CS101 Algorithms and Data Structures
Fall 2022
Homework 1

Due date: 23:59, September 18, 2022

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. (5 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') Bob lent Alice his laptop to play Elden Ring. 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

(c) (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

(d) (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

(e) (1') Bob is Alice's boyfriend. In order to enhance their romantic relationship, Alice and Bob studied together in the ShanghaiTech library and collaborated on one homework question. They discussed about some algorithm using the 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)
C	AD	AC	D

(a) (2') Assume we insert one element before the element at index $i(0 \leqslant i \leqslant n-1)$ in an array of length n and get a new array of length $n+1$. What is the minimum number of moves that the elements in the array need in total?

A. 0
B. i
C. $n-i$
D. $n-i-1$

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

A. Gaining access to the k-th element in an array takes constant time.
B. Gaining access to the k-th element in a linked-list takes constant time.
C. Erasing the k-th element in an array takes constant time.
D. With access to the k-th element, inserting an element after the k-th element in a linked-list takes constant time.

(c) (2') Please evaluate the following reverse-Polish expressions using stacks. Which of the following expressions are **illegal**?

A. `1 2 * + 3 5 +`
B. `4 5 6 / * 1 /`
C. `1 + 2 - 3 + 4`
D. `7 8 9 1 + - *`

(d) (2') Assume we implement a queue with a circular array indexed from 0 to $n-1$, and the `front` pointer is currently at index m. We will know that the queue is full if the `back` pointer is at index _____. (Options below are mapped to integers modulo n: $\mathbb{Z}_n = \{0, 1, 2, \cdots, n-1\}$.)

A. 0
B. m
C. $n-1$
D. $m-1$

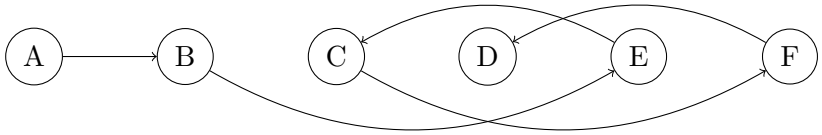
3. (10 points) Array Representation of Linked-List

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 combined 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 Linked-Lists in an Array**

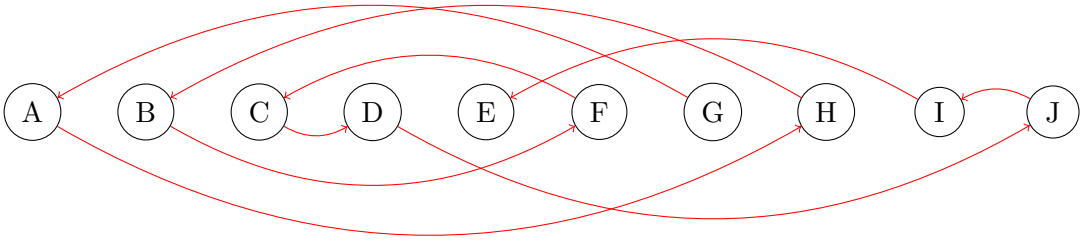
Fill in the table below to finish the array representation of the linked-list shown below. Fill in -1 to represent the end of the linked-list.



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

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

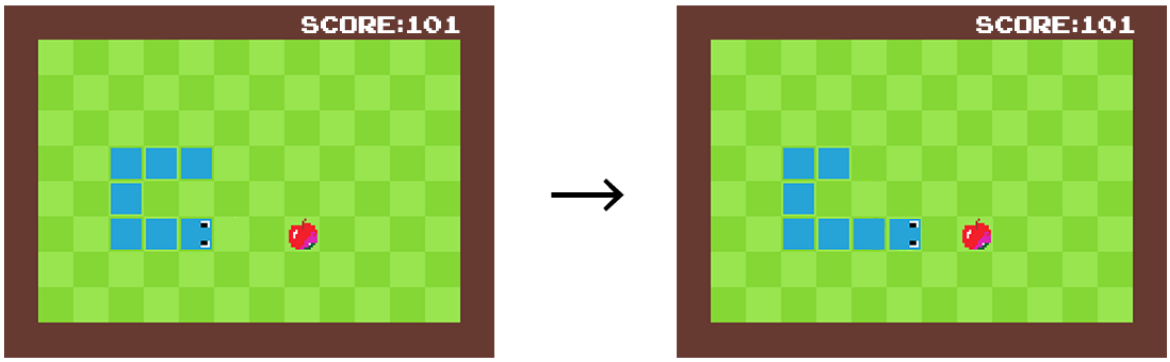
Here are the arrays `next` and `value`. Please draw the linked-list below represented by these two arrays.



index	0	1	2	3	4	5	6	7	8	9
`value`	A	B	C	D	E	F	G	H	I	J
`next`	7	5	3	9	-1	2	0	1	4	8

4. (12 points) Snake Game with Doubly-Linked-List

In this question, you will implement the movement of the snake in a basic Snake game with the help of a doubly-linked-list. As the figure shown below, the snake can take a step forward by deleting its tail node and creating a new head node according to the direction of the snake.



In this question, you will make use of the list node class '`Node`' defined below to implement several operations of the doubly-linked-list in C++ style to support this game.

```
struct Node {
  int x, y;
  Node *prev, *next;
  Node(int _x, int _y, Node *_prev, Node *_next)
      : x(_x), y(_y), prev(_prev), next(_next) {}
};
Node *head, *tail;
```

Please pay attention to the following rules.

1. Use `new` and `delete` to deal with memory issues.
2. Use `nullptr` for null pointer value.
3. Make sure your code **compiles** and does not cause **runtime-errors** or **memory-leaks**.
4. You can fill in each blank line with at most one statement. You don't have to use all of the blank lines.
5. For simplicity, you can directly access node attributes like `node->prev`, `node->next`, etc. This is not recommended for practical programming but OK for this question.

(a) (5') **Update the Snake Head**

We want to create a new node to store the new location of the snake head, and insert it before the original head. You may assume that the snake has at least two nodes. Please fill in the blanks below.

Solution: Possible answer 1:

```
void update_head(int x, int y) {
  Node *new_head = new Node(x, y, nullptr, head);
  head->prev = new_head;
  head = new_head;
}
```

Possible answer 2:

```
void update_head(int x, int y) {
  Node *new_head = new Node(x, y, nullptr, head);
  head = new_head;
  head->next->prev = head;
}
```

(b) (5') **Remove the Snake Tail**

We want to remove the tail node of the snake and update the information of the tail. You may assume that the snake has at least two nodes. Please fill in the blanks below.

Solution: Possible answer 1:

```
void remove_tail() {
  Node *old_tail = tail;
  tail = old_tail->prev;
  tail->next = nullptr;
  delete old_tail;
}
```

Possible answer 2:

```
void remove_tail() {
  Node *old_tail = tail;
  tail = tail->prev;
  delete tail->next;
  tail->next = nullptr;
}
```

(c) (2') **Eat the Fruit**

If the snake head would touch the fruit at (x, y) after a certain step, the snake will eat the fruit and increase its length by one node. Then for this step, which of the following operations should we call? Tick ($\surd$) the correct answer.

◯ `remove_tail()`

$\surd$ `update_head(x, y)`

◯ `update_head(x, y)` and `remove_tail()`

5. (10 points) Let's Play with Linked-List, Stack and Queue

In the following questions, we want to implement the `push` and `pop` operations of stacks and queues with singly-linked-lists in the most efficient way. You can only access the head node of the linked-list. Access to the tail node is not allowed.

You can answer this question using pseudocode or natural language. Please make sure your description is clear and easy to understand.

(a) (5') **Stack Using Linked-List**

Please specify how you implement stack's `push` and `pop` operations.

Solution:

- `push`: Assume we push an element x. We create a new node n for x, set its next node to be the head node, and update the head node to be n. (Call `push_front(x)` in the lecture slides.)
- `pop`: Suppose the original head node is h. Update the head node to be $h.next$, and then remove h. (Call `pop_front()` in the lecture slides.)

(b) (5') **Queue Using Linked-Lists**

Please specify how you implement queue's `push` and `pop` operations with linked-list operations. You may directly use the operations you implemented in part (a).

Hint: *Use multiple stacks to implement a queue.*

Solution: We use two stacks to implement a queue. According to part (a), two linked-lists are used as two stacks. Let the two stacks be A and B.

- `push`: Push to A.
- `pop`: If B is empty, pop each element in A out of A and push it to B once popped. Then pop from B.