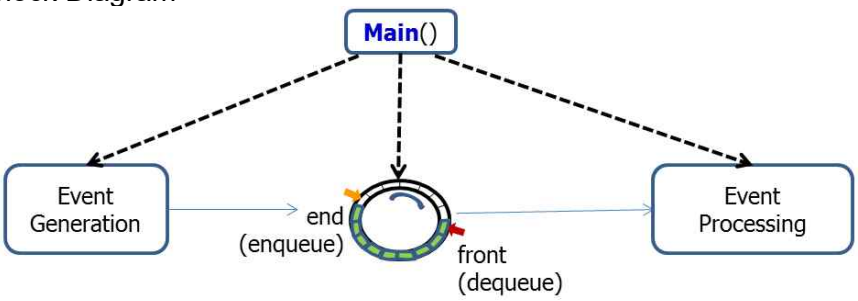
2021-1 프로그래밍언어 실습 11

11.1 Event processing in FIFO with Circular Queue

(1) Functional Block Diagram



(2) Event

```
/* Event.h */

#ifndef EVENT_H
#define EVENT_H

#include <stdio.h>

#define NUM_PRIORITY 100
#define EVENT_PER_LINE 5

enum EventStatus { GENERATED, ENQUEUED, PROCESSED, UNDEFINED };
extern const char *strEventStatus[];

typedef struct
{
	int event_no;
	int event_gen_addr;
	int event_handler_addr;
	int event_pri;   // event_priority
	EventStatus eventStatus;
} Event;

void printEvent(Event* pEvt);
void fprintEvent(FILE *fout, Event* pEvent);
Event *genEvent(Event *pEvent, int event_Gen_ID, int event_no, int event_pri);

#endif
```

(3) Event 처리를 위한 Circular Queue 구현

```
/* CirQ_Event.h */

#ifndef CIRCULAR_QUEUE_H
#define CIRCULAR_QUEUE_H
#include "Event.h"

typedef struct
{
	Event *CirBuff_Ev; // circular queue for events
	int capacity;
	int front;
	int end;
	int num_elements;
```

```
} CirQ_Event;

CirQ_Event *initCirQ(CirQ_Event *pCirQ, int capacity);
void printCirQ(CirQ_Event *cirQ);
void fprintCirQ(FILE *fout, CirQ_Event *cirQ);
bool isCirQFull(CirQ_Event *cirQ);
bool isCirQEmpty(CirQ_Event *cirQ);
Event *enCirQ(FILE *fout, CirQ_Event *cirQ, Event ev);
Event *deCirQ(FILE *fout, CirQ_Event *cirQ);
void delCirQ(CirQ_Event *cirQ);

#endif
```

(4) Example of main() function

```
/* main() for Priority-Queue for Events */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "Event.h"
#include "CirQ_Event.h"
#include "PriQ_Event.h"

#define EVENT_GENERATOR 0
#define TOTAL_NUM_EVENTS 50
#define MAX_ROUND 100

#define INIT_PriQ_SIZE 1
void test_FIFO_CirQ_Event(FILE *fout, int max_events_per_round);
void test_PriQ_Event(FILE *fout, int max_events_per_round);

void main()
{
	FILE *fout;
	int menu;
	int max_events_per_round;

	fout = fopen("output.txt", "w");
	if (fout == NULL)
	{
		printf("Error in creation of output.txt file !!\n");
		exit(-1);
	}
	srand(time(0));

	while (1)
	{
		printf("\nAvailable Menu : \n");
		printf(" 1. Test FIFO/CirQ Event.\n");
		printf(" 2. Test PriQ Event.\n");

		printf("Input menu (0 to quit) : ");
		scanf("%d", &menu);
		if (menu == 0)
			break;
		printf("Input num_events per round :");
		scanf("%d", &max_events_per_round);
		switch (menu)
		{
```

```
            case 1:
                test_FIFO_CirQ_Event(fout, max_events_per_round);
                break;
            case 2:
                test_PriQ_Event(fout, max_events_per_round);
                break;
            default:
                break;
            }
        }
        fclose(fout);
}

void test_FIFO_CirQ_Event(FILE *fout, int max_events_per_round)
{
        CirQ_Event* pCirQ_Event;
        Event ev, * pEv = NULL;
        Event processed_events[TOTAL_NUM_EVENTS];
        int total_processed_events = 0;
        int total_generated_events = 0;
        int num_events = 0;
        int num_generated_round = 0;
        int num_processed_round = 0;

        fprintf(fout, "Testing Event Handling with FIFO Circular Queue\n");
        pCirQ_Event = (CirQ_Event*)calloc(1, sizeof(CirQ_Event));
        printf("Initializing FIFO_CirQ of capacity (%d)\n", max_events_per_round);
        fprintf(fout, "Initializing FIFO_CirQ of capacity (%d)\n", max_events_per_round);
        pCirQ_Event = initCirQ_Event(pCirQ_Event, max_events_per_round);
        //fprintQueue(fout, pCirQ_Event);
        //fprintf(fout, "\nEnqueuing data into event circular queue: \n");

        for (int round = 0; round < MAX_ROUND; round++)
        {
            fprintf(fout, "start of Round(%2d) ****\n", round);
            if (total_generated_events < TOTAL_NUM_EVENTS)
            {
                num_events = max_events_per_round;
                if ((total_generated_events + num_events) > TOTAL_NUM_EVENTS)
                    num_events = TOTAL_NUM_EVENTS - total_generated_events;
                fprintf(fout, "generate and enque %2d events\n", num_events);
                num_generated_round = 0;
                for (int i = 0; i < num_events; i++)
                {
                    if (isCirQFull(pCirQ_Event))
                    {
                        fprintf(fout, "    !!! CirQ_Event is full --> skip generation and enqueueing of
                            event. \n");
                        break;
                    }
                    pEv = genEvent(pEv, EVENT_GENERATOR, total_generated_events,
                            TOTAL_NUM_EVENTS - total_generated_events - 1);
                    fprintf(fout, ">>> Enqueue event = ");
                    fprintEvent(fout, pEv);
                    fprintf(fout, "\n");
                    enCirQ_Event(fout, pCirQ_Event, *pEv);
                    fprintCirQ_Event(fout, pCirQ_Event);
                    free(pEv);
                    total_generated_events++;
                    num_generated_round++;
                } // end for
```

```
            }   // end if
            //fprintf(fout, "\nDequeuing data from event circular queue: \n");
            num_events = max_events_per_round;
            if ((total_processed_events + num_events) > TOTAL_NUM_EVENTS)
                num_events = TOTAL_NUM_EVENTS - total_processed_events;
            fprintf(fout, "dequeue %2d events\n", num_events);
            num_processed_round = 0;
            for (int i = 0; i < num_events; i++)
            {
                if (isCirQEmpty(pCirQ_Event))
                    break;
                pEv = deCirQ_Event(fout, pCirQ_Event);
                if (pEv != NULL)
                {
                    fprintf(fout, "<<< Dequed event = ");
                    fprintEvent(fout, pEv);
                    fprintf(fout, "\n");
                    processed_events[total_processed_events] = *pEv;
                    total_processed_events++;
                    num_processed_round++;
                }
                fprintCirQ_Event(fout, pCirQ_Event);
            } // end for

            /* Monitoring simulation status */
            fprintf(fout, "Round(%2d): generated_in_this_round(%3d), total_generated_events(%3d),
               processed_in_this_round (%3d), total_processed_events(%3d),
               events_in_queue(%3d)\n\n", round, num_generated_round, total_generated_events,
               num_processed_round, total_processed_events, pCirQ_Event->num_elements);
            printf("Round(%2d): generated_in_this_round(%3d), total_generated(%3d),
               processed_in_this_round (%3d), total_processed_events(%3d),
               events_in_queue(%3d)\n", round, num_generated_round, total_generated_events,
               num_processed_round, total_processed_events, pCirQ_Event->num_elements);
            if (total_processed_events >= TOTAL_NUM_EVENTS)
                break;
        } // end for()
        printf("Processed Events :\n");
        for (int i = 0; i < TOTAL_NUM_EVENTS; i++)
        {
            printf("Ev(id:%3d, pri:%3d), ", processed_events[i].event_no,
                processed_events[i].event_pri);
            if ((i + 1) % 5 == 0)
                printf("\n");
        }
        printf("\n");
        delCirQ_Event(pCirQ_Event);
}
```

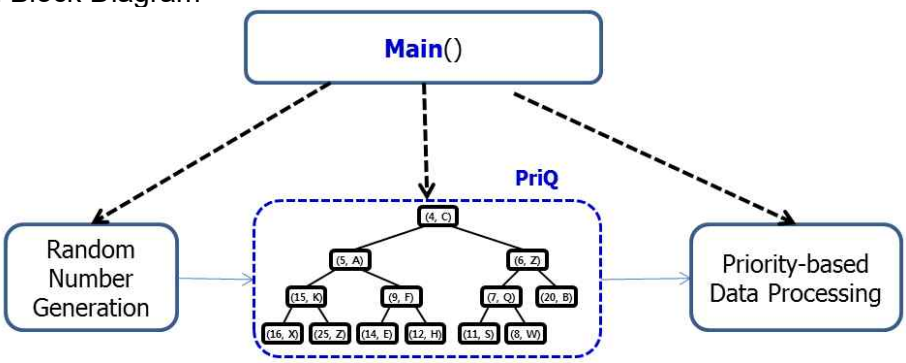
(5) Example output

```
Available Menu :
 1. Test FIFO/CirQ Event.
 2. Test PriQ Event.
Input menu (0 to quit) : 1
Input num_events per round :10
Initializing FIFO_CirQ of capacity (10)
Round( 0): generated_in_this_round( 10), total_generated( 10), processed_in_this_round ( 10), total_processed_events( 10), events_in_queue(  0)
Round( 1): generated_in_this_round( 10), total_generated( 20), processed_in_this_round ( 10), total_processed_events( 20), events_in_queue(  0)
Round( 2): generated_in_this_round( 10), total_generated( 30), processed_in_this_round ( 10), total_processed_events( 30), events_in_queue(  0)
Round( 3): generated_in_this_round( 10), total_generated( 40), processed_in_this_round ( 10), total_processed_events( 40), events_in_queue(  0)
Round( 4): generated_in_this_round( 10), total_generated( 50), processed_in_this_round ( 10), total_processed_events( 50), events_in_queue(  0)
Processed Events :
Ev(id:  0, pri: 49), Ev(id:  1, pri: 48), Ev(id:  2, pri: 47), Ev(id:  3, pri: 46), Ev(id:  4, pri: 45),
Ev(id:  5, pri: 44), Ev(id:  6, pri: 43), Ev(id:  7, pri: 42), Ev(id:  8, pri: 41), Ev(id:  9, pri: 40),
Ev(id: 10, pri: 39), Ev(id: 11, pri: 38), Ev(id: 12, pri: 37), Ev(id: 13, pri: 36), Ev(id: 14, pri: 35),
Ev(id: 15, pri: 34), Ev(id: 16, pri: 33), Ev(id: 17, pri: 32), Ev(id: 18, pri: 31), Ev(id: 19, pri: 30),
Ev(id: 20, pri: 29), Ev(id: 21, pri: 28), Ev(id: 22, pri: 27), Ev(id: 23, pri: 26), Ev(id: 24, pri: 25),
Ev(id: 25, pri: 24), Ev(id: 26, pri: 23), Ev(id: 27, pri: 22), Ev(id: 28, pri: 21), Ev(id: 29, pri: 20),
Ev(id: 30, pri: 19), Ev(id: 31, pri: 18), Ev(id: 32, pri: 17), Ev(id: 33, pri: 16), Ev(id: 34, pri: 15),
Ev(id: 35, pri: 14), Ev(id: 36, pri: 13), Ev(id: 37, pri: 12), Ev(id: 38, pri: 11), Ev(id: 39, pri: 10),
Ev(id: 40, pri:  9), Ev(id: 41, pri:  8), Ev(id: 42, pri:  7), Ev(id: 43, pri:  6), Ev(id: 44, pri:  5),
Ev(id: 45, pri:  4), Ev(id: 46, pri:  3), Ev(id: 47, pri:  2), Ev(id: 48, pri:  1), Ev(id: 49, pri:  0),


Available Menu :
 1. Test FIFO/CirQ Event.
 2. Test PriQ Event.
Input menu (0 to quit) : 1
Input num_events per round :50
Initializing FIFO_CirQ of capacity (50)
Round( 0): generated_in_this_round( 50), total_generated( 50), processed_in_this_round ( 50), total_processed_events( 50), events_in_queue(  0)
Processed Events :
Ev(id:  0, pri: 49), Ev(id:  1, pri: 48), Ev(id:  2, pri: 47), Ev(id:  3, pri: 46), Ev(id:  4, pri: 45),
Ev(id:  5, pri: 44), Ev(id:  6, pri: 43), Ev(id:  7, pri: 42), Ev(id:  8, pri: 41), Ev(id:  9, pri: 40),
Ev(id: 10, pri: 39), Ev(id: 11, pri: 38), Ev(id: 12, pri: 37), Ev(id: 13, pri: 36), Ev(id: 14, pri: 35),
Ev(id: 15, pri: 34), Ev(id: 16, pri: 33), Ev(id: 17, pri: 32), Ev(id: 18, pri: 31), Ev(id: 19, pri: 30),
Ev(id: 20, pri: 29), Ev(id: 21, pri: 28), Ev(id: 22, pri: 27), Ev(id: 23, pri: 26), Ev(id: 24, pri: 25),
Ev(id: 25, pri: 24), Ev(id: 26, pri: 23), Ev(id: 27, pri: 22), Ev(id: 28, pri: 21), Ev(id: 29, pri: 20),
Ev(id: 30, pri: 19), Ev(id: 31, pri: 18), Ev(id: 32, pri: 17), Ev(id: 33, pri: 16), Ev(id: 34, pri: 15),
Ev(id: 35, pri: 14), Ev(id: 36, pri: 13), Ev(id: 37, pri: 12), Ev(id: 38, pri: 11), Ev(id: 39, pri: 10),
Ev(id: 40, pri:  9), Ev(id: 41, pri:  8), Ev(id: 42, pri:  7), Ev(id: 43, pri:  6), Ev(id: 44, pri:  5),
Ev(id: 45, pri:  4), Ev(id: 46, pri:  3), Ev(id: 47, pri:  2), Ev(id: 48, pri:  1), Ev(id: 49, pri:  0),
```

11.2 Priority Queue 기반의 우선 순위에 따른 Event 처리

(1) Functional Block Diagram



(2) 구조체 PriQ_Event 정의

```
/* PriorityQueue_Event.h */

#ifndef PRIORITY_QUEUE_H
#define PRIORITY_QUEUE_H

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

#define POS_ROOT 1
#define MAX_NAME_LEN 80
#define TRUE 1
#define FALSE 0


typedef struct CBTN_Event
{
        int priority;
        Event *pEvent;
} CBTN_Event;

typedef struct PriorityQueue
{
        char name[MAX_NAME_LEN];
        int capacity;
        int num_entry;
        int pos_last;
        CBTN_Event *pCBT_Event;
} PriQ_Event;

PriQ_Event *initPriQ_Event(PriQ_Event *pPriQ_Event, const char *name, int capacity);
int insertPriQ_Event(PriQ_Event *pPriQ_Event, Event *pEvent);
Event *removeMinPriQ_Event(PriQ_Event *pPriQ_Event);
void printPriQ_Event(PriQ_Event *pPriQ_Event);
void fprintPriQ_Event(FILE *fout, PriQ_Event *pPriQ_Event);
void deletePriQ_Event(PriQ_Event *pPriQ_Event);
#endif
```

(3) Example of main() function

```
void test_PriQ_Event(FILE *fout, int max_events_per_round)
{
    PriQ_Event *pPriQ_Ev;
    Event *pEv = NULL;
    Event processed_events[TOTAL_NUM_EVENTS];
```

```
int data;
int total_processed_events = 0;
int total_generated_events = 0;
int num_events = 0;
int num_generated_round = 0;
int num_processed_round = 0;

fprintf(fout, "Testing Event Handling with Priority Queue\n");
pPriQ_Ev = (PriQ_Event *)malloc(sizeof(PriQ_Event));
if (pPriQ_Ev == NULL)
{
    printf("Error in malloc() for PriorityQueue_Event !\n");
    fclose(fout);
    exit(-1);
}
printf("Initializing PriorityQueue_Event of capacity (%d)\n", INIT_PriQ_SIZE);
initPriQ_Event(pPriQ_Ev, "PriorityQueue_Event", INIT_PriQ_SIZE);

for (int round = 0; round < MAX_ROUND; round++)
{
    fprintf(fout, "\n*** Start of round(%2d)...\n", round);
    num_generated_round = 0;
    if (total_generated_events < TOTAL_NUM_EVENTS)
    {
        num_events = max_events_per_round;
        if ((total_generated_events + num_events) > TOTAL_NUM_EVENTS)
            num_events = TOTAL_NUM_EVENTS - total_generated_events;
        fprintf(fout, ">>> enque %2d events\n", num_events);
        for (int i = 0; i < num_events; i++)
        {
            pEv = genEvent(pEv, 0, total_generated_events, TOTAL_NUM_EVENTS -
                total_generated_events - 1);
            if (pEv == NULL)
            {
                printf("Error in generation of event !!\n");
                fclose(fout);
                exit(-1);
            }
            fprintf(fout, " *** enqued event : ");
            fprintEvent(fout, pEv);
            insertPriQ_Event(pPriQ_Ev, pEv);
            total_generated_events++;
            num_generated_round++;
            fprintPriQ_Event(fout, pPriQ_Ev);
        }
    }
    //fprintf(fout, "\n=====================================================\n");
    //fprintf(fout, "\nRemoving min data from Priority Queue . . . \n");
    num_events = max_events_per_round;
    if ((total_processed_events + num_events) > TOTAL_NUM_EVENTS)
        num_events = TOTAL_NUM_EVENTS - total_processed_events;
    fprintf(fout, "<<< dequeue %2d events\n", num_events);
    num_processed_round = 0;
    for (int i = 0; i < num_events; i++)
    {
        pEv = removeMinPriQ_Event(pPriQ_Ev);
        if (pEv == NULL)
        {
            fprintf(fout, "    PriQ is empty\n");
            break;
        }
```

```
            fprintf(fout, " *** dequeued event : ");
            fprintEvent(fout, pEv);
            fprintPriQ_Event(fout, pPriQ_Ev);
            processed_events[total_processed_events] = *pEv;
            total_processed_events++;
            num_processed_round++;
        }
        /* Monitoring simulation status */
        fprintf(fout, "Round(%2d): generated_in_this_round(%3d), total_generated_events(%3d),
           processed_in_this_round (%3d), total_processed_events(%3d), events_in_queue(%3d)\n\n",
           round, num_generated_round, total_generated_events, num_processed_round,
           total_processed_events, pPriQ_Ev->num_entry);
        printf("Round(%2d): generated_in_this_round(%3d), total_generated(%3d),
           processed_in_this_round (%3d), total_processed_events(%3d), events_in_queue(%3d)\n",
           round, num_generated_round, total_generated_events, num_processed_round,
           total_processed_events, pPriQ_Ev->num_entry);
        fflush(fout);
        if (total_processed_events >= TOTAL_NUM_EVENTS)
            break;
    }
    printf("Processed Events :\n");
    for (int i = 0; i < TOTAL_NUM_EVENTS; i++)
    {
        printf("Ev(id:%3d, pri:%3d), ", processed_events[i].event_no, processed_events[i].event_pri);
        if ((i + 1) % 5 == 0)
            printf("\n");
    }
    printf("\n");
    deletePriQ_Event(pPriQ_Ev);
    fprintf(fout, "\n");
}
```

(4) Example output

```
Available Menu :
 1. Test FIFO/CirQ Event.
 2. Test PriQ Event.
Input menu (0 to quit) : 2
Input num_events per round :10
Initializing PriorityQueue_Event
Round( 0): generated_in_this_round( 10), total_generated( 10), processed_in_this_round ( 10), total_processed_events( 10), events_in_queue(  0)
Round( 1): generated_in_this_round( 10), total_generated( 20), processed_in_this_round ( 10), total_processed_events( 20), events_in_queue(  0)
Round( 2): generated_in_this_round( 10), total_generated( 30), processed_in_this_round ( 10), total_processed_events( 30), events_in_queue(  0)
Round( 3): generated_in_this_round( 10), total_generated( 40), processed_in_this_round ( 10), total_processed_events( 40), events_in_queue(  0)
Round( 4): generated_in_this_round( 10), total_generated( 50), processed_in_this_round ( 10), total_processed_events( 50), events_in_queue(  0)
Processed Events :
Ev(id:  9, pri: 40), Ev(id:  8, pri: 41), Ev(id:  7, pri: 42), Ev(id:  6, pri: 43), Ev(id:  5, pri: 44),
Ev(id:  4, pri: 45), Ev(id:  3, pri: 46), Ev(id:  2, pri: 47), Ev(id:  1, pri: 48), Ev(id:  0, pri: 49),
Ev(id: 19, pri: 30), Ev(id: 18, pri: 31), Ev(id: 17, pri: 32), Ev(id: 16, pri: 33), Ev(id: 15, pri: 34),
Ev(id: 14, pri: 35), Ev(id: 13, pri: 36), Ev(id: 12, pri: 37), Ev(id: 11, pri: 38), Ev(id: 10, pri: 39),
Ev(id: 29, pri: 20), Ev(id: 28, pri: 21), Ev(id: 27, pri: 22), Ev(id: 26, pri: 23), Ev(id: 25, pri: 24),
Ev(id: 24, pri: 25), Ev(id: 23, pri: 26), Ev(id: 22, pri: 27), Ev(id: 21, pri: 28), Ev(id: 20, pri: 29),
Ev(id: 39, pri: 10), Ev(id: 38, pri: 11), Ev(id: 37, pri: 12), Ev(id: 36, pri: 13), Ev(id: 35, pri: 14),
Ev(id: 34, pri: 15), Ev(id: 33, pri: 16), Ev(id: 32, pri: 17), Ev(id: 31, pri: 18), Ev(id: 30, pri: 19),
Ev(id: 49, pri:  0), Ev(id: 48, pri:  1), Ev(id: 47, pri:  2), Ev(id: 46, pri:  3), Ev(id: 45, pri:  4),
Ev(id: 44, pri:  5), Ev(id: 43, pri:  6), Ev(id: 42, pri:  7), Ev(id: 41, pri:  8), Ev(id: 40, pri:  9),


Available Menu :
 1. Test FIFO/CirQ Event.
 2. Test PriQ Event.
Input menu (0 to quit) : 2
Input num_events per round :50
Initializing PriorityQueue_Event
Round( 0): generated_in_this_round( 50), total_generated( 50), processed_in_this_round ( 50), total_processed_events( 50), events_in_queue(  0)
Processed Events :
Ev(id: 49, pri:  0), Ev(id: 48, pri:  1), Ev(id: 47, pri:  2), Ev(id: 46, pri:  3), Ev(id: 45, pri:  4),
Ev(id: 44, pri:  5), Ev(id: 43, pri:  6), Ev(id: 42, pri:  7), Ev(id: 41, pri:  8), Ev(id: 40, pri:  9),
Ev(id: 39, pri: 10), Ev(id: 38, pri: 11), Ev(id: 37, pri: 12), Ev(id: 36, pri: 13), Ev(id: 35, pri: 14),
Ev(id: 34, pri: 15), Ev(id: 33, pri: 16), Ev(id: 32, pri: 17), Ev(id: 31, pri: 18), Ev(id: 30, pri: 19),
Ev(id: 29, pri: 20), Ev(id: 28, pri: 21), Ev(id: 27, pri: 22), Ev(id: 26, pri: 23), Ev(id: 25, pri: 24),
Ev(id: 24, pri: 25), Ev(id: 23, pri: 26), Ev(id: 22, pri: 27), Ev(id: 21, pri: 28), Ev(id: 20, pri: 29),
Ev(id: 19, pri: 30), Ev(id: 18, pri: 31), Ev(id: 17, pri: 32), Ev(id: 16, pri: 33), Ev(id: 15, pri: 34),
Ev(id: 14, pri: 35), Ev(id: 13, pri: 36), Ev(id: 12, pri: 37), Ev(id: 11, pri: 38), Ev(id: 10, pri: 39),
Ev(id:  9, pri: 40), Ev(id:  8, pri: 41), Ev(id:  7, pri: 42), Ev(id:  6, pri: 43), Ev(id:  5, pri: 44),
Ev(id:  4, pri: 45), Ev(id:  3, pri: 46), Ev(id:  2, pri: 47), Ev(id:  1, pri: 48), Ev(id:  0, pri: 49),
```

<Oral Test>

Q 11.1 Stack 의 First In Last Out (FILO)기본 동작 (push(), pop(), top())들이 어떻게 실행되는가에 대하여 설명하고, Queue 의 First In First Out (FIFO) 동작이 Stack 과 어떻게 차이가 나는가에 대하여 설명하라.

Q 11.2 Circular buffer 를 기반으로 FIFO queue 를 구성하는 방법을 설명하고, queue 의 기본 동작 (enQueue(), deQueue(), isFull(), isEmpty())이 어떻게 실행되는가에 대하여 설명하라.

Q 11.3 Complete Binary Tree 를 기반으로 구현되는 우선 순위 큐 (priority queue)에서 새로운 항목이 추가 될 때 실행되는 up-heap bubbling 과 우선 순위 큐에서 우선 순위가 가장 높은 항목이 추출될 때 실행되는 down-heap bubbling 의 동작이 어떻게 실행되는가에 대하여 설명하라.

Q 11.4 Circular buffer 기반의 FIFO Queue 를 사용한 Event 처리와 Complete Binary Tree 기반의 Priority Queue 를 사용한 Event 처리의 차이점에 대하여 설명하라.