```
//2020118008 박보경
//2020118008 박보경
//본인은 이 소스파일을 다른 사람의 소스를 복사하지 않고 직접 작성하였습니다.

#define _CRT_SECURE_NO_WARNINGS

#include<stdio.h>
#include <string.h>
#include<stdlib.h>

#define MAX_QUEUE_SIZE 5
#define MAX_NAME_SIZE 10

typedef struct
{
	int id;
	char name[MAX_QUEUE_SIZE];
}element;

element queue[MAX_QUEUE_SIZE];
int rear = -1;
int front = -1;
int cnt = 0;

element queueEmpty();
void queueFull();
element deleteq();
void addq(element item);

int main()
{
	char input[80];
	char* delimiter = " \n";
	char* op = NULL;
	element student;
	int cnt= 0;

	printf("<< linear queue operations where MAX_QUEUE_SIZE is 5>>\n");
	printf("add 1 Jung\n");
	printf("delete\n");
	printf("***********************************************\n");

	while (1)
```

```
	{
		gets(input);
		op = strtok(input, delimiter);

		if (!strcmp(op, "add"))
		{
			sscanf(input + strlen(op) + 1, "%d%s", &student.id,
student.name);
			addq(student);
		}
		else if (!strcmp(op, "delete"))
		{
			element item;
			item = deleteq();
			cnt++;
			//if (item.id == -1)
				//queueEmpty();

			//if (item.id == 1)
			//exit(EXIT_FAILURE);
		}
		else if (!strcmp(op, "quit"))
			break;
		else
			printf("wrong command!try again!\n");
	}

	return 0;
}

element queueEmpty()
{
	fprintf(stderr, "queue is empty, cannot delete element.\n");
	exit(EXIT_FAILURE);
}

void queueFull()
{

	if (front == -1)
	{
		fprintf(stderr, "Queue is full, cannot add element!\n");
		fprintf(stderr, "currnet queue element\n");
```

```
                for (int i = 0; i <= MAX_QUEUE_SIZE - 1; i++)
                {
                        printf("%d  %s \n", queue[i].id, queue[i].name);
                        deleteq();
                }

                exit(EXIT_FAILURE);
        }

        else
        {
                printf("array shifiting...\n");
                int f = front;
                int i = 0;
                for ( i = 0; i < (rear - front); i++)
                {
                        queue[i] = queue[++f];
                }
                rear = i - 1;
                front = -1;
        }

}


void addq(element item)
{
        if (rear == MAX_QUEUE_SIZE - 1)
                queueFull();
        queue[++rear] = item;
}

element deleteq()
{
        if (front == rear)
                return queueEmpty();
        return queue[++front];

}


//02
//2020118008 박보경
```

```
//본인은 이 소스파일을 다른 사람의 소스를 복사하지 않고 직접 작성하였습니다.

#define _CRT_SECURE_NO_WARNINGS

#include<stdio.h>
#include <string.h>
#include<stdlib.h>

#define MAX_QUEUE_SIZE 2
#define MAX_NAME_SIZE 10
#define MALLOC(p,s) if(!((p)=malloc(s))) { fprintf(stderr, "Insuffient memory");
exit(EXIT_FAILURE); }

typedef struct
{
	int id;
	char name[MAX_NAME_SIZE];
}element;

element* queue;

int capacity = 2;
int rear = 0;
int front = 0;
int cnt = 0;

element queueEmpty();
void queueFull();
void addq(element item);
element deleteq();
void copy(element *a, element *b, element *c);

int main()
{
	char input[80];
	char* delimiter = " \n";
	char* op = NULL;
	element student;
	int cnt = 0;

	MALLOC(queue, capacity*sizeof(*queue));

	printf("<< Circular queue operations where MAX_QUEUE_SIZE is 2>>\n");
```

```
	printf("add 1 Jung\n");
	printf("delete\n");
	printf("************************************************\n");

	while (1)
	{
		gets(input);
		op = strtok(input, delimiter);

		if (!strcmp(op, "add"))
		{
			sscanf(input + strlen(op) + 1, "%d%s", &student.id,
student.name);
			addq(student);
		}
		else if (!strcmp(op, "delete"))
		{
			element item;
			item = deleteq();
			cnt++;
			//if (item.id == -1)
				//queueEmpty();

			//if (item.id == 1)
			//exit(EXIT_FAILURE);
		}
		else if (!strcmp(op, "quit"))
			break;
		else
			printf("wrong command!try again!\n");
	}

	return 0;
}


element queueEmpty()
{
	fprintf(stderr, "queue is empty, cannot delete element.\n");
	exit(EXIT_FAILURE);
}

void queueFull()
```

```
{
        int start;
        element* newQueue;
        MALLOC(newQueue, 2 * capacity * sizeof(*queue));

        start = (front + 1) % capacity;
        if(start < 2)
                copy(queue + start, queue + start + capacity - 1, newQueue);
        else
        {
                copy(queue + start, queue + capacity, newQueue);
                copy(queue, queue + rear + 1, newQueue + capacity - start);
        }

        front = 2 * capacity - 1;
        rear = capacity - 1;
        capacity *= 2;

        printf("queue capacity is doubled \n");
        printf("current queue capacity is %d \n", capacity);

        free(queue);
        queue=newQueue;
}

void addq(element item)
{
        rear = (rear + 1) % capacity;
        if (front == rear)
                queueFull();
        queue[rear] = item;
}

element deleteq()
{
        if (front == rear)
                return queueEmpty();
        front = (front + 1) % capacity;
        printf("delete item: %d %s \n", queue[front].id, queue[front].name);
        return queue[front];
}
```

```
void copy(element *a, element *b, element *c)
{
        while (a != b)
                *c++ = *a++;
}
```