Assignment 2

Course: Data Structures
Course id:14461002
Student id: 202033762

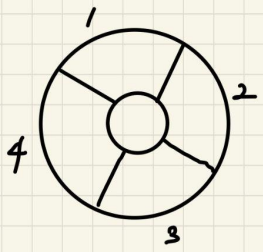
Name: 장민호

Major : 설비소방공학과

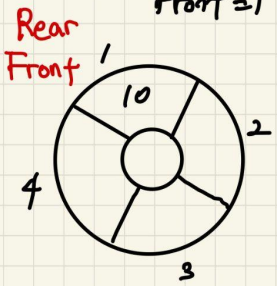
Submission Date : 2022_03_17

HW2-P1

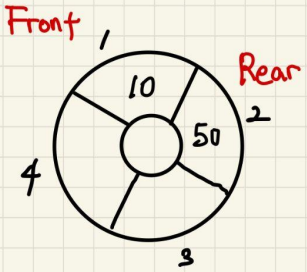
1. Initially, Rear = 0, Front = 0



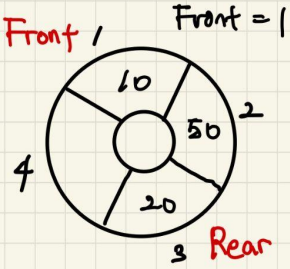
2. Insert 10, Rear = 1, Front = 1



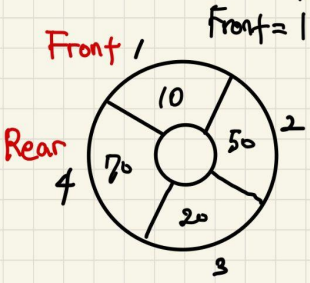
3. Insert 50, Rear = 2, Front = 1



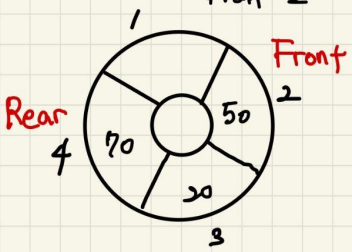
4. Insert 20, Rear = 3, Front = 1



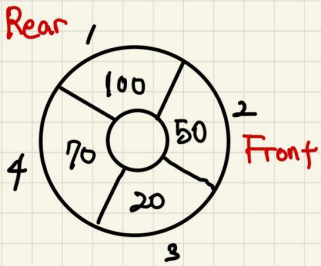
5. Insert 70, Rear = 4, Front = 1



6. Delete front, Rear = 4, Front = 2



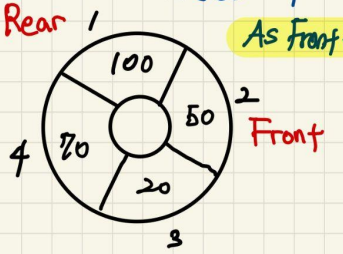
7. Insert 100, Rear = 1, Front = 2



8. Insert 40, Rear = 1, Front = 2

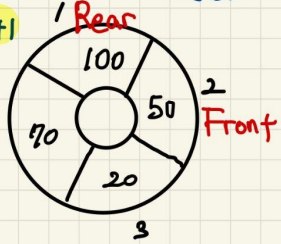
Queue overflow

As Front = Rear + 1

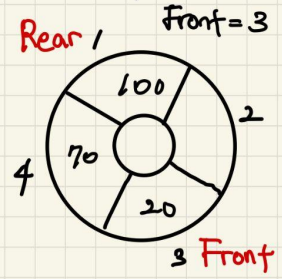


9. Insert 140, Rear = 1, Front = 2

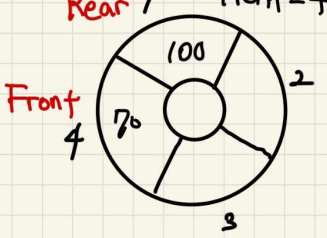
Queue overflow



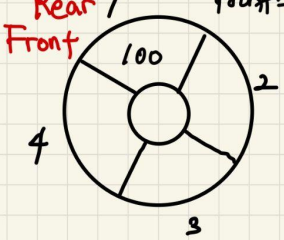
10. Delete front, Rear = 1, Front = 3



11. Delete front, Rear = 1, Front = 4



12. Delete front, Rear = 1, Front = 1



HW2-P2

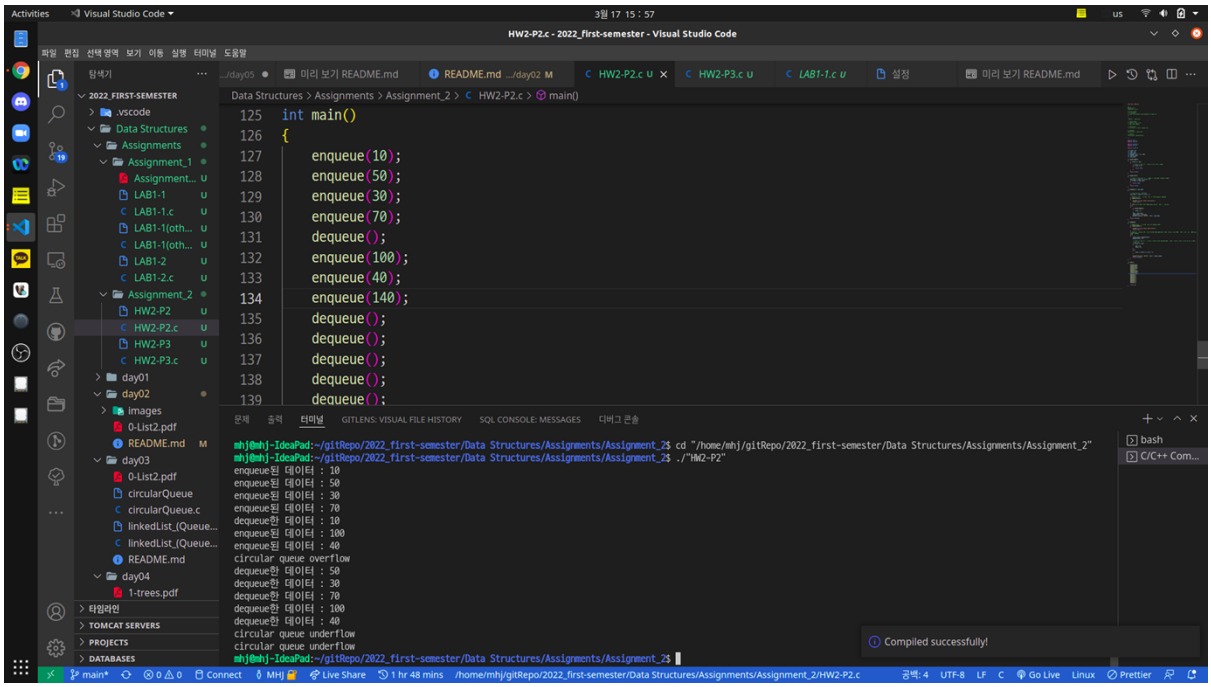
- Source Code

```
#include <stdio.h>
/*
HW2-P2 과제
202033762 장민호
Circular Queue
-> 5 elements
-> Test the program using sequence of page 8-9
*/
/*
구현해야할 함수 정보
1. Queue_empty
-> front == rear == 0
-> 0인 경우 (Empty)
2. Queue_full
-> If front == rear+1, Queue full
3. enqueue
queue[rear] = input_num;
4. dequeue
return_num = queue[front];
*/
#define TRUE 1
#define FALSE 0
#define SUCCESS 1
#define FAIL 0
#define Q_SIZE 5
int front = 0;
int rear = 0;
int next_rear;
int queue[Q_SIZE + 1] = {0};
int return_item;
int new_item;
int queue_empty()
{
  if (front == rear)
  {
     // front == rear 이고, front == 0일 경우 빈 큐임
     if (front == 0)
     {
        return TRUE;
     }
  }
  return FALSE;
}
int queue_full()
{
  // front == next_rear 일 때, 큐가 꽉 차있으므로 enqueue 불가능.
  next_rear = (rear % Q_SIZE) + 1;
```

```
    if (front == next_rear)
    {
        return TRUE;
    }
    return FALSE;
}
int enqueue(int new_item)
{
    // next_rear 을 만들어둔다.
    next_rear = (rear % Q_SIZE) + 1;
    // 큐가 full 인지 검사한다. full 일 경우 enqueue 불가하다.
    if (queue_full())
    {
        printf("circular queue overflow\n");
        return FAIL;
    }
    // 큐가 full이 아닐 경우, 큐가 empty 상태인지 아닌지 구분한다.
    else
    {
        if (queue_empty())
        {
            front = 1;
        }
        rear = next_rear;
        queue[rear] = new_item;
        printf("enqueue된 데이터 : %d\n", new_item);
    }
    return SUCCESS;
}
int dequeue()
{
    // 큐가 비었는지 검사한다. 비었으면 dequeue 실패
    if (queue_empty())
    {
        printf("circular queue underflow\n");
        return FAIL;
    }
    // 큐가 비지 않았을 경우, 일단 front의 값을 가져온다. 이후, front과 rear 값의 관계를 구하고 그 관계에 따라 분기
를 나눈다.
    else
    {
        return_item = queue[front];
        queue[front] = 0;
        // front == rear 이지만 front != 0이면 front 위치의 값을 저장하고 front = rear = 0 을 해주어 빈 큐로 만들어준
다.
        if (front == rear)
        {
            rear = 0;
            front = 0;
        }
        else
        {
            front = (front % Q_SIZE) + 1;
        }
```

```
        printf("dequeue한 데이터 : %d\n", return_item);
        return SUCCESS;
    }
}
int main()
{
    enqueue(10);
    enqueue(50);
    enqueue(30);
    enqueue(70);
    dequeue();
    enqueue(100);
    enqueue(40);
    enqueue(140);
    dequeue();
    dequeue();
    dequeue();
    dequeue();
    dequeue();
    dequeue();
    dequeue();
    return 0;
}
```

- ScreenShot (P2)



```
mhj@mhj-IdeaPad:~/gitRepo/2022_first-semester/Data Structures/Assignments/Assignment_2$ cd "/home/mhj/gitRepo/2022_first-semester/Data Structures/Assignments/Assignment_2"
mhj@mhj-IdeaPad:~/gitRepo/2022_first-semester/Data Structures/Assignments/Assignment_2$ ./"HW2-P2"
enqueue된 데이터 : 10
enqueue된 데이터 : 50
enqueue된 데이터 : 30
enqueue된 데이터 : 70
dequeue한 데이터 : 10
enqueue된 데이터 : 100
enqueue된 데이터 : 40
circular queue overflow
dequeue한 데이터 : 50
dequeue한 데이터 : 30
dequeue한 데이터 : 70
dequeue한 데이터 : 100
dequeue한 데이터 : 40
circular queue underflow
circular queue underflow
mhj@mhj-IdeaPad:~/gitRepo/2022_first-semester/Data Structures/Assignments/Assignment_2$
```

HW2-P3

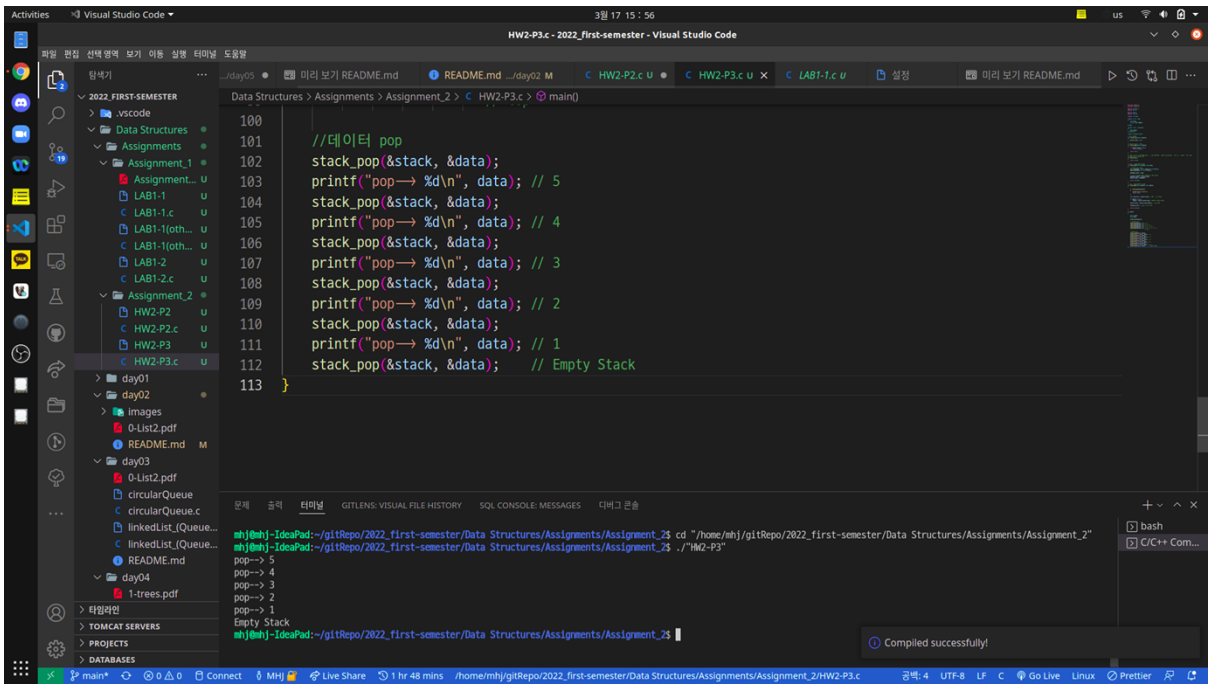
- Source Code

```
#include <stdio.h>
#include <stdlib.h>
#define SUCCESS 1
#define FAIL 0
#define TRUE 1
#define FALSE 0
typedef int Data;
typedef struct _node
{
   Data data;
   struct _node *pNext;
} Node;
typedef struct _listStack
{
   Node *pTop;
} ListStack;
typedef ListStack Stack;
// 스택 초기화
void stack_init(Stack *pStack)
{
   pStack->pTop = NULL;
}
// 스택이 비었는지 확인
int stack_empty(Stack *pStack)
{
   if (pStack->pTop == NULL)
      return SUCCESS;
   return FAIL;
}
// malloc 방식으로 데이터가 들어올 때마다 메모리에 할당하는 방식의 스택은 full 되지 않는다고 보면 된다. 그러
므로 반드시 False 리턴
int stack_full()
{
   return FALSE;
}
```

```c
// push : top에 데이터 추가
int stack_push(Stack *pStack, Data data)
{
    if (stack_full())
    {
        return FAIL;
    }
    // 새로운 노드 생성
    Node *pNewNode = (Node *)malloc(sizeof(Node));
    memset(pNewNode, 0, sizeof(Node));
    pNewNode->data = data;
    // 노드를 스택에 추가 (리스트 맨 앞에 insert)
    pNewNode->pNext = pStack->pTop;
    pStack->pTop = pNewNode;
    return SUCCESS;
}
// pop : top의 데이터 꺼내기
int stack_pop(Stack *pStack, Data *pData)
{
    if (stack_empty(pStack))
    {
        printf("Empty Stack\n");
        return FAIL;
    }
    Node *pCurrent = pStack->pTop; // 삭제할 노드 기억
    if (pData != NULL)
        *pData = pStack->pTop->data; //데이터 꺼내기
    pStack->pTop = pStack->pTop->pNext; // top 이동
    free(pCurrent); // pop 된 노드 제거
    return SUCCESS;
}
int main()
{
    Stack stack;
    Data data;
    stack_init(&stack);
    // 데이터 push
    stack_push(&stack, 1); // 1
    stack_push(&stack, 2); // 2 - 1
    stack_push(&stack, 3); // 3 - 2 - 1
    stack_push(&stack, 4); // 4 - 3 - 2 - 1
    stack_push(&stack, 5); // 5 - 4 - 3 - 2 - 1
                           // top
    //데이터 pop
    stack_pop(&stack, &data);
    printf("pop--> %d\n", data); // 5
    stack_pop(&stack, &data);
    printf("pop--> %d\n", data); // 4
    stack_pop(&stack, &data);
    printf("pop--> %d\n", data); // 3
    stack_pop(&stack, &data);
    printf("pop--> %d\n", data); // 2
    stack_pop(&stack, &data);
```

```
    printf("pop--> %d\n", data); // 1
    stack_pop(&stack, &data);    // Empty Stack
}
```

- ScreenShot (P3)




```
mhj@mhj-IdeaPad:~/gitRepo/2022_first-semester/Data Structures/Assignments/Assignment_2$ cd "/home/mhj/gitRepo/2022_first-semester/Data Structures/Assignments/Assignment_2"
mhj@mhj-IdeaPad:~/gitRepo/2022_first-semester/Data Structures/Assignments/Assignment_2$ ./"HW2-P3"
pop--> 5
pop--> 4
pop--> 3
pop--> 2
pop--> 1
Empty Stack
```

HW2-P4

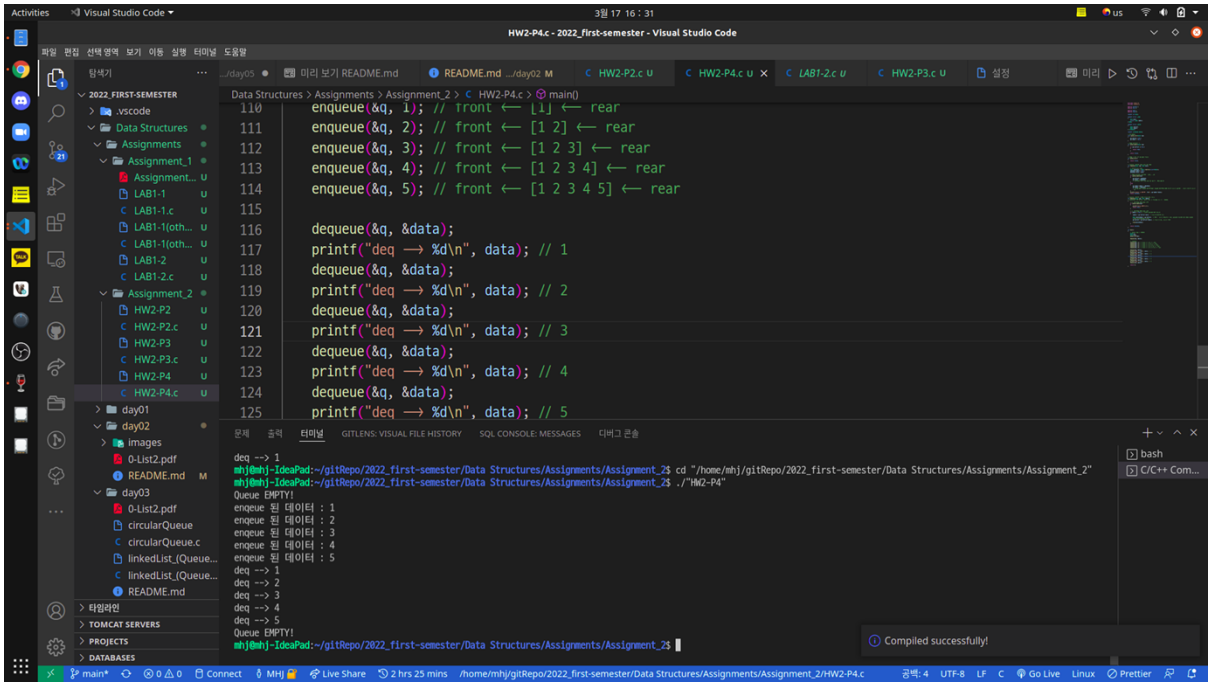
- Source Code

```
#include <stdio.h>
#include <stdlib.h>
#define SUCCESS 1
#define FAIL 0
#define TRUE 1
#define FALSE 0
typedef int Data;
typedef struct _node
{
   Data data;
   struct _node *pNext;
} Node;
typedef struct _queue
{
   Node *pFront;
   Node *pRear;
} ListQueue;
typedef ListQueue Queue;
// 큐 초기화
void queue_init(Queue *pq)
{
   pq->pFront = NULL;
   pq->pRear = NULL;
}
// 큐가 비어있는지?
int queue_empty(Queue *pq)
{
   if (pq->pFront == NULL)
   {
      return TRUE;
   }
   return FALSE;
}
// 큐가 꽉 찰 일이 없으므로 False
int queue_full()
{
   return FALSE;
}
// enqueue, 데이터 추가 (rear 에 추가)
int enqueue(Queue *pq, Data data)
{
   // 새로운 노드 생성
   Node *pNewNode = (Node *)malloc(sizeof(Node));
   pNewNode->pNext = NULL;
   pNewNode->data = data;
   // 큐가 비어있을때와 그렇지 않을때 구분
   if (queue_empty(pq))
   {
      pq->pFront = pNewNode;
      pq->pRear = pNewNode;
```

```
        // 비어있을 경우 front 와 rear 가 같은 값을 가리킴
    }
    else
    {
        pq->pRear->pNext = pNewNode;
        pq->pRear = pNewNode;
        // rear가 가장 나중의 데이터이므로 나중의 데이터의 뒤에 새로운 노드를 집어넣고, rear를 새로운 노드로 지
정
    }
    printf("enqeue 된 데이터 : %d\n", pq->pRear->data);
    return SUCCESS;
}
// dequeue, 데이터 꺼내기 (front 에서 추출)
int dequeue(Queue *pq, Data *pData)
// dequeue의 경우 맨 처음 들어온 값, 즉 front가 제일 먼저 나간다.
{
    // 1. 데이터가 큐에 없을 경우
    if (queue_empty(pq))
    {
        printf("Queue EMPTY!\n");
        return FAIL;
    }
    // 2. 데이터가 큐에 있을 경우
    if (pData != NULL) // 포인터 pData에 값이 있으면
    {
        *pData = pq->pFront->data; // front 의 데이터 추출
        Node *deleteNode = pq->pFront;  // 삭제할 노드를 기억해야 한다. queue의 front에 있는 값을 node형 포인터
deleteNode에 담는다.
        pq->pFront = pq->pFront->pNext; // front 는 다음 노드로 이동
        free(deleteNode);
    }
    return SUCCESS;
}
int main()
{
    // Queue 생성 및 초기화
    Queue q;
    Data data;
    queue_init(&q);
    dequeue(&q, &data);
    // enqueue
    enqueue(&q, 1); // front <-- [1] <-- rear
    enqueue(&q, 2); // front <-- [1 2] <-- rear
    enqueue(&q, 3); // front <-- [1 2 3] <-- rear
    enqueue(&q, 4); // front <-- [1 2 3 4] <-- rear
    enqueue(&q, 5); // front <-- [1 2 3 4 5] <-- rear
    dequeue(&q, &data);
    printf("deq --> %d\n", data); // 1
    dequeue(&q, &data);
    printf("deq --> %d\n", data); // 2
    dequeue(&q, &data);
```

```
    printf("deq --> %d\n", data); // 3
    dequeue(&q, &data);
    printf("deq --> %d\n", data); // 4
    dequeue(&q, &data);
    printf("deq --> %d\n", data); // 5
    dequeue(&q, &data);
    return 0;
}
```

- ScreenShot




```
mhj@mhj-IdeaPad:~/gitRepo/2022_first-semester/Data Structures/Assignments/Assignment_2$ ./"HW2-P4"
Queue EMPTY!
enqeue 된 데이터 : 1
enqeue 된 데이터 : 2
enqeue 된 데이터 : 3
enqeue 된 데이터 : 4
enqeue 된 데이터 : 5
deq --> 1
deq --> 2
deq --> 3
deq --> 4
deq --> 5
Queue EMPTY!
```

Compiled successfully!