 enters the scheduling area to wait. Car c_2 enters and drives out directly. Car c_1 leaves from the scheduling area. Car c_3 enters and drives out.
5. $\{c_3, c_2, c_1\}$: Cars c_1 and c_2 both enter the scheduling area to wait. Car c_3 enters and drives out directly. Then Car c_2 exits the scheduling area, followed by Car c_1.

It can be proved that $\{c_3, c_1, c_2\}$ is not a feasible exit sequence.

(a) (2') Which data structure is the operation of this scheduling area most similar to?

◯ Array ◯ Linked list √ **Stack** ◯ Queue

(b) (6') According to question (a), we could use this data structure to simulate the enter-exit progress. Please complete the function. Hint: you can access the front element of a queue or the top element of a stack using the member function `front()` or `top()`. The class `car` has `==` operator.

```
bool is_feasible(const std::deque<car>& enter_seq, std::queue<car>
    exit_seq) {
     /* (1) */ x;   // Choose your data structure
     for(auto& enter_car : enter_seq) {
        /* (2) */
        while( !x.empty() && /* (3) */ ) {
            /* (4) */
            /* (5) */
        }
     }
     if (/* (6) */) {
        return true;  // Given enter_seq, the exit_seq is feasible.
     }
     return false; // Given enter_seq, the exit_seq is not feasible.
}
```

Solution:

```
bool is_feasible(const std::deque<car>& enter_seq, std::queue<car>
    exit_seq) {
    std::stack<car> x;
    for(auto& enter_car :enter_seq) {
        x.push(enter_car);
        while(!x.empty() && x.top() == exit_seq.front()) {
            x.pop();
            exit_seq.pop();
        }
    }
    if (x.empty()) {
        return true;
    }
    return false;
}
```

(c) (2') **Try your algorithm!**

Now you can use the algorithm you implemented in question (b) to check whether these sequences are feasible exit sequences. The enter sequence is $\{c_1, c_2, c_3, c_4, c_5, c_6\}$.

- ◯ $\{c_3, c_1, c_5, c_6, c_2, c_4\}$
- √ $\{c_3, c_5, c_4, c_2, c_6, c_1\}$
- ◯ $\{c_2, c_6, c_3, c_4, c_1, c_5\}$
- ◯ $\{c_4, c_5, c_6, c_1, c_3, c_2\}$
- √ $\{c_1, c_4, c_5, c_3, c_2, c_6\}$