Bài tập thực hành
Cấu trúc dữ liệu và thuật toán
Hàng đợi, hàng đợi hai đầu

1. Bài 1

Cho khai báo kiểu của một nút trong danh sách liên kết như sau:

```
typedef struct _listnode {
  int num;
  struct _listnode *next;
} ListNode;
```

Cho khai báo kiểu của một danh sách liên kết như sau:

```
typedef struct _linkedlist {
  ListNode *head;
  int size;
} LinkedList;
```

Cho khai báo kiểu của một hàng đợi như sau:

```
typedef struct _queue {
  LinkedList ll;
} Queue;
```

Cho khai báo của các hàm thực hiện các thao tác trên danh sách liên kết (đã cài đặt trong bài thực hành về danh sách liên kết) như sau:

```
void printList(ListNode *head);
```

Hiện ra màn hình giá trị của các phần tử trong danh sách liên kết trỏ bởi head. Nếu danh sách liên kết không có phần tử nào thì hiện thông báo "Danh sach lien ket khong co phan tu nao".

```
ListNode* findNode(ListNode *head, int i);
```

Trả về địa chỉ của node thứ i trong danh sách liên kết trỏ bởi head. Node đầu tiên của danh sách liên kết là node thứ 0.

void insertNode(ListNode **pHead, int index, int value);

Chèn một nút có giá trị value vào vị trí index trong danh sách liên kết trỏ bởi *pHead.

void removeNode(ListNode **ptrHead, int index);

Xóa nút ở vị trí index trong danh sách liên kết trỏ bởi *pHead.

Cho khai báo của các hàm thực hiện các thao tác trên hàng đợi như sau:

void enqueue(Queue *q, int item)

Đưa vào hàng đợi một phần tử có giá trị item.

int dequeue(Queue *q)

Lấy một phần tử ra khỏi hàng đợi. Giá trị trả về của hàm là giá trị của phần tử lấy ra.

int peek(Queue *q)

Trả về giá trị của nút ở đầu hàng đợi, không tạo sự thay đổi gì trong hàng đợi.

int isEmptyQueue(Queue *q)

Trả về 1 nếu hàng đợi rỗng, trả về 0 nếu hàng đợi không rỗng.

1) Khai báo các cấu trúc dữ liệu và các hàm đã cho

2) Hoàn thiện các hàm thao tác trên danh sách liên kết trong bài thực hành về danh sách liên kết.

3) Hoàn thiện đoạn chương trình dưới đây trong hàm main() để tạo một hàng đợi rỗng:

```
Queue *q = malloc( ___ );
q->ll.size = ___ ;
q->ll.head= ___ ;
```

4) Trong hàm main(), dùng hàm printList() để đưa ra danh sách các phần tử trong hàng đợi.

5) Hoàn thiện hàm enqueue()

```
void enqueue(Queue *q, int item) {
  insertNode( ___ , ___, item);
  q->ll.size = ___ ;
}
```

6) Hoàn thiện hàm dequeue()

```
int dequeue(Queue *q) {
  int item;
  if (q->ll.head != NULL) {
            item = ___ ;
            removeNode( ___, ___ );
            (q->ll).size = ___ ;
            return item;
  }
  else {
            prinft("Hang doi rong");
            return 0;
  }
}
```

7) Hoàn thiện hàm peek()

```
int peek(Queue *q){
  if (q->ll.head != NULL)
            return ___ ;
  else {
            prinft("Hang doi rong");
            return 0;
  }
}
```

8) Hoàn thiện hàm isEmptyQueue()

```
int isEmptyQueue(Queue *q){
  if  (  ___ ) return 1;
  return 0;
}
```

2. Bài 2

Cho định nghĩa hằng số và khai báo kiểu của một hàng đợi sử dụng mảng tĩnh như sau:

```
#include <stdio.h>
#include <stdbool.h>

#define MAX_SIZE 10

typedef struct {
    int data[MAX_SIZE];
    int front; // vị trí đầu hàng đợi
    int rear;  // vị trí cuối hàng đợi
    int size;  // số lượng phần tử trong hàng đợi
} Queue;
```

Cho khai báo của các hàm thực hiện các thao tác trên hàng đợi như sau:

void initQueue(Queue *q);
Khởi tạo một hàng đợi rỗng.

bool isEmptyQueue(Queue *q);
Trả về true nếu hàng đợi rỗng, false nếu không rỗng.

bool isFullQueue(Queue *q);
Trả về true nếu hàng đợi đầy, false nếu không đầy.

void enqueue(Queue *q, int value);
Thêm một phần tử có giá trị value vào cuối hàng đợi. Nếu hàng đợi đầy, in thông báo lỗi.

int dequeue(Queue *q);
Lấy và xóa phần tử ở đầu hàng đợi. Giá trị trả về của hàm là giá trị của phần tử lấy ra. Nếu hàng đợi rỗng, in thông báo lỗi và trả về -1.

int peek(Queue *q);
Trả về giá trị của phần tử ở đầu hàng đợi, không xóa phần tử đó. Nếu hàng đợi rỗng, in thông báo lỗi và trả về -1.

1) Khai báo các cấu trúc dữ liệu và các hàm đã cho.
2) Hoàn thiện các hàm thao tác trên hàng đợi sử dụng mảng tĩnh.
3) Hoàn thiện hàm initQueue():

```
void initQueue(Queue *q) {
    q->front = ___ ;
    q->rear = ___ ;
    q->size = ___ ;
}
```

4) Hoàn thiện hàm isEmptyQueue():

```
bool isEmptyQueue(Queue *q) {
    return q->size == ___ ;
}
```

5) Hoàn thiện hàm isFullQueue():

```
bool isFullQueue(Queue *q) {
    return q->size == ___ ;
}
```

6) Hoàn thiện hàm enqueue():

```
void enqueue(Queue *q, int value) {
    if ( ___ (q)) { // Kiểm tra hàng đợi đầy
        printf("Queue is full. Cannot enqueue %d\n", value);
        return;
    }
    q->rear = ( ___ + 1) % ___ ; // Tính vị trí rear mới (vòng tròn)
    q->data[ ___ ] = value;     // Gán giá trị vào vị trí rear
    q->size ___ ;               // Tăng kích thước
}
```

7) Hoàn thiện hàm dequeue():

```
int dequeue(Queue *q) {
    if ( ___ (q)) { // Kiểm tra hàng đợi rỗng
        printf("Queue is empty. Cannot dequeue.\n");
        return ___ ; // Giá trị trả về khi lỗi
    }
    int value = q->data[ ___ ]; // Lấy giá trị ở đầu hàng đợi
    q->front = ( ___ + 1) % ___ ; // Cập nhật vị trí front (vòng tròn)
    q->size ___ ; // Giảm kích thước
    return value;
}
```

8) Hoàn thiện hàm peek():

```
int peek(Queue *q) {
    if ( ___ (q)) { // Kiểm tra hàng đợi rỗng
        printf("Queue is empty. Cannot peek.\n");
```

```
        return ___ ; // Giá trị trả về khi lỗi
    }
    return q->data[ ___ ]; // Trả về giá trị ở đầu hàng đợi
}
```

9) Viết đoạn mã trong hàm main() để:
   - Khởi tạo một hàng đợi q.
   - Thêm các phần tử 10, 20, 30 vào hàng đợi.
   - Xem giá trị phần tử đầu tiên (dùng peek).
   - Lấy ra 2 phần tử khỏi hàng đợi (dùng dequeue).
   - In ra các phần tử còn lại trong hàng đợi bằng cách lặp và dùng dequeue cho đến khi hàng đợi rỗng (tham khảo main() trong code gốc).

3. Bài 3

Cài đặt các thao tác trên hàng đợi hai đầu.

Cho cấu trúc dữ liệu sau:

```
typedef struct _dlistnode{
  int num;
  struct _dlistnode *next;
    struct _dlistnode *prev;
} DListNode;
typedef struct Deque {
   DListNode* head;
   DListNode* tail;
} Deque;
```

1) Hoàn thiện hàm tạo nút mới

```
DListNode* createNode(int data) {
   DListNode* newNode = (DListNode*) malloc( ___ );
   if (newNode == NULL) {
      printf("Memory allocation failed.\n");
      return NULL;
   }
   newNode->data = ___ ;
   newNode->next = ___ ;
```

```
    newNode->prev = ___ ;
    return newNode;
}
```

2) Hoàn thiện hàm khởi tạo hàng đợi hai đầu

```
Deque* createDeque() {
    Deque* deque = (Deque*)malloc( ___ );
    if (deque == NULL) {
        printf("Memory allocation failed.\n");
        return NULL;
    }
    deque->head = ___ ;
    deque->tail = ___ ;
    return ___ ;
}
```

3) Hoàn thiện hàm kiểm tra hàng đợi hai đầu có rống không

```
int isEmpty(Deque* deque) {
    return deque->head ==  ___ ;
}
```

4) Hoàn thiện hàm liệt kê các phần từ trong hàng đợi hai đầu

```
void printDeque(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
        return;
    }
    DListNode* current = deque->head;
    while (current ___  NULL) {
        printf("%d ", ___ );
        current = ___ ;
    }
    printf("\n");
```

```
}
```

5) Hoàn thiện hàm chèn một phần tử vào đầu hàng đợi hai đầu

```
void pushFront(Deque* deque, int data) {
    DListNode* newNode = createNode(data);
    if (isEmpty(deque)) {
        deque->head = ___ ;
        deque->tail = ___ ;
    } else {
        newNode->next = ___ ;
        deque->head->prev = ___ ;
        deque->head = ___ ;
    }
}
```

6) Hoàn thiện hàm chèn một phần tử vào cuối hàng đợi hai đầu

```
void pushRear(Deque* deque, int data) {
    DListNode* newNode = createNode(data);
    if (isEmpty(deque)) {
        deque->head = ___ ;
        deque->tail = ___ ;
    } else {
        newNode->prev = ___ ;
        deque->tail->next = ___ ;
        deque->tail = ___ ;
    }
}
```

7) Hoàn thiện hàm lấy một phần tử ở đầu hàng đợi hai đầu

```
int popFront(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
```

```
        return EXIT_FAILURE;
    }
    DListNode* frontNode = deque->head;
    int data = frontNode->data;
    deque->head = ___ ;
    if (deque->head != NULL) {
        deque->head->prev = ___ ;
    } else {
        deque->tail = __ ;
    }
    free(frontNode);
    return data;
}
```

8) Hoàn thiện hàm lấy một phần tử ở cuối hàng đợi hai đầu

```
int popRear(Deque* deque) {
    if (isEmpty(deque)) {
        printf("Deque is empty.\n");
        return EXIT_FAILURE;
    }
    DListNode* rearNode = deque->tail;
    int data = rearNode->data;
    deque->tail = ___ ;
    if (deque->tail != NULL) {
        deque->tail->next = ___ ;
    } else {
        deque->head = ___ ;
    }
    free( ___ );
    return data;
}
```

9) Hoàn thiện hàm xem giá trị của phần tử ở đầu hàng đợi hai đầu

```
int peekFront(Deque* deque) {
   if (isEmpty(deque)) {
      printf("Deque is empty.\n");
      return EXIT_FAILURE;
   }
return ___ ;
}
```

10) Hoàn thiện hàm xem giá trị của phần tử ở cuối hàng đợi hai đầu

```
int popRear(Deque* deque) {
   if (isEmpty(deque)) {
      printf("Deque is empty.\n");
      return EXIT_FAILURE;
   }
   return ___ ;
}
```