**Homework 11**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 프로그래밍 언어 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.05.24.월 |

// main.cpp(2)

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]; // 처리한 이벤트 기록

int total\_processed\_events = 0;

int total\_generated\_events = 0;

int num\_events = 0;

int num\_generated\_round = 0;

int num\_processed\_round = 0;

int c = 0, n = 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(pCirQ\_Event, max\_events\_per\_round); // 초기화 작업

//fprintQueue(fout, pCirQ\_Event);

//fprintf(fout, "\nEnqueuing data into event circular queue: \n");

for (int round = 0; round < MAX\_ROUND; round++) { // 라운드 반복

printf("Expand CirQ Size(CS NS) : ");

scanf("%d %d", &c, &n);

pCirQ\_Event->capacity = expand\_CirQ(pCirQ\_Event, c, n);

fprintf(fout, "start of Round(%2d) \*\*\*\*\n", round);

if (total\_generated\_events < TOTAL\_NUM\_EVENTS) { // 생성된 이벤트 수가 일정 수준을 넘지 않도록 제한

num\_events = max\_events\_per\_round; // 원하는 이벤트 수

if ((total\_generated\_events + num\_events) > TOTAL\_NUM\_EVENTS)

num\_events = TOTAL\_NUM\_EVENTS - total\_generated\_events; // 남아있는 개수 만큼만 실행

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)) { // 큐가 꽉 차있으면 탈출

fprintf(fout, " !!! CirQ\_Event is full --> skip generation and enqueueing of event. \n");

printf(" !!! CirQ\_Event is full --> skip generation and enqueueing of event. \n");

break;

}

/\*if (total\_generated\_events == TOTAL\_NUM\_EVENTS - 1) {

if ((TOTAL\_NUM\_EVENTS % 2) == 0)

total\_generated\_events++;

break;

}\*/

pEv = genEvent(pEv, EVENT\_GENERATOR, total\_generated\_events, // 이벤트 생성

TOTAL\_NUM\_EVENTS - total\_generated\_events - 1);

fprintf(fout, ">>> Enqueue event = ");

fprintEvent(fout, pEv);

fprintf(fout, "\n");

enCirQ(fout, pCirQ\_Event, \*pEv); // 생성된 이벤트를 순환 큐에 집어 넣기

fprintCirQ(fout, pCirQ\_Event);

free(pEv);

total\_generated\_events++;

num\_generated\_round++;

} // end for

} // end if

/\*

\* 파일명 : main.cpp

\* 목적 및 기본 기능

\* Circular queue, Priority Queue

\* 작성자 : 김주환(21812158)

\* 작성일 : 2021년 5월 21일

\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "Event.h"

#include "CirQ\_Event.h"

#include "PriorityQueue\_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);

int 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); // 동작 선택

if (menu == 0)

break;

printf("Input num\_events per round :");

scanf("%d", &max\_events\_per\_round); // 이벤트

switch (menu) {

case 1:

test\_FIFO\_CirQ\_Event(fout, max\_events\_per\_round); // 선입 선출 순환 큐

break;

case 2:

test\_PriQ\_Event(fout, max\_events\_per\_round); // 우선순위 큐

break;

default:

break;

}

}

fclose(fout);

return 0;

}

// main.cpp(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; // 이번 라운드에서 몇개를 처리했는가

for (int i = 0; i < num\_events; i++) {

if (isCirQEmpty(pCirQ\_Event)) { // 비었으면 탈출

fprintf(fout, " !!! CirQ\_Event is full --> skip generation and enqueueing of event. \n");

printf(" !!! CirQ\_Event is full --> skip generation and enqueueing of event. \n");

break;

}

pEv = deCirQ(fout, pCirQ\_Event); // 뽑아내기

if (pEv != NULL) {

fprintf(fout, "<<< Dequed event = ");

fprintEvent(fout, pEv);

fprintf(fout, "\n");

processed\_events[total\_processed\_events] = \*pEv; // 어떤 순서로 처리됐는지 확인

total\_processed\_events++;

num\_processed\_round++;

}

fprintCirQ(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(pCirQ\_Event); // 반납

}

// main.cpp(4)

void test\_PriQ\_Event(FILE\* fout, int max\_events\_per\_round) {

PriQ\_Ev\* 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\_Ev\*)malloc(sizeof(PriQ\_Ev));

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\_Ev(pPriQ\_Ev, "PriorityQueue\_Event", INIT\_PriQ\_SIZE); // 초기화

for (int round = 0; round < MAX\_ROUND; round++) { // 라운드

fprintf(fout, "\n\*\*\* Start of round(%2d)...\n", round);

num\_generated\_round = 0;

if (total\_generated\_events < TOTAL\_NUM\_EVENTS) { // 처리할 것이 남아있는 경우

num\_events = max\_events\_per\_round;

if ((total\_generated\_events + num\_events) > TOTAL\_NUM\_EVENTS) // 남은 개수만 생성

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); // 이벤트 생성

if (pEv == NULL) {

printf("Error in generation of event !!\n");

fclose(fout);

exit(-1);

}

fprintf(fout, " \*\*\* enqued event : ");

fprintEvent(fout, pEv);

insertPriQ\_Ev(pPriQ\_Ev, pEv); // 이벤트 추가

total\_generated\_events++;

num\_generated\_round++;

fprintPriQ\_Ev(fout, pPriQ\_Ev);

}

}

// main.cpp(5)

//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\_Ev(pPriQ\_Ev); // 빼기

if (pEv == NULL) {

fprintf(fout, " PriQ is empty\n");

break;

}

fprintf(fout, " \*\*\* dequeued event : ");

fprintEvent(fout, pEv);

fprintPriQ\_Ev(fout, pPriQ\_Ev);

processed\_events[total\_processed\_events] = \*pEv; // 순서 기록

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->priQ\_size);

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->priQ\_size);

fflush(fout);

if (total\_processed\_events >= TOTAL\_NUM\_EVENTS) // 초과 시 탈출

break;

}

printf("Processed Events :\n");

for (int i = 0; i < TOTAL\_NUM\_EVENTS; i++) { // 처리 순서 출력

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\_Ev(pPriQ\_Ev); // 반환

fprintf(fout, "\n");

}

1. **Circular Queue 기반의 Event 처리 시뮬레이션**

/\* 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);

void initEvent(Event\* pEv, int ev\_gen\_ID, int ev\_no, int ev\_pri, int ev\_handler\_addr, EventStatus ev\_status);

Event\* genEvent(Event\* pEvent, int event\_Gen\_ID, int event\_no, int event\_pri);

void printEventArray(Event\* pEv, int size, int items\_per\_line);

#endif

/\* Event.cpp \*/

#include <stdio.h>

#include <stdlib.h>

#include "Event.h"

extern const 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);

}

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;

}

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;

}

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 ");

}

}

/\* CirQ\_Event.cpp \*/

#ifndef CIRCULAR\_QUEUE\_H

#define CIRCULAR\_QUEUE\_H

#include <stdio.h>

#include <stdlib.h>

#include "CirQ\_Event.h"

#include "Event.h"

typedef struct {

Event\* pEv; // circular queue for events

// 버퍼

int capacity;

int front; // 맨 처음 누가 서비스를 받을 것인가

int end; // 어디 집어넣을지

int num\_elements;

} CirQ\_Event;

CirQ\_Event\* initCirQ(CirQ\_Event\* pCirQ, int capacity) {

Event\* pEv2;

pEv2 = (Event\*)calloc(capacity, sizeof(Event));

if (pEv2 == NULL) {

printf("Error in memory allocation for Event array of size (%d)\n", capacity);

exit;

}

pCirQ->pEv = pEv2;

pCirQ->capacity = capacity;

pCirQ->front = pCirQ->end = 0;

pCirQ->num\_elements = 0;

return pCirQ;

}

bool isCirQEmpty(CirQ\_Event\* pCirQ) {

if (pCirQ->num\_elements == 0)

return true;

else

return false;

}

/\* void printCirQ(CirQ\_Event\* cirQ) {} \*/

/\* CirQ\_Event.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\_Event;

CirQ\_Event\* initCirQ(CirQ\_Event\* pCirQ, int capacity);

bool isCirQEmpty(CirQ\_Event\* pCirQ);

// void printCirQ(CirQ\_Event\* cirQ);

void fprintCirQ(FILE\* fout, CirQ\_Event\* pCirQ);

bool isCirQFull(CirQ\_Event\* pCirQ);

Event\* enCirQ(FILE\* fout, CirQ\_Event\* pCirQ, Event ev);

Event\* deCirQ(FILE\* fout, CirQ\_Event\* pCirQ);

void delCirQ(CirQ\_Event\* pCirQ);

int expand\_CirQ(CirQ\_Event\* array, int curS, int newS);

#endif

텍스트이(가) 표시된 사진

자동 생성된 설명

// CirQ\_Event.cpp(3)

Event\* enCirQ(FILE\* fout, CirQ\_Event\* pCirQ, Event ev) { // 집어넣기

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]);

}

Event\* deCirQ(FILE\* fout, CirQ\_Event\* pCirQ) { // 뽑아내기

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\_Event\* pCirQ) { // 메모리 배열 반납

if (pCirQ->pEv != NULL)

free(pCirQ->pEv);

pCirQ->pEv = NULL;

pCirQ->capacity = 0;

pCirQ->front = pCirQ->end = 0;

pCirQ->num\_elements = 0;

free(pCirQ);

}

int expand\_CirQ(CirQ\_Event\* A, int curS, int newS) {

Event\* newA;

Event\* empty\_sp = (Event\*)calloc(1, sizeof(Event));

newA = (Event\*)realloc(A->pEv, sizeof(Event) \* newS);

for (int i = curS; i < newS; i++)

newA[i] = \*empty\_sp;

return newS;

}

#endif

// CirQ\_Event.cpp(2)

void fprintCirQ(FILE\* fout, CirQ\_Event\* 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");

}

bool isCirQFull(CirQ\_Event\* pCirQ) {

if (pCirQ->num\_elements == pCirQ->capacity)

return true;

else

return false;

}

1. **우선 순위 큐 기반의 Event 처리 기능 시뮬레이션**

/\* PriorityQueue\_Event.cpp \*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#ifndef PRIORITY\_QUEUE\_H

#define PRIORITY\_QUEUE\_H

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "PriorityQueue\_Event.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\* pEv;

} CBTN\_Event;

typedef struct PriorityQueue {

char priQ\_name[MAX\_NAME\_LEN];

int priQ\_capacity;

int priQ\_size;

int pos\_last;

CBTN\_Event\* pCBT\_Ev;

} PriQ\_Ev;

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, const 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;

}

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.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\* pEv; // 저장되는 곳 주소

} CBTN\_Event;

typedef struct PriorityQueue

{

char priQ\_name[MAX\_NAME\_LEN]; // 이름

int priQ\_capacity; // 용량

int priQ\_size; // 몇개가 포함되어 있는가

int pos\_last; // 마지막 위치

CBTN\_Event\* pCBT\_Ev; // 배열을 가르키는 역할

} PriQ\_Ev;

bool hasLeftChild(int pos, PriQ\_Ev\* pPriQ\_Ev);

bool hasRightChild(int pos, PriQ\_Ev\* pPriQ\_Ev);

PriQ\_Ev\* initPriQ\_Ev(PriQ\_Ev\* pPriQ\_Ev, const char\* name, int capacity = 1);

void deletePriQ\_Ev(PriQ\_Ev\* pPriQ\_Ev);

void insertPriQ\_Ev(PriQ\_Ev\* pPriQ\_Event, Event\* pEvent);

Event\* removeMinPriQ\_Ev(PriQ\_Ev\* pPriQ\_Event);

void printPriQ\_Ev(PriQ\_Ev\* pPriQ\_Event);

void fprintPriQ\_Ev(FILE\* fout, PriQ\_Ev\* pPriQ\_Event);

#endif

// PriorityQueue\_Event.cpp(3)

/\* 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]; // 스왑

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;

}

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);

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");

}

#endif

// PriorityQueue\_Event.cpp(2)

void 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) { // 꽉 차면 사이즈 키우는 역할

// 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); // 기존 할당 해제

pPriQ\_Ev->pCBT\_Ev = newCBT\_Event;

pPriQ\_Ev->priQ\_capacity = newCapacity;

} // end - if

/\* insert at the last position \*/

pos = ++pPriQ\_Ev->priQ\_size;

pPriQ\_Ev->pCBT\_Ev[pos].priority = pEv->event\_pri;

pPriQ\_Ev->pCBT\_Ev[pos].pEv = pEv;

/\* 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]; // 스왑

pPriQ\_Ev->pCBT\_Ev[pos\_parent] = pPriQ\_Ev->pCBT\_Ev[pos];

pPriQ\_Ev->pCBT\_Ev[pos] = CBTN\_Ev\_tmp;

pos = pos\_parent; // 주소도 스왑

}

} // end - while

}

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; // 우선순위 큐가 비었으면 탈출

pEv = pPriQ\_Ev->pCBT\_Ev[1].pEv; // 최고 우선순위 주소 반환

pos\_last = pPriQ\_Ev->priQ\_size; // 최 하단 주소 반환

--pPriQ\_Ev->priQ\_size; // 뺄거니까 빼줌

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]; // 최하단 주소를 최상단 주소로 이동

pos\_last--;

/\* down heap bubbling \*/

pos = pos\_root;

while (hasLeftChild(pos, pPriQ\_Ev)) { // 왼쪽에 자식이 있는가

pos\_child = pos \* 2;

if (hasRightChild(pos, pPriQ\_Ev)) { // 오른쪽에 자식이 있는가

if (pPriQ\_Ev->pCBT\_Ev[pos\_child].priority > // 오른쪽 우선순위가 높으면 그놈 선택

pPriQ\_Ev->pCBT\_Ev[pos\_child + 1].priority)

pos\_child = pos \* 2 + 1;

}

**텍스트이(가) 표시된 사진

자동 생성된 설명**