프로그래밍언어 (실습)

실습 11 (보충설명) 확장성 배열 기반 기본 자료 구조 - Event Processing with FIFO CirQ and PriQ

교수 김 영 탁

영남대학교 정보통신공학과

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- ◆Event with Priority
- ◆Event Generation and Handling
- ◆FIFO Circular Queue
- ◆Heap Priority Queue

Event

Event

```
/* Event.h */
#ifndef EVENT_H
#define EVENT_H

#include <stdio.h>
#define NUM_PRIORITY 100
#define EVENT_PER_LINE 5
#define SIZE_DESCRIPTION 2048

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

typedef struct
{
        int event_no;
        int event_gen_addr;
        int event_handler_addr;
        int event_pri;  // event_priority
        EventStatus eventStatus;
        //char description[SIZE_DESCRIPTION];
} Event;

void initEvent(Event *pEv, int ev_gen_ID, int ev_no, int ev_pri, int ev_handler_addr, EventStatus ev_status);
void printEvent(Event* pEvt);
void fprintEvent(FILE *fout, Event* pEvent);
void printEventArray(Event* pEvent, int size, int items_per_line);
Event *genEvent(Event *pEvent, int event_Gen_ID, int event_no, int event_pri);
#endif
```

```
/* Event.cpp (1) */

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

char *strEventStatus[] = { "GENERATED", "ENQUED", "PROCESSED", "UNDEFINED" };

void printEvent(Event* pEvent)
{
    char str_pri[6];

    printf("Ev(no:%3d, pri:%2d)  ", pEvent->event_no, pEvent->event_pri);
}

void fprintEvent(FILE *fout, Event* pEvent)
{
    char str_pri[6];

    fprintf(fout, "Ev(no:%3d, pri:%2d)  ", pEvent->event_no, pEvent->event_pri);
}

Event *genEvent(Event *pEv, int ev_gen_ID, int ev_no, int ev_pri)
{
    pEv = (Event *)calloc(1, sizeof(Event));
    if (pEv == NULL)
        return NULL;
    initEvent(pEv, ev_gen_ID, ev_no, ev_pri, -1, GENERATED);

    return pEv;
}
```

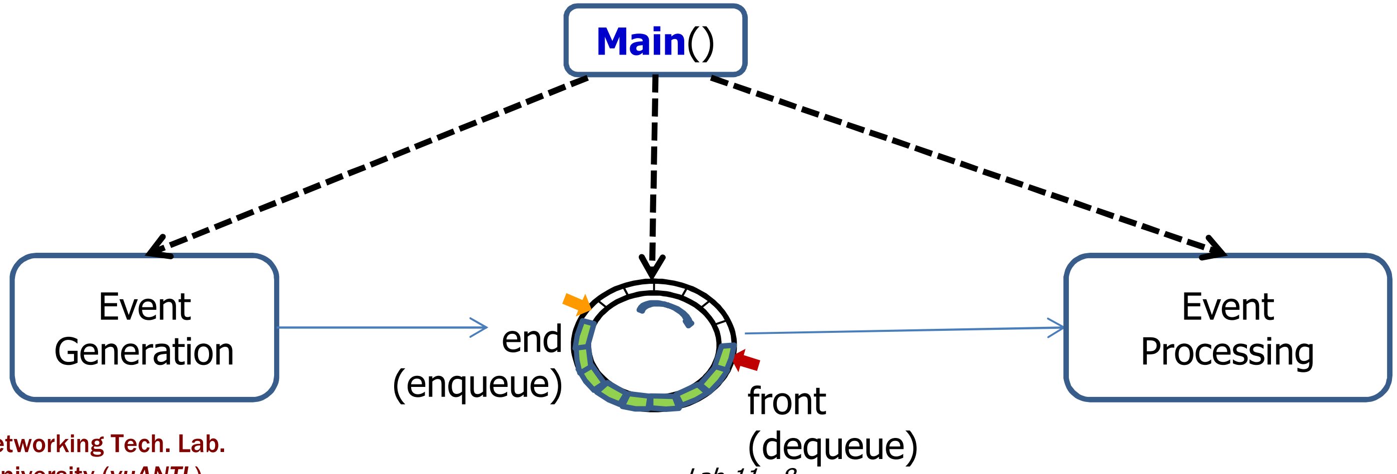
```
/* Event.cpp (2) */

void initEvent(Event *pEv, int ev_gen_ID, int ev_no, int ev_pri, int ev_handler_addr,
   EventStatus ev_status)
{
    pEv->event_gen_addr = ev_gen_ID;
    pEv->event_handler_addr = -1; // event handler is not defined yet !!
    pEv->event_no = ev_no;
    pEv->event_pri = ev_pri;
    pEv->event_handler_addr = ev_handler_addr;
    pEv->eventStatus = ev_status;
}

void printEventArray(Event* pEv, int size, int items_per_line)
{
    for (int i = 0; i < size; i++)
    {
        printEvent(&pEv[i]);
        if (((i + 1) % items_per_line) == 0)
           printf("₩n  ");
    }
}
```

FIFO Queue

실습 11.1 환형 버퍼 (Circular Buffer) 기반 FIFO CirQ의 응용 예제

◆ Event Processing with Circular Queue

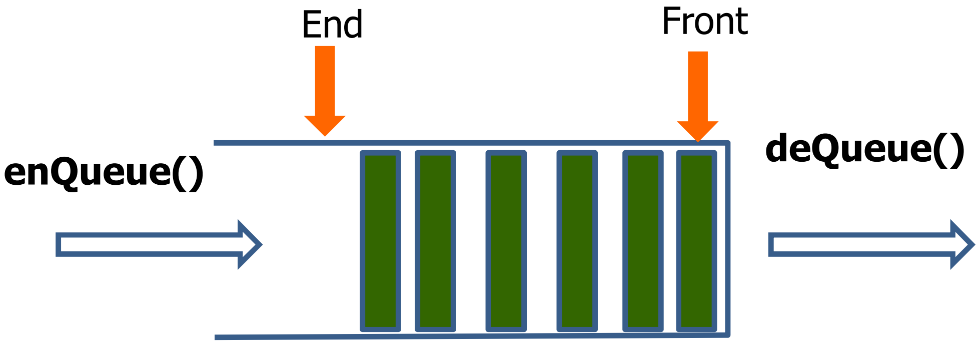
- Event Generation
  - Event generation with event_no, event_priority
  - Enqueue the event into circular queue
- Event Processing
  - Dequeue an event from circular queue
  - Process the event
- Shared Queue
  - Circular Queue (CirQ) with First In First Out (FIFO) process ordering



First In First Out (FIFO) Queues

◆ The Queue stores arbitrary objects

- Insertions and deletions follow the first-in first-out (FIFO) scheme
- Insertions are at the rear of the queue and removals are at the front of the queue



FIFO Queue

Queue Operations

◆ Main queue operations:

- enqueue(object): inserts an element at the end of the queue
- dequeue(): removes the element at the front of the queue

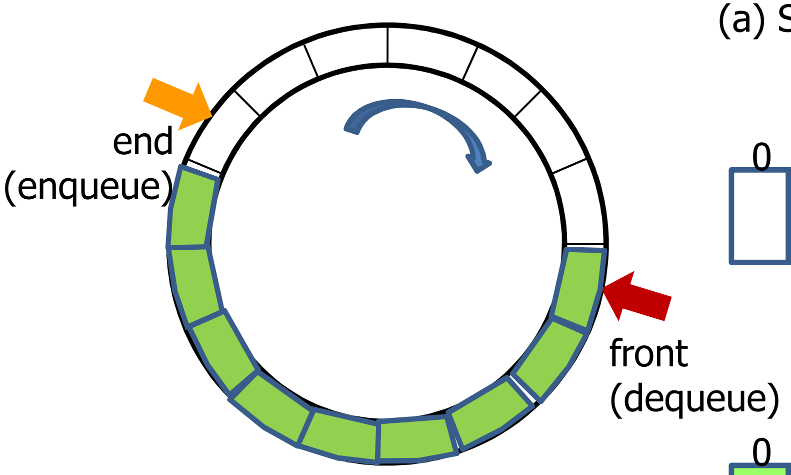
◆ Auxiliary queue operations:

- object front(): returns the element at the front without removing it
- integer size(): returns the number of elements stored
- boolean empty(): indicates whether no elements are stored

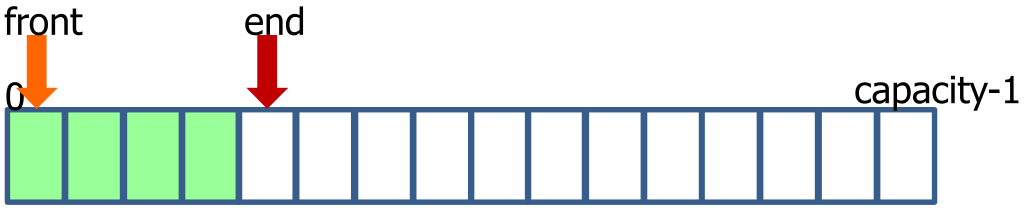
◆ Exceptions

- Attempting the execution of dequeue or front on an empty queue throws an QueueEmpty

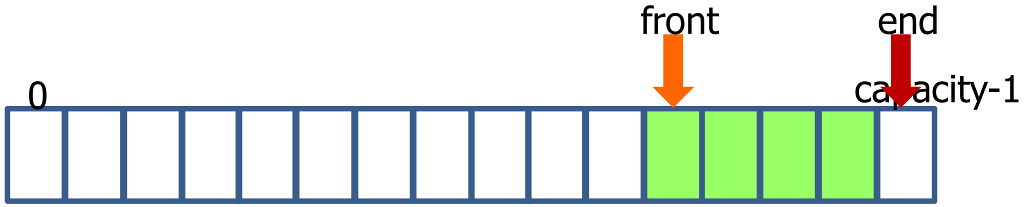
Implementation of Queue with Circular Buffer



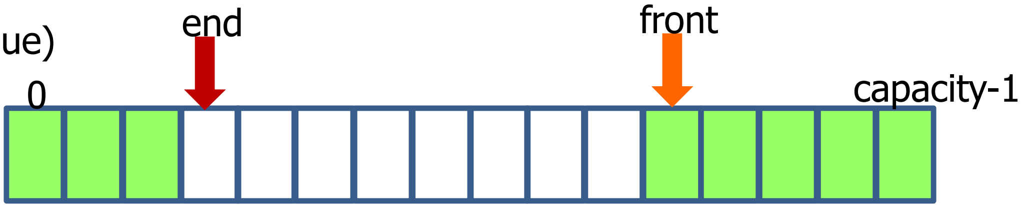
Operations in Circular Buffer



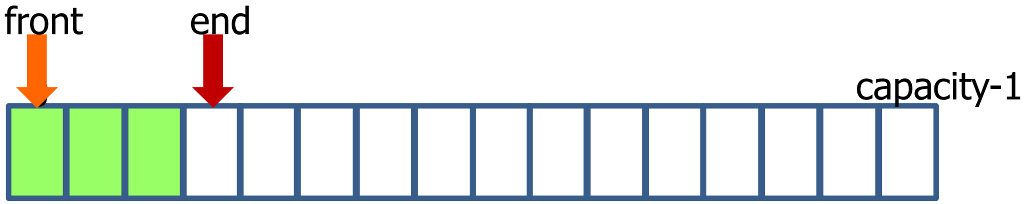
(a) Status of circular buffer after 4 enqueues after initialization



(b) Current status of circular buffer



(c) Circular buffer after 4 enqueues



(d) Circular buffer after 5 dequeues

Circular-Buffer 기반 Queue의 분석

◆ FIFO Queue로 동작

- 큐에 도착 하는 순서에 따라 선착순으로 처리
- 우선 순위를 고려하지 않음

CirQ_Event.h

```c
/* 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_Event(CirQ_Event *pCirQ, int capacity);
void printCirQ_Event(CirQ_Event *cirQ);
void fprintCirQ_Event(FILE *fout, CirQ_Event *cirQ);
bool isCirQFull(CirQ_Event *cirQ);
bool isCirQEmpty(CirQ_Event *cirQ);
Event *enCirQ_Event(FILE *fout, CirQ_Event *cirQ, Event ev);
Event *deCirQ_Event(FILE *fout, CirQ_Event *cirQ);
void delCirQ_Event(CirQ_Event *cirQ);

#endif
```

Lab. 11.1의 main() 프로그램 구성

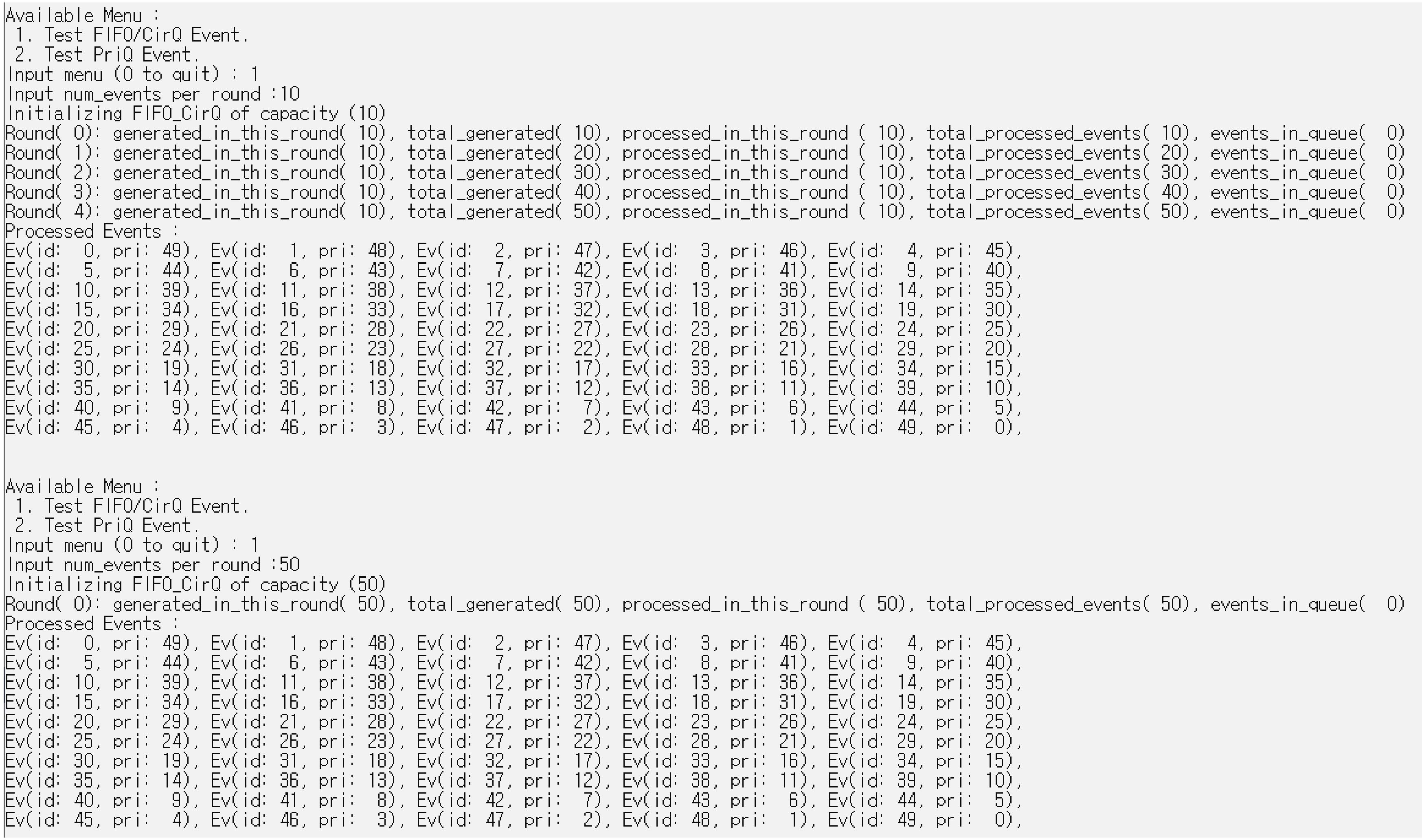
```
/* Lab. 11 – Expandable Array-based Circular Queue and Priority Queue for Event Processing */
/* main() for Priority-Queue for Events */

#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) /* (1) */
{
    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);
/* (2) */

            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
```

```
/* (3) */

        //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
```

```
/* (4) */

        /* 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);
}
```

CirQ_Event 기능 시험 결과



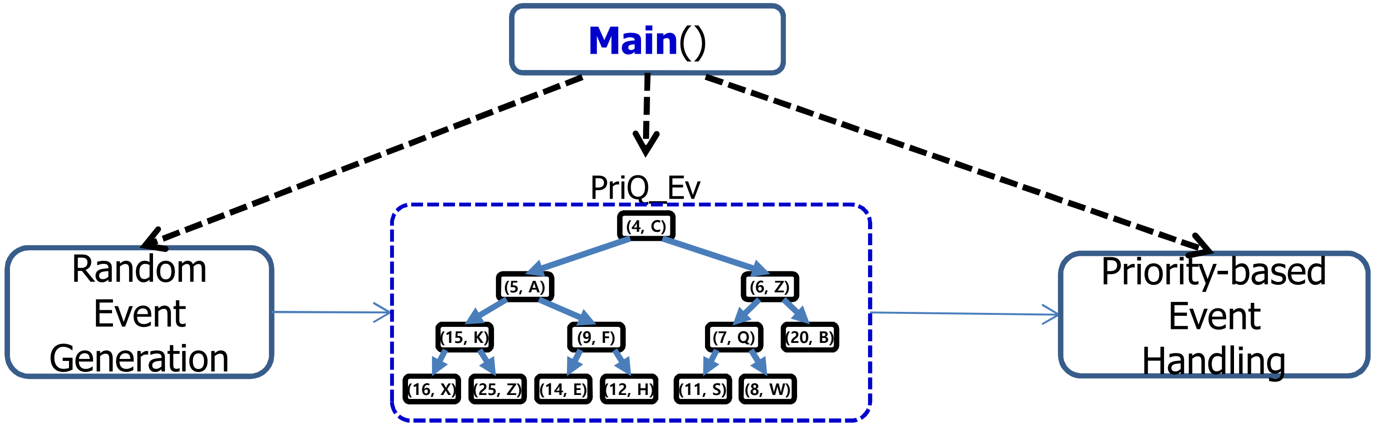
```
Available Menu :
 1. Test FIFO/CirQ Event.
 2. Test PriQ Event.
Input menu (0 to quit) : 1
Input num_events per round :10
Initializing FIFO_CirQ of capacity (10)
Round( 0): generated_in_this_round( 10), total_generated( 10), processed_in_this_round ( 10), total_processed_events( 10), events_in_queue(  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),
```

Heap Priority Queue

실습 11.2 Priority Queue 기반의 우선순위에 따른 Event 처리

◆ Event Processing with Circular Queue

- Event Generation
  - Event generation with event_no, event_priority
  - Enqueue the event into priority queue
- Event Processing
  - Dequeue an event from priority queue
  - Process the event
- Shared Priority Queue
  - Priority Queue for event processing with priority



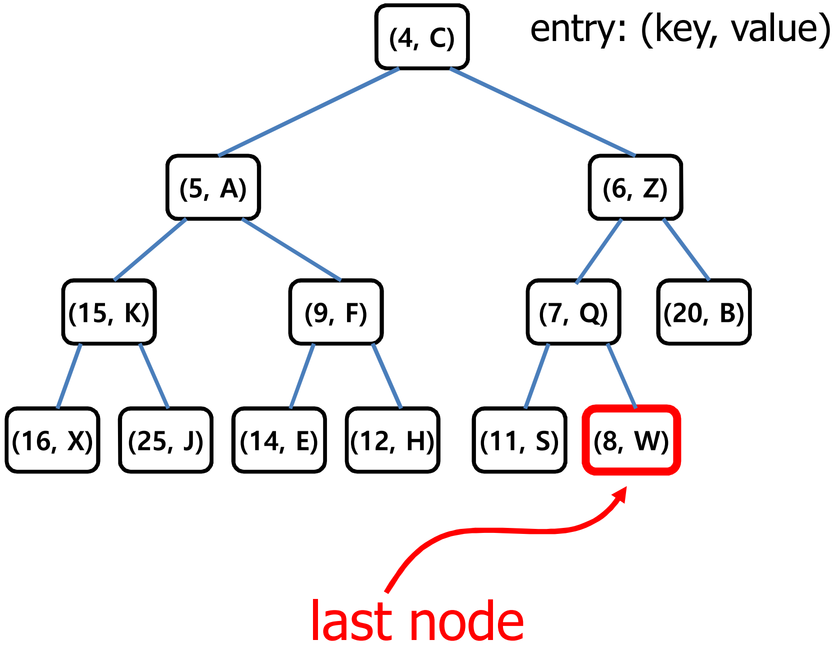
Priority Queue (우선 순위 큐)

- **A priority queue stores a collection of entries**
- **Typically, an entry is a pair (key, value), where the key indicates the priority**
- **Main methods of the Priority Queue ADT**
  - insert(e) : inserts an entry e
  - e = removeMin() : removes the entry with smallest key (highest priority)
- **Additional methods**
  - min() : returns, but does not remove, an entry with smallest key
  - size(), empty()
- **Applications:**
  - Standby flyers
  - Auctions
  - Stock market

Heaps

- A heap is a complete binary tree storing keys at its nodes and satisfying the following properties:
- Heap-Order: for every internal node v other than the root, *key*(*v*) ≥ *key*(*parent*(*v*))
- Complete Binary Tree: let *h* be the height of the heap
  - for ***i*** = 0, … , ***h*** - 1, there are 2^i nodes of depth ***i***
  - at depth ***h*** - 1, the internal nodes are to the left of the external nodes

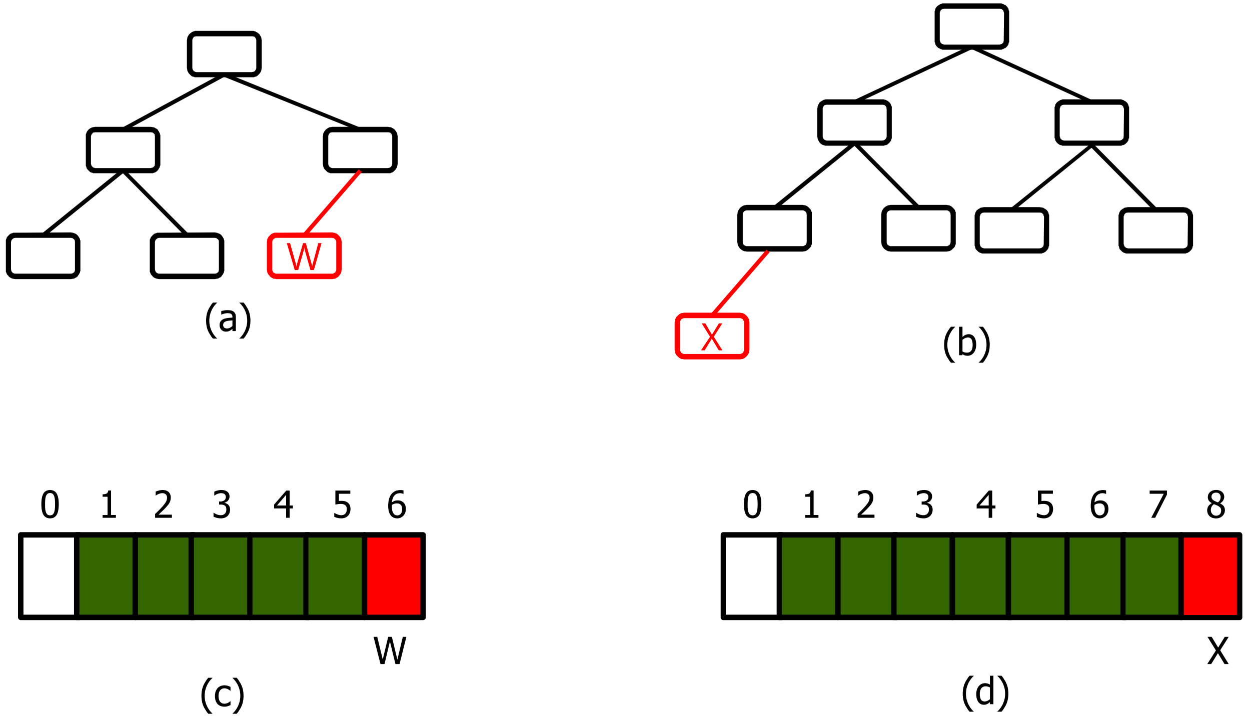
- The last node of a heap is the rightmost node of maximum depth



Array Representation of a Complete Binary Tree

◆ Array representation of a complete binary tree

- if v is the root of CBT, then pos(v) = 1
- if lc is the left child of node u, then pos(lc) = 2 × pos(u)
- if rc is the right child of node u, then pos(rc) = 2 × pos(u) + 1



(a)

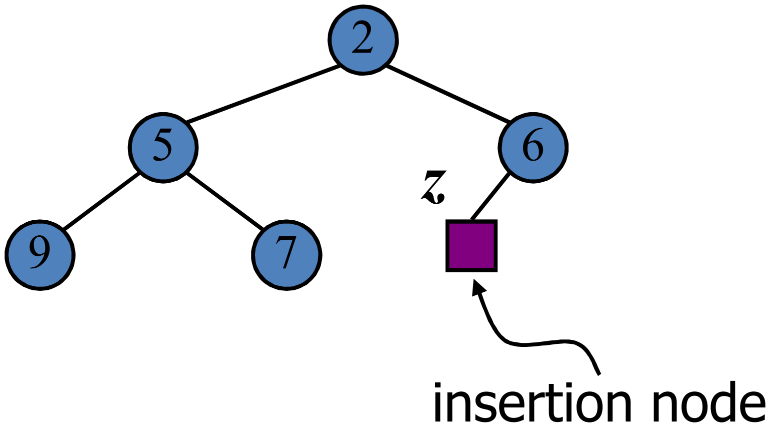
(b)

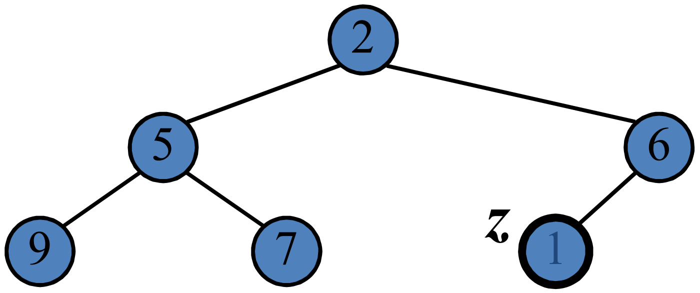
(c)

(d)

Insertion into a Heap

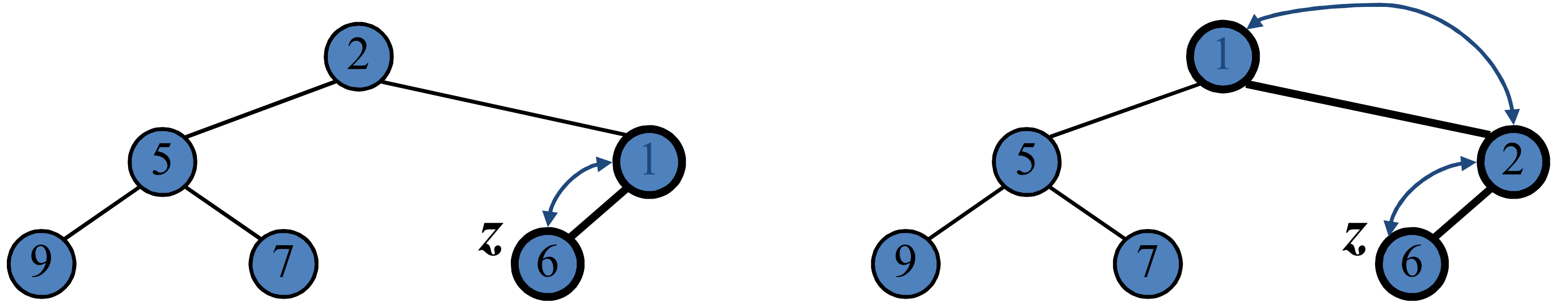
- Method **insertItem()** of the priority queue ADT corresponds to the insertion of a **key** *k* to the heap
- The insertion algorithm consists of three steps
  - Find the insertion node *z* (the new last node)
  - Store *k* at *z*
  - Restore the heap-order property (discussed next)





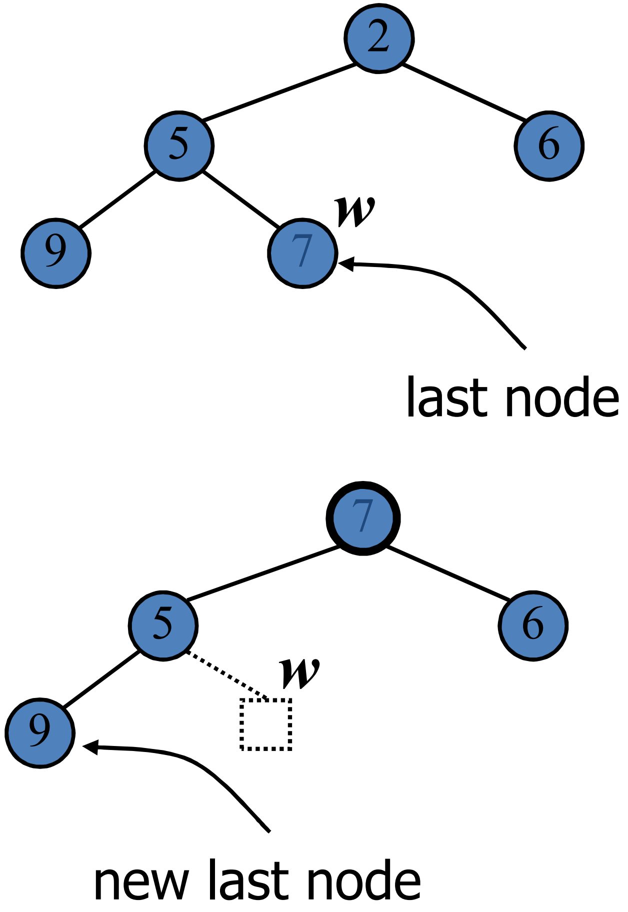
Up-heap Bubbling

- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an upward path from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k
- Since a heap has height $O(\log n)$, upheap runs in $O(\log n)$ time



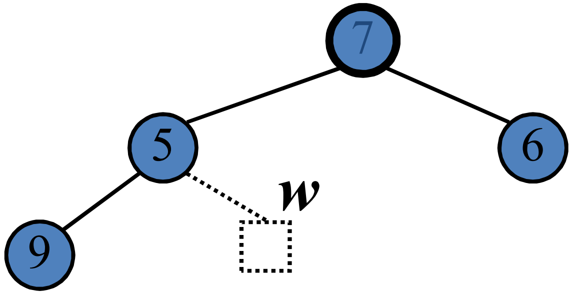
Removal from a Heap

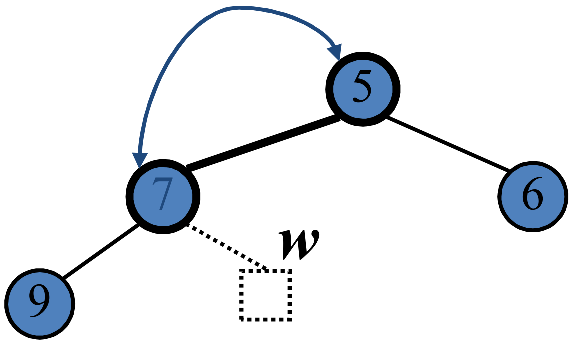
- **Method removeMin of the priority queue ADT corresponds to the removal of the root key from the heap**
- **The removal algorithm consists of three steps**
  - Replace the root key with the key of the last node *w*
  - Remove *w*
  - Restore the heap-order property (discussed next)



Down-heap Bubbling

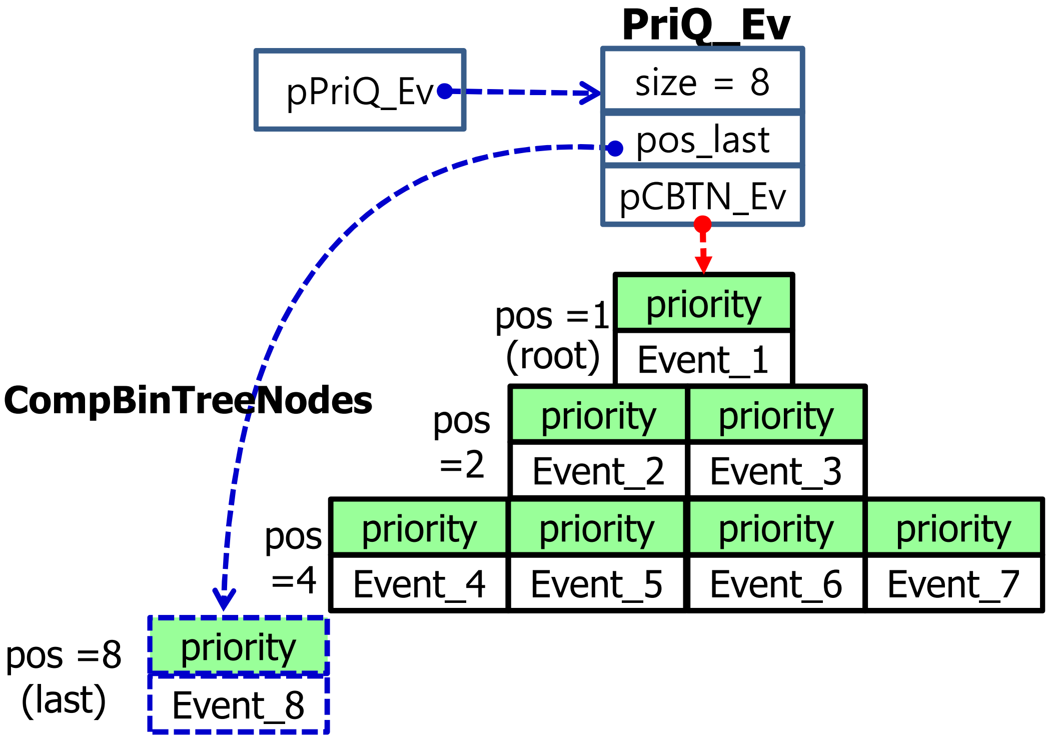
- After replacing the root key with the key k of the last node, the heap-order property may be violated
- Algorithm down-heap restores the heap-order property by swapping key k along a downward path from the root
- Down-heap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k
- Since a heap has height $O(\log n)$, down-heap runs in $O(\log n)$ time





Heap Priority Queue

◆ Heap Priority Queue



PriQ_Event.h

```
/* PriorityQueue_Event.h (1) */

#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;
```

```
/* PriorityQueue_Event.h (2) */

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
```

Lab. 11.2의 main() 프로그램 구성

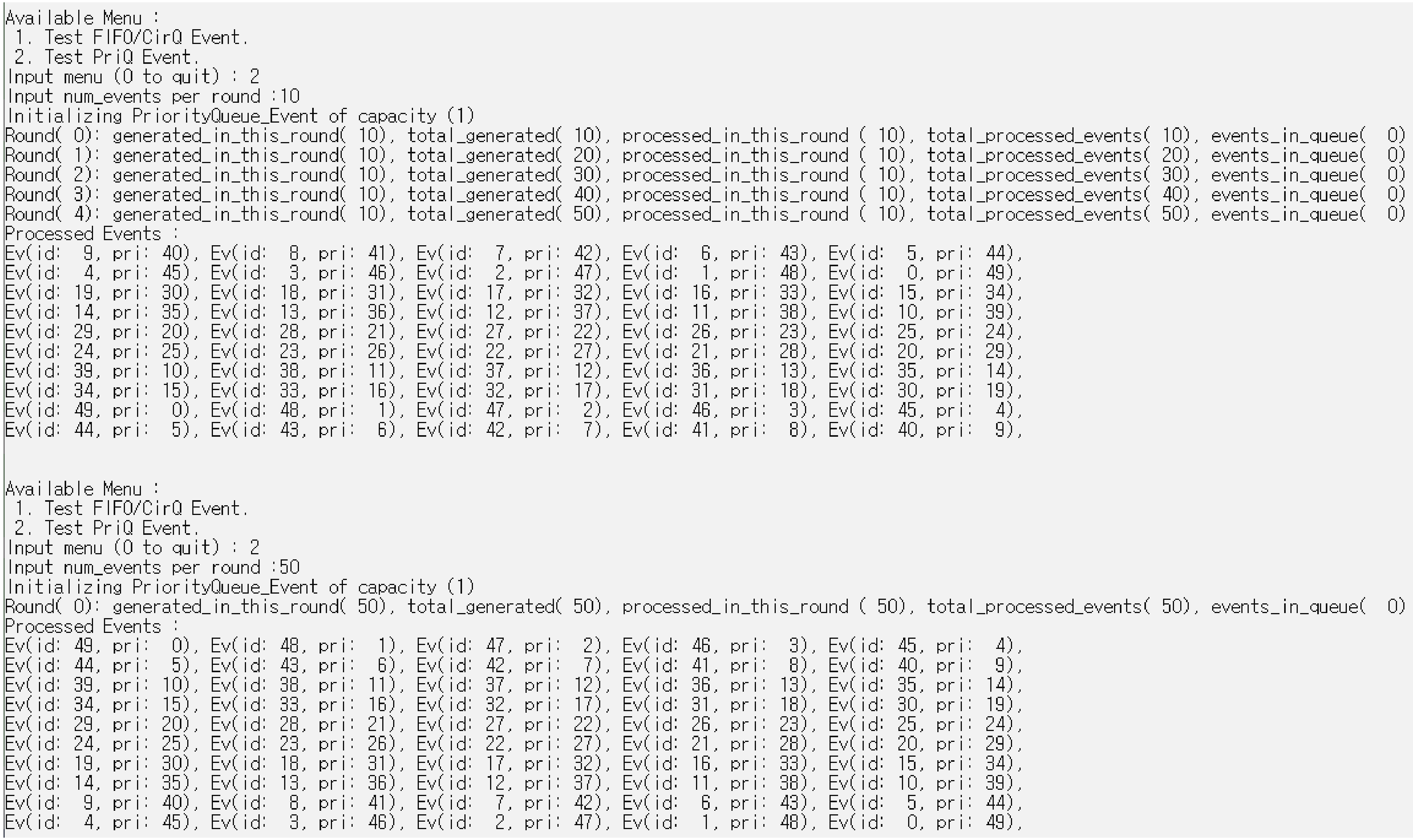
```
void test_PriQ_Event(FILE *fout, int max_events_per_round) /* (1) */
{
    PriQ_Event *pPriQ_Ev;
    Event *pEv = NULL;
    Event processed_events[TOTAL_NUM_EVENTS];
    int data;
    int total_processed_events = 0;
    int total_generated_events = 0;
    int num_events = 0;
    int num_generated_round = 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);
```

```
/* (2) */

 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);
         }
     } // end if
```

```
/* (3) */

    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++;
    }
```

```
  /* (4) */

      /* 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");
}
```

PriQ_Event 기능 시험 결과



```
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 of capacity (1)
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 of capacity (1)
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 11

Oral Test 11

11.1 Stack의 Last In First Out (LIFO)기본 동작 (push(), pop(), top())들이 어떻게 실행되는가에 대하여 설명하고, Queue의 First In First Out (FIFO) 동작이 Stack과 어떻게 차이가 나는가에 대하여 설명하라.

11.2 Circular buffer를 기반으로 FIFO queue를 구성하는 방법을 설명하고, queue의 기본 동작 (enQueue(), deQueue(), isFull(), isEmpty())이 어떻게 실행되는가에 대하여 설명하라.

11.3 Complete Binary Tree를 기반으로 구현되는 우선 순위 큐 (priority queue)에서 새로운 항목이 추가 될 때 실행되는 up-heap bubbling과 우선 순위 큐에서 우선 순위가 가장 높은 항목이 추출될 때 실행되는 down-heap bubbling의 동작이 어떻게 실행되는가에 대하여 설명하라.

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