프로그래밍언어

14. 자기참조 구조체, 연결형 리스트, Stack, Queue

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

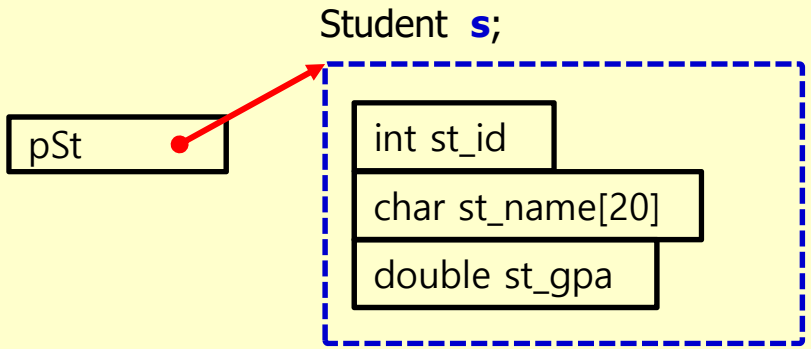
- 자기참조 구조체 (self-referential structure)
- 연결리스트 (Linked List)
- 연결리스트 연산 (insert, delete)
- 연결형 리스트 기반의 Stack
- 연결형 리스트 기반의 Queue
- Multi-thread와 Linked List Queue의 응용
  - Event generator thread
  - Event processor thread
  - Linked List Queue for High Priority Events
  - Linked List Queue for Low Priority Events

자기참조 구조체
연결형 리스트

구조체와 구조체 포인터

◆ 구조체와 구조체 포인터

```
typedef struct
{
   int st_id;
   char st_name[20];
   double st_gpa;
} Student;

Student *pSt;
Student s = { 20070001, "홍길동", 4.3 };

pSt = &s;

printf("학번=%d 이름=%s 학점=%lf \n", s.st_id, s.st_name, s.st_gpa);
printf("학번=%d 이름=%s 학점=%lf \n", (*pSt).st_id,(*pSt).st_name,(*pSt).st_gpa);
printf("학번=%d 이름=%s 학점=%lf \n", pSt->st_id, pSt->st_name, pSt->st_gpa);
```



자기참조 구조체 (Self-referential Structure)

◆ 자기참조 구조체 (self-referential structure)

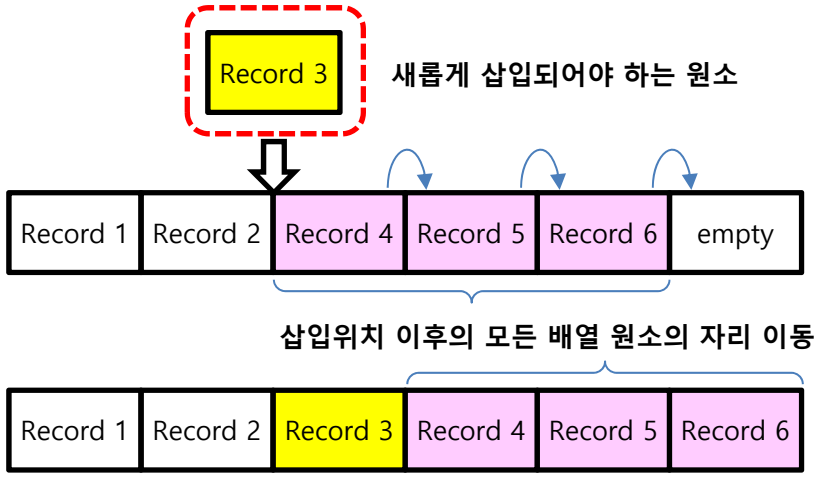
- 구조체에 포함된 항목으로 자신과 동일한 구조체를 가리키는 포인터를 포함하는 구조체
- 다양한 형태의 자료구조를 만들 수 있게 함

◆ 자기참조 구조체의 예

- Linked List Node (연결형 리스트의 리스트 노드)
- Binary Search Tree Node (이진 탐색 트리의 트리 노드)

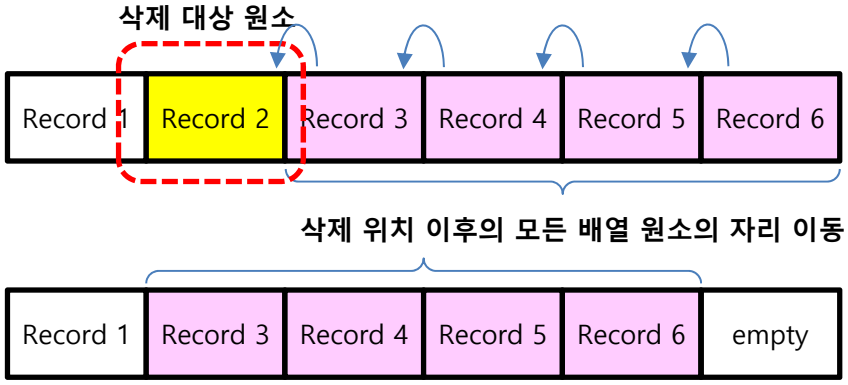
배열의 원소 삽입과 삭제 성능 분석

◆ 새로운 원소의 삽입



Operation	Complexity in Array
search	O(n)
insert	O(n)
remove	O(n)

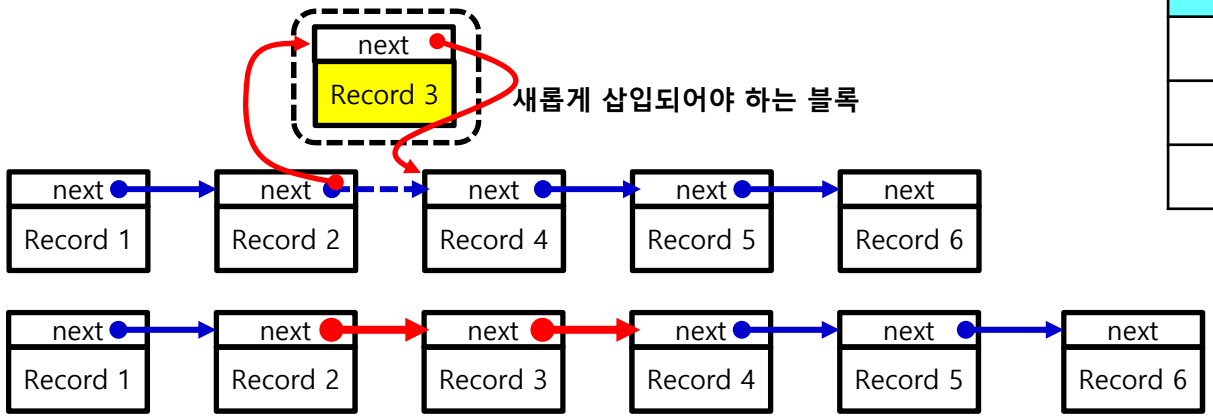
◆ 배열 원소의 삭제



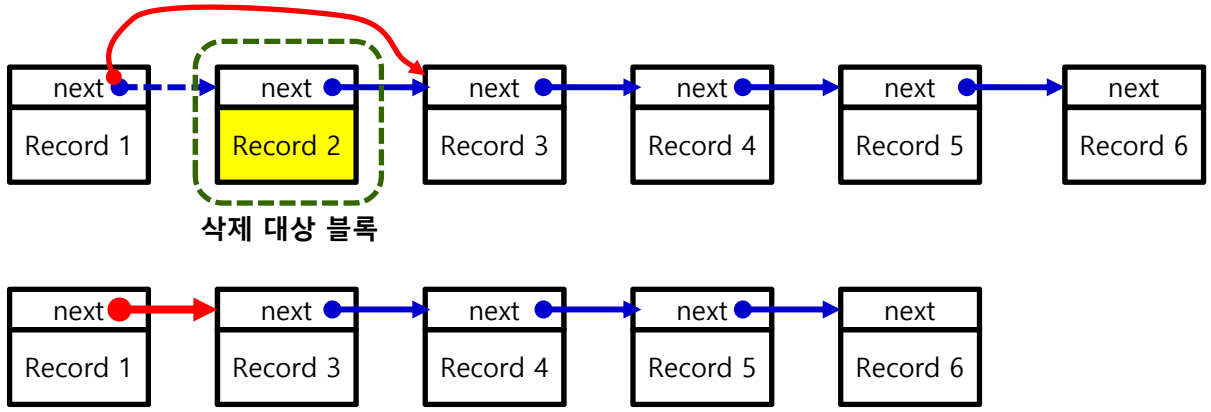
연결형 리스트에서의 삽입과 삭제

◆ Insert in Linked List

Operation	Complexity in Linked List
search	O(n)
insert	O(1)
remove	O(1)

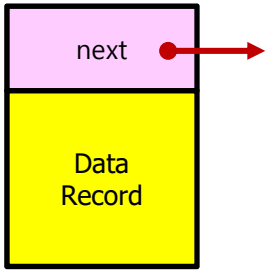


◆ Delete/Remove in Linked List

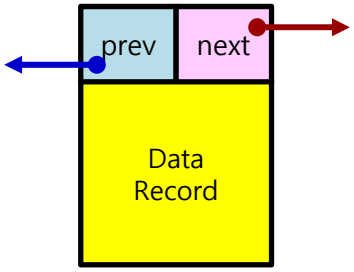


연결형 리스트의 구조 (1)

- **List Node = data field + link field (next, prev)**
- **List Node with Data**

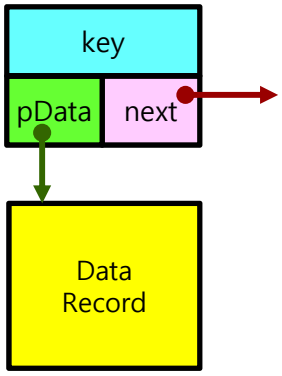


(a) List Node with Data for Singly Linked List

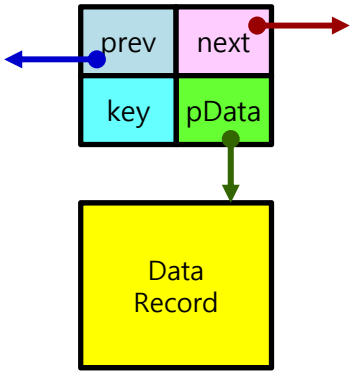


(b) Doubly Linked List Node with Data
for Doubly Linked List

- **List Node with Data Pointer**



(a) List Node with Data Pointer
for Singly Linked List



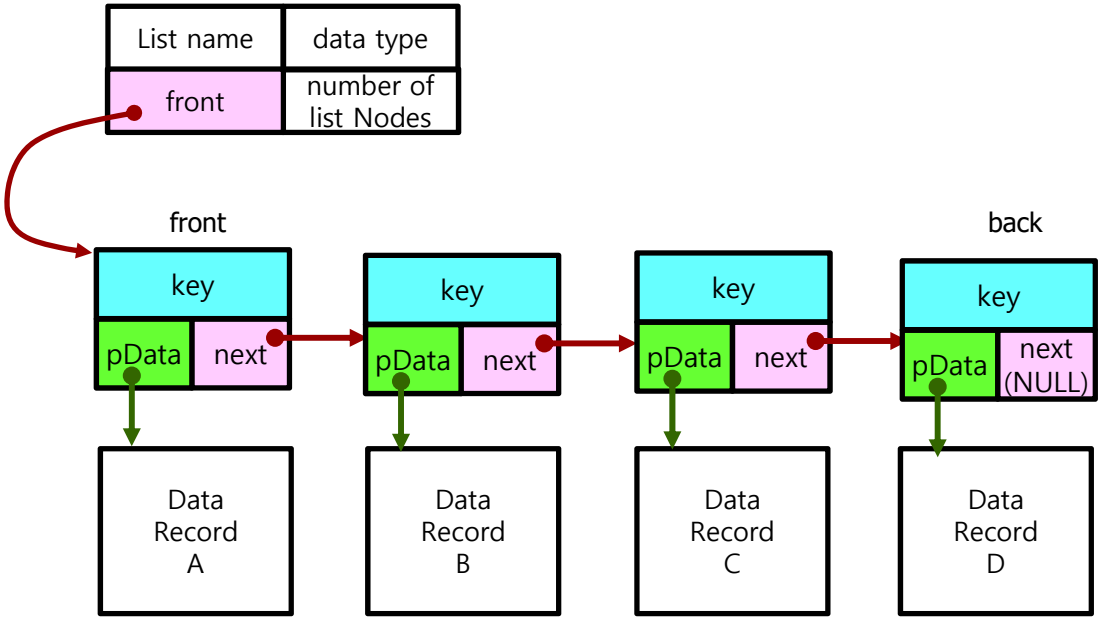
(b) List Node with Data Pointer
for Doubly Lniked List

연결형 리스트의 구조 (2)

◆ List

- linked list of Nodes
- list abstract data type

◆ Single Linked List (SLL)

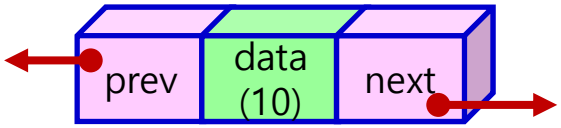


Doubly Linked List (DLL)
DLLN 자기 참조 구조체

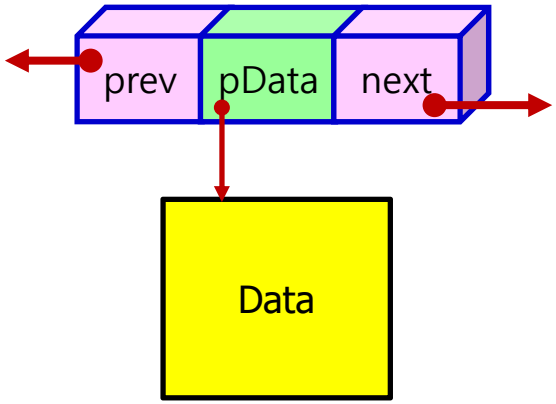
◆ Doubly Linked List (DLL) DLLN

- 2 pointers: next, prev
- pointer to data: pData
- optional key value

```
typedef  struct node
{
      //int data;
      int *pData;
      struct node *next;
      struct node *prev;
} DLLN;
```



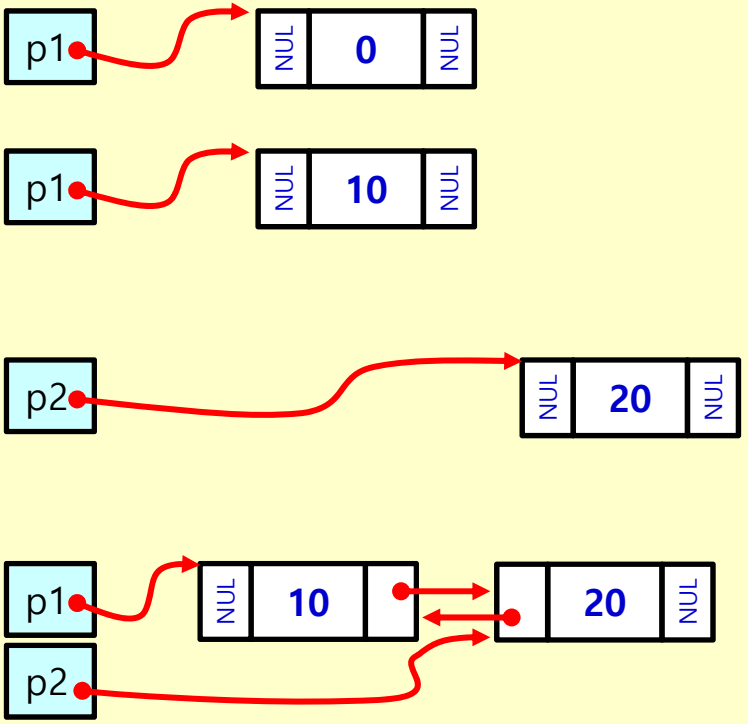
(a) Doubly Linked List Node with Data



(b) Doubly Linked List Node
with Data Pointer

간단한 doubly linked list (DLL) 생성

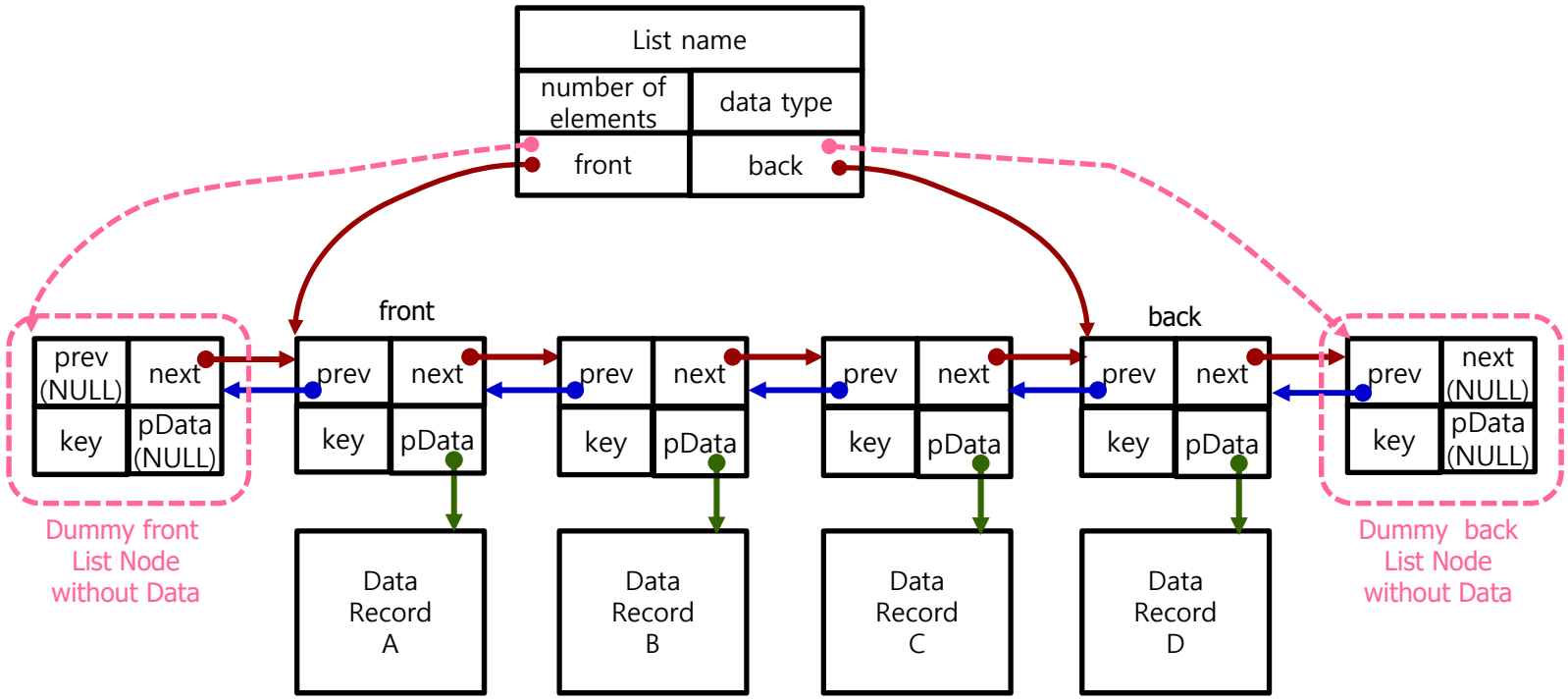
```
DLLN *p1;
p1 = (DLLN *)calloc(1, sizeof(DLLN));

p1->data = 10;
p1->next = p1->prev = NULL;

DLLN *p2;
p2 = (DLLN *)calloc(1, sizeof(DLLN));
p2->data = 20;
p2->next = NULL;

p2->prev = p1;
p1->next = p2;

free(p1);
free(p2);
```



이중 연결형 리스트 (Doubly Linked List)의 구조

◆ Structure of Doubly Linked List

- **front** pointer**:** the pointer that points the front (head, first) Node
- **back** pointer: the pointer that points the last (end, tail, last) Node
- number of elements: current number of elements in DLL



Doubly Linked List의 DLLN, 관리 구조체

◆ List Node, DLL_xyz

```
typedef struct list_node
{
    Key_Type key;
    Data_Type *pData;
    struct list_node *prev;
    struct list_node *next;
} DLLN;
```

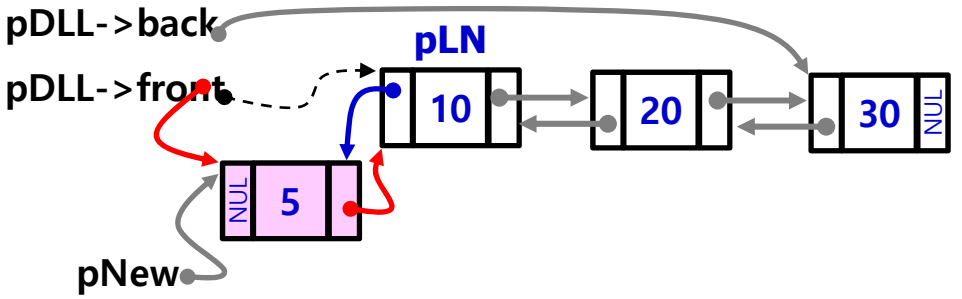
```
typedef struct
{
    char name[MAX_NAME_LENGTH];
    DLLN *front;
    DLLN *back;
    int num_element;
} DLL_xyz;  // e.g., Galaxy, School, Library
```

Doubly Linked List에서 지정된 위치에 Node 삽입 (1)

```
DLLN *insert_node(DLL_xyz *pDLL, DLLN *pLN, DATA *pD);
```

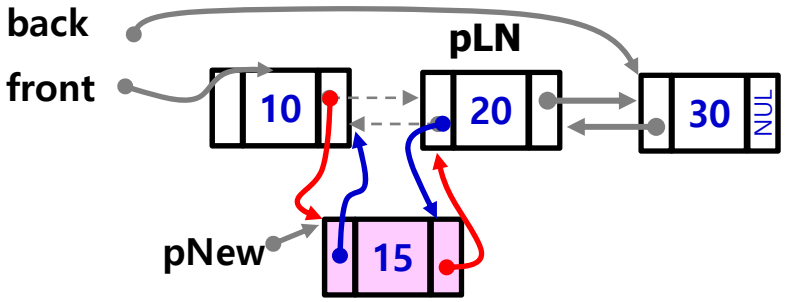
1. 리스트의 처음에 삽입하는 경우

```
pNew = (DLLN *)calloc(1, sizeof(DLLN));
pNew->pData = pD;
//pLN == pDLL->front
pDLL->front->prev = pNew;
pNew->next = pDLL->front;
pNew->prev = NULL;
pDLL->front = pNew;
```



2. 리스트의 중간에 삽입하는 경우

```
pNew = (DLLN *)calloc(1, sizeof(DLLN));
pNew->pData = pD;
pLN->prev->next = pNew;
pNew->prev = pLN->prev;
pLN->prev = pNew;
pNew->next = pLN;
```



Doubly Linked List에서 지정된 위치에 Node 삽입 (2)

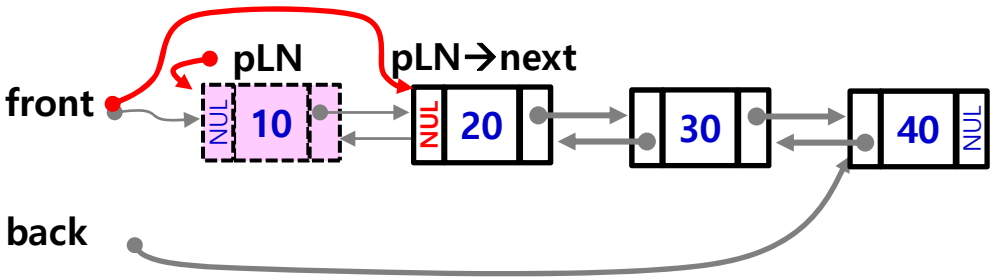
```
DLLN *insert_node(DLLN *front, DLLN *pLN, DATA *pD)
{
   DLLN *pNew = NULL;
   DLLN *prev = pLN->prev;

   if ( !(pNew = (DLLN *)calloc(1, sizeof(DLLN))) ) {
      printf("메모리 동적 할당 오류\n");
      exit(1);
   }
   pNew->pData = pD;
   if( prev == NULL ) {    // 연결 리스트의 처음에 삽입
      pNew->next = front;
      front->prev = pNew;
      front = pNew;
   } else {    // 연결 리스트의 중간 (pLN 앞)에 삽입
      pLN->prev->next = pNew;
      pNew->prev = pLN->prev;
      pLN->prev = pNew;
      pNew->next = pLN;
   }
   return front;
}
```

Doubly Linked List에서 지정된 리스트 노드 삭제 (1)

```
DLLN *delete_node(DLLN *front, DLLN *pLN);
```

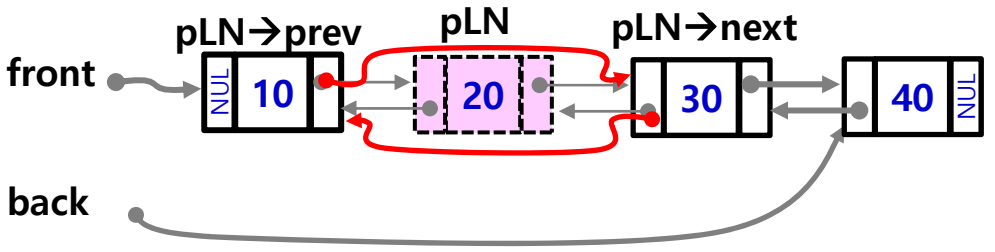
1. 리스트의 처음 노드를 삭제하는 경우

```
DLLN *prev, next;
prev = pLN->prev;
next = pLN->next;
if (prev == NULL)
{
    front = pLN->next;
    pLN->next->prev = NULL;
    free(pLN);
}
```



2. 리스트의 중간 노드를 삭제하는 경우

```
else
{
   pLN->prev->next = pLN->next;
   pLN->next->prev = pLN->prev;
   free(pLN);
}
```

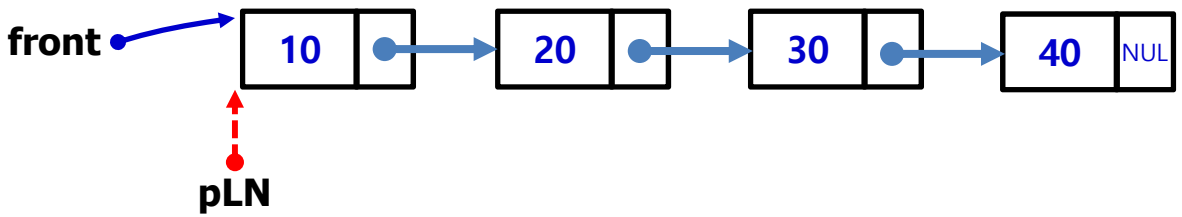


Doubly Linked List에서 지정된 리스트 노드 삭제 (2)

```
DLLN *delete_node(DLLN *front, DLLN *pLN)
{

    if( pLN->prev == NULL )  // First Node
    {
        front = pLN→next;
        pLN→next->prev = NULL;
    } else {
        pLN->prev->next = pLN->next;
        pLN->next->prev = pLN->prev;
    }
    free(pLN);
    return front;
}
```

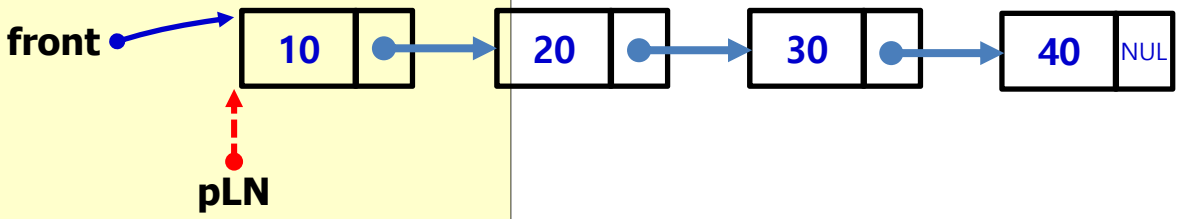
Doubly Linked List에서 탐색 (Search)

```
DLLN * search_DLL(DLLN *front, DATA key)
{
    DLLN *pLN;
    pLN = front;

    while( pLN != NULL ) {
        if (key == *(pLN->pData))
            return pLN;
        else
            pLN = pLN→next;
    }
    return NULL;
}
```



Print, Count, Sum Nodes in Linked List

```
void print_list(DLLN *front)
{
    DLLN *pLN;
    int count = 0;
    int sum = 0;
    pLN = front;

    while( pLN != NULL ) {
        printf("%d,  ", pLN→data);
        count++;
        sum += pLN→data;
        pLN = pLN→next;
    }
    printf("\n");
    printf("Length of list: %d\n", count);
    printf("Sum of list: %d\n", sum);
}
```



연결형 리스트 응용 프로그램

Student.h

```
/* Student.h */
#ifndef Student_H
#define Student_H
#define MAX_NAME_LEN 20
#define NUM_STUDENTS 10
typedef struct
{
    int st_id;
    char name[MAX_NAME_LEN];
    double GPA; // Grade Point Average
} Student;

void printStudent(Student st);
void printStudent(Student *pST);
void printStudents(Student *stArr, int num);
#endif
```

Student.cpp

```
/* Student.cpp (1) */
#include <stdio.h>
#include <string.h>
#include "Student.h"

Student students[NUM_STUDENTS] =
{
    { 21711000, "Kim, G-M", 3.57 },
    { 21611075, "Yoon, S-M", 4.37 },
    { 21411015, "Hwang, S-S", 2.72 },
    { 21611054, "Lee, K-M", 3.35 },
    { 21611340, "Hong, G-M", 3.89 },
    { 21712056, "Jang, S-M", 4.42 },
    { 21411017, "Park, S-S", 4.12 },
    { 21511053, "Choi, Y-H", 3.85 },
    { 21411017, "Shin, D-J", 3.21 },
    { 21511051, "Kwak, S-B", 4.45 },
};
```

Student.cpp

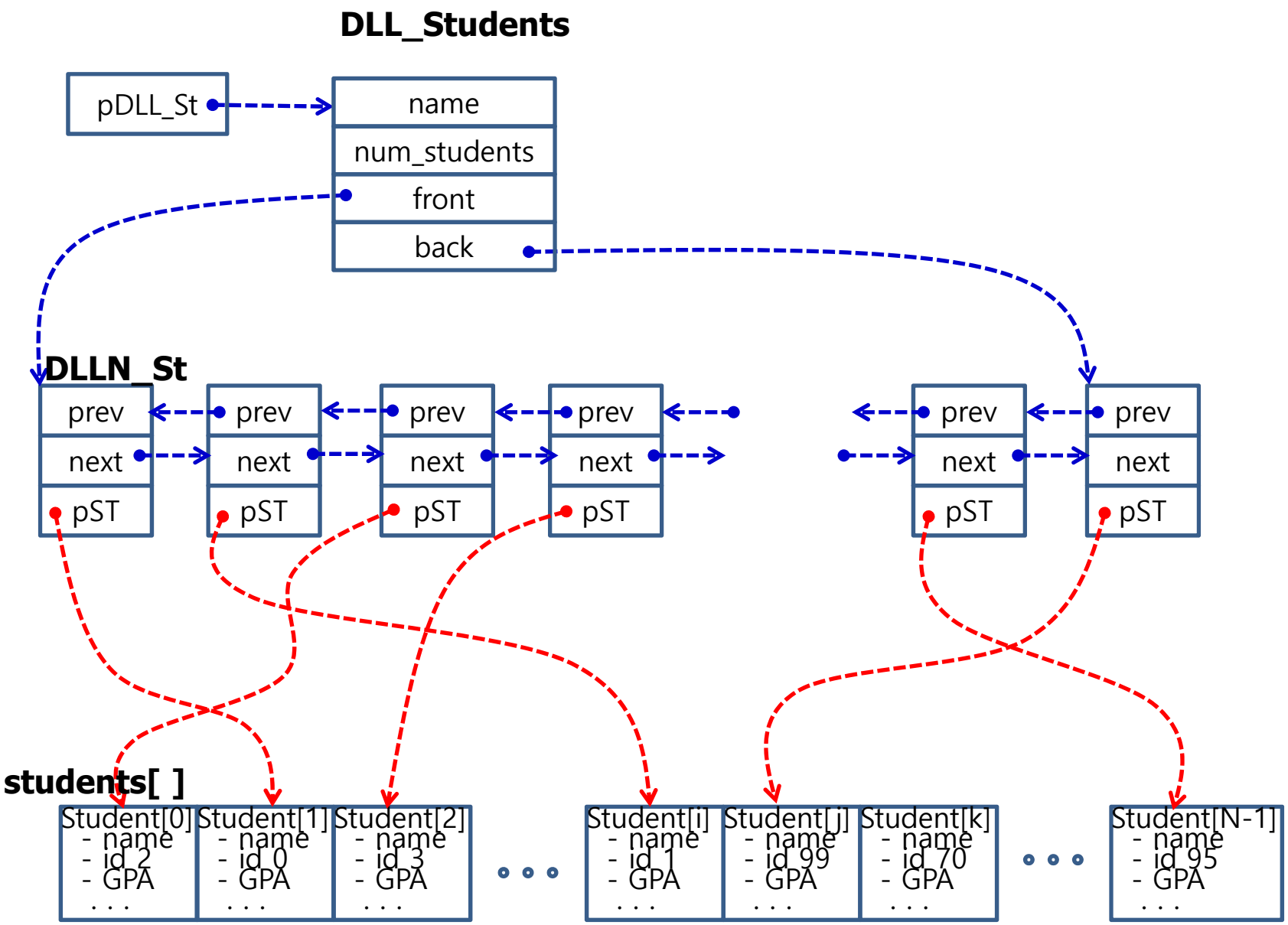
```
/* Student.cpp (2) */

void printStudent(Student st)
{
    printf("Student [ID: %08d, %-10s", st.st_id, st.name);
    printf(", GPA: %5.2lf", st.GPA);
    printf("]");
}

void printStudent(Student *pST)
{
    printf("Student [ID: %08d, %-10s", pST->st_id, pST->name);
    printf(", GPA: %5.2lf", pST->GPA);
    printf("]");
}

void printStudents(Student stArr[], int num)
{
    Student *st = stArr;
    for (int i = 0; i < num; i++)
    {
        printStudent(st);
        printf("₩n");
        st++;

    }
}
```

DLL_Student.h

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

#ifndef DLL_Student_H
#define DLL_Student_H
#include "Student.h"

#define MAX_NAME_LEN 64

struct DLLN_ST {
    Student* pST;  // data field
    struct DLLN_ST* next;
    struct DLLN_ST* prev;
};

typedef struct
{
    char name[MAX_NAME_LEN];
    int num_students;
    struct DLLN_ST *front;
    struct DLLN_ST *back;
} DLL_Student;
```

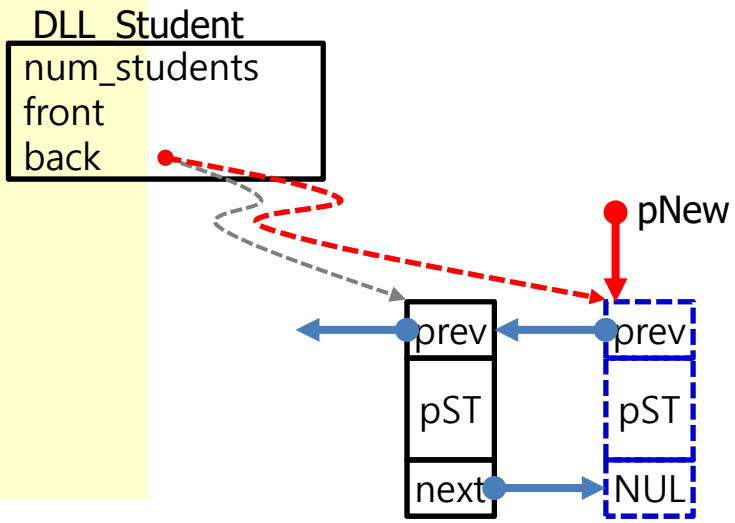
DLL_Student.h

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

void insertDLLN_atBack(DLL_Student* pDLL_St, Student* pST);
void insertDLLN_atFront(DLL_Student* pDLL_St, Student* pST);
void insertDLLN_inOrder_StID(DLL_Student* pDLL_St, Student* pST);
void insertDLLN_inReverseOrder_GPA(DLL_Student* pDLL_St, Student* pST);
Student* deleteNode_atFront(DLL_Student* pDLL_St);
Student* deleteNode_atBack(DLL_Student* pDLL_St);
Student* peek_atFront(DLL_Student* pDLL_St);
void printDLL(DLL_Student* pDLL_St);
void printDLL_reverse(DLL_Student* pDLL_St);

Student* DLL_getBestST_GPA(DLL_Student *ppDLL_St);
Student* DLL_findStudent_St_ID(DLL_Student *ppDLL_St, int st_id);


#endif
```

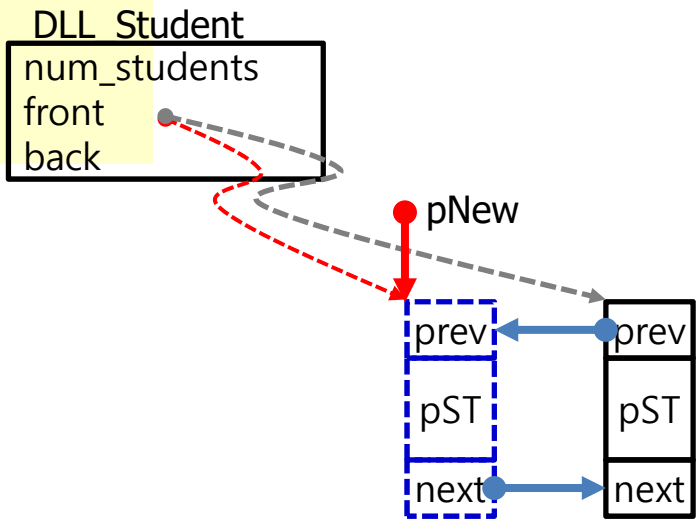

DLL_Students
pDLL_St
name
num_students
front
back
DLLN_St
prev
next
pST
students[ ]
Student[0]
- name
- id_2
- GPA
Student[1]
- name
- id_0
- GPA
Student[2]
- name
- id_3
- GPA
Student[i]
- name
- id_1
- GPA
Student[j]
- name
- id_99
- GPA
Student[k]
- name
- id_70
- GPA
Student[N-1]
- name
- id_95
- GPA

DLL_Student.cpp

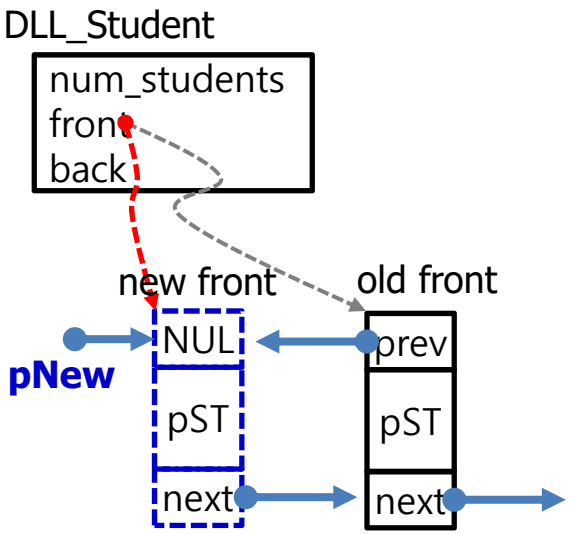
```
/* DLL_Student.cpp (1) */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "DLL_Student.h"

void insertDLLN_atBack(DLL_Student* pDLL_St, Student* pST)
{
    struct DLLN_ST* pNew = NULL;

    pNew = (struct DLLN_ST*)calloc(1, sizeof(struct DLLN_ST));
    pNew->pST = pST;
    pNew->next = pNew->prev = NULL;
    if (pDLL_St->num_students == 0)
    {
        pDLL_St->front = pDLL_St->back = pNew;
        pDLL_St->num_students++;
    }
    else {
        pNew->prev = pDLL_St->back;
        pDLL_St->back->next = pNew;
        pDLL_St->back = pNew;
        pDLL_St->num_students++;
    }
}
```

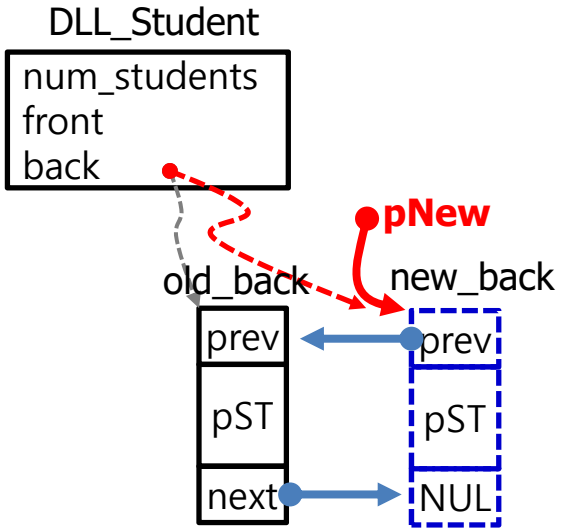


```
/* DLL_Student.cpp (2) */

void insertDLLN_atFront(DLL_Student* pDLL_St, Student* pST)
{
    struct DLLN_ST* pNew = NULL;

    pNew = (struct DLLN_ST*)calloc(1, sizeof(struct DLLN_ST));
    pNew->pST = pST;
    pNew->next = pNew->prev = NULL;
    if (pDLL_St->num_students == 0)
    {
        pDLL_St->front = pDLL_St->back = pNew;
        pDLL_St->num_students++;
    }
    else {
        pNew->next = pDLL_St->front;
        pDLL_St->front->prev = pNew;
        pDLL_St->front = pNew;
        pDLL_St->num_students++;
    }
}
```

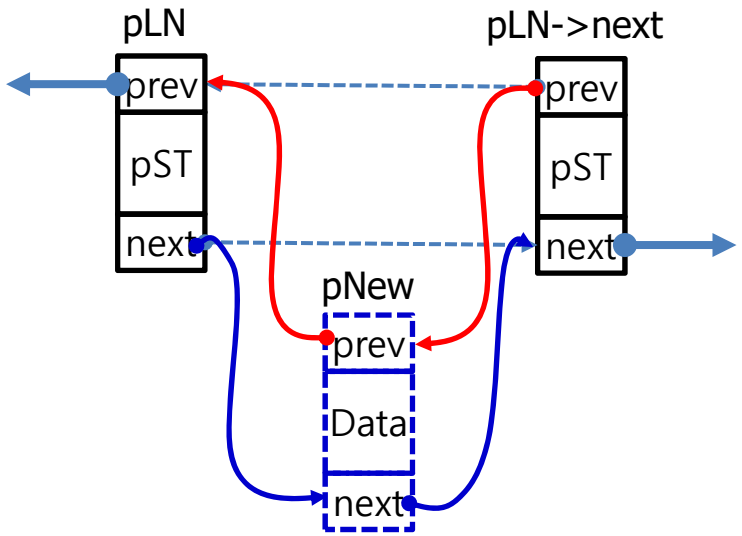


```
/* DLL_Student.cpp (3) */

void insertDLLN_inOrder_StID(DLL_Student*
      pDLL_St, Student* pST)
{
    DLLN_ST* pNew, * pLN;

    pNew = (DLLN_ST*)calloc(1, sizeof(DLLN_ST));
    pNew->next = pNew->prev = NULL;
    pNew->pST = pST;

    if (pDLL_St->num_students  ==  0)
    {
        pDLL_St->front = pDLL_St->back = pNew;
    }
    else if (pDLL_St->front->pST->st_id  > pST->st_id)
    {
        pNew->next = pDLL_St->front;
        pDLL_St->front->prev = pNew;
        pDLL_St->front = pNew;
    }
    else if (pST->st_id  >=  pDLL_St->back->pST->st_id)
    {
        pNew->prev = pDLL_St->back;
        pDLL_St->back->next = pNew;
        pDLL_St->back = pNew;
    }
```





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

else
   {
      pLN = pDLL_St->back;
      while (pLN != NULL)
      {
         if (pLN->pST->st_id  <=  pST->st_id)
            break;
         pLN = pLN->prev;
      }
      pNew->prev = pLN;
      pNew->next = pLN->next;
      pLN->next->prev = pNew;
      pLN->next = pNew;
   }
   pDLL_St->num_students++;
}
```



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

void insertDLLN_inReverseOrder_GPA(
     DLL_Student*pDLL_St, Student* pST)
{
    DLLN_ST* pNew, * pLN;

    pNew = (DLLN_ST*)calloc(1, sizeof(DLLN_ST));
    pNew->next = pNew->prev = NULL;
    pNew->pST = pST;

    if (pDLL_St->num_students == 0)
    {
        pDLL_St->front = pDLL_St->back = pNew;
    }
    else if (pDLL_St->front->pST->GPA < pST->GPA)
    {
        pNew->next = pDLL_St->front;
        pDLL_St->front->prev = pNew;
        pDLL_St->front = pNew;
    }
    else if (pST->GPA <= pDLL_St->back->pST->GPA)
    {
        pNew->prev = pDLL_St->back;
        pDLL_St->back->next = pNew;
        pDLL_St->back = pNew;
    }
```

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

    else
    {
        pLN = pDLL_St->back;
        while (pLN != NULL)
        {
            if (pLN->pST->GPA > pST->GPA)
                break;
            pLN = pLN->prev;
        }
        pNew->prev = pLN;
        pNew->next = pLN->next;
        pLN->next->prev = pNew;
        pLN->next = pNew;
    }
    pDLL_St->num_students++;
}
```

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

Student* deleteNode_atFront(
    DLL_Student* pDLL_St)
{
    struct DLLN_ST* pDLN_st = NULL;
    Student* pST;

    if (pDLL_St->num_students <= 0)
    {
        printf("DLL_Student is Empty !\n");
        return NULL;
    }
    pDLN_st = pDLL_St->front;
    pST = pDLN_st->pST;
    pDLL_St->front = pDLL_St->front->next;
    if (pDLL_St->front != NULL)
        pDLL_St->front->prev = NULL;
    pDLL_St->num_students--;
    free(pDLN_st);
    return pST;
}
```

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

Student* deleteNode_atEnd(
     DLL_Student* pDLL_St)
{
    struct DLLN_ST* pDLN_st = NULL;
    Student* pST;

    if (pDLL_St->num_students <= 0)
    {
        printf("DLL_Student is Empty !\n");
        return NULL;
    }
    pDLN_st = pDLL_St->back;
    pST = pDLN_st->pST;
    pDLL_St->back = pDLL_St->back->prev;
    if (pDLL_St->back != NULL)
        pDLL_St->back->next = NULL;
    pDLL_St->num_students--;
    free(pDLN_st);
    return pST;
}
```

```
/* DLL_Student.cpp (9) */

Student* peek_atFront(DLL_Student* pDLL_St)
{
    struct DLLN_ST* pDLN_st = NULL;
    Student* pST;

    if (pDLL_St->num_students <= 0)
    {
        printf("DLL_Student is Empty !\n");
        return NULL;
    }
    pDLN_st = pDLL_St->front;
    pST = pDLN_st->pST;

    return pST;
}
```

```
/* DLL_Student.cpp (10) */

void printDLL(DLL_Student* pDLL_St)
{
    struct DLLN_ST* pDLN_st = NULL;
    Student* pST;
    int count;

    pDLN_st = pDLL_St->front;
    count = 0;
    while ((pDLN_st != NULL) && (count < pDLL_St->num_students))
    {
        pST = pDLN_st->pST;
        if (pST != NULL)
            printStudent(pST);
        printf("\n");
        pDLN_st = pDLN_st->next;
        count++;
    }
}

void printDLL_reverse(DLL_Student* pDLL_St)
{
    struct DLLN_ST* pDLN_st = NULL;
    Student* pST;
    int count;

    pDLN_st = pDLL_St->back;
    count = 0;
    while ((pDLN_st != NULL) && (count < pDLL_St->num_students))
    {
        pST = pDLN_st->pST;
        if (pST != NULL)
            printStudent(pST);
        printf("\n");
        pDLN_st = pDLN_st->prev;
        count++;
    }
}
```

```
/* DLL_Student.cpp (11) */

Student* DLL_getBestST_GPA(DLL_Student *ppDLL_St)
{
    Student *pST, *pST_bestGPA = NULL;
    struct DLLN_ST *pDLLN_st = NULL;
    int num_students;
    int count = 0;
    double bestGPA = 0.0;

    if ((num_students = ppDLL_St->num_students) == 0)
        return NULL;
    pDLLN_st = ppDLL_St->front;
    count = 0;
    while ((pDLLN_st != NULL) && (count < num_students))
    {
        pST = pDLLN_st->pST;
        if ((pST != NULL) && (pST->GPA > bestGPA))
        {
            pST_bestGPA = pST;
            bestGPA = pST_bestGPA->GPA;
         }
        pDLLN_st = pDLLN_st->next;
    }
    return pST_bestGPA;
}
```

```
/* DLL_Student.cpp (12) */

Student* DLL_findStudent_St_ID(DLL_Student *ppDLL_St, int st_id)
{
    Student *pST= NULL;
    struct DLLN_ST *pDLLN_st = NULL;
    int num_students;
    int count = 0;
    double bestGPA = 0.0;

    if ((num_students = ppDLL_St->num_students) == 0)
        return NULL;
    pDLLN_st = ppDLL_St->front;
    count = 0;
    while ((pDLLN_st != NULL) && (count < num_students))
    {
        pST = pDLLN_st->pST;
        if ((pST != NULL) && (pST->st_id == st_id))
            break;
        pDLLN_st = pDLLN_st->next;
        count++;
    }
    return pST;
}
```

main()

```
/* main.c (1) */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Student.h"
#include "DLL_Student.h"

extern Student students[num_students];

void main()
{
    Student *pST;

    printf("Array of students at initialization : \n");
    printStudents(students, NUM_STUDENTS);
    printf("\n");

    DLL_Student *pDLL_St = (DLL_Student *)calloc(1, sizeof(DLL_Student));
    strcpy(pDLL_St->name, (const char*)"DLL_Students");
    pDLL_St->front = pDLL_St->back = NULL;
    pDLL_St->num_students = 0;
    for (int i = 0; i < NUM_STUDENTS; i++)
    {
        pST = &students[i];
        insertDLLN_inOrder_StID(pDLL_St, pST);
    }
    printDLL(pDLL_St);
    printf("\n");
```

```
Array of students at initialization :
Student [ID: 21711000, Kim, G-M  , GPA:  3.57]
Student [ID: 21611075, Yoon, S-M , GPA:  4.37]
Student [ID: 21411015, Hwang, S-S, GPA:  2.72]
Student [ID: 21611054, Lee, K-M  , GPA:  3.35]
Student [ID: 21611340, Hong, G-M , GPA:  3.89]
Student [ID: 21712056, Jang, S-M , GPA:  4.42]
Student [ID: 21411017, Park, S-S , GPA:  4.12]
Student [ID: 21511053, Choi, Y-H , GPA:  3.85]
Student [ID: 21411017, Shin, D-J , GPA:  3.21]
Student [ID: 21511051, Kwak, S-B , GPA:  4.45]
```

```
DLL_Students (number of students: 10):
Student [ID: 21411015, Hwang, S-S, GPA:  2.72]
Student [ID: 21411017, Park, S-S , GPA:  4.12]
Student [ID: 21411017, Shin, D-J , GPA:  3.21]
Student [ID: 21511051, Kwak, S-B , GPA:  4.45]
Student [ID: 21511053, Choi, Y-H , GPA:  3.85]
Student [ID: 21611054, Lee, K-M  , GPA:  3.35]
Student [ID: 21611075, Yoon, S-M , GPA:  4.37]
Student [ID: 21611340, Hong, G-M , GPA:  3.89]
Student [ID: 21711000, Kim, G-M  , GPA:  3.57]
Student [ID: 21712056, Jang, S-M , GPA:  4.42]
```

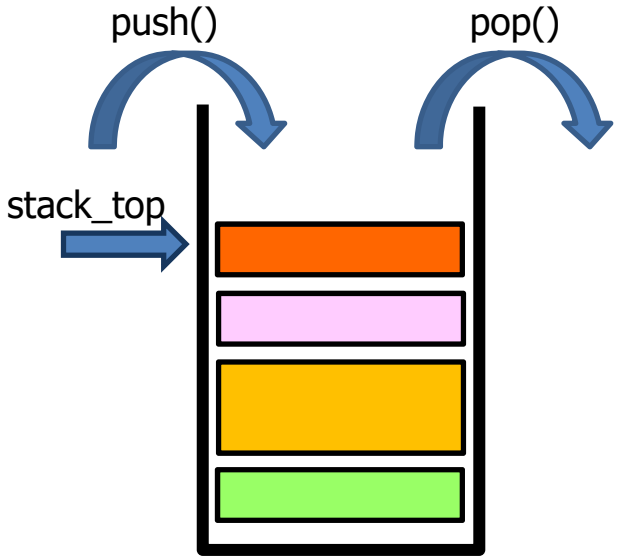
```
/* main.c (2) */

    DLL_Student* pDLL_St_GPA = (DLL_Student*)calloc(1, sizeof(DLL_Student));
    strcpy(pDLL_St_GPA->name, (const char*)"DLL_Students in reverse order of GPA");
    pDLL_St_GPA->front = pDLL_St_GPA->back = NULL;
    pDLL_St_GPA->num_students = 0;
    for (int i = 0; i < NUM_STUDENTS; i++)
    {
        pST = &students[i];
        insertDLLN_inReverseOrder_GPA(pDLL_St_GPA, pST);
    }
    printDLL(pDLL_St_GPA);
    printf("\n");

    int st_id = students[num_students - 1].st_id;
    pST = DLL_findStudent_St_ID(pDLL_St, st_id);
    printf("The student with st_id (%d) is :\n   ", st_id);
    printStudent(pST);     printf("\n\n");

    pST = DLL_getBestST_GPA(pDLL_St);
    printf("The student with best GPA is :\n   ");
    printStudent(pST);     printf("\n\n");
    deleteDLL(pDLL_St);
    deleteDLL(pDLL_St_GPA);

}
```

```
DLL_Students in reverse order of GPA (number of students: 10):
Student [ID: 21511051, Kwak, S-B , GPA:  4.45]
Student [ID: 21712056, Jang, S-M , GPA:  4.42]
Student [ID: 21611075, Yoon, S-M , GPA:  4.37]
Student [ID: 21411017, Park, S-S , GPA:  4.12]
Student [ID: 21611340, Hong, G-M , GPA:  3.89]
Student [ID: 21511053, Choi, Y-H , GPA:  3.85]
Student [ID: 21711000, Kim, G-M  , GPA:  3.57]
Student [ID: 21611054, Lee, K-M  , GPA:  3.35]
Student [ID: 21411017, Shin, D-J , GPA:  3.21]
Student [ID: 21411015, Hwang, S-S, GPA:  2.72]

The student with st_id (21511051) is :
   Student [ID: 21511051, Kwak, S-B , GPA:  4.45]

The student with best GPA is :
   Student [ID: 21511051, Kwak, S-B , GPA:  4.45]
```

연결형 리스트 구조의
LIFO 스택 (Stack)

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 LIFO 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 method calls in the C++ run-time system

◆ Indirect applications

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

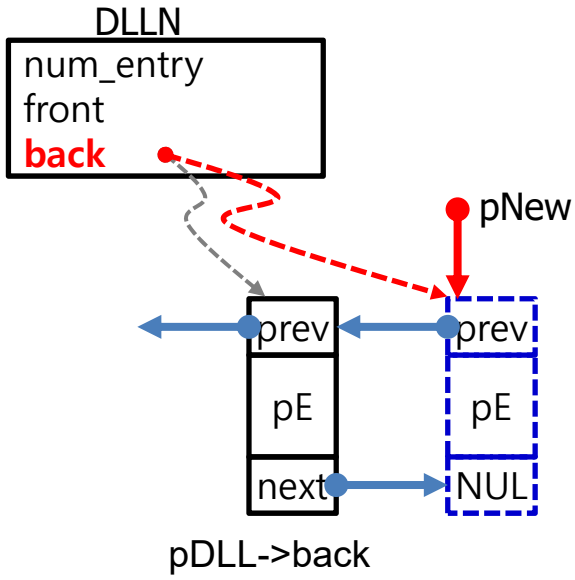
DoublyLinkedList.h

```
/* DoublyLinkedList.h  (1) */
#ifndef DOUBLY_LINKED_LIST_H
#define DOUBLY_LINKED_LIST_H
#include <stdio.h>
#include <stdlib.h>

typedef int Entry_T;

typedef struct DLLN
{
    Entry_T *pE;
    struct DLLN *prev;
    struct DLLN *next;
} DLLN;

typedef struct DoublyLinkedList
{
    char name[50];
    int num_entry;
    DLLN *front;
    DLLN *back;
} DLL;
```

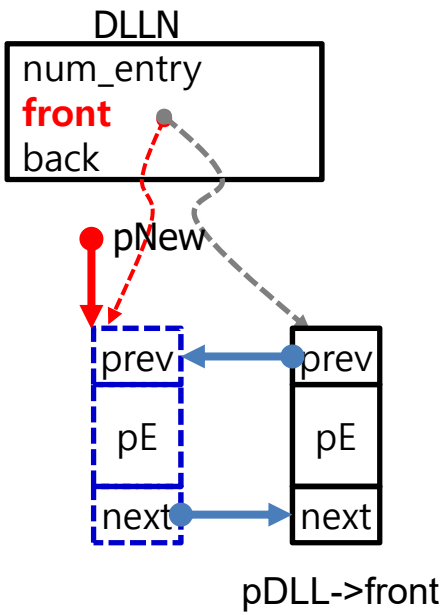
```
/* DoublyLinkedList.h  (2) */

void initDLL(DLL* pDLL, const char* nm);
DLLN* insertEntry_atEnd(DLL *pDLL, Entry_T *pE);
DLLN* insertEntry_atFront(DLL *pDLL, Entry_T *pE);
Entry_T* deleteEntry_atFront(DLL *pDLL);
Entry_T* deleteEntry_atEnd(DLL *pDLL);
Entry_T* peekEntry_atFront(DLL *pDLL);
void printDLL(DLL *pDLL);

#endif
```

DoublyLinkedList.c

```
/* DoublyLinkedList.c (1) */
#include <stdio.h>
#include <string.h>
#include "DoublyLinkedList.h"

void initDLL(DLL* pDLL, const char* nm)
{
    strcpy(pDLL->name, nm);
    pDLL->num_entry = 0;
    pDLL->front = pDLL->back = NULL;
}

DLLN* insertEntry_atEnd(DLL *pDLL, Entry_T *pE)
{
    DLLN *pNew = NULL;

    pNew = (DLLN *)calloc(1, sizeof(DLLN));
    pNew->pE = pE;
    pNew->next = pNew->prev = NULL;
    if (pDLL->num_entry == 0)
    {
        pDLL->front = pDLL->back = pNew;
        pDLL->num_entry++;
    }
    else {
        pNew->prev = pDLL->back;
        pDLL->back->next = pNew;
        pDLL->back = pNew;
        pDLL->num_entry++;
    }
    return pNew;
}
```



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

DLLN* insertEntry_atFront(DLL *pDLL, Entry_T *pE)
{
    DLLN *pNew = NULL;

    pNew = (DLLN *)calloc(1, sizeof(DLLN));
    pNew->pE = pE;
    pNew->next = pNew->prev = NULL;
    if (pDLL->num_entry == 0)
    {
        pDLL->front = pDLL->back = pNew;
        pDLL->num_entry++;
    }
    else {
        pNew->next = pDLL->front;
        pDLL->front->prev = pNew;
        pDLL->front = pNew;
        pDLL->num_entry++;
    }
    return pNew;
}

void printEntry(Entry_T* pE)
{
    printf("%3d", *pE); // Entry_T is defined as int
}
```

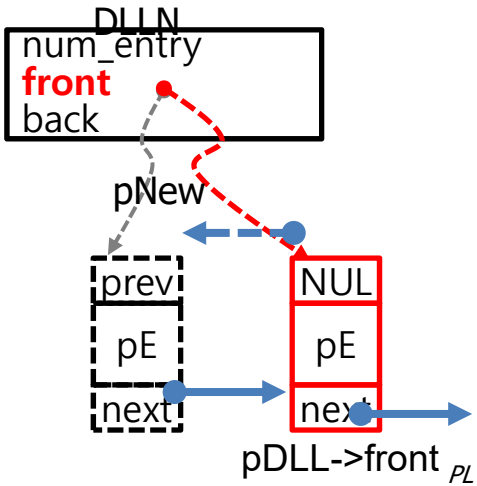


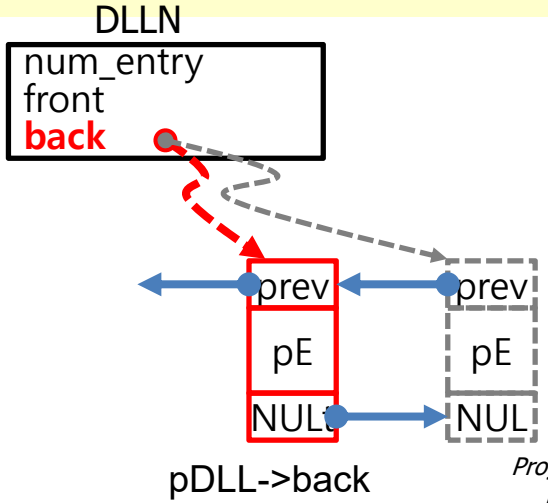
/* DoublyLinkedList.c (3) */

```
Entry_T* deleteEntry_atFront(DLL *pDLL)
{
    DLLN *pDLN_st = NULL;
    Entry_T *pE;

    if (pDLL->num_entry <= 0)
    {
        printf("LinkedList of DLL is Empty !\n");
        return NULL;
    }
    pDLN_st = pDLL->front;
    pE = pDLN_st->pE;
    pDLL->front = pDLL->front->next;
    if (pDLL->front != NULL)
        pDLL->front->prev = NULL;
    pDLL->num_entry--;
    free(pDLN_st);
    return pE;
}
```

/* DoublyLinkedList.c (4) */

```
Entry_T* deleteEntry_atEnd(DLL *pDLL)
{
    DLLN *pDLN_st = NULL;
    Entry_T *pE;

    if (pDLL->num_entry <= 0)
    {
        printf("LinkedList of DLL is Empty !\n");
        return NULL;
    }
    pDLN_st = pDLL->back;
    pE = pDLN_st->pE;
    pDLL->back = pDLL->back->prev;
    if (pDLL->back != NULL)
        pDLL->back->next = NULL;
    pDLL->num_entry--;
    free(pDLN_st);
    return pE;
}
```





```
/* DoublyLinkedList.c (5) */

Entry_T* peekEntry_atFront(DLL *pDLL)
{
    DLLN *pDLN_st = NULL;
    Entry_T *pE;

    if (pDLL->num_entry <= 0)
    {
        printf("LinkedList of DLL is Empty !\n");
        return NULL;
    }
    pDLN_st = pDLL->front;
    pE = pDLN_st->pE;

    return pE;
}
```

```
/* DoublyLinkedList.c (6) */

void printDLL(DLL *pDLL)
{
    DLLN *pDLN_st = NULL;
    Entry_T *pE;
    int count;

    printf("%s (num_entry %2d) : ", pDLL->name, pDLL->num_entry);
    if (pDLL->num_entry == 0)
    {
        printf("DLL is Empty !\n");
        return;
    }

    pDLN_st = pDLL->front;
    count = 0;
    while ((pDLN_st != NULL) && (count < pDLL->num_entry))
    {
        pE = pDLN_st->pE;
        if (pE != NULL)
            printEntry(pE);
        pDLN_st = pDLN_st->next;
        count++;
    }
    printf("\n");
}
```

```
/* DoublyLinkedList.c (7) */

void printDLL_reverse(DLL *pDLL)
{
    DLLN *pDLN_st = NULL;
    Entry_T *pE;
    int count;

    printf("%s (num_entry %2d) in reverse order : ", pDLL->name, pDLL->num_entry);
    if (pDLL->num_entry == 0)
    {
        printf("DLL is Empty !\n");
        return;
    }

    pDLN_st = pDLL->back;
    count = 0;
    while ((pDLN_st != NULL) && (count < pDLL->num_entry))
    {
        pE = pDLN_st->pE;
        if (pE != NULL)
            printEntry(pE);
        pDLN_st = pDLN_st->prev;
        count++;
    }
    printf("\n");
}
```

DLL_Stack.h

```
/* DLL_Stack.h */
#ifndef DLL_STACK_H
#define DLL_STACK_H

#include "DoublyLinkedList.h"
#define MAX_NAME_LEN 30
typedef struct DLL_Stack
{
    char name[MAX_NAME_LEN];
    DLL *pDLL;
    int size;
} DLL_Stack;

void initStack(DLL_Stack *pS, const char *name);
void push(DLL_Stack *pS, Entry_T *pE);
Entry_T* pop(DLL_Stack *pS);
void printStack(DLL_Stack *pS);
#endif
```

DLL_Stack.c

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

void initStack(DLL_Stack *pS, const char *name)
{
    DLL *pDLL;

    strcpy(pS->name, name);
    pDLL = (DLL *)calloc(1, sizeof(DLL));
    pDLL->front = pDLL->back = NULL;
    pDLL->num_entry = 0;
    pS->pDLL = pDLL;
    initDLL(pDLL, name);
}

void push(DLL_Stack *pS, Entry_T *pE)
{
    DLLN* pDLLN;

    pDLLN = insertEntry_atFront(pS->pDLL, pE);
}
```

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

Entry_T* pop(DLL_Stack *pS)
{
    Entry_T *pE;

    pE = deleteEntry_atFront(pS->pDLL);
    return pE;
}

void printStack(DLL_Stack *pS)
{
    if (pS->pDLL->num_entry == 0)
    {
        printf("%s is empty !\n", pS->name);
        return;
    }
    printDLL(pS->pDLL);
}
```

main()

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

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

#define NUM_DATA 10
void main()
{
    DLL_Stack *stack;
    Entry_T data[NUM_DATA] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    Entry_T *pE;

    stack = (DLL_Stack *)calloc(1, sizeof(DLL_Stack));
    initStack(stack, (const char *)("Int_DLL_Stack"));
    printf("After initialization of stack\n");
    printStack(stack);
```

```
After initialization of stack
Int_DLL_Stack is empty !

Test Int_DLL_Stack with repeated push operations...
Pushed entry:   0
 Int_DLL_Stack (num_entry  1) :   0
Pushed entry:   1
 Int_DLL_Stack (num_entry  2) :   1  0
Pushed entry:   2
 Int_DLL_Stack (num_entry  3) :   2  1  0
Pushed entry:   3
 Int_DLL_Stack (num_entry  4) :   3  2  1  0
Pushed entry:   4
 Int_DLL_Stack (num_entry  5) :   4  3  2  1  0
Pushed entry:   5
 Int_DLL_Stack (num_entry  6) :   5  4  3  2  1  0
Pushed entry:   6
 Int_DLL_Stack (num_entry  7) :   6  5  4  3  2  1  0
Pushed entry:   7
 Int_DLL_Stack (num_entry  8) :   7  6  5  4  3  2  1  0
Pushed entry:   8
 Int_DLL_Stack (num_entry  9) :   8  7  6  5  4  3  2  1  0
Pushed entry:   9
 Int_DLL_Stack (num_entry 10) :   9  8  7  6  5  4  3  2  1  0

Test Int_DLL_Stack with repeated pop operations...
Poped entry:   9
 Int_DLL_Stack (num_entry  9) :   8  7  6  5  4  3  2  1  0
Poped entry:   8
 Int_DLL_Stack (num_entry  8) :   7  6  5  4  3  2  1  0
Poped entry:   7
 Int_DLL_Stack (num_entry  7) :   6  5  4  3  2  1  0
Poped entry:   6
 Int_DLL_Stack (num_entry  6) :   5  4  3  2  1  0
Poped entry:   5
 Int_DLL_Stack (num_entry  5) :   4  3  2  1  0
Poped entry:   4
 Int_DLL_Stack (num_entry  4) :   3  2  1  0
Poped entry:   3
 Int_DLL_Stack (num_entry  3) :   2  1  0
Poped entry:   2
 Int_DLL_Stack (num_entry  2) :   1  0
Poped entry:   1
 Int_DLL_Stack (num_entry  1) :   0
Poped entry:   0
 Int_DLL_Stack is empty !
```

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

   printf("\nTest %s with repeated push operations...\n", stack->name);
   for (int i = 0; i < 10; i++)
   {
      pE = &data[i];
      push(stack, pE);
      printf("Pushed entry: ");
      printEntry(pE);
      printf("\n ");
      printStack(stack);
   }

   printf("\nTest %s with repeated
      pop operations...\n", stack->name);
   for (int i = 0; i < 10; i++)
   {
      pE = pop(stack);
      printf("Poped entry: ");
      printEntry(pE);
      printf("\n ");
      printStack(stack);
   }
}
```

```
After initialization of stack
Int_DLL_Stack is empty !

Test Int_DLL_Stack with repeated push operations...
Pushed entry:   0
 Int_DLL_Stack (num_entry  1) :   0
Pushed entry:   1
 Int_DLL_Stack (num_entry  2) :   1  0
Pushed entry:   2
 Int_DLL_Stack (num_entry  3) :   2  1  0
Pushed entry:   3
 Int_DLL_Stack (num_entry  4) :   3  2  1  0
Pushed entry:   4
 Int_DLL_Stack (num_entry  5) :   4  3  2  1  0
Pushed entry:   5
 Int_DLL_Stack (num_entry  6) :   5  4  3  2  1  0
Pushed entry:   6
 Int_DLL_Stack (num_entry  7) :   6  5  4  3  2  1  0
Pushed entry:   7
 Int_DLL_Stack (num_entry  8) :   7  6  5  4  3  2  1  0
Pushed entry:   8
 Int_DLL_Stack (num_entry  9) :   8  7  6  5  4  3  2  1  0
Pushed entry:   9
 Int_DLL_Stack (num_entry 10) :   9  8  7  6  5  4  3  2  1  0

Test Int_DLL_Stack with repeated pop operations...
Poped entry:   9
 Int_DLL_Stack (num_entry  9) :   8  7  6  5  4  3  2  1  0
Poped entry:   8
 Int_DLL_Stack (num_entry  8) :   7  6  5  4  3  2  1  0
Poped entry:   7
 Int_DLL_Stack (num_entry  7) :   6  5  4  3  2  1  0
Poped entry:   6
 Int_DLL_Stack (num_entry  6) :   5  4  3  2  1  0
Poped entry:   5
 Int_DLL_Stack (num_entry  5) :   4  3  2  1  0
Poped entry:   4
 Int_DLL_Stack (num_entry  4) :   3  2  1  0
Poped entry:   3
 Int_DLL_Stack (num_entry  3) :   2  1  0
Poped entry:   2
 Int_DLL_Stack (num_entry  2) :   1  0
Poped entry:   1
 Int_DLL_Stack (num_entry  1) :   0
Poped entry:   0
 Int_DLL_Stack is empty !
```

연결형 리스트 구조의
FIFO 큐(Queue)

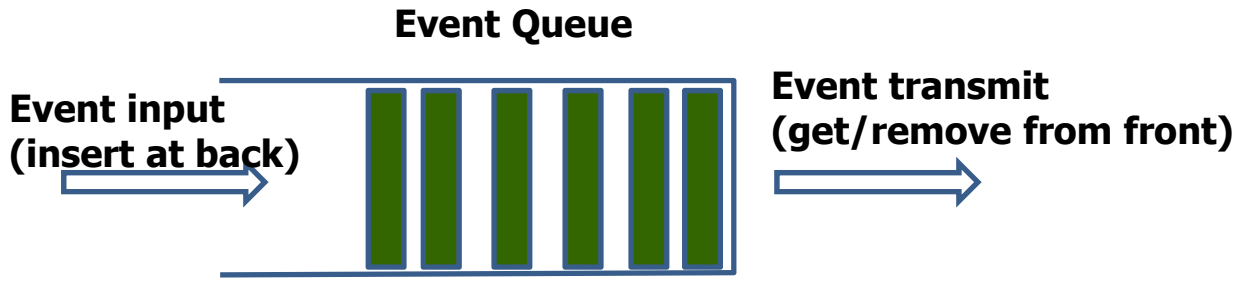
First-In First-Out (FIFO) Queues

- The Queue stores arbitrary objects
- Insertions and deletions follow the first-in first-out (FIFO) scheme
- Insertions are at the back(rear) of the queue and removals are at the front of the queue
- Main queue operations:
  - **enqueue(object)**: inserts an element at the back(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

FIFO Queue의 응용 – 이벤트 시스템의 큐

◆ 이벤트 시스템에서의 입력/출력 Queue

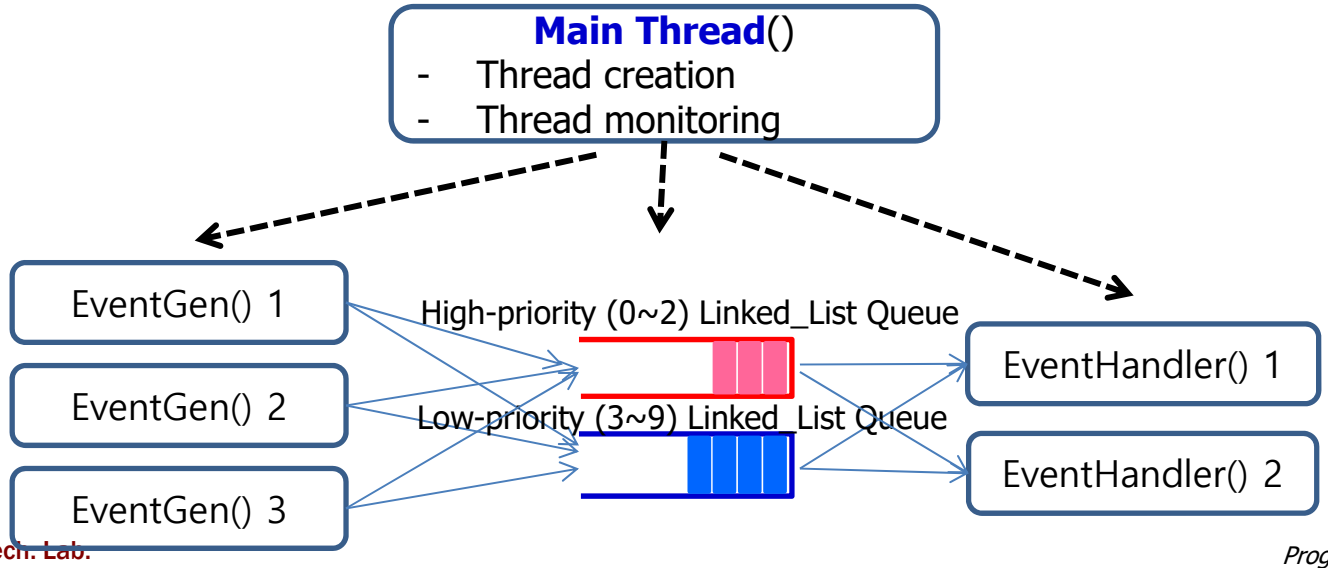
- queue: first come, first served
- ***enqueue*** **()** - inserts a newly arrived event in the tail of the queue
- ***front()*** - gets the event at the front of the queue
- ***dequeue()*** - deletes the acknowledged event from the queue



Multi-thread와 Linked List Queue의 응용 예제

◆ Simple Simulation of Event Generator, Linked List Queues, Event Handler

- Two kinds of Threads
  - Event Generator
  - Event Handler
- Two shared Doubly Linked List Queue
  - EventQ_HighPriority
  - EventQ_LowPriority



이벤트 (Event) 이란 ?

◆ 이벤트 (Event)

- 다양하게 발생되는 사건들을 모델링
- 이벤트의 중요 항목
  - 생성자 주소: 이벤트가 발생된 노드의 주소
  - 처리자 주소: 이벤트를 처리한 노드의 주소
  - 이벤트번호 (event id): 동일한 송신-수신 단말장치간에 전달되는 이벤트들을 구분하기 위한 번호.
  - 우선순위 (priority 또는 precedence): 이벤트의 종류에 따라 우선적으로 처리하여야 하는 필요성을 나타내는 정보

Event

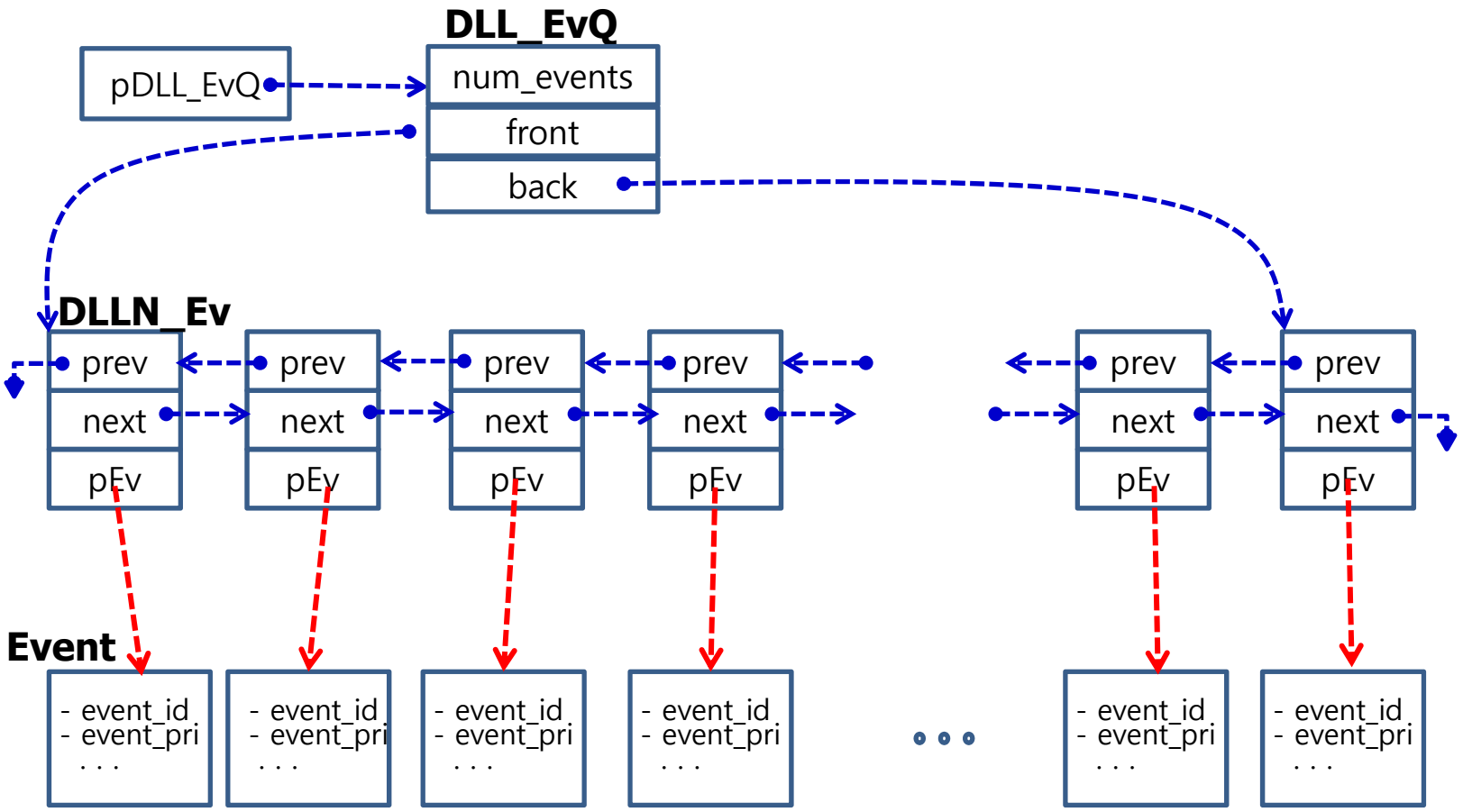
```
/* Event.h  (1) */
#ifndef EVENT_H
#define EVENT_H

#include <stdio.h>
#include <Windows.h>
#include "ConsoleDisplay.h"
#include "SimParams.h"

#define NUM_PRIORITY 10
#define PRIORITY_THRESHOLD 3
#define EVENT_PER_LINE 5

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

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

typedef struct
{
    int event_no;
    int event_gen_addr;
    int event_handler_addr;
    int event_pri;  // event_priority
    LARGE_INTEGER t_gen;
    LARGE_INTEGER t_proc;
    double t_elapsed;
    EventStatus eventStatus;
} Event;

void printEvent(Event* pEvt);
void printEvent_withTime(Event* pEv);
void calc_elapsed_time(Event *pEv,
      LARGE_INTEGER freq);

#endif
```

```
/* Event.cpp */
#include <stdio.h>
#include <stdlib.h>
#include "Event.h"

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

void printEvent(Event* pEvent)
{
      printf("Ev(no:%3d, pri:%2d)  ", pEvent->event_no, pEvent->event_pri);
}
void printEvent_withTime(Event* pEv)
{
       printf("Ev(no:%3d, pri:%2d, %6.0lf[ms])  ", pEv->event_no, pEv->event_pri,
          pEv->t_elapsed * 1000);
}

void calc_elapsed_time(Event *pEv, LARGE_INTEGER freq)
{
       LONGLONG t_diff_LL;
       double t_elapsed;

       t_diff_LL = pEv->t_proc.QuadPart - pEv->t_gen.QuadPart;
       t_elapsed = (double) t_diff_LL / (double)freq.QuadPart;
       pEv->t_elapsed = t_elapsed;
}
```

Doubly Linked List Event Queue

◆ Doubly Linked List Event Queue



Linked List Queue for Event Handling

```
/* DLL_EvQ.h  (1) */
#ifndef DLL_EvQ_H
#define DLL_EvQ_H

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

using namespace std;

// doubly linked list node (DLLN)
typedef struct DLLN
{
    DLLN *prev;
    DLLN *next;
    Event *pEv;
} DLLN_Ev;
```

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

typedef struct
{
    char name[50];
    mutex cs_EvQ;
    int priority;
    DLLN_Ev *front;
    DLLN_Ev *back;
    int num_event;
} DLL_EvQ;

void initDLL_EvQ(DLL_EvQ *DLL_EvQ, int priority);
Event *enDLL_EvQ(DLL_EvQ *DLL_EvQ, Event *pEv);
Event *deDLL_EvQ(DLL_EvQ *DLL_EvQ);
void printDLL_EvQ(DLL_EvQ *DLL_EvQ);
#endif
```

```
/* DLL_EvQ.cpp (1) */
#include "DLL_EvQ.h"

void initDLL_EvQ(DLL_EvQ *pEvQ, int pri)
{
   pEvQ->cs_EvQ.lock();
   pEvQ->priority = pri;
   pEvQ->front = pEvQ->back = NULL;
   pEvQ->num_event = 0;
   pEvQ->cs_EvQ.unlock();
}

Event * enDLL_EvQ(DLL_EvQ *pEvQ, Event *pEv)
{
   DLLN_Ev *pLN_Ev;
   if (pEv == NULL)
   {
      printf("Error in enDLL_EvQ :: DLL_EvQ is NULL !!\n");
      printf("Press any key to continue ...\n");
      getc(stdin);
      return NULL;
   }
   pLN_Ev = (DLLN_Ev *)calloc(1, sizeof(DLLN_Ev));
   if (pLN_Ev == NULL)
   {
      printf("Error in enDLL_EvQ :: memory allocation for new DLLN failed !!\n");
      printf("Press any key to continue ...\n");
      getc(stdin);
      return NULL;
   }
```

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

   pLN_Ev->pEv = pEv;
   pEvQ->cs_EvQ.lock();
   if (pEvQ->num_event == 0) // currently empty
   {
      pEvQ->front = pEvQ->back = pLN_Ev;
      pLN_Ev->prev = pLN_Ev->next = NULL;
      pEvQ->num_event = 1;
   }
   else
   {
      pLN_Ev->prev = pEvQ->back;
      pEvQ->back->next = pLN_Ev;
      pEvQ->back = pLN_Ev;
      pLN_Ev->next = NULL;
      pEvQ->num_event++;
   }
   pEvQ->cs_EvQ.unlock();
   return pLN_Ev->pEv;
}
```

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

Event *deDLL_EvQ(DLL_EvQ *pEvQ)
{
   Event *pEv;
   DLLN_Ev *pLN_Ev_OldFront;

   pEvQ->cs_EvQ.lock();
   if (pEvQ->num_event <= 0)
   {
      pEvQ->cs_EvQ.unlock();
      return NULL;  // DLL_EvQ is Empty
   }
   else
   {
      pLN_Ev_OldFront = pEvQ->front;
      pEv = pEvQ->front->pEv;
      pEvQ->front = pEvQ->front->next;
      if (pEvQ->front != NULL)
         pEvQ->front->prev = NULL;
      pEvQ->num_event--;
      free(pLN_Ev_OldFront); // release memory for the current front DLLN
      pEvQ->cs_EvQ.unlock();
      return pEv;
   }
}
```

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

void printDLL_EvQ(DLL_EvQ *pEvQ)
{
   int index = 0;
   int count;
   Event *pEv;
   DLLN_Ev *pLN_Ev;
   if (pEvQ == NULL)
   {
      printf("Error in printDLL_EvQ :: DLL_EvQ is NULL !!");
      printf("Press any key to continue ...\n");
      getc(stdin);
   }
   //printf("DLL_EvQ(num_event: %2d):\n   ", pEvQ->num_event);
   if (pEvQ->num_event <= 0)
      return;
   pLN_Ev = pEvQ->front;
   count = 0;
   while (pLN_Ev != NULL)
   {
      pEv = pLN_Ev->pEv;
      if (pEv == NULL)
         break;
      printEvent(pEv); printf(" ");
      count++;
      if ((count % 5) == 0)
         printf("\n   ");
      pLN_Ev = pLN_Ev->next;
   }
}
```

Doubly Linked List 구조의 FIFO Queue와 Multi-thread 응용 프로그램

Thread.h

```
/* Thread.h  (1)*/
#ifndef THREAD_H
#define THREAD_H
#include <Windows.h>
#include <thread>
#include <mutex>
#include <process.h>
#include "Event.h"
#include "SimParams.h"
#include "DLL_EvQ.h"

using namespace std;

enum ROLE {EVENT_GENERATOR, EVENT_HANDLER};
enum THREAD_FLAG {INITIALIZE, RUN, TERMINATE};

typedef struct
{
    int eventsGen[NUM_EVENT_GENERATORS];
    int eventsProc[NUM_EVENT_HANDLERS];
    int totalEventGen;
    int totalEventProc;
    int numEventProcs_priH;
    int numEventProcs_priL;
    THREAD_FLAG *pFlagThreadTerminate;
    Event eventGenerated[TOTAL_NUM_EVENTS];
    Event eventProcessed[TOTAL_NUM_EVENTS];
} ThreadStatMon;
```

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

typedef struct
{
    FILE *fout;
    mutex *pCS_main;
    mutex *pCS_thrd_mon;
    DLL_EvQ *EvQ_PriH;
    DLL_EvQ *EvQ_PriL;
    ROLE role;
    int myAddr;
    int maxRound;
    int targetEventGen;
    LARGE_INTEGER PC_freq;
      // frequency of performance counter
      // that is used to measure elapsed time
    ThreadStatMon *pThrdMon;
} ThreadParam_Ev;

void Thread_EventHandler(ThreadParam_Ev*
    pParam);
void Thread_EventGenerator(ThreadParam_Ev*
    pParam);
#endif
```

Thread_EventGenerator

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

#include <Windows.h>
#include <time.h>
#include <thread>
#include "Thread.h"
#include "DLL_EvQ.h"
#include "Event.h"
using namespace std;

void Thread_EventGenerator(ThreadParam_Ev* pThrdParam)
{

      Event *pEv;
   int event_no = 0;
   int event_pri = 0;
   int event_gen_count = 0;
   int myRole = pThrdParam->role;
   int myGenAddr = pThrdParam->myAddr;
   int targetEventGen = pThrdParam->targetEventGen;
   DLL_EvQ* pEvQ;
   DLL_EvQ* priH_EvQ = pThrdParam->EvQ_PriH;
   DLL_EvQ* priL_EvQ = pThrdParam->EvQ_PriL;
   ThreadStatMon* pThrdMon = pThrdParam->pThrdMon;
   int maxRound = pThrdParam->maxRound;
   pThrdParam->pCS_main->lock();
   printf("Thread_EventGenerator(%d): targetEventGen(%d)₩n", myGenAddr, targetEventGen);
   pThrdParam->pCS_main->unlock();
```

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

    for (int round = 0; round < maxRound; round++)
    {
        if (event_gen_count < targetEventGen)
        {
            pEv = (Event *)calloc(1, sizeof(Event));
            pEv->event_gen_addr = myGenAddr;
            pEv->event_no = event_no = event_gen_count + (NUM_EVENTS_PER_GEN * myGenAddr);
            pEv->event_pri = event_pri = rand() % NUM_PRIORITY;
            pEv->event_handler_addr = -1;
            QueryPerformanceCounter(&pEv->t_gen);
            pEvQ = (event_pri < PRIORITY_THRESHOLD) ? priH_EvQ : priL_EvQ;
            while (enDLL_EvQ(pEvQ, pEv) == NULL)
            {
                Sleep(100);
            } // end while
            pThrdParam->pCS_thrd_mon->lock();
            pThrdMon->eventsGen[myGenAddr]++;
            pThrdMon->eventGenerated[pThrdMon->totalEventGen] = *pEv;
            pThrdMon->totalEventGen++;
            pThrdParam->pCS_thrd_mon->unlock();
            event_gen_count++;
        }
        else
        {
            if (*pThrdMon->pFlagThreadTerminate == TERMINATE)
                break;
        }  // end if
        Sleep(100 + rand() % 100);
    }  // end for round
}
```

Thread_EventHandler

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

#include <Windows.h>
#include <time.h>
#include <thread>
#include <mutex>
#include "Thread.h"
#include "DLL_EvQ.h"
#include "Event.h"

using namespace std;
void Thread_EventHandler(ThreadParam_Ev* pThrdParam)
{

    int myRole = pThrdParam->role;
    int myProcAddr = pThrdParam->myAddr;
    Event* pEv;
    DLL_EvQ* pEvQ;
    DLL_EvQ *priH_EvQ = pThrdParam->EvQ_PriH;
    DLL_EvQ *priL_EvQ = pThrdParam->EvQ_PriL;
    ThreadStatMon* pThrdMon = pThrdParam->pThrdMon;
    int maxRound = pThrdParam->maxRound;
    Event* evProc = pThrdParam->pThrdMon->eventProcessed;
    int targetEventGen = pThrdParam->targetEventGen;
    LARGE_INTEGER PC_freq = pThrdParam->PC_freq; // frequence of performance counter
    pThrdParam->pCS_main->lock();
    printf("Thread_EventHandler(%d): targetEventGen(%d)₩n", myProcAddr, targetEventGen);
    pThrdParam->pCS_main->unlock();
```

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

    for (int round = 0; round < maxRound; round++)
    {
        if (*pThrdMon->pFlagThreadTerminate == TERMINATE)
            break;
        while ((pEv = deDLL_EvQ(priH_EvQ)) != NULL)
        {
            pThrdParam->pCS_thrd_mon->lock();
            pEv->event_handler_addr = myProcAddr;
            QueryPerformanceCounter(&pEv->t_proc);
            calc_elapsed_time(pEv, PC_freq);
            pThrdMon->eventProcessed[pThrdMon->totalEventProc] = *pEv;
            pThrdMon->eventsProc[myProcAddr]++;
            pThrdMon->totalEventProc++;
            pThrdMon->numEventProcs_priH++;
            free(pEv);  // free the memory space for a Packet
            pThrdParam->pCS_thrd_mon->unlock();
            Sleep(300 + rand() % 500);
        } // end while
        if ((pEv = deDLL_EvQ(priL_EvQ)) != NULL)
        {
            pThrdParam->pCS_thrd_mon->lock();
            pEv->event_handler_addr = myProcAddr;
            QueryPerformanceCounter(&pEv->t_proc);
            calc_elapsed_time(pEv, PC_freq);
            pThrdMon->eventProcessed[pThrdMon->totalEventProc] = *pEv;
            pThrdMon->eventsProc[myProcAddr]++;
            pThrdMon->totalEventProc++;
            pThrdMon->numEventProcs_priL++;
            free(pEv);
            pThrdParam->pCS_thrd_mon->unlock();
        } // end if
        _sleep(100 + rand() % 100);
    }  // end while
}
```

Console Display

```
/* ConsoleDisplay.h */
#ifndef CONSOLE_DISPLAY_H
#define CONSOLE_DISPLAY_H
#include <Windows.h>

HANDLE initConsoleHandler();
void closeConsoleHandler(HANDLE hndlr);
int gotoxy(HANDLE consoleHandler, int x, int y);
#endif
```

```
/* ConsoleDisplay.cpp */
#include <stdio.h>
#include "ConsoleDisplay.h"

HANDLE consoleHandler;
HANDLE initConsoleHandler()
{
   HANDLE stdCnslHndlr;
   stdCnslHndlr =
      GetStdHandle(STD_OUTPUT_HANDLE);
   consoleHandler = stdCnslHndlr;
   return consoleHandler;
}

void closeConsoleHandler(HANDLE hndlr)
{
   CloseHandle(hndlr);
}

int gotoxy(HANDLE consHndlr, int x, int y)
{
   if (consHndlr == INVALID_HANDLE_VALUE)
   return 0;
   COORD coords = { static_cast<short>(x),
      static_cast<short>(y) };
   SetConsoleCursorPosition(consHndlr, coords);
}
```

SimParams.h

```
/* SimParam.h Simulation Parameters */

#ifndef SIMULATION_PARAMETERS_H
#define SIMULATION_PARAMETERS_H

#define NUM_EVENT_GENERATORS 3
#define NUM_EVENTS_PER_GEN 20
#define NUM_EVENT_HANDLERS 2
#define TOTAL_NUM_EVENTS (NUM_EVENTS_PER_GEN * NUM_EVENT_GENERATORS)
#define PLUS_INF INT_MAX
#define MAX_ROUND 1000

#define NUM_PRIORITY 10
#define PRIORITY_THRESHOLD 3 // 0 ~ 2: High Priority, 3 ~ 9: low priority
#define EVENT_PER_LINE 5

#endif
```

main()

```
/* main_EventGen_DLL_EvQ_EventProc.cpp (1) */
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
#include <time.h>
#include <thread>
#include <mutex>
#include "Thread.h"
#include "DLL_EvQ.h"
#include "Event.h"
#include "ConsoleDisplay.h"

using namespace std;

void main()
{
    FILE *fout;
    DLL_EvQ dll_EvQ_PriH, dll_EvQ_PriL;
    Event *pEvent;
    int myAddr = 0;
    int event_handler_addr, eventPriority;
    LARGE_INTEGER pc_freq;

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

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


    initDLL_EvQ(&dll_EvQ_PriH, 0);
    initDLL_EvQ(&dll_EvQ_PriL, 1);
    srand(time(NULL));
    ThreadParam_Ev thrdParam_EventGen[NUM_EVENT_GENERATORS],
        thrdParam_EventHndlr[NUM_EVENT_HANDLERS];
    thread thread_evHandlers[NUM_EVENT_HANDLERS];
    thread thread_evGens[NUM_EVENT_GENERATORS];
    mutex cs_main;
    mutex cs_thrd_mon;
    ThreadStatMon thrdMon;
    HANDLE consHndlr;
    THREAD_FLAG eventThreadFlag = RUN;
    int count, totalEventGenerated, totalEventProcessed;
    Event eventProcessed[TOTAL_NUM_EVENTS];

    consHndlr = initConsoleHandler();
    QueryPerformanceFrequency(&pc_freq);

    thrdMon.pFlagThreadTerminate = &eventThreadFlag;
    thrdMon.totalEventGen = 0;
    thrdMon.totalEventProc = 0;
    thrdMon.numEventProcs_priH = 0;
    thrdMon.numEventProcs_priL = 0;
    for (int ev = 0; ev < TOTAL_NUM_EVENTS; ev++)
    {
        thrdMon.eventProcessed[ev].event_no = -1; // mark as not-processed
        thrdMon.eventProcessed[ev].event_pri = -1;
    }
```

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

    /* Create and Activate Thread_EventHandler */
    for (int p = 0; p < NUM_EVENT_HANDLERS; p++)
    {
        thrdMon.eventsProc[p] = 0;
        thrdParam_EventHndlr[p].fout = fout;
        thrdParam_EventHndlr[p].role = EVENT_HANDLER;
        thrdParam_EventHndlr[p].myAddr = p; // Event handler address
        thrdParam_EventHndlr[p].pCS_main = &cs_main;
        thrdParam_EventHndlr[p].pCS_thrd_mon = &cs_thrd_mon;
        thrdParam_EventHndlr[p].EvQ_PriH = &dll_EvQ_PriH;
        thrdParam_EventHndlr[p].EvQ_PriL = &dll_EvQ_PriL;
        thrdParam_EventHndlr[p].maxRound = MAX_ROUND;
        thrdParam_EventHndlr[p].pThrdMon = &thrdMon;
        thrdParam_EventHndlr[p].PC_freq = pc_freq;

        thread_evHandlers[p] = thread(Thread_EventHandler, &thrdParam_EventHndlr[p]);
        //cs_main.lock();
        printf("%d-th thread_EventHandler is created and activated (id: %d)\n", p,
            thread_evHandlers[p].get_id());
        //cs_main.unlock();
    }
```

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

    /* Create and Activate Thread_EventGenerators */
    for (int g = 0; g < NUM_EVENT_GENERATORS; g++)
    {
        thrdMon.eventsGen[g] = 0;
        thrdParam_EventGen[g].role = EVENT_GENERATOR;
        thrdParam_EventGen[g].myAddr = g; // my Address of event generator
        thrdParam_EventGen[g].pCS_main = &cs_main;
        thrdParam_EventGen[g].pCS_thrd_mon = &cs_thrd_mon;
        thrdParam_EventGen[g].EvQ_PriH = &dll_EvQ_PriH;
        thrdParam_EventGen[g].EvQ_PriL = &dll_EvQ_PriL;
        thrdParam_EventGen[g].targetEventGen = NUM_EVENTS_PER_GEN;
        thrdParam_EventGen[g].maxRound = MAX_ROUND;
        thrdParam_EventGen[g].pThrdMon = &thrdMon;
        thrdParam_EventGen[g].PC_freq = pc_freq;

        thread_evGens[g] = thread(Thread_EventGenerator, &thrdParam_EventGen[g]);
        //cs_main.lock();
        printf("%d-th thread_EventGen is created and activated (id: %d)\n", g, thread_evGens[g].get_id());
        //cs_main.unlock();
    }
```

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

    /* Monitoring thread progress in rounds */
    for (int round = 0; round < MAX_ROUND; round++)
    {
        cs_main.lock();
        system("cls");
        gotoxy(consHndlr, 0, 0);
        printf("Thread monitoring by main() :: round(%2d): \n", round);
        cs_thrd_mon.lock();
        for (int i = 0; i < NUM_EVENT_GENERATORS; i++)
        {
            printf("  Event_Gen[%d] generated %2d events.\n", i, thrdMon.eventsGen[i]);
        }

        printf("Event_Generators have generated total %2d events\n", thrdMon.totalEventGen);
        totalEventGenerated = thrdMon.totalEventProc;
        printf("\nTotal Generated Events (current total %d events)\n   ", totalEventGenerated);
        for (int ev = 0; ev < totalEventGenerated; ev++)
        {
            pEvent = &thrdMon.eventGenerated[ev];
            if (pEvent != NULL)
            {
                printEvent(pEvent);
                if (((ev + 1) % EVENT_PER_LINE) == 0)
                    printf("\n   ");
            }
        }
        printf("\n");
```

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


        printf("\nEvent_Handlers have processed total %2d events ", thrdMon.totalEventProc);
        printf("(event__PriH (%2d), event_PriL (%2d))\n", thrdMon.numEventProcs_priH,
            thrdMon.numEventProcs_priL);
        for (int i = 0; i < NUM_EVENT_HANDLERS; i++)
        {
            printf("  Event_Proc[%d] processed %2d events.\n", i, thrdMon.eventsProc[i]);
        }

        printf("\nDLL_EvQ_PriH (%3d events):\n   ", dll_EvQ_PriH.num_event);
        printDLL_EvQ(&dll_EvQ_PriH);
        printf("\nDLL_EvQ_PriL (%3d events):\n   ", dll_EvQ_PriL.num_event);
        printDLL_EvQ(&dll_EvQ_PriL);
        printf("\n");

        totalEventProcessed = thrdMon.totalEventProc;
        printf("\nTotal Processed Events (current total %d events):\n   ", totalEventProcessed);
        count = 0;
        for (int ev = 0; ev < totalEventProcessed; ev++)
        {
            pEvent = &thrdMon.eventProcessed[ev];
            if (pEvent != NULL)
            {
                printEvent(pEvent);
                if (((ev + 1) % EVENT_PER_LINE) == 0)
                    printf("\n   ");
            }
        }
        printf("\n");
```

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

        cs_thrd_mon.unlock();
        if (totalEventProcessed >= TOTAL_NUM_EVENTS)
        {
            eventThreadFlag = TERMINATE; // set 1 to terminate threads
            cs_main.unlock();
            break;
        }
        cs_main.unlock();
        Sleep(100);
    } // end for (int round .....)

    for (int p = 0; p < NUM_EVENT_HANDLERS; p++)
    {
        thread_evHandlers[p].join();
    }
    printf("All threads of event handlers are terminated !!\n");

    for (int g = 0; g < NUM_EVENT_GENERATORS; g++)
    {
        thread_evGens[g].join();
    }
    printf("All threads of event generators are terminated !!\n");
```

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

    //calc_elapsed_time(thrdMon.eventProcessed, thrdMon.numPktProcs, freq);
    double min, max, avg, sum;
    int min_event, max_event;
    min = max = sum = thrdMon.eventProcessed[0].t_elapsed;
    min_event = max_event = 0;
    for (int i = 1; i < TOTAL_NUM_EVENTS; i++)
    {
        sum += thrdMon.eventProcessed[i].t_elapsed;
        if (min > thrdMon.eventProcessed[i].t_elapsed)
        {
            min = thrdMon.eventProcessed[i].t_elapsed;
            min_event = i;
        }
        if (max < thrdMon.eventProcessed[i].t_elapsed)
        {
            max = thrdMon.eventProcessed[i].t_elapsed;
            max_event = i;
        }
    }
    avg = sum / (double) TOTAL_NUM_EVENTS;
    printf("Minimum event processing time: %8.2lf[ms] for ", min * 1000);
    printEvent_withTime(&thrdMon.eventProcessed[min_event]); printf("\n");
    printf("Maximum event processing time: %8.2lf[ms] for ", max * 1000);
    printEvent_withTime(&thrdMon.eventProcessed[max_event]); printf("\n");
    printf("Average event processing time: %8.2lf[ms] for total %d events\n", avg * 1000,
     TOTAL_NUM_EVENTS);
    printf("\n");
}
```

◆ Thread Monitoring 결과 (중간 단계)

```
Thread monitoring by main() :: round(21):
  Event_Gen[0] generated 19 events.
  Event_Gen[1] generated 19 events.
  Event_Gen[2] generated 19 events.
Event_Generators have generated total 57 events

Total Generated Events (current total 10 events)
   Ev[  0, pri( 1), gen( 0), proc(-1)] Ev[ 20, pri( 1), gen( 1), proc(-1)] Ev[ 40, pri( 1), gen( 2), proc(-1)] Ev[  1, pri( 4), gen( 0), proc(-1)] Ev[ 41, pri( 4), gen( 2), proc(-1)]
   Ev[ 21, pri( 4), gen( 1), proc(-1)] Ev[ 22, pri( 9), gen( 1), proc(-1)] Ev[ 42, pri( 9), gen( 2), proc(-1)] Ev[  2, pri( 9), gen( 0), proc(-1)] Ev[ 43, pri( 8), gen( 2), proc(-1)]


Event_Handlers have processed total 10 events (event__PriH (10), event_PriL ( 0))
  Event_Proc[0] processed  5 events.
  Event_Proc[1] processed  5 events.

DLL_EvQ_PriH ( 15 events):
   Ev[ 30, pri( 1), gen( 1), proc(-1)]  Ev[ 10, pri( 1), gen( 0), proc(-1)]  Ev[ 50, pri( 1), gen( 2), proc(-1)]  Ev[ 51, pri( 2), gen( 2), proc(-1)]  Ev[ 31, pri( 2), gen( 1), proc(-1)]
   Ev[ 11, pri( 2), gen( 0), proc(-1)]  Ev[ 32, pri( 2), gen( 1), proc(-1)]  Ev[ 52, pri( 2), gen( 2), proc(-1)]  Ev[ 12, pri( 2), gen( 0), proc(-1)]  Ev[ 33, pri( 1), gen( 1), proc(-1)]
   Ev[ 53, pri( 1), gen( 2), proc(-1)]  Ev[ 13, pri( 1), gen( 0), proc(-1)]  Ev[ 36, pri( 1), gen( 1), proc(-1)]  Ev[ 56, pri( 1), gen( 2), proc(-1)]  Ev[ 16, pri( 1), gen( 0), proc(-1)]

DLL_EvQ_PriL ( 30 events):
   Ev[  1, pri( 4), gen( 0), proc(-1)]  Ev[ 41, pri( 4), gen( 2), proc(-1)]  Ev[ 21, pri( 4), gen( 1), proc(-1)]  Ev[ 22, pri( 9), gen( 1), proc(-1)]  Ev[ 42, pri( 9), gen( 2), proc(-1)]
   Ev[  2, pri( 9), gen( 0), proc(-1)]  Ev[ 43, pri( 8), gen( 2), proc(-1)]  Ev[ 23, pri( 8), gen( 1), proc(-1)]  Ev[  3, pri( 8), gen( 0), proc(-1)]  Ev[  5, pri( 5), gen( 0), proc(-1)]
   Ev[ 25, pri( 5), gen( 1), proc(-1)]  Ev[ 45, pri( 5), gen( 2), proc(-1)]  Ev[ 28, pri( 5), gen( 1), proc(-1)]  Ev[ 48, pri( 5), gen( 2), proc(-1)]  Ev[  8, pri( 5), gen( 0), proc(-1)]
   Ev[ 49, pri( 7), gen( 2), proc(-1)]  Ev[ 29, pri( 7), gen( 1), proc(-1)]  Ev[  9, pri( 7), gen( 0), proc(-1)]  Ev[ 54, pri( 8), gen( 2), proc(-1)]  Ev[ 34, pri( 8), gen( 1), proc(-1)]
   Ev[ 14, pri( 8), gen( 0), proc(-1)]  Ev[ 15, pri( 7), gen( 0), proc(-1)]  Ev[ 35, pri( 7), gen( 1), proc(-1)]  Ev[ 55, pri( 7), gen( 2), proc(-1)]  Ev[ 17, pri( 9), gen( 0), proc(-1)]
   Ev[ 57, pri( 9), gen( 2), proc(-1)]  Ev[ 37, pri( 9), gen( 1), proc(-1)]  Ev[ 58, pri( 7), gen( 2), proc(-1)]  Ev[ 38, pri( 7), gen( 1), proc(-1)]  Ev[ 18, pri( 7), gen( 0), proc(-1)]


Total Processed Events (current total 10 events):
   Ev[  0, pri( 1), gen( 0), proc( 0)] Ev[ 20, pri( 1), gen( 1), proc( 1)] Ev[ 40, pri( 1), gen( 2), proc( 1)] Ev[ 44, pri( 2), gen( 2), proc( 0)] Ev[ 24, pri( 2), gen( 1), proc( 1)]
   Ev[  4, pri( 2), gen( 0), proc( 0)] Ev[ 46, pri( 1), gen( 2), proc( 1)] Ev[ 26, pri( 1), gen( 1), proc( 0)] Ev[  6, pri( 1), gen( 0), proc( 1)] Ev[ 47, pri( 1), gen( 2), proc( 0)]
```

◆ Thread Monitoring 결과 (최종 단계)

```
Thread monitoring by main() :: round(80):
  Event_Gen[0] generated 20 events.
  Event_Gen[1] generated 20 events.
  Event_Gen[2] generated 20 events.
Event_Generators have generated total 60 events

Total Generated Events (current total 60 events)
   Ev[  0, pri( 1), gen( 0), proc(-1)] Ev[ 20, pri( 1), gen( 1), proc(-1)] Ev[ 40, pri( 1), gen( 2), proc(-1)] Ev[ 41, pri( 4), gen( 2), proc(-1)] Ev[  1, pri( 4), gen( 0), proc(-1)]
   Ev[ 21, pri( 4), gen( 1), proc(-1)] Ev[ 22, pri( 9), gen( 1), proc(-1)] Ev[  2, pri( 9), gen( 0), proc(-1)] Ev[ 42, pri( 9), gen( 2), proc(-1)] Ev[  3, pri( 8), gen( 0), proc(-1)]
   Ev[ 43, pri( 8), gen( 2), proc(-1)] Ev[ 23, pri( 8), gen( 1), proc(-1)] Ev[ 44, pri( 2), gen( 2), proc(-1)] Ev[ 24, pri( 2), gen( 1), proc(-1)] Ev[  4, pri( 2), gen( 0), proc(-1)]
   Ev[ 25, pri( 5), gen( 1), proc(-1)] Ev[ 45, pri( 5), gen( 2), proc(-1)] Ev[  5, pri( 5), gen( 0), proc(-1)] Ev[  6, pri( 1), gen( 0), proc(-1)] Ev[ 26, pri( 1), gen( 1), proc(-1)]
   Ev[ 46, pri( 1), gen( 2), proc(-1)] Ev[  7, pri( 1), gen( 0), proc(-1)] Ev[ 27, pri( 1), gen( 1), proc(-1)] Ev[ 47, pri( 1), gen( 2), proc(-1)] Ev[  8, pri( 5), gen( 0), proc(-1)]
   Ev[ 48, pri( 5), gen( 2), proc(-1)] Ev[ 28, pri( 5), gen( 1), proc(-1)] Ev[  9, pri( 7), gen( 0), proc(-1)] Ev[ 29, pri( 7), gen( 1), proc(-1)] Ev[ 49, pri( 7), gen( 2), proc(-1)]
   Ev[ 50, pri( 1), gen( 2), proc(-1)] Ev[ 30, pri( 1), gen( 1), proc(-1)] Ev[ 10, pri( 1), gen( 0), proc(-1)] Ev[ 51, pri( 2), gen( 2), proc(-1)] Ev[ 31, pri( 2), gen( 1), proc(-1)]
   Ev[ 11, pri( 2), gen( 0), proc(-1)] Ev[ 52, pri( 2), gen( 2), proc(-1)] Ev[ 12, pri( 2), gen( 0), proc(-1)] Ev[ 32, pri( 2), gen( 1), proc(-1)] Ev[ 33, pri( 1), gen( 1), proc(-1)]
   Ev[ 53, pri( 1), gen( 2), proc(-1)] Ev[ 13, pri( 1), gen( 0), proc(-1)] Ev[ 54, pri( 8), gen( 2), proc(-1)] Ev[ 34, pri( 8), gen( 1), proc(-1)] Ev[ 14, pri( 8), gen( 0), proc(-1)]
   Ev[ 15, pri( 7), gen( 0), proc(-1)] Ev[ 35, pri( 7), gen( 1), proc(-1)] Ev[ 55, pri( 7), gen( 2), proc(-1)] Ev[ 16, pri( 1), gen( 0), proc(-1)] Ev[ 36, pri( 1), gen( 1), proc(-1)]
   Ev[ 56, pri( 1), gen( 2), proc(-1)] Ev[ 37, pri( 9), gen( 1), proc(-1)] Ev[ 17, pri( 9), gen( 0), proc(-1)] Ev[ 57, pri( 9), gen( 2), proc(-1)] Ev[ 58, pri( 7), gen( 2), proc(-1)]
   Ev[ 38, pri( 7), gen( 1), proc(-1)] Ev[ 18, pri( 7), gen( 0), proc(-1)] Ev[ 59, pri( 5), gen( 2), proc(-1)] Ev[ 39, pri( 5), gen( 1), proc(-1)] Ev[ 19, pri( 5), gen( 0), proc(-1)]


Event_Handlers have processed total 60 events (event__PriH (27), event_PriL (33))
  Event_Proc[0] processed 32 events.
  Event_Proc[1] processed 28 events.

DLL_EvQ_PriH (  0 events):

DLL_EvQ_PriL (  0 events):


Total Processed Events (current total 60 events):
   Ev[  0, pri( 1), gen( 0), proc( 0)] Ev[ 20, pri( 1), gen( 1), proc( 1)] Ev[ 40, pri( 1), gen( 2), proc( 1)] Ev[ 44, pri( 2), gen( 2), proc( 0)] Ev[  4, pri( 2), gen( 0), proc( 1)]
   Ev[ 24, pri( 2), gen( 1), proc( 0)] Ev[ 26, pri( 1), gen( 1), proc( 1)] Ev[  6, pri( 1), gen( 0), proc( 0)] Ev[ 46, pri( 1), gen( 2), proc( 0)] Ev[  7, pri( 1), gen( 0), proc( 1)]
   Ev[ 47, pri( 1), gen( 2), proc( 0)] Ev[ 27, pri( 1), gen( 1), proc( 1)] Ev[ 50, pri( 1), gen( 2), proc( 1)] Ev[ 30, pri( 1), gen( 1), proc( 0)] Ev[ 51, pri( 2), gen( 2), proc( 1)]
   Ev[ 10, pri( 1), gen( 0), proc( 0)] Ev[ 11, pri( 2), gen( 0), proc( 1)] Ev[ 31, pri( 2), gen( 1), proc( 0)] Ev[ 52, pri( 2), gen( 2), proc( 1)] Ev[ 12, pri( 2), gen( 0), proc( 0)]
   Ev[ 32, pri( 2), gen( 1), proc( 1)] Ev[ 33, pri( 1), gen( 1), proc( 0)] Ev[ 53, pri( 1), gen( 2), proc( 0)] Ev[ 13, pri( 1), gen( 0), proc( 1)] Ev[ 16, pri( 1), gen( 0), proc( 0)]
   Ev[ 36, pri( 1), gen( 1), proc( 1)] Ev[ 56, pri( 1), gen( 2), proc( 1)] Ev[ 41, pri( 4), gen( 2), proc( 0)] Ev[  1, pri( 4), gen( 0), proc( 0)] Ev[ 21, pri( 4), gen( 1), proc( 0)]
   Ev[ 22, pri( 9), gen( 1), proc( 0)] Ev[  2, pri( 9), gen( 0), proc( 0)] Ev[ 42, pri( 9), gen( 2), proc( 1)] Ev[  3, pri( 8), gen( 0), proc( 0)] Ev[ 43, pri( 8), gen( 2), proc( 1)]
   Ev[ 23, pri( 8), gen( 1), proc( 0)] Ev[ 25, pri( 5), gen( 1), proc( 1)] Ev[ 45, pri( 5), gen( 2), proc( 0)] Ev[  5, pri( 5), gen( 0), proc( 0)] Ev[  8, pri( 5), gen( 0), proc( 1)]
   Ev[ 48, pri( 5), gen( 2), proc( 0)] Ev[ 28, pri( 5), gen( 1), proc( 1)] Ev[  9, pri( 7), gen( 0), proc( 0)] Ev[ 29, pri( 7), gen( 1), proc( 1)] Ev[ 49, pri( 7), gen( 2), proc( 0)]
   Ev[ 54, pri( 8), gen( 2), proc( 1)] Ev[ 34, pri( 8), gen( 1), proc( 1)] Ev[ 14, pri( 8), gen( 0), proc( 0)] Ev[ 15, pri( 7), gen( 0), proc( 1)] Ev[ 35, pri( 7), gen( 1), proc( 0)]
   Ev[ 55, pri( 7), gen( 2), proc( 1)] Ev[ 37, pri( 9), gen( 1), proc( 0)] Ev[ 17, pri( 9), gen( 0), proc( 0)] Ev[ 57, pri( 9), gen( 2), proc( 1)] Ev[ 58, pri( 7), gen( 2), proc( 0)]
   Ev[ 38, pri( 7), gen( 1), proc( 1)] Ev[ 18, pri( 7), gen( 0), proc( 0)] Ev[ 59, pri( 5), gen( 2), proc( 1)] Ev[ 39, pri( 5), gen( 1), proc( 0)] Ev[ 19, pri( 5), gen( 0), proc( 1)]

All threads of event handlers are terminated !!
All threads of event generators are terminated !!
Minimum event processing time:   143.47[ms] for Ev(no: 20, pri: 1,    143[ms])
Maximum event processing time:  8641.41[ms] for Ev(no: 23, pri: 8,   8641[ms])
Average event processing time:  5657.17[ms] for total 60 events
```

Homework 14

Homework 14

14.1 Queuing System and Multi-Threads

(1) 구조체 struct Event은 다음과 같은 멤버를 가진다:

```
int genAddr, handlerAddr ; // generator address, handler address
int priority; // priority of protocol data unit (사용자/프로토콜 정보의 우선 순위)
int seqNo; // sequence number
LARGE_INTEGER  t_gen, t_proc;  // time stamps of Event generation and processing
double  t_elapse;  // elapsed time of Event processing from Event generation
```

(2) 다음 함수들이 구조체 struct Event를 위하여 사용된다:

```
void initEvent(Event *pEvent, unsigned int sAddr, unsigned int sN);
FILE * fprintEvent(FILE *fout, Event* pEvent);
```

(3) 구조체 struct DLLN 는 다음과 같은 데이터 멤버를 가진다:

```
Event *pEv;
DLLN *next;
DLLN *prev;
```

(4) 구조체 struct EventQueue (Doubly Linked List)는 다음과 같은 데이터 멤버를 가진다:

```
mutex cs_EvQ;
int numEvents;
DLLN* front;
DLLN* back;
```

(5) 다음 함수들은 구조체 struct EventQueue 와 관련되어 사용된다:

```
Event* enQueue(Queue *pQ, Event* pEvent); // enQueue a Event into the queue.
  // The pEvent is pointing an Event.
Event* deQueue(Queue *pQ);
  // remove a list Node from the queue, and return the address of event.
void fprintQueue(FILE *fout, Queue *pQ); // print all list Node in the queue to file
```

(6) 다중 스레드로 파라메터를 전달하기 위하여 다음과 같은 구조체 (ThreadParam)를 사용하라. 추가적으로 필요한 항목이 있는 경우 추가할 것.

```
typedef struct
{
        mutex *pCS_main; // pointer to the shared critical section
        mutex *pCS_thrd_mon;
        EventQueue  *pPriH_EvQ;
        EventQueue  *pPriL_EvQ;
                // pointer to the two shared queue (high, low priority)
        int role; // EventGen or EventHandler
        UINT_32 addr;
        int max_queue;
        THREAD_FLAG *pFlagThreadTerminate;
        FILE *fout; // pointer to the output file stream
} ThreadParam;
```

(8) Thread_EventGen() 는 주기적으로 이벤트들을 생성하며, 생성된 이벤트들을 queue에 삽입 (enqueue)한다. 각 스레드 EventGen()은 지정된 발생지 주소 (genAddr) 를 사용한다. 이벤트 생성 시간 간격은 100 ~ 400ms 사이의 값이 임의로 설정된다.

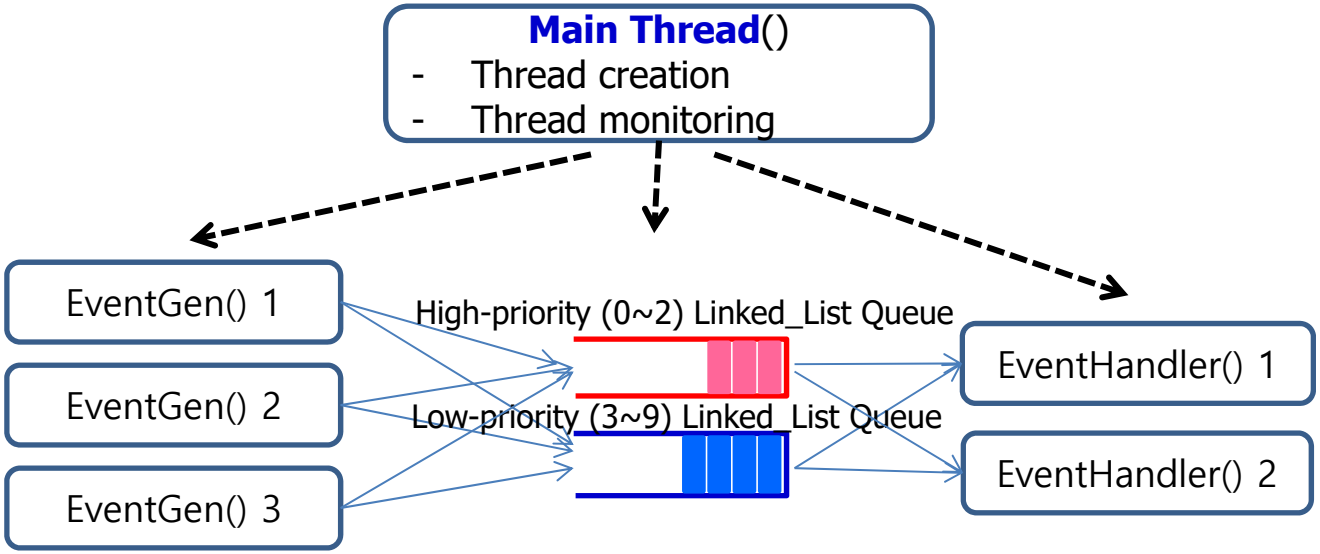
(9) Thread_EventHandler()는 high_priority EvQ와 low_priority EvQ를 지속적으로 점검하며, high_priority EvQ에 event가 남아 있으면 이를 우선적으로 처리한다. 만약 high_priority EvQ가 empty 상태이면 low_priority EvQ의 이벤트를 처리한다. 각 이벤트를 처리한 후, 100ms ~ 400ms를 sleep 한다. 처리된 이벤트의 정보는 스레드 모니터링 정보로 기록한다.

(10) Thread_EventGen()과 Thread_EventHandler()는 main() 함수에서 제어하는 pFlagThreadTerminate 값이 TERMINATE로 설정되면 스레드 동작을 종료한다.

(11) main() 함수는 다음과 같은 내용을 실행한다:

- 3개의 Thread_EventGen 스레드를 생성하고, 초기화 한다.
- 2개의 Thread_EventHandler 스레드를 생성하고, 초기화 한다.
- main() 함수는 2의 구조체 struct Queue 변수를 생성하여 3개의 Thread_EventGen 스레드와 2개의 Thread_EventHandler()가 공유하게 한다. 각 Queue는 각각의 mutex를 자체적으로 생성하여, enqueue()와 dequeue() 등에서 임계구역이 관리되게 한다.
- 각 이벤트 생성 스레드 (Thread_EventGen())는 각각 50개의 이벤트 (우선 순위 (priority)는 0 ~ 10 사이의 값을 random하게 설정하며, 생성된 이벤트를 우선순위에 따라 지정된 queue에 넣는다 (0~2의 우선 순위 값은 high-priority queue에, 3 ~ 9의 우선 순위 값은 low-priority queue에).
- 이벤트 생성 스레드는 이벤트 생성 시점 (performance counter 값)을 기록한다.

- Thread_EventHandler() 스레드는 항상 우선 순위가 높은 queue를 먼저 점검하며, 만약 이벤트가 존재하는 경우, 이를 우선 처리한다.
- Thread_EventHandler() 스레드는 이벤트 처리 시점 (performance counter 값)을 기록하며, 이 이벤트를 처리할 때 까지 걸린 경과시간을 계산하여 기록한다.
- 이벤트 생성 스레드에서 생성되어 enqueue된 이벤트들은 Thread_EventHandler()스레드에 의하여 dequeue된 후, 이벤트가 생성되어 처리된 시점까지의 경과시간 (elapsed time)들이 측정된다.
- main() 함수의 마지막 단계에서는 각 이벤트들의 처리시간 중 최소값과 최대값을 찾고, 평균값을 계산하여, event 처리 성능을 분석한다.
- 프로그램의 수행 내용은 각 스레드의 동작 상황 및 이벤트 생성 및 처리 상황은 thread monitoring 기능을 사용하여 console 창에 상황판의 형식으로 출력한다.



◆ 실행결과 (초기단계)

```
Thread monitoring by main() :: round( 8):
Event_Generators have generated total 30 events
  Event_Genr[0] generated 10 events.
  Event_Genr[1] generated 10 events.
  Event_Genr[2] generated 10 events.
Event_Handlers   have processed total  8 events (event__PriH ( 2), event_PriL ( 6))
  Event_Proc[0] processed  4 events.
  Event_Proc[1] processed  4 events.

DLL_EvQ_PriH (  4 events):
   Ev[  7, pri( 2), gen( 0), proc( 0)]  Ev[  8, pri( 2), gen( 0), proc( 1)]  Ev[ 58, pri( 2), gen( 1), proc( 1)]  Ev[108, pri( 2), gen( 2), proc( 1)]
DLL_EvQ_PriL ( 18 events):
   Ev[  2, pri( 8), gen( 0), proc( 0)]  Ev[ 52, pri( 8), gen( 1), proc( 0)]  Ev[102, pri( 8), gen( 2), proc( 0)]  Ev[103, pri( 5), gen( 2), proc( 2)]  Ev[ 53, pri( 5), gen( 1), proc( 2)]
   Ev[  3, pri( 5), gen( 0), proc( 2)]  Ev[  4, pri( 7), gen( 0), proc( 1)]  Ev[104, pri( 7), gen( 2), proc( 1)]  Ev[ 54, pri( 7), gen( 1), proc( 1)]  Ev[105, pri( 5), gen( 2), proc( 2)]
   Ev[  5, pri( 5), gen( 0), proc( 2)]  Ev[ 55, pri( 5), gen( 1), proc( 2)]  Ev[  6, pri( 6), gen( 0), proc( 0)]  Ev[ 56, pri( 6), gen( 1), proc( 0)]  Ev[106, pri( 6), gen( 2), proc( 0)]
   Ev[ 59, pri( 8), gen( 1), proc( 2)]  Ev[109, pri( 8), gen( 2), proc( 2)]  Ev[  9, pri( 8), gen( 0), proc( 2)]
Events processed:
   Ev[  0, pri( 7), gen( 0), proc( 2)] Ev[ 50, pri( 7), gen( 1), proc( 2)] Ev[100, pri( 7), gen( 2), proc( 2)] Ev[  1, pri( 9), gen( 0), proc( 1)] Ev[101, pri( 9), gen( 2), proc( 1)]
   Ev[ 51, pri( 9), gen( 1), proc( 1)] Ev[107, pri( 2), gen( 2), proc( 0)] Ev[ 57, pri( 2), gen( 1), proc( 0)]
```

◆ 실행결과 (최종)

```
Thread monitoring by main() :: round(188):
Event_Generators have generated total 150 events
  Event_Genr[0] generated 50 events.
  Event_Genr[1] generated 50 events.
  Event_Genr[2] generated 50 events.
Event_Handlers   have processed total 150 events (event__PriH (33), event_PriL (117))
  Event_Proc[0] processed 75 events.
  Event_Proc[1] processed 75 events.

DLL_EvQ_PriH (  0 events):

DLL_EvQ_PriL (  0 events):

Events processed:
   Ev[  0, pri( 7), gen( 0), proc( 2)] Ev[ 50, pri( 7), gen( 1), proc( 2)] Ev[100, pri( 7), gen( 2), proc( 2)] Ev[  1, pri( 9), gen( 0), proc( 1)] Ev[101, pri( 9), gen( 2), proc( 1)]
   Ev[ 51, pri( 9), gen( 1), proc( 1)] Ev[  7, pri( 2), gen( 0), proc( 0)] Ev[107, pri( 2), gen( 2), proc( 0)] Ev[ 57, pri( 2), gen( 1), proc( 0)] Ev[  8, pri( 2), gen( 0), proc( 1)]
   Ev[ 58, pri( 2), gen( 1), proc( 1)] Ev[108, pri( 2), gen( 2), proc( 1)] Ev[ 15, pri( 1), gen( 0), proc( 2)] Ev[ 65, pri( 1), gen( 1), proc( 2)] Ev[115, pri( 1), gen( 2), proc( 2)]
   Ev[ 17, pri( 2), gen( 0), proc( 2)] Ev[ 67, pri( 2), gen( 1), proc( 2)] Ev[117, pri( 2), gen( 2), proc( 2)] Ev[ 72, pri( 2), gen( 1), proc( 1)] Ev[ 22, pri( 2), gen( 0), proc( 1)]
   Ev[122, pri( 2), gen( 2), proc( 1)] Ev[128, pri( 0), gen( 2), proc( 0)] Ev[ 78, pri( 0), gen( 1), proc( 0)] Ev[ 28, pri( 0), gen( 0), proc( 0)] Ev[ 29, pri( 1), gen( 0), proc( 2)]
   Ev[ 79, pri( 1), gen( 1), proc( 2)] Ev[129, pri( 1), gen( 2), proc( 2)] Ev[ 82, pri( 2), gen( 1), proc( 2)] Ev[132, pri( 2), gen( 2), proc( 2)] Ev[ 32, pri( 2), gen( 0), proc( 2)]
   Ev[ 40, pri( 0), gen( 0), proc( 1)] Ev[140, pri( 0), gen( 2), proc( 1)] Ev[ 90, pri( 0), gen( 1), proc( 1)] Ev[ 93, pri( 0), gen( 1), proc( 1)] Ev[143, pri( 0), gen( 2), proc( 1)]
   Ev[ 43, pri( 0), gen( 0), proc( 1)] Ev[ 99, pri( 1), gen( 1), proc( 2)] Ev[149, pri( 1), gen( 2), proc( 2)] Ev[ 49, pri( 1), gen( 0), proc( 2)] Ev[  2, pri( 8), gen( 0), proc( 0)]
   Ev[ 52, pri( 8), gen( 1), proc( 0)] Ev[102, pri( 8), gen( 2), proc( 0)] Ev[103, pri( 5), gen( 2), proc( 2)] Ev[ 53, pri( 5), gen( 1), proc( 2)] Ev[  3, pri( 5), gen( 0), proc( 2)]
   Ev[ 54, pri( 7), gen( 1), proc( 1)] Ev[104, pri( 7), gen( 2), proc( 1)] Ev[  4, pri( 7), gen( 0), proc( 1)] Ev[  5, pri( 5), gen( 0), proc( 2)] Ev[105, pri( 5), gen( 2), proc( 2)]
   Ev[ 55, pri( 5), gen( 1), proc( 2)] Ev[ 56, pri( 6), gen( 1), proc( 0)] Ev[  6, pri( 6), gen( 0), proc( 0)] Ev[106, pri( 6), gen( 2), proc( 0)] Ev[109, pri( 8), gen( 2), proc( 2)]
   Ev[ 59, pri( 8), gen( 1), proc( 2)] Ev[  9, pri( 8), gen( 0), proc( 2)] Ev[ 60, pri( 6), gen( 1), proc( 2)] Ev[110, pri( 6), gen( 2), proc( 2)] Ev[ 10, pri( 6), gen( 0), proc( 2)]
   Ev[111, pri( 9), gen( 2), proc( 0)] Ev[ 61, pri( 9), gen( 1), proc( 0)] Ev[ 11, pri( 9), gen( 0), proc( 0)] Ev[112, pri( 9), gen( 2), proc( 2)] Ev[ 62, pri( 9), gen( 1), proc( 2)]
   Ev[ 12, pri( 9), gen( 0), proc( 2)] Ev[113, pri( 3), gen( 2), proc( 0)] Ev[ 63, pri( 3), gen( 1), proc( 0)] Ev[ 13, pri( 3), gen( 0), proc( 0)] Ev[ 14, pri( 3), gen( 0), proc( 2)]
   Ev[ 64, pri( 3), gen( 1), proc( 2)] Ev[114, pri( 3), gen( 2), proc( 2)] Ev[ 66, pri( 8), gen( 1), proc( 2)] Ev[116, pri( 8), gen( 2), proc( 2)] Ev[ 16, pri( 8), gen( 0), proc( 2)]
   Ev[ 18, pri( 9), gen( 0), proc( 0)] Ev[118, pri( 9), gen( 2), proc( 0)] Ev[ 68, pri( 9), gen( 1), proc( 0)] Ev[ 69, pri( 9), gen( 1), proc( 0)] Ev[ 19, pri( 9), gen( 0), proc( 0)]
   Ev[119, pri( 9), gen( 2), proc( 0)] Ev[ 20, pri( 5), gen( 0), proc( 1)] Ev[120, pri( 5), gen( 2), proc( 1)] Ev[ 70, pri( 5), gen( 1), proc( 1)] Ev[121, pri( 8), gen( 2), proc( 0)]
   Ev[ 71, pri( 8), gen( 1), proc( 0)] Ev[ 21, pri( 8), gen( 0), proc( 0)] Ev[123, pri( 6), gen( 2), proc( 1)] Ev[ 73, pri( 6), gen( 1), proc( 1)] Ev[ 23, pri( 6), gen( 0), proc( 1)]
   Ev[124, pri( 9), gen( 2), proc( 2)] Ev[ 74, pri( 9), gen( 1), proc( 2)] Ev[ 24, pri( 9), gen( 0), proc( 2)] Ev[125, pri( 6), gen( 2), proc( 2)] Ev[ 75, pri( 6), gen( 1), proc( 2)]
   Ev[ 25, pri( 6), gen( 0), proc( 2)] Ev[126, pri( 8), gen( 2), proc( 0)] Ev[ 76, pri( 8), gen( 1), proc( 0)] Ev[ 26, pri( 8), gen( 0), proc( 0)] Ev[ 27, pri( 4), gen( 0), proc( 2)]
   Ev[ 77, pri( 4), gen( 1), proc( 2)] Ev[127, pri( 4), gen( 2), proc( 2)] Ev[ 80, pri( 9), gen( 1), proc( 0)] Ev[130, pri( 9), gen( 2), proc( 0)] Ev[ 30, pri( 9), gen( 0), proc( 0)]
   Ev[131, pri( 7), gen( 2), proc( 1)] Ev[ 81, pri( 7), gen( 1), proc( 1)] Ev[ 31, pri( 7), gen( 0), proc( 1)] Ev[ 83, pri( 3), gen( 1), proc( 2)] Ev[133, pri( 3), gen( 2), proc( 2)]
   Ev[ 33, pri( 3), gen( 0), proc( 2)] Ev[ 84, pri( 8), gen( 1), proc( 1)] Ev[134, pri( 8), gen( 2), proc( 1)] Ev[ 34, pri( 8), gen( 0), proc( 1)] Ev[ 35, pri( 7), gen( 0), proc( 0)]
   Ev[135, pri( 7), gen( 2), proc( 0)] Ev[ 85, pri( 7), gen( 1), proc( 0)] Ev[ 36, pri( 6), gen( 0), proc( 2)] Ev[136, pri( 6), gen( 2), proc( 2)] Ev[ 86, pri( 6), gen( 1), proc( 2)]
   Ev[ 87, pri( 4), gen( 1), proc( 2)] Ev[137, pri( 4), gen( 2), proc( 2)] Ev[ 37, pri( 4), gen( 0), proc( 2)] Ev[ 88, pri( 9), gen( 1), proc( 0)] Ev[138, pri( 9), gen( 2), proc( 0)]
   Ev[ 38, pri( 9), gen( 0), proc( 0)] Ev[ 89, pri( 7), gen( 1), proc( 0)] Ev[139, pri( 7), gen( 2), proc( 0)] Ev[ 39, pri( 7), gen( 0), proc( 0)] Ev[ 91, pri( 5), gen( 1), proc( 0)]
   Ev[141, pri( 5), gen( 2), proc( 0)] Ev[ 41, pri( 5), gen( 0), proc( 0)] Ev[142, pri( 4), gen( 2), proc( 0)] Ev[ 92, pri( 4), gen( 1), proc( 0)] Ev[ 42, pri( 4), gen( 0), proc( 0)]
   Ev[144, pri( 4), gen( 2), proc( 0)] Ev[ 94, pri( 4), gen( 1), proc( 0)] Ev[ 44, pri( 4), gen( 0), proc( 0)] Ev[ 45, pri( 7), gen( 0), proc( 1)] Ev[145, pri( 7), gen( 2), proc( 1)]
   Ev[ 95, pri( 7), gen( 1), proc( 1)] Ev[ 46, pri( 7), gen( 0), proc( 1)] Ev[ 96, pri( 7), gen( 1), proc( 1)] Ev[146, pri( 7), gen( 2), proc( 1)] Ev[ 47, pri( 3), gen( 0), proc( 2)]
   Ev[147, pri( 3), gen( 2), proc( 2)] Ev[ 97, pri( 3), gen( 1), proc( 2)] Ev[ 98, pri( 9), gen( 1), proc( 2)] Ev[148, pri( 9), gen( 2), proc( 2)] Ev[ 48, pri( 9), gen( 0), proc( 2)]

All threads of event handlers are terminated !!
All threads of event generators are terminated !!
Minimum event processing time:   299.18[ms] for Ev(no:107, pri: 2,    299[ms])
Maximum event processing time: 25032.86[ms] for Ev(no: 48, pri: 9,  25033[ms])
Average event processing time: 14199.73[ms] for total 150 events
```