프로그래밍언어

12. 동적 확장성 배열기반의 기본 자료 구조 – 스택, 큐, 환형 큐, 우선 순위 큐

교수 김 영 탁

영남대학교 기계IT대학 정보통신공학과

(Tel : +82-53-810-2497; Fax : +82-53-810-4742

http://antl.yu.ac.kr/; E-mail : ytkim@yu.ac.kr)

Outline

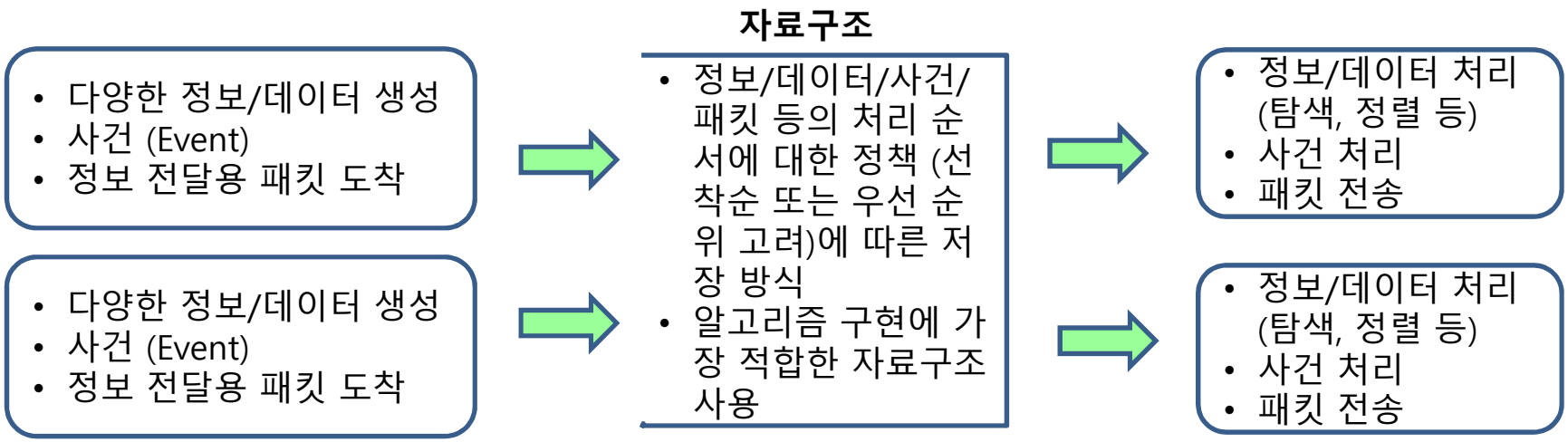
- 자료구조 (Data Structure)
- 동적 확장성이 있는 배열 (동적 배열 크기 조정)
- 스택 (Stack)
- 큐 (Queue)
- 환형큐 (Circular Queue) 및 응용
- 우선순위큐 (Priority Queue) 및 응용

자료구조 개요

자료 구조 (Data Structure) 란 ?

◆ 자료 구조 (Data Structure)

- 다양한 정보와 사건들을 효율적으로 처리 (탐색, 정렬, 순차적 처리, 우선 순위 고려 처리 등) 하기 위하여 체계적으로 저장할 수 있는 버퍼링 구조
- 그 문제 해결 (또는 응용 프로그램)에 적합한 효율적인 알고리즘을 구현하기 위하여 가장 적합한 자료구조를 사용할 수 있어야 함



대표적인 자료 구조

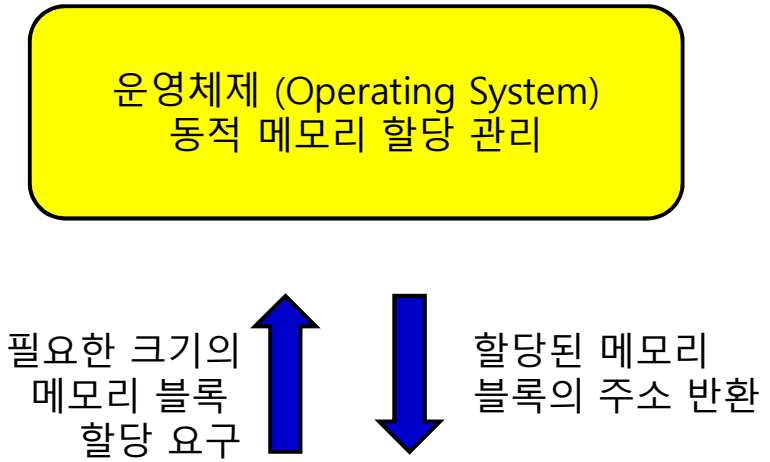
자료구조	특성
단순 배열	컴파일 단계에서 크기가 지정되어 변경되지 않는 크기의 배열
동적 확장성 배열	프로그램 실행 단계에서 크기가 지정되며, 프로그램 실행 중에 크기를 변경할 수 있는 확장성 배열
구조체 배열	구조체로 배열원소가 지정되는 배열
연결형 리스트	확장성이 있으며, 단일연결형 또는 이중연결형으로 구성 가능 다양한 컨테이너 자료구조의 내부적인 자료구조로도 활용됨
스택 (Stack)	Last In First Out (LIFO) 특성을 가짐. 배열 또는 연결형 리스트로 구현할 수 있음.
큐 (Queue)	First In First Out (FIFO) 특성을 가짐. 배열 또는 연결형 리스트로 구현할 수 있음.
우선 순위 큐 (Priority Queue)	컨테이너 내부에 가장 우선순위가 높은 데이터 항목을 추출할 수 있도록 관리하며, 배열 또는 자기 참조 구조체로 구현할 수 있음
이진 탐색 트리 (Binary Search Tree)	컨테이너 내부에 포함된 데이터 항목들을 정렬된 상태로 관리하여야 할 때 매우 효율적임. 단순 이진 탐색 트리의 경우 편중될 수 있으며, 편중된 경우 검색 성능이 저하되기 때문에, 밸런싱이 필요함.
해시 테이블 (Hash Table)	컨테이너 자료구조에 포함된 항목들을 문자열 (string) 또는 긴 숫자를 키 (key)로 사용하여 관리하여야 하는 경우, key로부터 해시 값을 구하고, 이 해시 값을 배열의 인덱스로 사용함.
맵(Map)	key와 항목 간에 1:1 관계가 유지되어야 하는 경우에 사용되며, 해시 테이블을 기반으로 구현할 수 있음
딕셔너리(Dictionary)	key와 항목 간에 1:N 관계가 유지되어야 하는 경우에 사용되며, 해시 테이블을 기반으로 구현할 수 있음
트라이(trie)	텍스트 또는 비트열의 검색을 신속하게 처리할 수 있도록 하며, 동일한 접두어 (prefix)를 사용하는 문장 또는 주소를 신속하게 검색할 수 있게 함. 예측 구문 (predictive text) 제시, longest prefix matching 등의 텍스트 처리 응용 분야에 사용됨.
그래프	정점 (vertex)/노드 (node)로 개체 (object)가 표현되고, 간선 (edge)/링크(link)들을 사용하여 개체 간의 관계를 표현하는 경우에 적합함. 그래프를 기반으로 경로 탐색, 최단거리경로 탐색, 신장트리 (spanning tree) 탐색 등에 활용됨.

확장성 있는 배열 (Expandable Array)

동적 (dynamic) 메모리 할당

◆ 동적 메모리 할당

- 실행 도중에 동적으로 메모리를 할당 받는 것
- 사용이 끝나면 시스템에 메모리를 반납
- 필요한 만큼만 할당을 받고 메모리를 매우 효율적으로 사용
- calloc(), malloc() 계열의 라이브러리 함수를 사용




```
int array_size;
int *intArray;

printf("Input array_size : ");
scanf("%d", &array_size);
intArray = (int *)calloc(array_size, sizeof(int));
if( intArray == NULL )
{
        ... // 오류 처리
}
 intArray[5] = 30;

 free(intArray);
```

동적 메모리 블록 할당 및 반환 함수

분류	함수 원형과 인수	기 능
동적 메모리 블록 할당 및 반환 <stdlib.h>	void* malloc(size_t size)	지정된 size 크기의 메모리 블록을 할당하고, 그 시작 주소를 void pointer로 반환
	void *calloc(size_t num_element, size_t element_size)	element_size 크기의 항목을 num_element개 할당하고, 0으로 초기화 한 후, 그 시작 주소를 void pointer로 반환
	void *realloc(void *p, size_t size)	이전에 할당받아 사용하고 있는 메모리 블록의 크기를 변경 p는 현재 사용하고 있는 메모리 블록의 주소, size는 변경하고자 하는 크기; 기존의 데이터 값은 유지된다
	void free(void *p)	동적 메모리 블록을 시스템에 반환; p는 현재 사용하였던 메모리 블록 주소

확장성이 있는 배열

◆ Expandable Array

- 필요에 따라 배열의 크기를 확장
- realloc() 함수 사용

```
/* ExpandableArray.h */

#ifndef EXPANDABLE_ARRAY
#define EXPANDABLE_ARRAY

typedef int Elem_type; // Elem_type can be defined as application-specific struct

Elem_type* expandArray(Elem_type *array, int curSize, int newSize);
void printArray(Elem_type *expandableArray, int array_size, int item_per_line);

#endif
```

```
/* ExpandableArray.c */
#include <stdio.h>
#include <stdlib.h>
#include "ExpandableArray.h"

Elem_type * expandArray(Elem_type *array, int curSize, int newSize)
{
    Elem_type *newArray;

    newArray = (Elem_type *)realloc(array, sizeof(Elem_type) * newSize);
    for (int i = curSize; i < newSize; i++)
    {
        newArray[i] = 0;  // 0 or other value that indicates initial value
    }
    return newArray;
}

void printArray(Elem_type *array, int array_size, int item_per_line = 10)
{
    int count = 0;
    for (int i = 0; i < array_size; i++)
    {
        printf("%5d", array[i]); // printing of element may be used here
        count++;
        if (count % item_per_line == 0)
            printf("\n");
    }
}
```

```c
/* main.c */
#include <stdio.h>
#include <stdlib.h>
#include "ExpandableArray.h"
#define INIT_ARRAY_SIZE 10
#define ITEMS_PER_LINE 10
/* typedef int Elem_Type; //defined in ExpandableArray.h */
void main()
{
    int array_size = INIT_ARRAY_SIZE;
    Elem_Type *DA_int = NULL, *old_DA;
    int old_array_size;

    DA_int = (Elem_Type *)calloc(array_size, sizeof(Elem_type));
    for (int i = 0; i < array_size; i++)
        DA_int[i] = i;
    printf("Dynamic array of %3d integers at address (%p):\n", array_size, DA_int);
    printArray(DA_int, array_size, ITEMS_PER_LINE);
    old_array_size = array_size;
    array_size = array_size * 2;
    old_DA = DA_int;
    DA_int = expandArray(old_DA, old_array_size, array_size);
    printf("Dynamic array of %3d integers at address (%p):\n", array_size, DA_int);
    printArray(DA_int, array_size, ITEMS_PER_LINE);
    printf("\n");
    free(DA_int);
}
```

```
Dynamic array of  10 integers at address (012550B8):
    0    1    2    3    4    5    6    7    8    9
Dynamic array of  20 integers at address (012423F8):
    0    1    2    3    4    5    6    7    8    9
    0    0    0    0    0    0    0    0    0    0
```

Example of Dynamic Array for Event Handling

```
/* 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 const 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* pEv);
void printEventArray(Event* pEv, int size, int items_per_line);
Event *genEvent(Event *pEv, int event_Gen_ID, int event_no, int event_pri);
#endif
```

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

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

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

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

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

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

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

Event *genEvent(Event *pEv, int event_Gen_ID, int event_no, int event_pri)
{
    pEv = (Event *)malloc(sizeof(Event));
    if (pEv == NULL)
        return NULL;
    pEv->event_gen_addr = event_Gen_ID;
    pEv->event_handler_addr = -1; // event handler is not defined yet !!
    pEv->event_no = event_no;
    pEv->event_pri = event_pri;

    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_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  ");
   }
}
```

```
/* main.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "ExpandableArray.h"
#include "Event.h"

#define INIT_ARRAY_SIZE 10
#define MAX_ARRAY_SIZE 100000

void main()
{
    Event *evArr = NULL;
    int array_size = INIT_ARRAY_SIZE;

    for (array_size = INIT_ARRAY_SIZE; array_size < MAX_ARRAY_SIZE;
            array_size *= 2)
    {
        evArr = (Event *)realloc(evArr, array_size *sizeof(Event));
        for (int n = 0; n < array_size; n++)
        {
            initEvent(& evArr[n], n, 0, MAX_ARRAY_SIZE - n, 1, ENQUEUED);
        }
        printf("Dynamic array of %8d Events (struct event size = %8d), address: %p\n",
            array_size, sizeof(Event), evArr);
        //printEventArray(evArr, array_size, EVENT_PER_LINE);
    }

    printf("\n");
    free(evArr);
}
```

◆ Execution Result

```
Dynamic array of       10 Events (struct event size =       20), address: 007D5288
Dynamic array of       20 Events (struct event size =       20), address: 007D5288
Dynamic array of       40 Events (struct event size =       20), address: 007D5288
Dynamic array of       80 Events (struct event size =       20), address: 007D5288
Dynamic array of      160 Events (struct event size =       20), address: 007D5288
Dynamic array of      320 Events (struct event size =       20), address: 007D8028
Dynamic array of      640 Events (struct event size =       20), address: 007D9958
Dynamic array of     1280 Events (struct event size =       20), address: 007DCB88
Dynamic array of     2560 Events (struct event size =       20), address: 007E2FB8
Dynamic array of     5120 Events (struct event size =       20), address: 007EF7E8
Dynamic array of    10240 Events (struct event size =       20), address: 00808818
Dynamic array of    20480 Events (struct event size =       20), address: 003B0068
Dynamic array of    40960 Events (struct event size =       20), address: 005E0040
Dynamic array of    81920 Events (struct event size =       20), address: 00890040

계속하려면 아무 키나 누르십시오 . . .
```

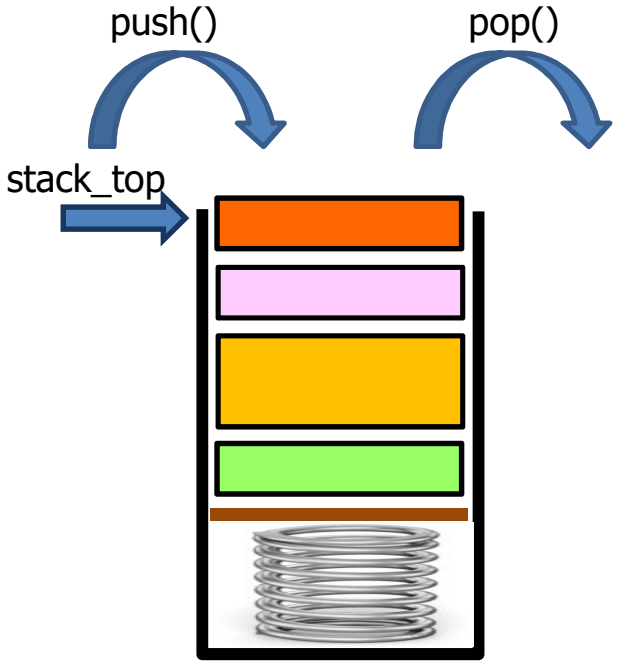
```
Dynamic array of       10 Events (struct event size =     2068), address: 007B6FF8
Dynamic array of       20 Events (struct event size =     2068), address: 007BD120
Dynamic array of       40 Events (struct event size =     2068), address: 007C72E0
Dynamic array of       80 Events (struct event size =     2068), address: 007DB630
Dynamic array of      160 Events (struct event size =     2068), address: 00803CA0
Dynamic array of      320 Events (struct event size =     2068), address: 00130040
Dynamic array of      640 Events (struct event size =     2068), address: 00370040
Dynamic array of     1280 Events (struct event size =     2068), address: 00870040
Dynamic array of     2560 Events (struct event size =     2068), address: 00B20040
Dynamic array of     5120 Events (struct event size =     2068), address: 01030040
Dynamic array of    10240 Events (struct event size =     2068), address: 01A50040
Dynamic array of    20480 Events (struct event size =     2068), address: 02E90040
Dynamic array of    40960 Events (struct event size =     2068), address: 05700040
Dynamic array of    81920 Events (struct event size =     2068), address: 0FD00040

계속하려면 아무 키나 누르십시오 . . .
```

스택과 큐 (Stack and Queue)

Last In First Out (LIFO) Stack

- The Stack stores arbitrary objects
- Insertions and deletions follow the last-in first-out (LIFO) or first-in last –out (FILO) scheme
- Think of a spring-loaded plate dispenser
- Main stack operations:
  - **push(Element elm)**: inserts an element
  - **Element pop()**: removes the last inserted element
- Auxiliary stack operations:
  - **Element top()**: returns the last inserted element without removing it
  - **integer size()**: returns the number of elements stored
  - **boolean empty()**: indicates whether no elements are stored



Applications of Stacks

◆ Direct applications

- Page-visited history in a Web browser
  (e.g., keeping recently visited web site URLs)
- Undo sequence in a text editor
- Chain of function/method calls in the C/C++ run-time system

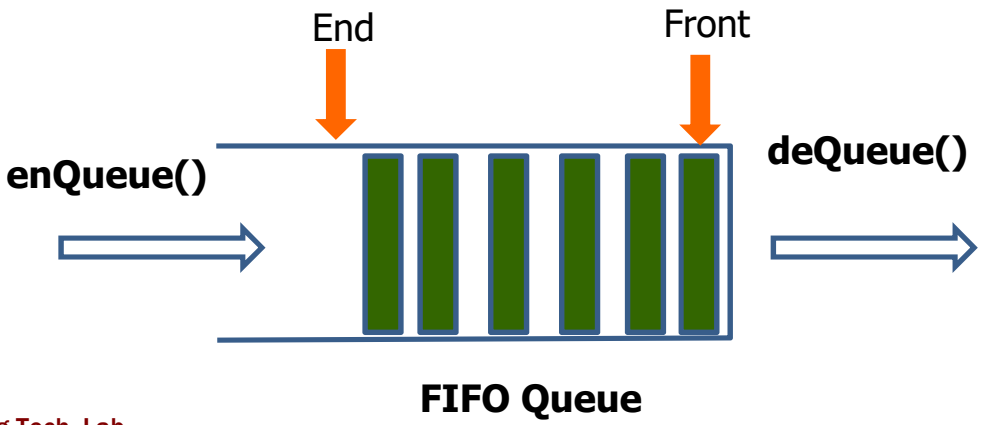
◆ Indirect applications

- Auxiliary data structure for algorithms
- Stack is used as a component of other data structures

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

큐 동작 (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

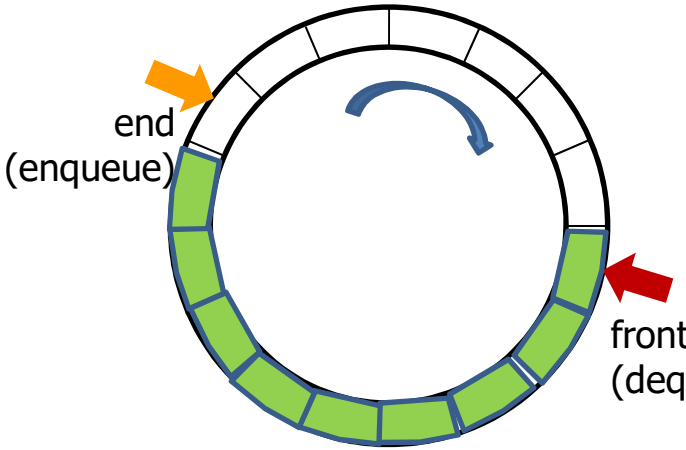
◆ Auxiliary queue operations:

- 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

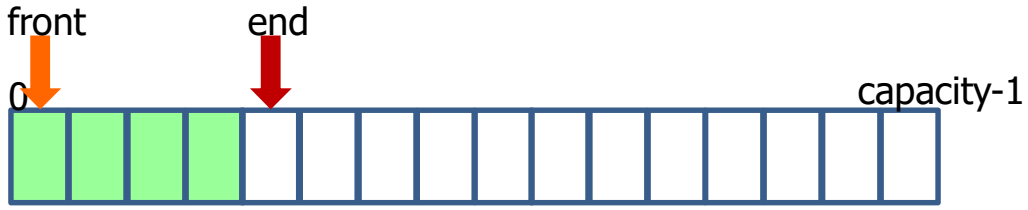
◆ Exceptions

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

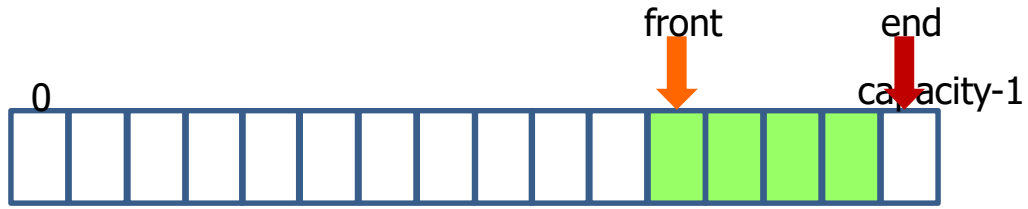
환형 버퍼 (Circular Buffer) 기반의 큐 구현



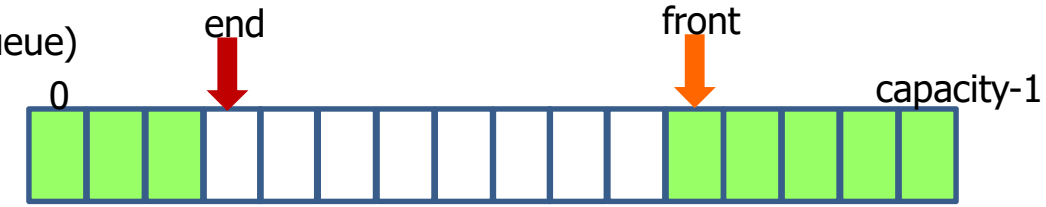
Operations in Circular Buffer



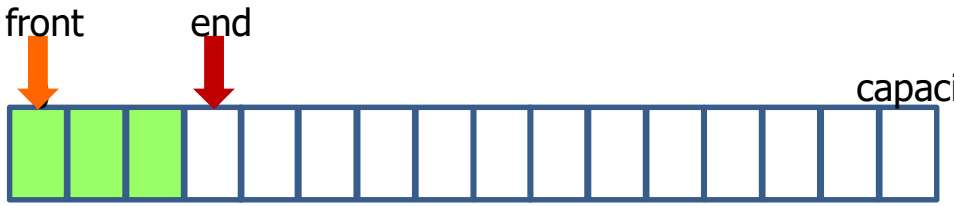
(a) Status of circular buffer after 4 enqueues after initialization



(b) Current status of circular buffer



(c) Circular buffer after 4 enqueues

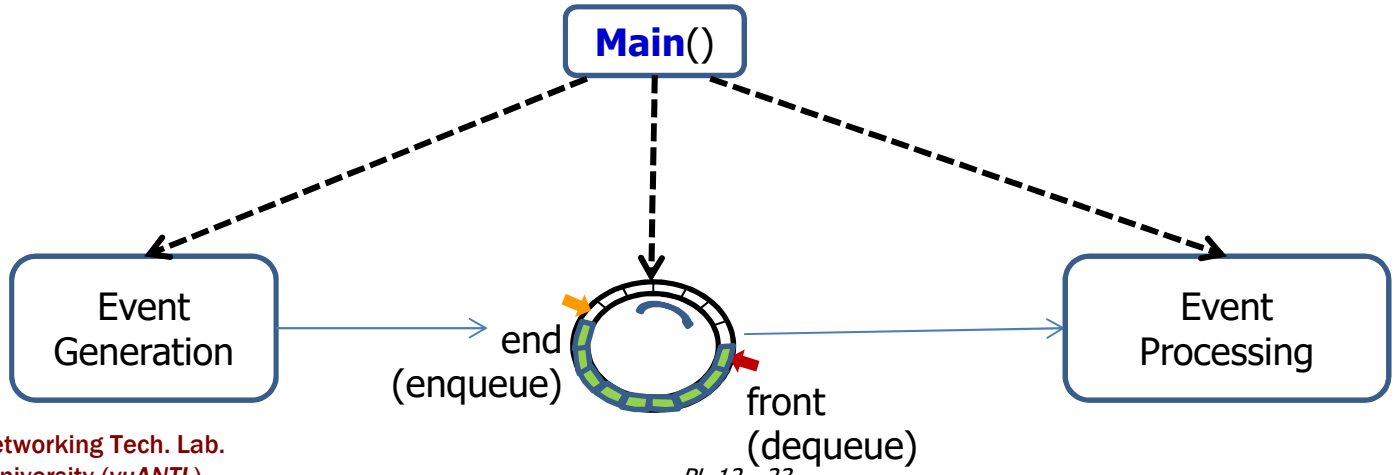


(d) Circular buffer after 5 dequeues

환형 버퍼 (Circular Buffer) 기반의 큐의 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 Priority Queue
  - Circular Queue (pCirQ) with First In First Out (FIFO) process ordering



Circular Queue for Event Handling

```
/* CirQ.h */

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

typedef struct
{
    Event *pEv; // circular queue for events
    int capacity;
    int front;
    int end;
    int num_elements;
} CirQ;

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

#endif
```

```
/* CirQ.c  (1) */
#include <stdio.h>
#include <stdlib.h>
#include "CirQ.h"

CirQ *initCirQ(CirQ *pCirQ, int capacity)
{
   Event * pEv;

   pEv = (Event *)calloc(capacity, sizeof(Event));
   if (pEv == NULL)
   {
      printf("Error in memory allocation for Event array of size (%d)\n", capacity);
      exit;
   }
   pCirQ-> pEv = pEv;
   pCirQ->capacity = capacity;
   pCirQ->front = pCirQ->end = 0;
   pCirQ->num_elements = 0;
   return pCirQ;
}
```

```
/* CirQ.c  (2) */

void fprintCirQ(FILE *fout, CirQ *pCirQ)
{
    Event ev;
    int index;

    if ((pCirQ == NULL) || (pCirQ-> pEv == NULL))
    {
       printf("Error in printArrayQueue: pCirQ is NULL or pCirQ->array is NULL");
       exit;
    }
    fprintf(fout, "  %2d Elements in CirQ_Event (index_front:%2d) :\n     ",
        pCirQ->num_elements, pCirQ->front);
    if (isCirQEmpty(pCirQ))
    {
       fprintf(fout, "pCirQ_Event is Empty");
    }
    else
    {
        for (int i = 0; i < pCirQ->num_elements; i++)
        {
            index = pCirQ->front + i;
            if (index >= pCirQ->capacity)
               index = index % pCirQ->capacity;
            ev = pCirQ-> pEv[index];
            fprintEvent(fout, &ev);
            if ((((i + 1) % EVENT_PER_LINE) == 0) && ((i + 1) != pCirQ->num_elements))
               fprintf(fout, "\n     ");
        }
    }
    fprintf(fout, "\n");
}
```

```
/* CirQ.c  (3) */

bool isCirQFull(CirQ *pCirQ)
{
    if (pCirQ->num_elements == pCirQ->capacity)
        return true;
    else
        return false;
}

bool isCirQEmpty(CirQ *pCirQ)
{
    if (pCirQ->num_elements == 0)
        return true;
    else
        return false;
}

Event *enCirQ(CirQ *pCirQ, Event ev)
{
    if (isCirQFull(pCirQ))
    {
        return NULL;
    }
    pCirQ-> pEv[pCirQ->end] = ev;
    pCirQ->num_elements++;
    pCirQ->end++;
    if (pCirQ->end >= pCirQ->capacity)
         pCirQ->end = pCirQ->end % pCirQ->capacity;
    return &(pCirQ-> pEv[pCirQ->end]);
}
```

```
/* CirQ.c  (4) */

Event *deCirQ(CirQ *pCirQ)
{
    if (isCirQEmpty(pCirQ))
        return NULL;

    Event *pEv = &(pCirQ-> pEv[pCirQ->front]);
    pCirQ->front++;
    if (pCirQ->front >= pCirQ->capacity)
        pCirQ->front = pCirQ->front % pCirQ->capacity;
    pCirQ->num_elements--;
    return pEv;
}

void delCirQ(CirQ *pCirQ)
{
    if (pCirQ->pEv != NULL)
        free(pCirQ->pEv);
    pCirQ->pEv = NULL;
    pCirQ->capacity = 0;
    pCirQ->front = pCirQ->end = 0;
    pCirQ->num_elements = 0;
}
```

```
/* main.c  (1) */

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

#define QUEUE_CAPACITY 10
#define EVENT_GEN 0
#define TOTAL_NUM_EVENTS 50
#define MAX_EVENTS_PER_ROUND 5
#define MAX_ROUND 100

void main()
{
      FILE *fout;
      CirQ *pCirQ_Ev;
      Event ev, *pEv = NULL;
      int processed_events = 0;
      int generated_events = 0;
      int num_events = 0;

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

```
/* main.c  (2) */
      pCirQ_Ev = (CirQ *)malloc(sizeof(CirQ));
      fprintf(fout, "Initializing event circular queue of capacity (%d)\n", QUEUE_CAPACITY);
      pCirQ_Ev = initCirQ(pCirQ_Ev, QUEUE_CAPACITY);
      srand(time(0));
      for (int round = 0; round < MAX_ROUND; round++)
      {
            if (generated_events < TOTAL_NUM_EVENTS)
            {
                num_events = rand() % MAX_EVENTS_PER_ROUND;
                if ((generated_events + num_events) > TOTAL_NUM_EVENTS)
                   num_events = TOTAL_NUM_EVENTS - generated_events;
                fprintf(fout, "generate and enque %2d events\n", num_events);
                for (int i = 0; i < num_events; i++)
                {
                    if (isCirQFull(pCirQ_Ev))
                    {
                       fprintf(fout, "  !!! CirQ_Event is full --> skip generation and
                          enqueueing of event. \n");
                       break;
                    }
                    /* Event *genEvent(Event *pEv, int event_Gen_ID, int event_no, int event_pri) */
                    pEv = genEvent(pEv, EVENT_GEN, generated_events,
                      TOTAL_NUM_EVENTS - generated_events - 1);
                    fprintf(fout, ">>> Enqueue event = ");
                    fprintEvent(fout, pEv);    fprintf(fout, "\n");
                    enCirQ(pCirQ_Ev, *pEv);
                    fprintCirQ(fout, pCirQ_Ev);
                    free(pEv);
                    generated_events++;
                } // end for
            }  // end if
```

```
/* main.c  (3) */

            num_events = rand() % MAX_EVENTS_PER_ROUND;
            if ((processed_events + num_events) > TOTAL_NUM_EVENTS)
                num_events = TOTAL_NUM_EVENTS - processed_events;
            fprintf(fout, "dequeue %2d events\n", num_events);
            for (int i = 0; i < num_events; i++)
            {
                if (isCirQEmpty(pCirQ_Ev))
                    break;
                pEv = deCirQ(pCirQ_Ev);
                if (pEv != NULL)
                {
                    fprintf(fout, "<<< Dequed event = ");
                    fprintEvent(fout, pEv); fprintf(fout, "\n");
                    processed_events++;
                }
                fprintCirQ(fout, pCirQ_Ev);
            } // end for

            /* Monitoring simulation status */
            fprintf(fout, "\nRound(%2d): total_generated_events(%3d),
            total_processed_events(%3d), pCirQ capacity (%2d),
            events_in_queue(%3d)\n",
                round, generated_events, processed_events, pCirQ_Ev->capacity,
                pCirQ_Ev->num_elements);
            if (processed_events >= TOTAL_NUM_EVENTS)
                break;
        } // end for()
        delCirQ(pCirQ_Ev);
}
```

◆ Result (1)

```
Initializing integer stack of capacity (10)
generate and enque  0 events
dequeue  2 events

Round( 0): total_generated_events(  0), total_processed_events(  0), CirQ capacity (10), events_in_queue(  0)
generate and enque  1 events
>>> Enqueue event = Ev(no:  0, pri:49)
   1 Elements in CirQ_Event (index_front: 0) :
     Ev(no:  0, pri:49)
dequeue  3 events
<<< Dequed event = Ev(no:  0, pri:49)
   0 Elements in CirQ_Event (index_front: 1) :
     CirQ_Event is Empty

Round( 1): total_generated_events(  1), total_processed_events(  1), CirQ capacity (10), events_in_queue(  0)
generate and enque  0 events
dequeue  2 events

Round( 2): total_generated_events(  1), total_processed_events(  1), CirQ capacity (10), events_in_queue(  0)
generate and enque  0 events
dequeue  3 events

Round( 3): total_generated_events(  1), total_processed_events(  1), CirQ capacity (10), events_in_queue(  0)
generate and enque  1 events
>>> Enqueue event = Ev(no:  1, pri:48)
   1 Elements in CirQ_Event (index_front: 1) :
     Ev(no:  1, pri:48)
dequeue  2 events
<<< Dequed event = Ev(no:  1, pri:48)
   0 Elements in CirQ_Event (index_front: 2) :
     CirQ_Event is Empty

Round( 4): total_generated_events(  2), total_processed_events(  2), CirQ capacity (10), events_in_queue(  0)
```

◆ Result (2)

```
Round(32): total_generated_events( 50), total_processed_events( 44), CirQ capacity (10), events_in_queue(  6)
dequeue  3 events
<<< Dequed event = Ev(no: 44, pri: 5)
   5 Elements in CirQ_Event (index_front: 5) :
     Ev(no: 45, pri: 4)  Ev(no: 46, pri: 3)  Ev(no: 47, pri: 2)  Ev(no: 48, pri: 1)  Ev(no: 49, pri: 0)
<<< Dequed event = Ev(no: 45, pri: 4)
   4 Elements in CirQ_Event (index_front: 6) :
     Ev(no: 46, pri: 3)  Ev(no: 47, pri: 2)  Ev(no: 48, pri: 1)  Ev(no: 49, pri: 0)
<<< Dequed event = Ev(no: 46, pri: 3)
   3 Elements in CirQ_Event (index_front: 7) :
     Ev(no: 47, pri: 2)  Ev(no: 48, pri: 1)  Ev(no: 49, pri: 0)

Round(33): total_generated_events( 50), total_processed_events( 47), CirQ capacity (10), events_in_queue(  3)
dequeue  0 events

Round(34): total_generated_events( 50), total_processed_events( 47), CirQ capacity (10), events_in_queue(  3)
dequeue  3 events
<<< Dequed event = Ev(no: 47, pri: 2)
   2 Elements in CirQ_Event (index_front: 8) :
     Ev(no: 48, pri: 1)  Ev(no: 49, pri: 0)
<<< Dequed event = Ev(no: 48, pri: 1)
   1 Elements in CirQ_Event (index_front: 9) :
     Ev(no: 49, pri: 0)
<<< Dequed event = Ev(no: 49, pri: 0)
   0 Elements in CirQ_Event (index_front: 0) :
     CirQ_Event is Empty

Round(35): total_generated_events( 50), total_processed_events( 50), CirQ capacity (10), events_in_queue(  0)
```

힙과 우선순위 큐 (Heap and Priority Queue)

Priority Queue (우선순위큐)

- **우선순위큐 구성**
  - (키, 값) 쌍의 구조체 원소를 저장하며, 키는 우선순위를 나타냄
  - 키 값이 작을 수록 우선 순위는 높음
- **우선순위큐의 구현**
  - 완전이진트리 (complete binary tree) 기반의 힙우선순위큐 (heap priority queue)
- **우선순위큐 응용 분야:**
  - 주식거래 (stock market)
  - 경매 (auction)
  - 항공편의 대기 승객 우선 순위 관리

힙 (Heap)

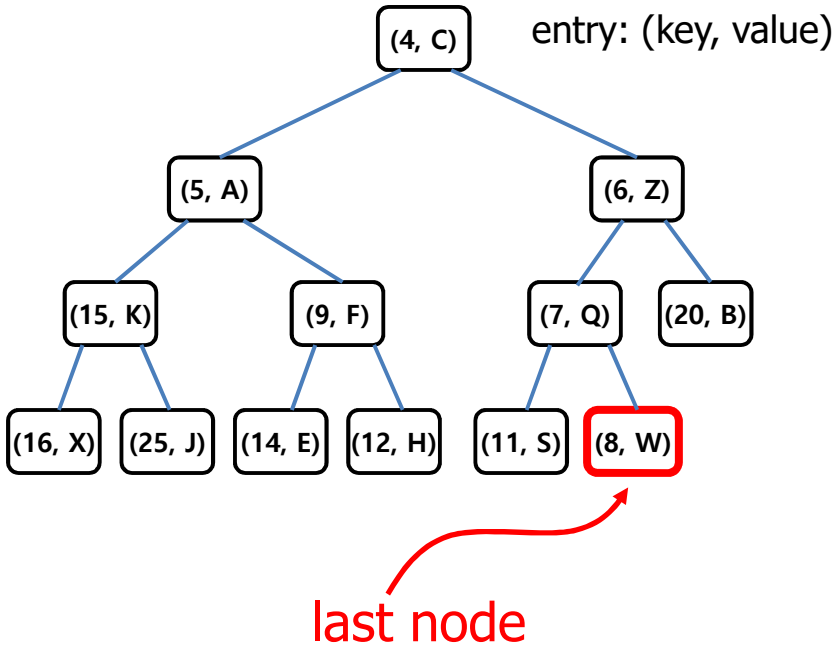
◆ 힙(heap)

- 완전이진트리 (complete binary tree) 구조로 데이터를 저장
- 힙순서(Heap-Order): 힙에 포함되는 모든 원소들의 순서

  $key(v) \geq key(parent(v))$

◆ 완전이진트리 (Complete Binary Tree):

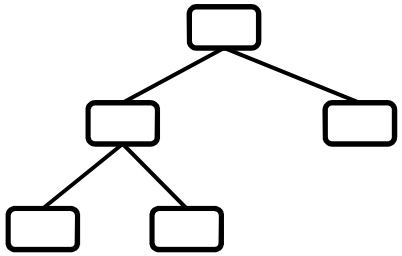
- 힙의 높이 (height)가 h 일 때
- depth $i = 0, \ldots, h - 1$에서 각각 최대 2^i 노드가 포함됨
- depth $h - 1$에서 왼쪽에서 오른쪽으로 차례대로 채워짐
- 힙의 마지막 노드 (last node)는 depth가 가장 깊은 (h-1)레벨의 가장 오른쪽 노드



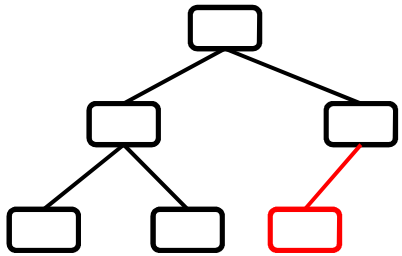
완전이진트리 (Complete Binary Tree)

◆ 완전이진트리의 특성

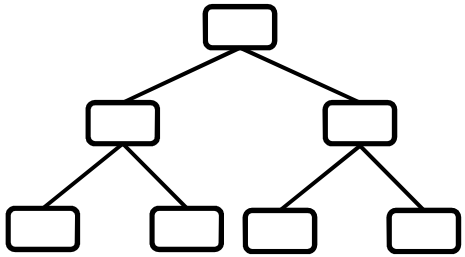
- 높이가 h인 힙 T는 완전이진트리로 구현되며, level = 0, 1, 2, …, h-1
- 각 level ($0 \le i \le h-1$) 에서 최대 2^i 노드를 가지게 됨.
- level (h-1)에서 $2^{(h-1)}$ 개의 노드가 채워지면, level h가 구성되기 시작함



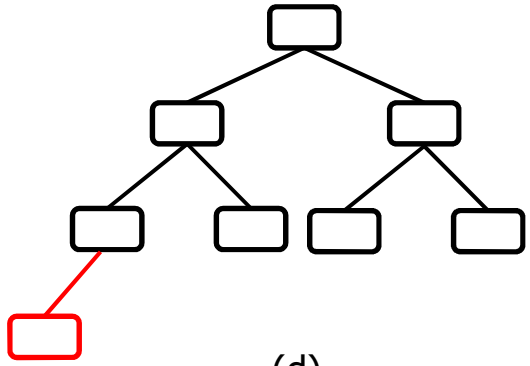
(a)



(b)



(c)

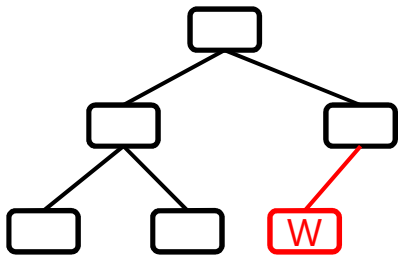


(d)

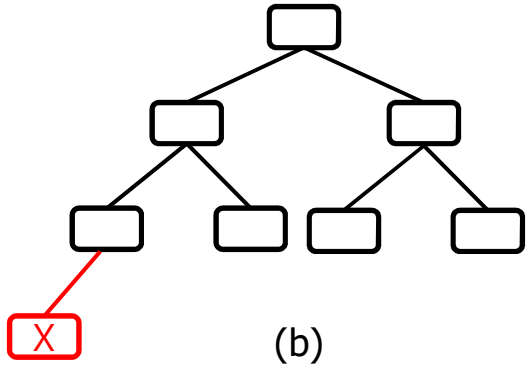
배열을 사용한 완전이진트리의 구현

◆ 배열 기반의 완전이진 트리 구현

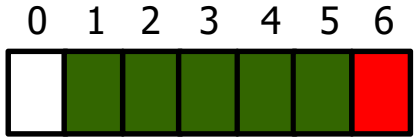
- 완전이진트리 (CBT)의 root가 v일 때, pos(v) = 1
- 만약 node u의 왼쪽 자식이 lc일 때, pos(lc) = 2 × pos(u)
- 만약 node u의 오른쪽 자식이 rc일 때, pos(rc) = 2 × pos(u) + 1



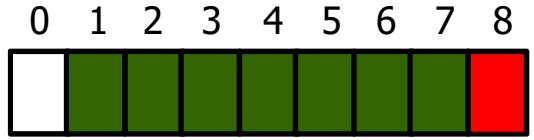
(a)



(b)



(c)

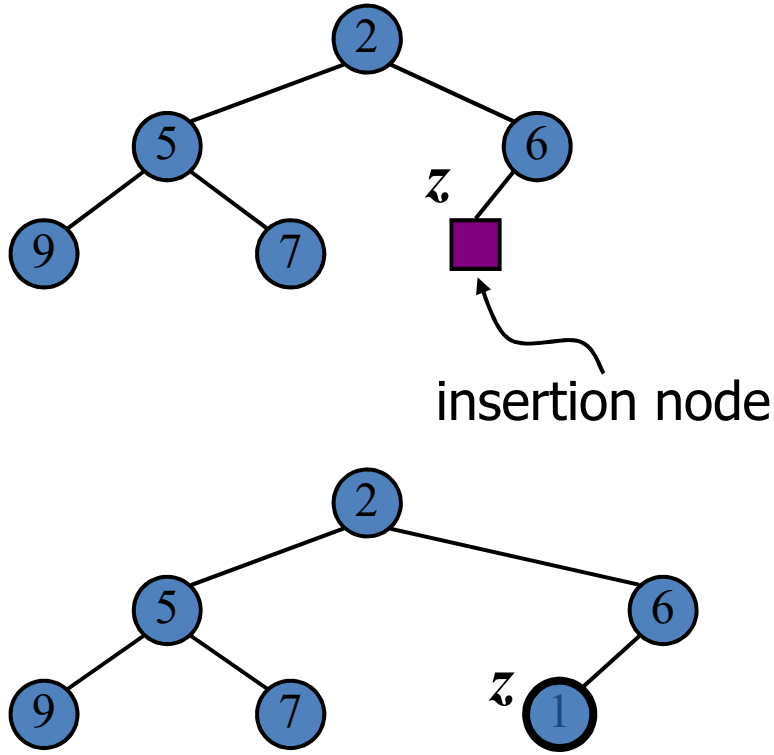


(d)

힙에 새로운 노드 삽입

◆ **힙에 새로운 key *k* 의 노드를 삽입할 때, 3 단계 실행**

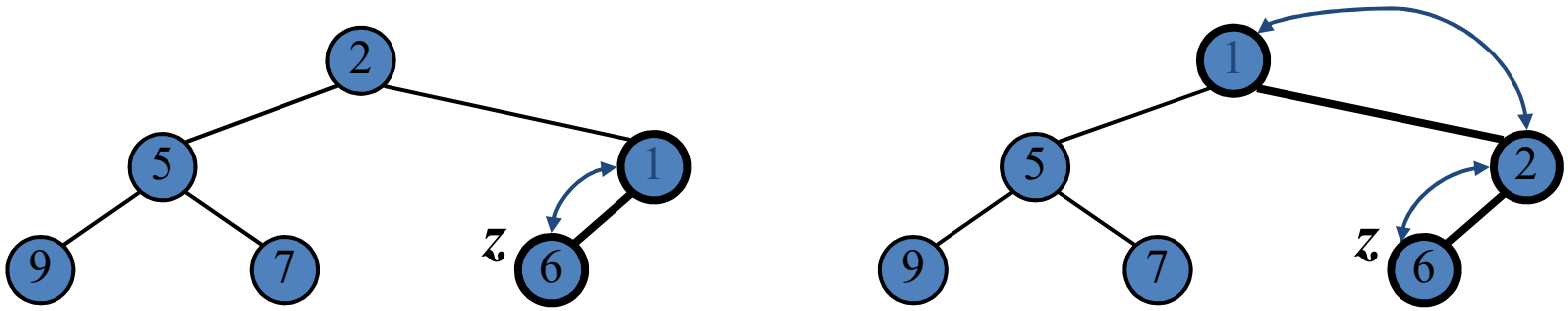
- 현재 힙 (완전이진트리)의 마지막 노드 (lost node)의 다음 위치 z 결정
- key K를 ***z*** 위치에 저장
- z 위치로 부터 root 노드까지의 경로에 있는 노드들을 heap order에 따라 정렬 (up heap bubbling)



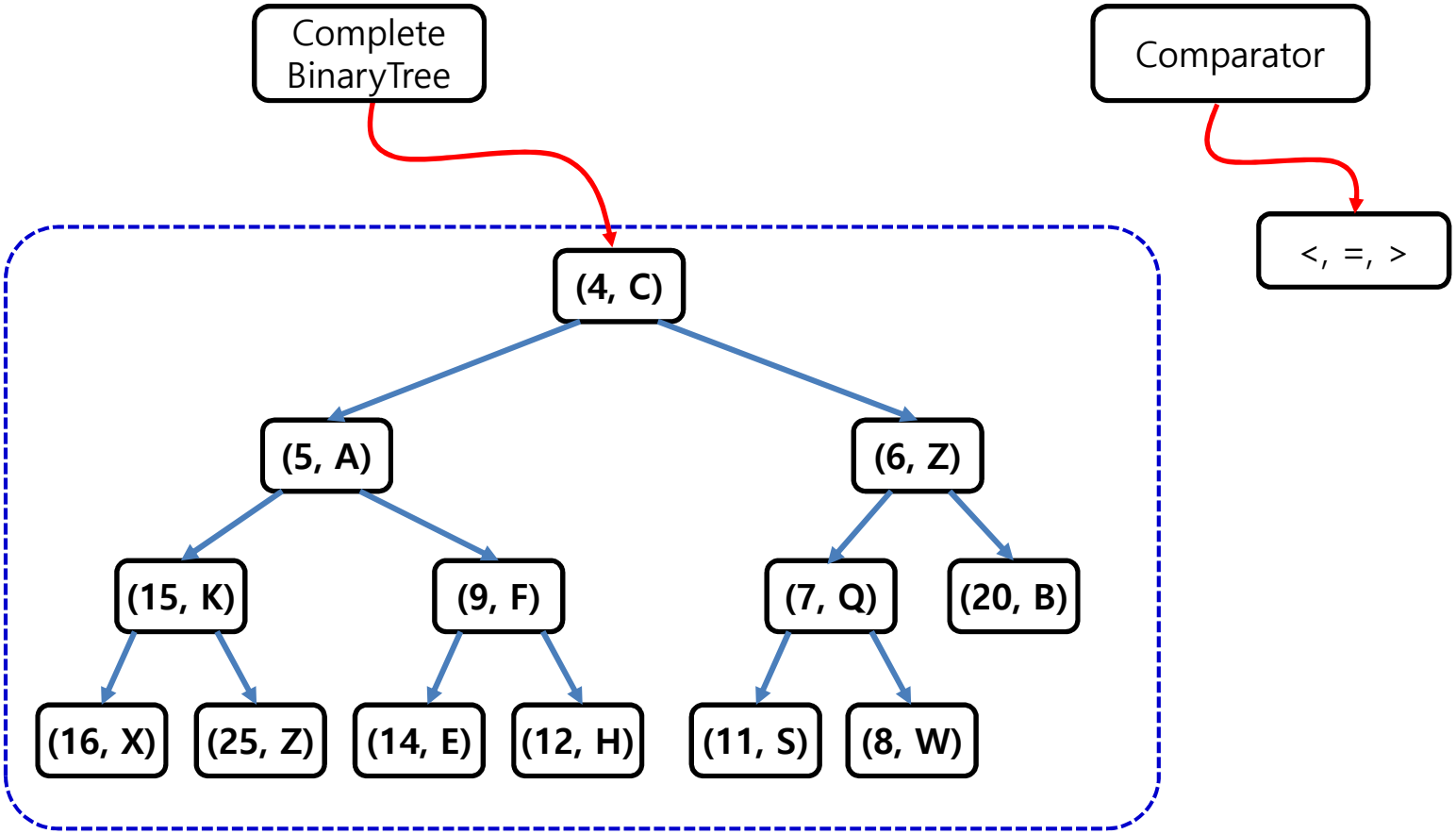
Up-heap Bubbling

◆ Up-heap Bubbling

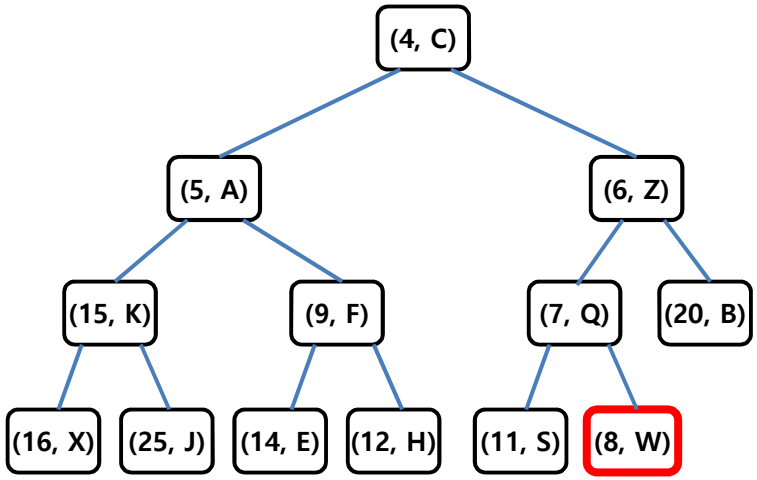
- 새로운 key k 를 노드 z에 삽입한 후, 노드 z로 부터 root 노드까지의 값들이 힙 정렬 순서 (heap order)가 아닐 수 있음
- Up-heap bubbling은 노드 z 에 저장된 key k 를 root노드까지의 경로상에 있는 다른 노드들과 비교하며, heap order가 유지될 수 있도록 조정
- Up-heap bubbling은 key k 가 root 노드에 도달하거나, 부모 노드의 key 값이 k보다 작을 때 중지됨
- 노드 개수가 n일 때, 힙은 $O(\log_2 n)$의 높이를 가지므로, up-heap bubbling은 최대 $O(\log_2 n)$ 번의 swapping이 실행될 수 있음



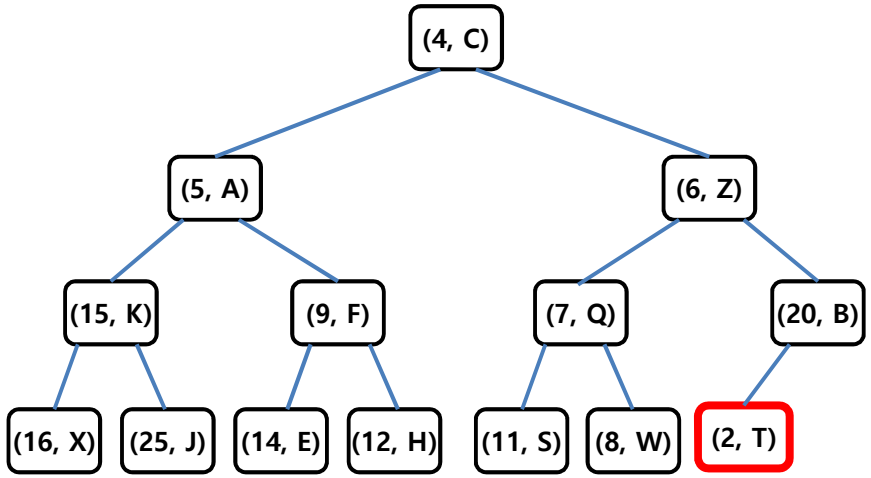
◆ Insertion with up-heap bubbling (1)



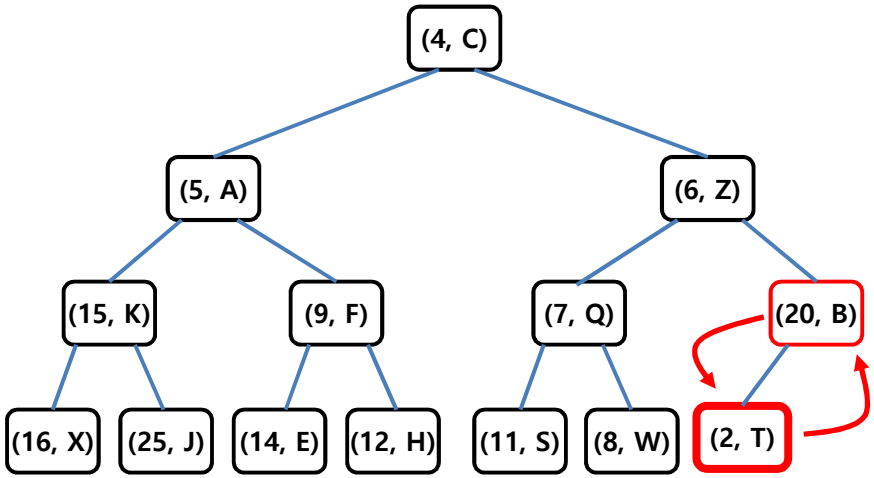
◆ Insertion with up-heap bubbling (2)



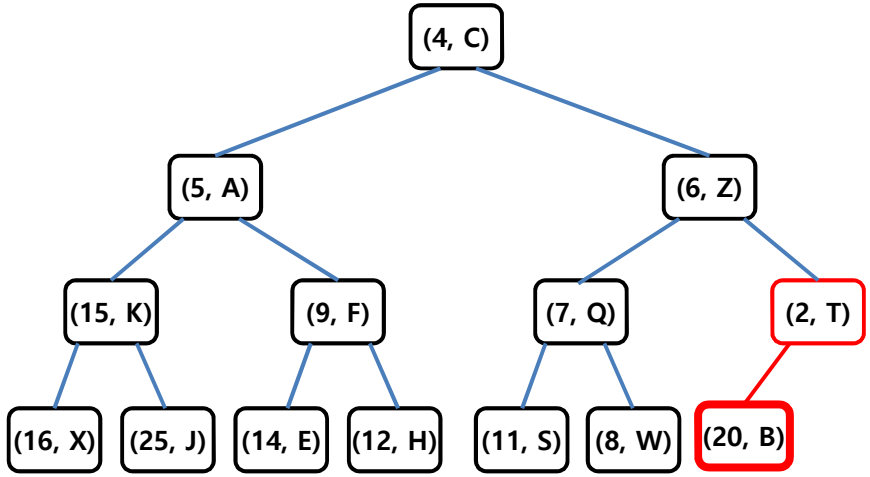
(a) last status



(b) insert a new node

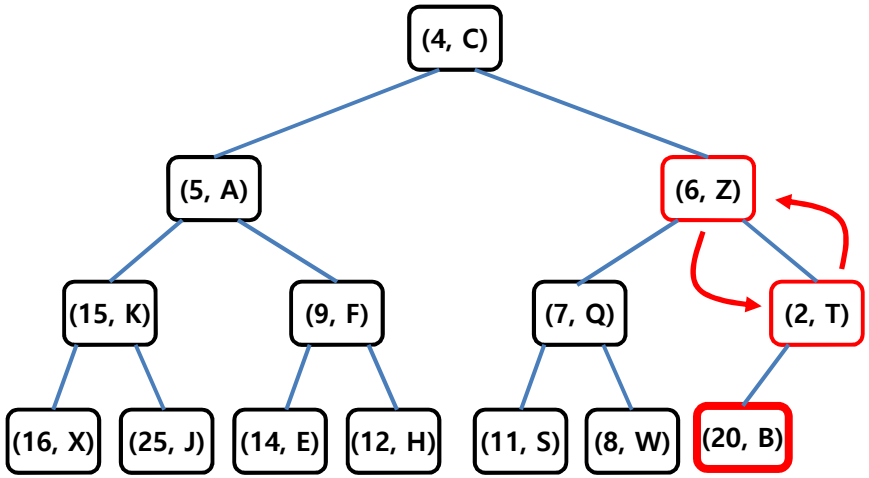


(c) swapping

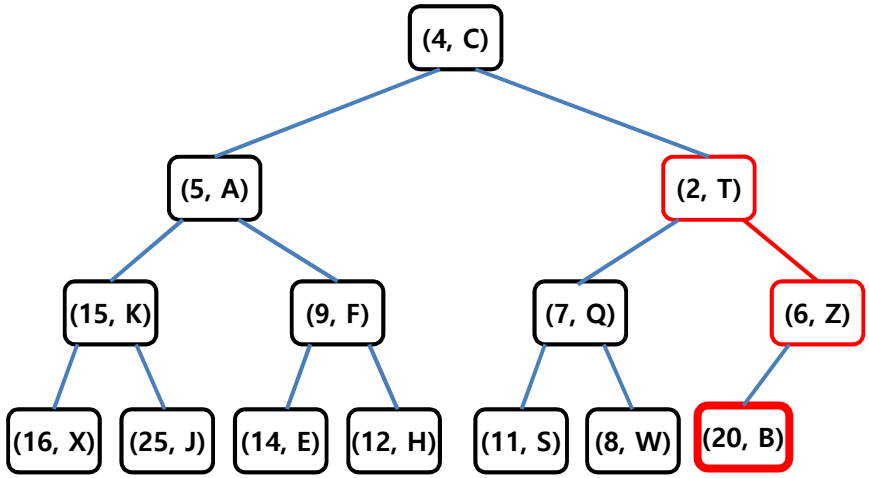


(d) after swapping

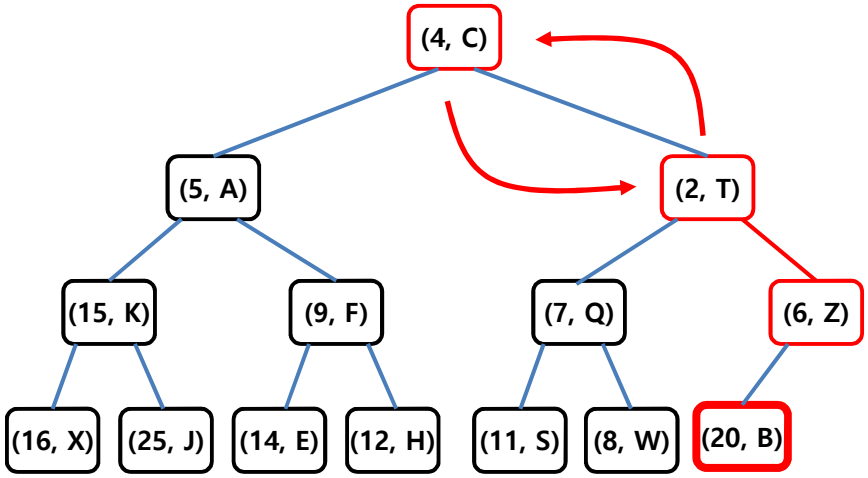
◆ Insertion with up-heap bubbling (3)



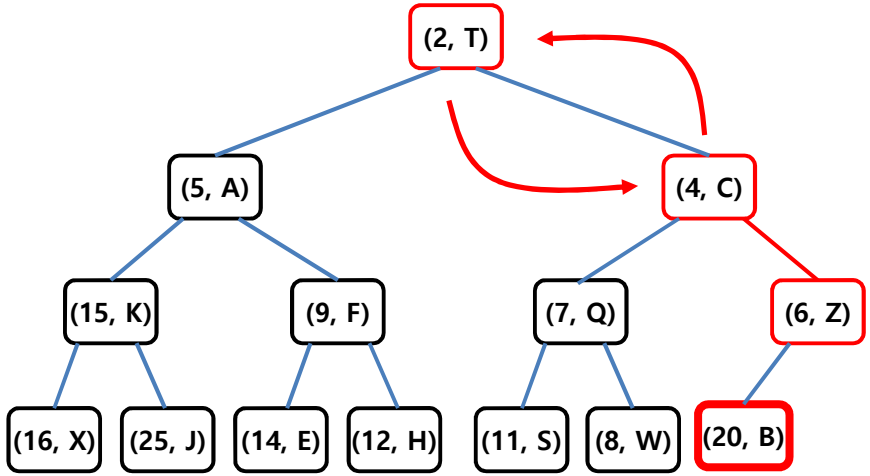
(e) swapping



(e) after swapping



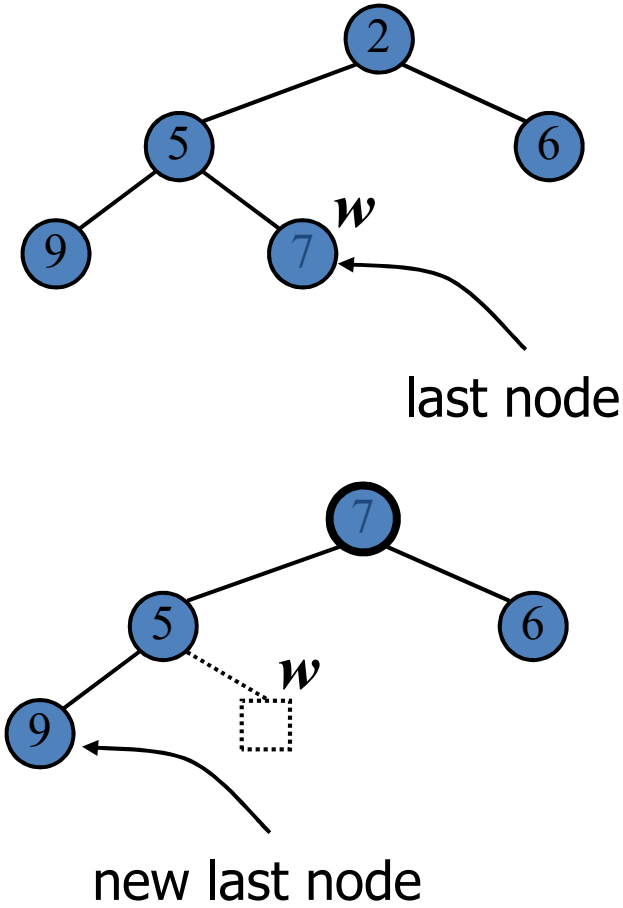
(e) swapping



(e) after swapping

힙으로 부터 최우선 순위 노드(key)의 제거

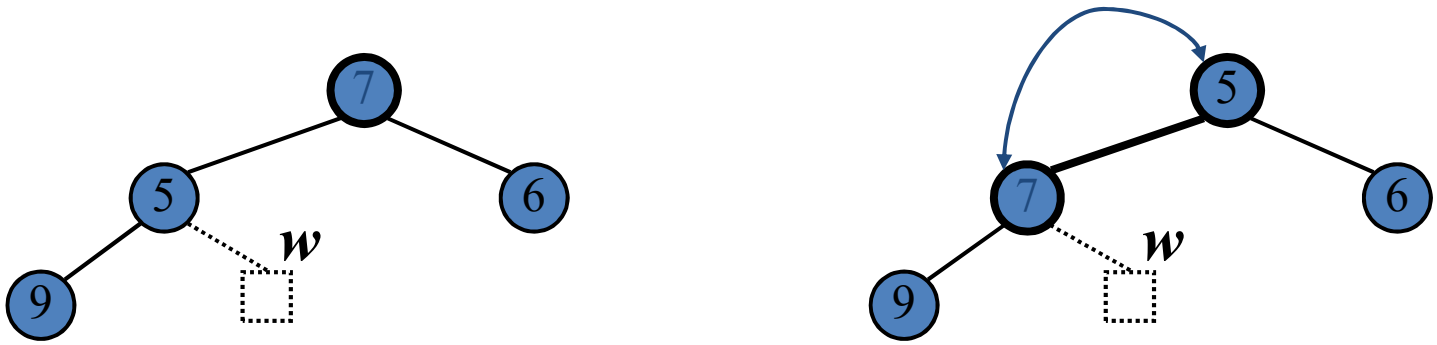
- **힙에서 최우선 순위의 노드 (key)를 제거할 때, 다음 3 단계로 실행**
  - 최우선 순위의 key를 root 노드로 부터 읽어 낸 후, last node 위치의 key ***w*** 를 root 노드 위치로 복사
  - last node의 key ***w***를 삭제하고, 그 직전의 노드로 last node를 조정
  - root 노드로 부터 하위 노드로 heap order로 재조정 (down-heap bubbling)



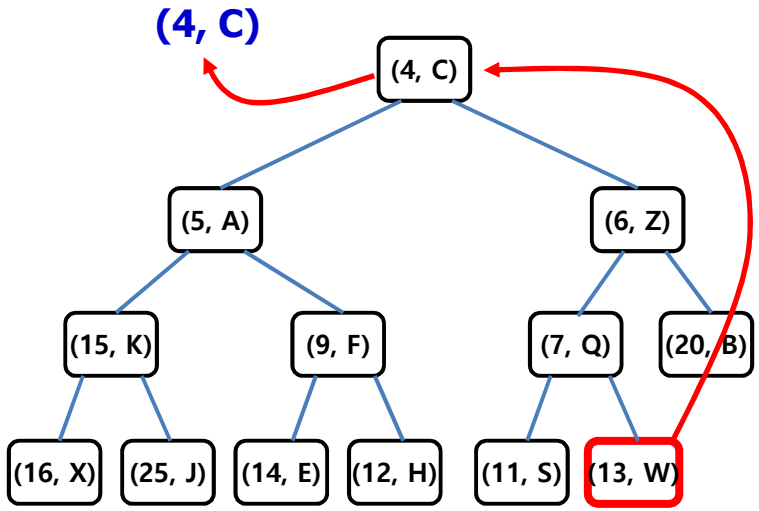
Down-heap Bubbling

◆ Down-heap Bubbling

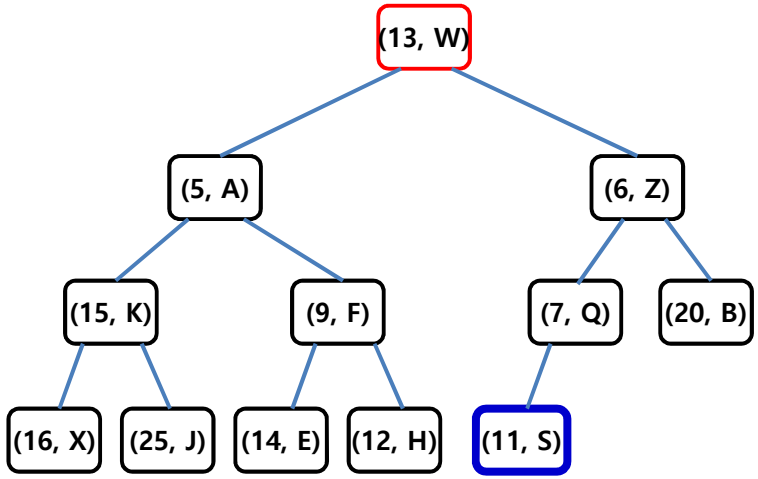
- root 노드에 있는 최우선 순위 key *k* 를 제거한 후, last node의 key *w*를 root 노드 위치로 복사하면 root 노드로 부터 하위 노드들 간에 heap order가 유지되지 않을 수 있음
- Down-heap bubbling은 root 노드와 하위 노드로의 힙 순서 (heap order)를 유지하도록 swapping을 실행하며 조정
- Down-heap bubbling은 key *k* 가 leaf 노드에 도달하거나 자식 노드가 key *k* 보다 더 큰 값일 경우 중지하게 됨
- 노드 개수가 n일 때, 힙은 $O(\log_2 n)$의 높이를 가지므로, down-heap bubbling은 최대 $O(\log_2 n)$ 번의 swapping이 실행될 수 있음



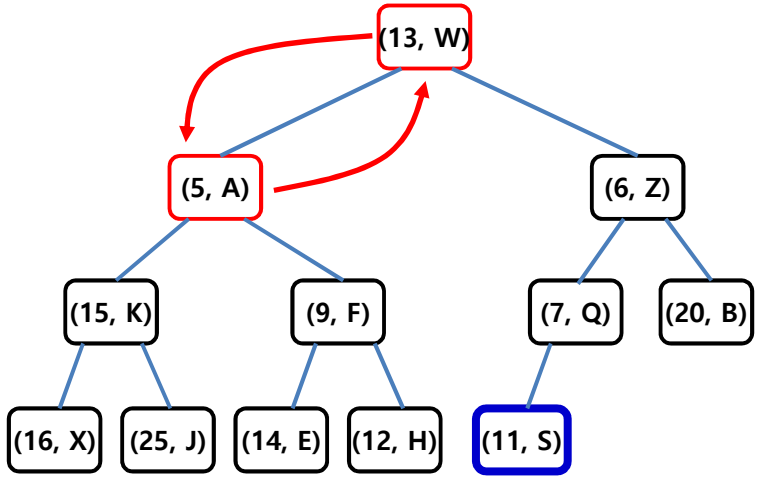
◆ Down-heap Bubbling after a RemoveMin() (1)



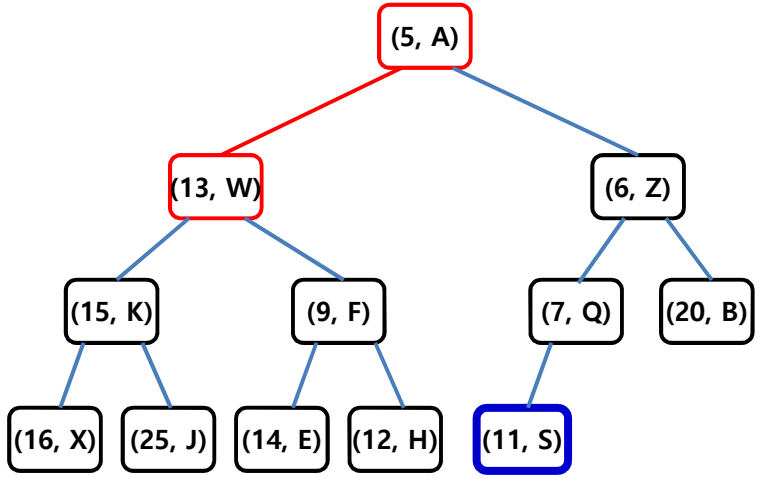
(a) last status



(b) after changing root by the last node

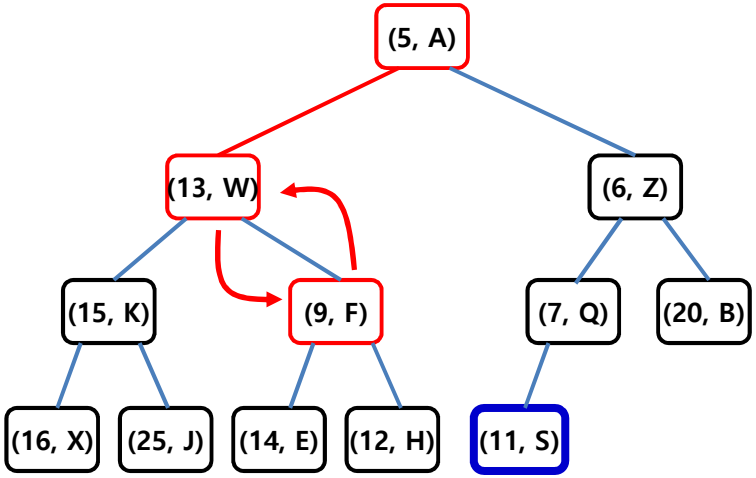


(c) compare and swap

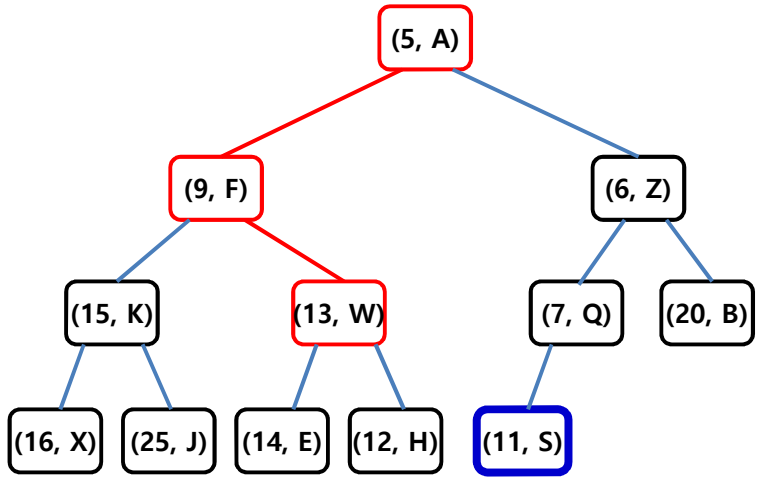


(d) after swapping

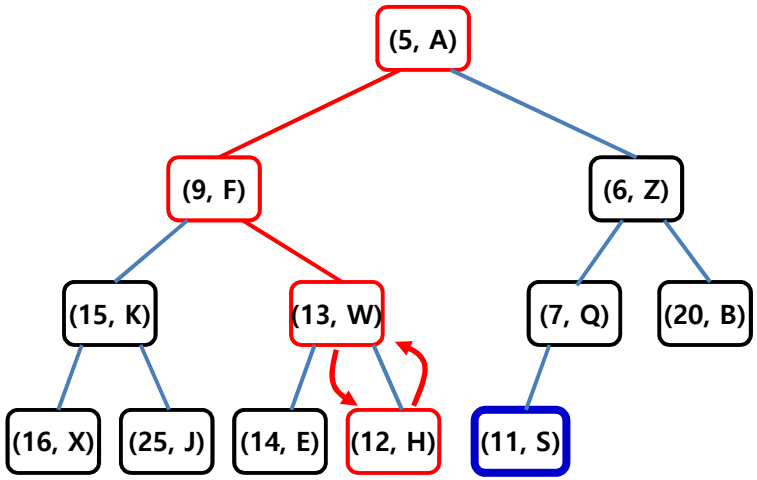
◆ Down-heap Bubbling after a RemoveMin() (2)



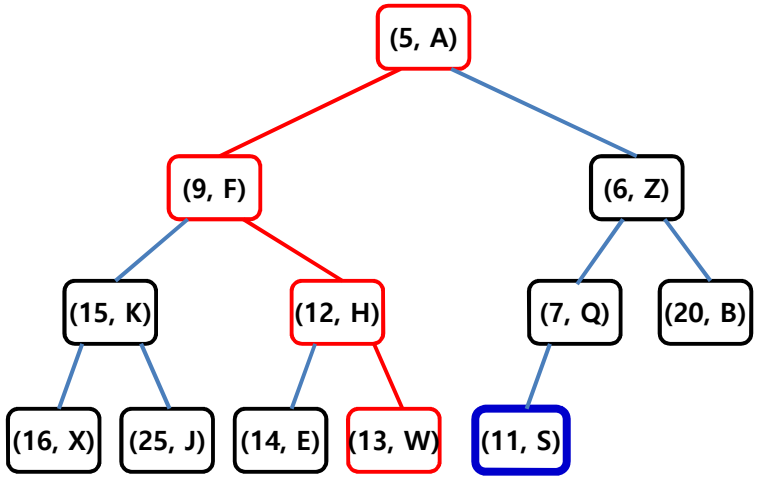
(e) compare and swap



(f) after swapping



(g) compare and swap



(h) after swapping

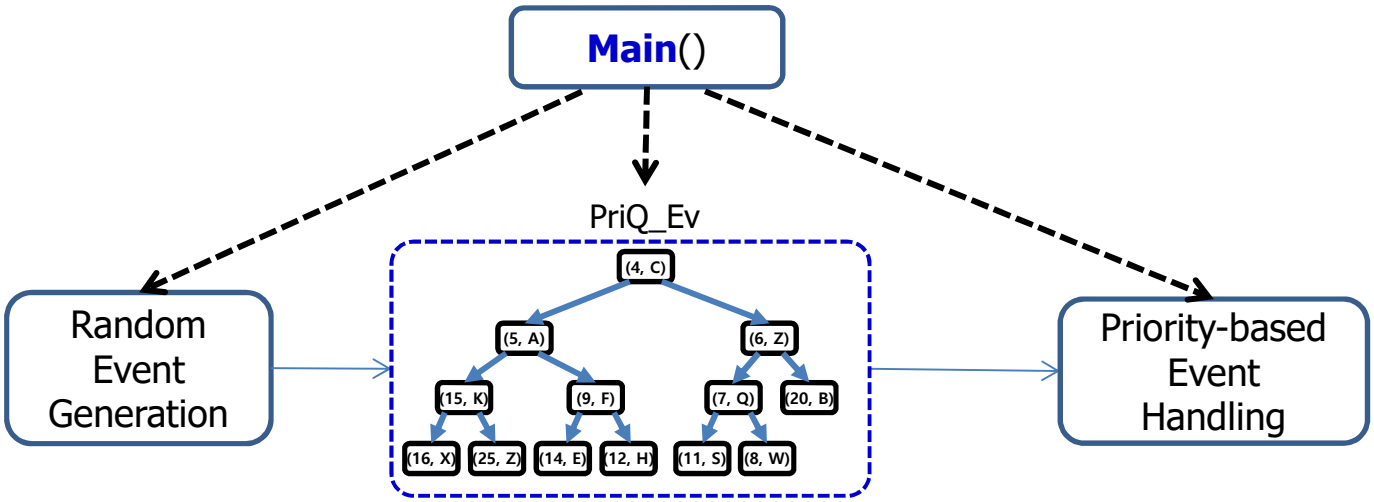
Priority Queue의 응용 예제

◆ Simple Simulation of Priority-based Event Handling

- Event Generation
- Event Handling

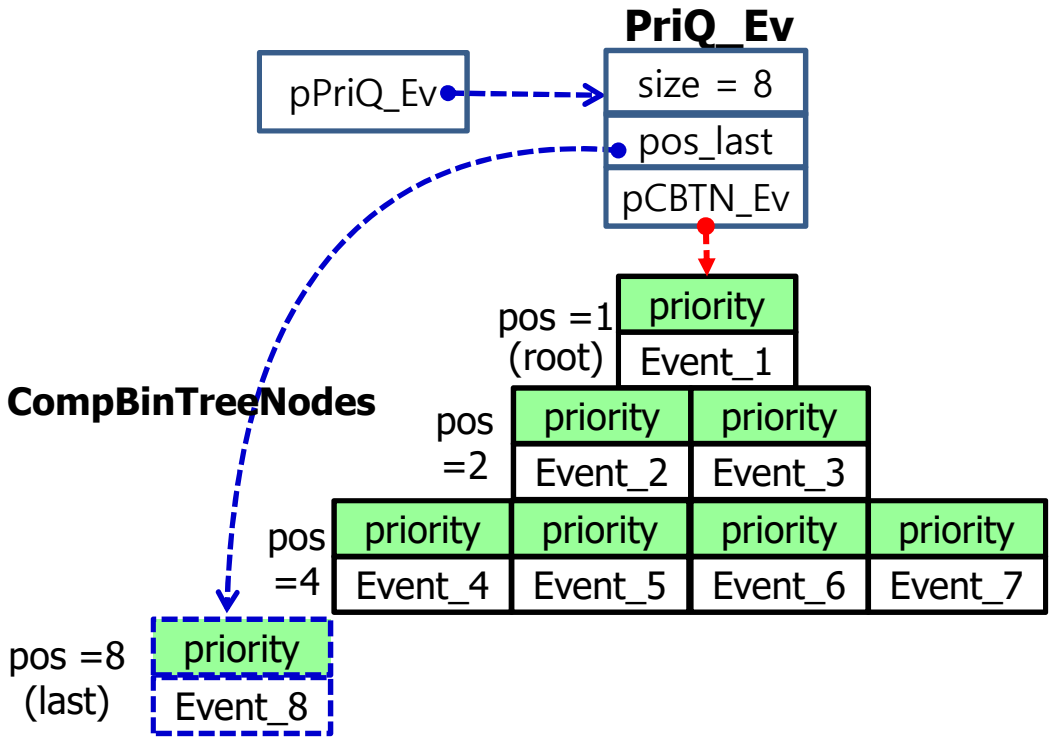
● Shared Priority Queue

- PriQ_Ev



Heap Priority Queue

◆ Heap Priority Queue



힙과 우선순위 큐 구현

Heap Priority Queue for Events

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

#ifndef PRIORITY_QUEUE_H
#define PRIORITY_QUEUE_H

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

#define POS_ROOT 1
#define MAX_NAME_LEN 80

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

typedef struct PriorityQueue
{
    CRITICAL_SECTION cs_priQ;
    char PriQ_name[MAX_NAME_LEN];
    int priQ_capacity;
    int priQ_size;
    int pos_last;
    CBTN_Event *pCBT_Ev;
} PriQ_Ev;
```

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

PriQ_Ev *initPriQ_Ev(PriQ_Ev *pPriQ_Ev, char
     *name, int capacity);
int insertPriQ_Ev(PriQ_Ev *pPriQ_Ev, Event *pEv);
Event *removeMinPriQ_Ev(PriQ_Ev *pPriQ_Ev);
void printPriQ_Ev(PriQ_Ev *pPriQ_Ev);
void fprintPriQ_Ev(FILE *fout, PriQ_Ev
     *pPriQ_Ev);
void deletePriQ_Ev(PriQ_Ev *pPriQ_Ev);
#endif
```

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

#include "PriorityQueue_Event.h"
#include "Event.h"

bool hasLeftChild(int pos, PriQ_Ev *pPriQ_Ev)
{
    if (pos * 2 <= pPriQ_Ev->priQ_size)
        return TRUE;
    else
        return FALSE;
}

bool hasRightChild(int pos, PriQ_Ev *pPriQ_Ev)
{
    if (pos * 2 + 1 <= pPriQ_Ev->priQ_size)
        return TRUE;
    else
        return FALSE;
}

PriQ_Ev *initPriQ_Ev(PriQ_Ev *pPriQ_Ev, char *name, int capacity = 1)
{
    strcpy(pPriQ_Ev->PriQ_name, name);
    pPriQ_Ev->priQ_capacity = capacity;
    pPriQ_Ev->pCBT_Ev = (CBTN_Event *)calloc((capacity + 1), sizeof(CBTN_Event));
    pPriQ_Ev->priQ_size = 0;
    pPriQ_Ev->pos_last = 0;

    return pPriQ_Ev;
}
```

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

void deletePriQ_Ev(PriQ_Ev *pPriQ_Ev)
{
    CBTN_Event *pCBTN_Ev;
    for (int i = 0; i < pPriQ_Ev->priQ_size; i++)
    {
        pCBTN_Ev = &(pPriQ_Ev->pCBT_Ev)[i];
        if (pCBTN_Ev != NULL)
        {
            if (pCBTN_Ev->pEv != NULL)
                free(pCBTN_Ev->pEv);
            free(pCBTN_Ev);
        }
    }
}
```

```
/* PriorityQueue_Event.cpp (3) */

int insertPriQ_Ev(PriQ_Ev *pPriQ_Ev, Event *pEv)
{
    int pos, pos_parent;
    CBTN_Event CBTN_Ev_tmp;

    if (pPriQ_Ev->priQ_size >= pPriQ_Ev->priQ_capacity)
    {
        // Priority Queue is full
        /* Expand the capacity twice, and copy the entries */
        CBTN_Event *newCBT_Event;
        int newCapacity;

        newCapacity = 2 * pPriQ_Ev->priQ_capacity;
        newCBT_Event = (CBTN_Event *)malloc((newCapacity + 1) * sizeof(CBTN_Event));
        if (newCBT_Event == NULL)
        {
            printf("Error in expanding CompleteBinaryTree for Priority Queue !!\n");
            exit(-1);
        }

        for (int pos = 1; pos <= pPriQ_Ev->priQ_size; pos++)
        {
           newCBT_Event[pos] = pPriQ_Ev->pCBT_Ev[pos];
        }
        free(pPriQ_Ev->pCBT_Ev);
        pPriQ_Ev->pCBT_Ev = newCBT_Event;
        pPriQ_Ev->priQ_capacity = newCapacity;
    } // end - if

    /* insert at the last position */
    pos = pos_last = ++pPriQ_Ev->priQ_size;
    pPriQ_Ev->pCBT_Ev[pos].priority = pEv->event_pri;
    pPriQ_Ev->pCBT_Ev[pos].pEv = pEv;
```

```
/* PriorityQueue_Event.cpp (4) */


    /* up-heap bubbling */
    while (pos != POS_ROOT)
    {
        pos_parent = pos / 2;
        if (pPriQ_Ev->pCBT_Ev[pos].priority > pPriQ_Ev->pCBT_Ev[pos_parent].priority)
        {
            break; // if the priority of the new packet is lower than its parent's priority,
                   // just stop up-heap     bubbling
        }
        else
        {
          CBTN_Ev_tmp = pPriQ_Ev->pCBT_Ev[pos_parent];
          pPriQ_Ev->pCBT_Ev[pos_parent] = pPriQ_Ev->pCBT_Ev[pos];
          pPriQ_Ev->pCBT_Ev[pos] = CBTN_Ev_tmp;
          pos = pos_parent;
        }
    } // end - while
}
```

```
/* PriorityQueue_Event.cpp (5) */

Event *removeMinPriQ_Ev(PriQ_Ev *pPriQ_Ev)
{
    Event *pEv;
    CBTN_Event CBT_Event_tmp;
    int pos, pos_root = 1, pos_last, pos_child;

    if (pPriQ_Ev->priQ_size <= 0)
        return NULL;  // Priority queue is empty
    pEv = pPriQ_Ev->pCBT_Ev[1].pEv; // get the packet address of current top
    pos_last = pPriQ_Ev->priQ_size;
    --pPriQ_Ev->priQ_size;
    if (pPriQ_Ev->priQ_size > 0)
    {
        /* put the last node into the top position */
        pPriQ_Ev->pCBT_Ev[pos_root] = pPriQ_Ev->pCBT_Ev[pos_last];
        pos_last--;

        /* down heap bubbling */
        pos = pos_root;
        while (hasLeftChild(pos, pPriQ_Ev))
        {
            pos_child = pos * 2;
            if (hasRightChild(pos, pPriQ_Ev))
            {
                if (pPriQ_Ev->pCBT_Ev[pos_child].priority >
                   pPriQ_Ev->pCBT_Ev[pos_child+1].priority)
                    pos_child = pos * 2 + 1;
                    // if right child has higher priority, then select it
            }
```

```
/* PriorityQueue_Event.cpp (6) */

            /* if the Event in pos_child has higher priority than Event in pos,
               swap them */
            if (pPriQ_Ev->pCBT_Ev[pos_child].priority < pPriQ_Ev->pCBT_Ev[pos].priority)
            {
                CBT_Event_tmp = pPriQ_Ev->pCBT_Ev[pos];
                pPriQ_Ev->pCBT_Ev[pos] =
                    pPriQ_Ev->pCBT_Ev[pos_child];
                pPriQ_Ev->pCBT_Ev[pos_child] = CBT_Event_tmp;
            }
            else {
                 break;
            }
            pos = pos_child;
        } // end while
    } // end if
    return pEv;
}
```

```
/* PriorityQueue_Event.cpp (7) */

void fprintPriQ_Ev(FILE *fout, PriQ_Ev *pPriQ_Ev)
{
    int pos, count, count_per_line;
    int eventPriority;
    int level = 0, level_count = 1;
    Event *pEv;

    if (pPriQ_Ev->priQ_size == 0)
    {
        fprintf(fout, "PriorityQueue_Event is empty !!\n");
        return;
    }
    pos = 1;
    count = 1;

    level = 0;
    level_count = 1; // level_count = 2^^level
    fprintf(fout, "\n CompBinTree :\n", level);
    while (count <= pPriQ_Ev->priQ_size)
    {
        fprintf(fout, " level%2d : ", level);
```

```
/* PriorityQueue_Event.cpp (8) */

        count_per_line = 0;
        while (count_per_line < level_count)
        {
            pEv = pPriQ_Ev->pCBT_Ev[pos].pEv;
            eventPriority = pPriQ_Ev->pCBT_Ev[pos].priority;
            fprintEvent(fout, pEv);
            pos++;
            count++;
            if (count > pPriQ_Ev->priQ_size)
               break;
            count_per_line++;
            if ((count_per_line % EVENT_PER_LINE) == 0)
            {
               if (count_per_line < level_count)
                  fprintf(fout, "\n          ");
               else
                  fprintf(fout, "\n");
            }
        }  // end - while
        if ((count_per_line % EVENT_PER_LINE) != 0)
           fprintf(fout, "\n");
        level++;
        level_count *= 2;
    } // end - while
    fprintf(fout, "\n");
}
```

main()

```
/* main() for Priority-Queue for Events  (1) */

#include <stdio.h>
#include <time.h>
#include "Event.h"
#include "PriorityQueue_Event.h"

#define TOTAL_NUM_EVENTS 50
#define MAX_EVENTS_PER_ROUND 5
#define MAX_ROUND 100
#define INIT_PriQ_SIZE 1

void main()
{
    PriQ_Ev *PriQ_Ev;
    Event *pEv = NULL;
    int data;
    FILE *fout;
    int processed_events = 0;
    int generated_events = 0;
    int num_events = 0;

    fout = fopen("output.txt", "w");
    if (fout == NULL)
    {
        printf("Error in output.txt file open in write mode !!\n");
        exit(-1);
    }
```

```
/* main() for Priority-Queue for Events  (2) */

    PriQ_Ev = (PriQ_Ev *)malloc(sizeof(PriQ_Ev));
    if (PriQ_Ev == NULL)
    {
        printf("Error in malloc() for PriorityQueue_Event !\n");
        fclose(fout);
        exit(-1);
    }
    initPriQ_Ev(PriQ_Ev, "PriorityQueue_Event", INIT_PriQ_SIZE);

    srand(time(0));
    for (int round = 0; round < MAX_ROUND; round++)
    {
        fprintf(fout, "\n*** Start of round(%2d)...\n", round);
        if (generated_events < TOTAL_NUM_EVENTS)
        {
            num_events = rand() % MAX_EVENTS_PER_ROUND;
            if ((generated_events + num_events) > TOTAL_NUM_EVENTS)
                num_events = TOTAL_NUM_EVENTS - generated_events;
            fprintf(fout, ">>> enque %2d events\n", num_events);
            for (int i = 0; i < num_events; i++)
            {
                pEv = genEvent(pEv, 0, generated_events,
                               TOTAL_NUM_EVENTS - 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_Ev(PriQ_Ev, pEv);
                generated_events++;
                fprintPriQ_Ev(fout, PriQ_Ev);
            }
        }  // end if
```

```
/* main() for Priority-Queue for Events  (3) */

        num_events = rand() % MAX_EVENTS_PER_ROUND;
        if ((processed_events + num_events) > TOTAL_NUM_EVENTS)
        num_events = TOTAL_NUM_EVENTS - processed_events;
        fprintf(fout, "<<< dequeue %2d events\n", num_events);
        for (int i = 0; i < num_events; i++)
        {
              pEv = removeMinPriQ_Ev(PriQ_Ev);
              if (pEv == NULL)
              {
                   fprintf(fout, "  PriQ is empty\n");
                   break;
              }
              fprintf(fout, " *** dequeued event : ");   fprintEvent(fout, pEv);
              fprintPriQ_Ev(fout, PriQ_Ev);
              processed_events++;
         }  // end for (int i = 0; …)
         /* Monitoring simulation status */
         fprintf(fout, "\*** At the end of round(%2d): total_generated_events(%3d),
            total_processed_events(%3d), PriQ capacity (%2d), events_in_PriQ(%3d)\n",
            round, generated_events, processed_events, PriQ_Ev->priQ_capacity,
            PriQ_Ev->priQ_size);
         fflush(fout);
         if (processed_events >= TOTAL_NUM_EVENTS)
               break;
    }  // end for (int round = 0; . . . )
    deletePriQ_Ev(PriQ_Ev);
    fprintf(fout, "\n");
    fclose(fout);
}
```

실행 결과 (1)

```
*** Start of round( 0)...
>>> enque  4 events
 *** enqued event : Ev(id:  0, pri:49)
 CompBinTree :
 level 0 : Ev(id:  0, pri:49)
 *** enqued event : Ev(id:  1, pri:48)
 CompBinTree :
 level 0 : Ev(id:  1, pri:48)
 level 1 : Ev(id:  0, pri:49)
 *** enqued event : Ev(id:  2, pri:47)
 CompBinTree :
 level 0 : Ev(id:  2, pri:47)
 level 1 : Ev(id:  0, pri:49)  Ev(id:  1, pri:48)

 *** enqued event : Ev(id:  3, pri:46)
 CompBinTree :
 level 0 : Ev(id:  3, pri:46)
 level 1 : Ev(id:  2, pri:47)  Ev(id:  1, pri:48)
 level 2 : Ev(id:  0, pri:49)
<<< dequeue  4 events
 *** dequeued event : Ev(id:  3, pri:46)
 CompBinTree :
 level 0 : Ev(id:  2, pri:47)
 level 1 : Ev(id:  0, pri:49)  Ev(id:  1, pri:48)

 *** dequeued event : Ev(id:  2, pri:47)
 CompBinTree :
 level 0 : Ev(id:  1, pri:48)
 level 1 : Ev(id:  0, pri:49)
 *** dequeued event : Ev(id:  1, pri:48)
 CompBinTree :
 level 0 : Ev(id:  0, pri:49)
 *** dequeued event : Ev(id:  0, pri:49)    PriorityQueue_Event is empty !!
*** At the end of round( 0): total_generated_events(  4), total_processed_events(  4), PriQ capacity ( 4), events_in_PriQ(  0)
```

실행 결과 (2)

```
*** Start of round(32)...
<<< dequeue  3 events
 *** dequeued event : Ev(id: 49, pri: 0)
 CompBinTree :
 level 0 : Ev(id: 48, pri: 1)
 level 1 : Ev(id: 47, pri: 2)  Ev(id: 45, pri: 4)
 level 2 : Ev(id: 44, pri: 5)  Ev(id: 46, pri: 3)

 *** dequeued event : Ev(id: 48, pri: 1)
 CompBinTree :
 level 0 : Ev(id: 47, pri: 2)
 level 1 : Ev(id: 46, pri: 3)  Ev(id: 45, pri: 4)
 level 2 : Ev(id: 44, pri: 5)
 *** dequeued event : Ev(id: 47, pri: 2)
 CompBinTree :
 level 0 : Ev(id: 46, pri: 3)
 level 1 : Ev(id: 44, pri: 5)  Ev(id: 45, pri: 4)

*** At the end of round(32): total_generated_events( 50), total_processed_events( 47), PriQ capacity ( 8), events_in_PriQ(  3)

*** Start of round(33)...
<<< dequeue  1 events
 *** dequeued event : Ev(id: 46, pri: 3)
 CompBinTree :
 level 0 : Ev(id: 45, pri: 4)
 level 1 : Ev(id: 44, pri: 5)
*** At the end of round(33): total_generated_events( 50), total_processed_events( 48), PriQ capacity ( 8), events_in_PriQ(  2)

*** Start of round(34)...
<<< dequeue  0 events
*** At the end of round(34): total_generated_events( 50), total_processed_events( 48), PriQ capacity ( 8), events_in_PriQ(  2)

*** Start of round(35)...
<<< dequeue  1 events
 *** dequeued event : Ev(id: 45, pri: 4)
 CompBinTree :
 level 0 : Ev(id: 44, pri: 5)
*** At the end of round(35): total_generated_events( 50), total_processed_events( 49), PriQ capacity ( 8), events_in_PriQ(  1)

*** Start of round(36)...
<<< dequeue  1 events
 *** dequeued event : Ev(id: 44, pri: 5)    PriorityQueue_Event is empty !!
*** At the end of round(36): total_generated_events( 50), total_processed_events( 50), PriQ capacity ( 8), events_in_PriQ(  0)
```

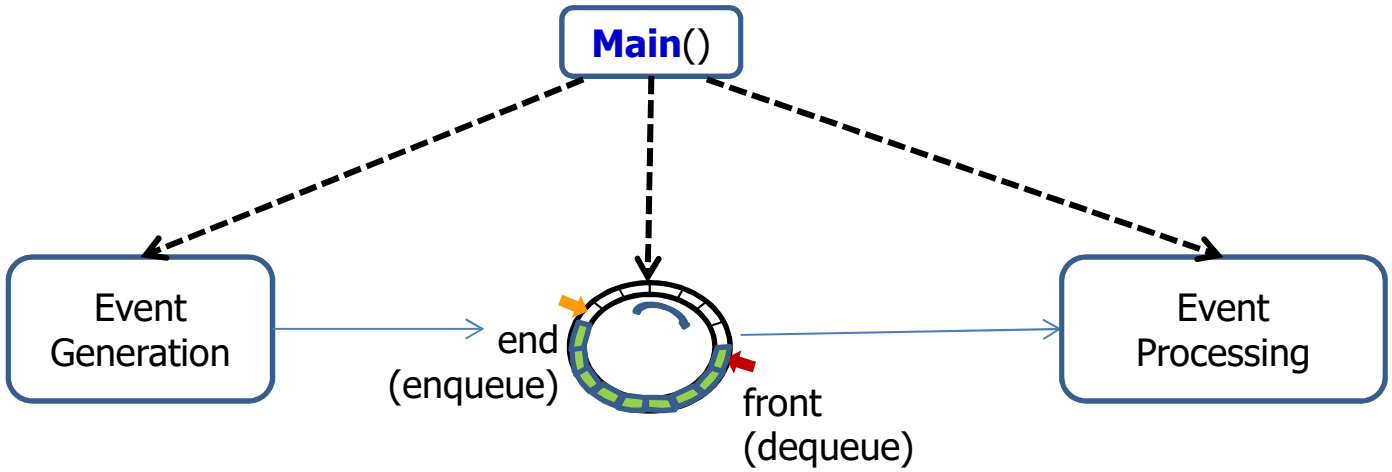
Homework 12

Homework 12.1

12.1 Circular Queue 기반의 Event 처리 시뮬레이션

- Event를 위한 구조체를 다음과 같이 정의할 것.

```
enum EventStatus { GENERATED, ENQUEUED, PROCESSED, UNDEFINED };
typedef struct
{
    int event_no;
    int event_gen_addr;
    int event_handler_addr;
    int event_pri;  // event_priority
    EventStatus eventStatus;
} Event;
```

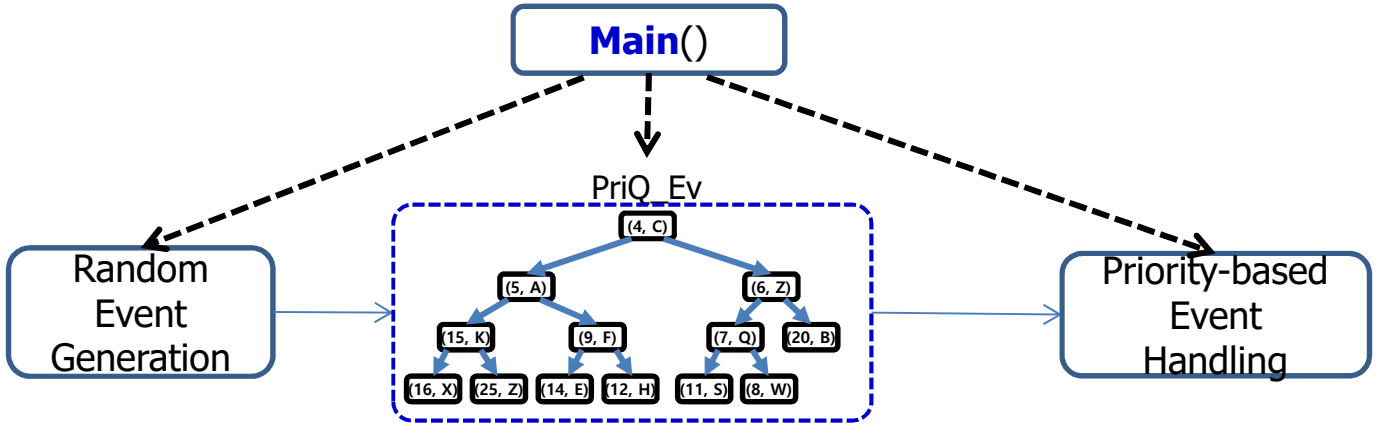


- 함수 Event generator (Event_Gen)은 총 128개의 Event를 우선 순위 127 ~ 0의 순으로 생성하여, 하나의 공유 자원인 Event 처리용 환형 큐 (CirQ_Ev)에 Event를 삽입하며, 출력 파일 (output.txt)에 생성된 순서에 따라 event를 저장
- 발생된 사건(event) 들을 도착 순서에 따라 처리하는 시뮬레이션 기능 구현
- Event는 번호 (event_no), 우선 순위 (event_prio)를 기본적인 정보로 가지며,
  우선 순위는 0 ~ 127의 값으로 설정되고, priority 0이 우선 순위가 가장 높은 것을 의미함
- 환형 큐 (CirQ_Ev)는 삽입되는 event들을 순서에 따라 저장하며, 항상 가장 먼저 도착한 event가 먼저 처리될 수 있도록 관리
- 환형 큐는 동적 확장성 있는 배열을 기반으로 구현되어야 하며, 필요에 따라 크기를 확장할 수 있도록 구현할 것.
- Event_gen() 함수가 먼저 실행한 후, Event_Handler가 뒤에 실행되도록 하며, 실제 처리된 Event 순서를 출력 파일 (output.txt)에 출력할 것.

Homework 12.2

12.2 우선 순위 큐 기반의 Event 처리 기능 시뮬레이션

- 이벤트 (event) 들을 우선 순위에 따라 처리하는 시뮬레이션 기능 구현



- Event를 위한 구조체는 HW 12.1에서 정의한 것을 사용할 것
- 함수 Event generator (Event_Gen)은 총 128개의 Event를 우선 순위 127 ~ 0의 순으로 생성하여, 하나의 공유 자원인 Event 처리용 Priority Queue (PriQ_Ev)에 Event를 삽입하며, 출력 파일 (output.txt)에 생성된 순서에 따라 event를 저장
- 우선 순위 큐 (PriQ_Ev)는 삽입되는 event들을 우선 순위에 따라 저장하며, 항상 가장 우선 순위가 높은 event가 root에 위치하도록 관리

- 함수 Event_Handler()는 우선 순위 이벤트 큐로 부터 가장 우선 순위가 높은 이벤트를 추출하여 처리
- 우선 순위 큐는 Heap Priority Queue로 구현되어야 하며, complete binary tree 구조를 기반으로 구현 할 것
- Event_gen() 함수가 먼저 실행한 후, Event_Handler가 뒤에 실행되도록 하며, 실제 처리된 Event 순서를 출력 파일 (output.txt)에 출력할 것.