Lab 5. Doubly-linked list의 구현

◎ 실험 실습 일시 : 2009. 4. 13.
◎ 담당 교수 : 정진우
◎ 담당 조교 : 곽문상
◎ 보고서 제출기한 : 2009. 4. 19.

학과 : _____
학번 : _____
성명 : _____

◎ 실습 과제 목적 : 이론시간에 배운 Doubly-linked list를 실제로 구현할 수 있다.
◎ 실습 과제 내용 : 주어진 소스를 이용해 Doubly-linked list의 각 함수를 구현한다.

◎ 실습순서

단계 1. 주어지는 코드와 함수 설계를 참고해, mp3 플레이어의 플레이 리스트(play list) 관리 프로그램을 구현하시오. 해당 플레이 리스트는 원형 이중-연결 리스트로 구현해야 하며, 구현해야하는 함수와 자료구조, main 함수는 주어진 코드를 이용해서 구현해야 한다.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

//MP3 플레이어의 최대 용량 128MB
#define MAX 128
//한 곡의 용량 4MB
#define SONGSIZE 4

//리스트 입력 모드 flag
#define NORMAL 0
#define FIRST 1

//리스트 노드 구조체
struct ListNode {
        char *title;          // 제목
        char *singer;         // 가수
        struct ListNode *next; // 다음 노드의 포인터
        struct ListNode *previous; // 이전 노드의 포인터
};
typedef struct ListNode LISTNODE;

//리스트 구조체
struct List{
        int listSize;         // 리스트의 크기
        LISTNODE *play;   // 현재 재생중인 곡
        LISTNODE *first;   // 리스트의 첫 번째 곡
};
typedef struct List LIST;
```

```
void initList(LIST *list);                    //리스트 초기화 함수
void printList(LIST *list);                   //리스트 출력 함수
bool isEmpty(LIST *list);                     //리스트가 비어있는지 확인하는 함수
bool isFull(LIST *list);                      //용량이 가득차 있는지 확인하는 함수
void insertMusic(LIST *list, int mode);  //곡을 추가하는 함수
void insertMusicSub(LIST *list, LISTNODE *node);          //리스트에 노드 추가
void insertMusicFirstSub(LIST *list, LISTNODE *node);     //리스트의 처음 위치에 노드 추가
void deleteMusic(LIST *list);  //곡을 삭제하는 함수
void deleteMusicSub(LIST *list, LISTNODE *node);          //리스트에서 노드를 삭제하는 함수
void movePrev(LIST *list);    //앞 곡으로 이동
void moveNext(LIST *list);    //뒷 곡으로 이동

void main()
{
        LIST *list;
        char cmd;

        //리스트에 동적으로 메모리 할당
        list = (LIST *)malloc(sizeof(LIST));

        //리스트 초기화
        initList(list);

        do
        {
                printf("*************** Choose Command!! ******************₩n");
                printf("+: Insert song S: Insert song in the first₩n");
                printf("-: Delete song P: Play previous song N: Play next song₩n");
                printf("F: full check   E: empty check   Q: Quit₩n");
                printf("*****************************************************₩n");


                // 리스트 출력
                printList(list);

                // 커맨드 입력
                printf("Command: ");
                cmd = getch();
                // 커맨드는 무조건 대문자로 입력
                cmd = toupper(cmd);
                putch(cmd);
                printf("₩n");

                switch (cmd)
                {
                        case '+' :          // 곡 추가
                                insertMusic(list,NORMAL);
                                break;
                        case 'S' :          // 처음 위치에 곡 추가
                                insertMusic(list,FIRST);
```

```
                                        break;
                                case '-' :  // 곡 삭제
                                        deleteMusic(list);
                                        break;
                                case 'E' :  // 비어있는 지 확인
                                        if (isEmpty(list)) printf ("List is empty!₩n");
                                        else printf ("List isn't empty₩n");
                                        break;
                                case 'F' :  // 가득찼는 지 확인
                                        if (isFull(list)) printf ("List is full!₩n");
                                        else printf ("List isn't full!₩n");
                                        break;
                                case 'P' : // 왼쪽으로 커서 이동
                                        movePrev(list);
                                        break;
                                case 'N' : // 오른쪽으로 커서 이동
                                        moveNext(list);
                                        break;
                                case 'Q' :  // 종료
                                        break;
                                default :   // 잘못된 커맨드 입력시
                                        printf("₩nWrong command! Retry!₩n");
                        }
                } while ( cmd !='Q' );

}
```

1) **initList(), isEmpty(), isFull(), printList()** 함수는 아래 코드를 참고할 것. **initList()**함수는 리스트를 초기화 하는 함수로 리스트의 멤버인 first, play, listSize를 각각 0과 NULL로 초기화 한다. isEmpty()와 isFull()은 리스트의 상태를 확인하는 함수로 **isEmpty()**는 리스트가 비어있는지 확인하는 함수로 listSize가 0이면 true를 반환하고 1이상이면 false를 반환한다. **isFull()**은 리스트에 들어있는 음악파일들의 크기를 계산해 (파일 한 개당 크기는 4MB로 가정) 최대 크기를 벗어나는지를 확인하는 함수로 최대 크기를 넘어가면 true, 아닌 경우에는 false를 반환한다. **printList()** 함수는 리스트에 들어있는 곡을 화면에 출력해주는 함수로 각 리스트 노드의 가수, 제목을 순서대로 출력한다.

```
// 리스트 초기화 함수
void initList(LIST *list)
{
        // 리스트 구조체의 멤버를 각각 초기화 한다.
        // 리스트 크기 = 0, 커서와 첫번째 노드 = NULL
        list->first = list->play = NULL;
        list->listSize = 0;
}
```

```
// 리스트가 비어있는지 확인
bool isEmpty(LIST *list)
{
	if (list->listSize == 0) return true;
	else return false;
}

// 용량이 가득 차있는지 확인
bool isFull(LIST *list)
{
	if (list->listSize*SONGSIZE >= MAX) return true;
	else return false;
}

// 리스트 출력
void printList(LIST *list)
{
	int nNum = 1;
	// 임시 사용 포인터 선언
	LISTNODE *tempCursor;
	// 리스트의 첫 노드를 가리킴
	tempCursor = list->first;

	if (isEmpty(list))	// 리스트가 비어있는 경우
	{
		printf("List is empty!₩n");
	}
	else
	{
		printf("=======================================₩n");
		printf("Play : %s, ₩t%s₩n",list->play->singer,list->play->title);
		printf("=======================================₩n");

		do
		{
			// Play list 출력 ->  [곡번호] : [가수], [제목]
			printf("%d : %s, ₩t%s₩n",nNum,tempCursor->singer,tempCursor->title);

			tempCursor = tempCursor->next;
			nNum++;
		}while(tempCursor != list->first);
		printf("₩n");
	}
}
```

2) 리스트에 곡을 추가하는 함수 **void insertMusic(LIST *list, int mode)**의 sub 함수들을 구현하시오. 이 함수는 매개변수로 받는 mode의 값에 따라 **NORMAL** 이면 void insertMusicSub(LIST *list, LISTNODE *node)함수를 호출하고 **FIRST**이면 void insertMusicFirstSub(LIST *list, LISTNODE *node)를 호출한다.

void insertMusicSub(LIST *list, LISTNODE *node)은 현재 플레이되고 있는 곡의 다음 위치에 곡을 추가하고 플레이를 하게 하는 함수이다. 매개변수로 LIST형의 list포인터와 LISTNODE 형의 포인터를 받으며 매개변수 node는 리스트에 추가돼야하는 노드이다.

void insertMusicFirstSub(LIST *list, LISTNODE *node)는 앞의 sub함수와 마찬가지 역할을 하지만 리스트의 제일 처음에 매개변수 node를 추가하는 함수이다.

위의 두 sub 함수들은 삽입이 불가능한 상황 (ex) 리스트가 가득 차있는 경우)에는 그에 따른 에러 메시지를 출력해야 한다.

```
void insertMusic(LIST *list,int mode)
{
	// 새로운 노드 선언
	LISTNODE *tempNode;
	// 새로운 노드에 동적 메모리 할당 후 data 저장
	tempNode = (LISTNODE *)malloc(sizeof(LISTNODE));
	tempNode->title = (char *)malloc(sizeof(char)*20);
	tempNode->singer = (char *)malloc(sizeof(char)*20);

	printf ("==========================================₩n");
	// 가수 입력
	printf ("Singer : ");
	gets(tempNode->singer);
	// 제목 입력
	printf ("Title : ");
	gets(tempNode->title);
	printf("₩n");

	// sub 함수 호출
	if (mode == NORMAL) insertMusicSub(list, tempNode);
	else if (mode == FIRST) insertMusicFirstSub(list, tempNode);
}
void insertMusicSub(LIST *list, LISTNODE *node)
{
	if(isFull(list))
	{
		printf("List is full! Can't insert.₩n");
	}
	else
	{
		if(isEmpty(list))		// 리스트가 비어있는 경우
		{
			list->first = node;
			list->play = node;
```

```
                        node->next = NULL;
                        node->previous = NULL;
                }
                else if(list->play->next == NULL) // 리스트의 끝에 삽입할 경우
                {
                        node->next = NULL;
                        node->previous = list->play;
                        list->play->next = node;

                        list->play = list->play->next;
                }
                else                    // 리스트에 노드가 존재하는 경우
                {
                        node->next = list->play->next;
                        list->play->next->previous = node;
                        list->play->next = node;
                        node->previous = list->play;

                        list->play = list->play->next;
                }
                list->listSize++;
        }
}
// 리스트에 data를 입력하는 함수
void insertMusicFirstSub(LIST *list, LISTNODE *node)
{
        if(isFull(list))
        {
                printf("List is full! Can't insert.₩n");
        }
        else
        {
                if(isEmpty(list))               // 리스트가 비어있는 경우
                {
                        list->first = node;
                        list->play = node;
                        node->next = NULL;
                        node->previous = NULL;
                }
                else                    // 리스트의 처음에 삽입할 경우
                {
                        list->first->previous = node;
                        node->next = list->first;
                        node->previous = NULL;
                        list->first = node;
                }
                list->listSize++;
        }
}
```

3) 리스트에서 현재 플레이 중인 곡을 삭제하는 void deleteMusic(LIST *list)의 sub 함수 void deleteMusicSub(LIST *list, LISTNODE *node)를 구현하시오. 매개변수로 받는 list의 현재 플레이 중인 곡을 리스트에서 삭제하고, 그 정보를 출력해주는 함수이다.

void deleteMusicSub(LIST *list, LISTNODE *node) 함수는 매개변수로 LIST형의 포인터 list를 받아 현재 플레이중인 곡을 삭제하고 그 곡의 정보를 두 번째 매개변수인 node에 저장하는 함수이다.

위 sub 함수는 삭제가 불가능한 상황 (ex) 리스트가 비어있는 경우)에는 그에 따른 에러 메시지를 출력해야 한다.

```
void deleteMusic(LIST *list)
{
	// 삭제된 노드의 data를 임시 저장할 변수
	LISTNODE *deletedNode;
	deletedNode = (LISTNODE *)malloc(sizeof(LISTNODE));
	deletedNode->singer = NULL;
	deletedNode->title = NULL;

	deleteMusicSub(list, deletedNode);

	printf ("₩nDeleted song is %s's %s₩n",deletedNode->singer,deletedNode->title);
	free(deletedNode);
}
// 리스트에서 data를 삭제하는 함수
void deleteMusicSub(LIST *list, LISTNODE *node)
{
	if(isEmpty(list))
	{
		printf("List is empty! Can't delete.₩n");
	}
	else
	{
		node->singer = list->play->singer;
		node->title = list->play->title;

		// 리스트에 노드가 한개만 있는 경우
		if(list->listSize == 1)
		{
			list->play = NULL;
			list->first = NULL;
		}
		// 커서가 리스트의 첫 노드에 있는 경우
		else if(list->first == list->play)
		{
			list->first = list->play->next;
			list->play = list->play->next;
			list->play->previous = NULL;
```

```
                }
                // 커서가 리스트의 마지막 노드에 있는 경우
                else if(list->play->next == NULL)
                {
                        list->play->previous->next = NULL;
                        list->play = list->play->previous;
                }
                else                    // 그 외의 경우
                {
                        list->play->previous->next = list->play->next;
                        list->play->next->previous = list->play->previous;
                        list->play = list->play->next;
                }
                list->listSize--;
        }
}
```

4) 플레이리스트에서 재생곡을 바꾸는 함수 **void movePrev (LIST *list)**와 **void moveNext (LIST *list)**를 구현한다. 이 함수들은 각각 현재 재생곡을 앞의 곡과 뒤의 곡으로 변경시키는 함수이다. 매개변수로 LIST형의 포인터 list를 받고 반환값은 없다. 두 함수는 커서를 이동시킬 수 없는 상황에서 이동을 시키려고 하면 에러메시지를 출력하며 변경이 불가능하게 구현되어야한다. (ex) 리스트가 비어있거나, 리스트에 노드가 한 개만 있는 경우)

```
// 앞 곡으로 변경
void movePrev(LIST *list)
{
        if(isEmpty(list))
        {
                printf("List is empty! Can't move.₩n");
        }
        else
        {
                // 리스트에 노드가 1개밖에 없거나 커서가 첫 노드에 있는 경우
                if(list->play == list->first || list->listSize == 1)
                {
                        printf("The course is in the first item! Can't move.₩n");
                }
                else
                {
                        // 커서를 왼쪽으로 이동
                        list->play = list->play->previous;
                }
        }
}
// 뒷 곡으로 변경
void moveNext(LIST *list)
{
        if(isEmpty(list))
```

```
        {
                printf("List is empty! Can't move.₩n");
        }
        else
        {
                // 리스트에 노드가 1개밖에 없거나 커서가 마지막 노드에 있는 경우
                if(list->play->next == NULL || list->listSize == 1)
                {
                        printf("The course is in the last item! Can't move.₩n");
                }
                else
                {
                        // 커서를 오른쪽으로 이동
                        list->play = list->play->next;
                }
        }
}
```

단계 2. 단계 1에서 완성된 프로그램을 실행시켜 각 함수들을 테스트해보고 그 결과 화면을 캡쳐하시오.

Extra 문제 (+3점). 단계 2까지 완성된 프로그램에서 리스트내의 모든 곡들을 삭제하는 void clearList(LIST *list)함수를 작성하시오. 리스트의 포인터만을 초기화하는 것뿐만 아니라, 동적 할당을 통해 할당받은 각 노드의 메모리 공간을 모두 해제해야 한다.

Extra 문제 (+1점). 단계 2까지 완성된 프로그램에서 LISTNODE의 자료구조와 insertMusic()와 sub 함수, deleteMusic()와 sub함수, isFull()을 수정해 각 곡의 크기를 다르게 저장할 수 있도록 하시오. LISTNODE의 자료구조에 파일의 크기를 저장할 수 있는 필드 unsigned int fileSize를 추가하시오. 이에 따라 insertMusic()에서는 용량을 입력하는 코드를 추가하고, deleteMusic()에서는 삭제된 파일의 크기를 출력하도록 변경하시오. isFull()함수에서는 리스트에 있는 모든 노드들의 크기를 합산해 최대 크기와 비교하도록 수정하시오.

◎ 과제 수행 결과 (결과 화면 및 소감)

[Source]

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

//MP3 플레이어의 최대 용량 128MB
#define MAX 128
//한 곡의 용량 4MB
#define SONGSIZE 4

//리스트 입력 모드 flag
#define NORMAL 0
#define FIRST 1

//리스트 노드 구조체
struct ListNode {
	char *title;		// 제목
	char *singer;		// 가수
	struct ListNode *next; // 다음 노드의 포인터
	struct ListNode *previous; // 이전 노드의 포인터
};
typedef struct ListNode LISTNODE;

//리스트 구조체
struct List{
	int listSize;		// 리스트의 크기
	LISTNODE *play;	// 현재 재생중인 곡
	LISTNODE *first;	// 리스트의 첫 번째 곡
};
typedef struct List LIST;

void initList(LIST *list);			//리스트 초기화 함수
void printList(LIST *list);			//리스트 출력 함수
bool isEmpty(LIST *list);			//리스트가 비어있는지 확인하는 함수
bool isFull(LIST *list);			//용량이 가득차 있는지 확인하는 함수
void insertMusic(LIST *list, int mode);  //곡을 추가하는 함수
void insertMusicSub(LIST *list, LISTNODE *node);		//리스트에 노드 추가
void insertMusicFirstSub(LIST *list, LISTNODE *node);	//리스트의 처음 위치에 노드 추가
void deleteMusic(LIST *list);  //곡을 삭제하는 함수
void deleteMusicSub(LIST *list, LISTNODE *node);		//리스트에서 노드를 삭제하는 함수
void movePrev(LIST *list);	//앞 곡으로 이동
void moveNext(LIST *list);	//뒷 곡으로 이동

void main()
{
	LIST *list;
	char cmd;

	//리스트에 동적으로 메모리 할당
	list = (LIST *)malloc(sizeof(LIST));
```

```
//리스트 초기화
initList(list);

do
{
        printf("*************** Choose Command!! ******************₩n");
        printf("+: Insert song S: Insert song in the first₩n");
        printf("-: Delete song P: Play previous song N: Play next song₩n");
        printf("F: full check   E: empty check   Q: Quit₩n");
        printf("*****************************************************₩n");


        // 리스트 출력
        printList(list);

        // 커맨드 입력
        printf("Command: ");
        cmd = getch();
        // 커맨드는 무조건 대문자로 입력
        cmd = toupper(cmd);
        putch(cmd);
        printf("₩n");

        switch (cmd)
        {
                case '+' :          // 곡 추가
                        insertMusic(list,NORMAL);
                        break;
                case 'S' :          // 처음 위치에 곡 추가
                        insertMusic(list,FIRST);
                        break;
                case '-' :  // 곡 삭제
                        deleteMusic(list);
                        break;
                case 'E' :  // 비어있는 지 확인
                        if (isEmpty(list)) printf ("List is empty!₩n");
                        else printf ("List isn't empty₩n");
                        break;
                case 'F' :  // 가득찼는 지 확인
                        if (isFull(list)) printf ("List is full!₩n");
                        else printf ("List isn't full!₩n");
                        break;
                case 'P' :// 왼쪽으로 커서 이동
                        movePrev(list);
                        break;
                case 'N' :// 오른쪽으로 커서 이동
                        moveNext(list);
                        break;
                case 'Q' :  // 종료
                        break;
```

```
                                default :    // 잘못된 커맨드 입력시
                                        printf("₩nWrong command! Retry!₩n");
                        }
                } while ( cmd !='Q' );

}

// 리스트 초기화 함수
void initList(LIST *list)
{
        // 리스트 구조체의 멤버를 각각 초기화 한다.
        // 리스트 크기 = 0, 커서와 첫번째 노드 = NULL
        list->first = list->play = NULL;
        list->listSize = 0;
}

// 리스트가 비어있는지 확인
bool isEmpty(LIST *list)
{
        if (list->listSize == 0) return true;
        else return false;
}

// 용량이 가득 차있는지 확인
bool isFull(LIST *list)
{
        if (list->listSize*SONGSIZE >= MAX) return true;
        else return false;
}

// 리스트 출력
void printList(LIST *list)
{
        int nNum = 1;
        // 임시 사용 포인터 선언
        LISTNODE *tempCursor;
        // 리스트의 첫 노드를 가리킴
        tempCursor = list->first;

        if (isEmpty(list))     // 리스트가 비어있는 경우
        {
                printf("List is empty!₩n");
        }
        else
        {
                printf("=====================================₩n");
                printf("Play : %s, ₩t%s₩n",list->play->singer,list->play->title);
                printf("=====================================₩n");

                do
                {
```

```
                        // Play list 출력 ->  [곡번호] : [가수], [제목]
                        printf("%d : %s, ₩t%s₩n",nNum,tempCursor->singer,tempCursor->title);

                        tempCursor = tempCursor->next;
                        nNum++;
                }while(tempCursor != NULL);
                printf("₩n");
        }
}

void insertMusic(LIST *list,int mode)
{
        // 새로운 노드 선언
        LISTNODE *tempNode;
        // 새로운 노드에 동적 메모리 할당 후 data 저장
        tempNode = (LISTNODE *)malloc(sizeof(LISTNODE));
        tempNode->title = (char *)malloc(sizeof(char)*20);
        tempNode->singer = (char *)malloc(sizeof(char)*20);

        printf ("=========================================₩n");
        // 가수 입력
        printf ("Singer : ");
        gets(tempNode->singer);
        // 제목 입력
        printf ("Title : ");
        gets(tempNode->title);
        printf("₩n");

        // sub 함수 호출
        if (mode == NORMAL) insertMusicSub(list, tempNode);
        else if (mode == FIRST) insertMusicFirstSub(list, tempNode);
}
void insertMusicSub(LIST *list, LISTNODE *node)
{
        if(isFull(list))
        {
                printf("List is full! Can't insert.₩n");
        }
        else
        {
                if(isEmpty(list))               // 리스트가 비어있는 경우
                {
                        list->first = node;
                        list->play = node;
                        node->next = NULL;
                        node->previous = NULL;
                }
                else if(list->play->next == NULL) // 리스트의 끝에 삽입할 경우
                {
                        node->next = NULL;
                        node->previous = list->play;
```

```
                        list->play->next = node;

                        list->play = list->play->next;
                }
                else                    // 리스트에 노드가 존재하는 경우
                {
                        node->next = list->play->next;
                        list->play->next->previous = node;
                        list->play->next = node;
                        node->previous = list->play;

                        list->play = list->play->next;
                }
                list->listSize++;
        }
}
// 리스트에 data를 입력하는 함수
void insertMusicFirstSub(LIST *list, LISTNODE *node)
{
        if(isFull(list))
        {
                printf("List is full! Can't insert.₩n");
        }
        else
        {
                if(isEmpty(list))               // 리스트가 비어있는 경우
                {
                        list->first = node;
                        list->play = node;
                        node->next = NULL;
                        node->previous = NULL;
                }
                else                    // 리스트의 처음에 삽입할 경우
                {
                        list->first->previous = node;
                        node->next = list->first;
                        node->previous = NULL;
                        list->first = node;
                }
                list->listSize++;
        }
}

void deleteMusic(LIST *list)
{
        // 삭제된 노드의 data를 임시 저장할 변수
        LISTNODE *deletedNode;
        deletedNode = (LISTNODE *)malloc(sizeof(LISTNODE));
        deletedNode->singer = NULL;
        deletedNode->title = NULL;
```

```
		deleteMusicSub(list, deletedNode);

		printf ("₩nDeleted song is %s's %s₩n",deletedNode->singer,deletedNode->title);
		free(deletedNode);
}
// 리스트에서 data를 삭제하는 함수
void deleteMusicSub(LIST *list, LISTNODE *node)
{
		if(isEmpty(list))
		{
			printf("List is empty! Can't delete.₩n");
		}
		else
		{
			node->singer = list->play->singer;
			node->title = list->play->title;

			// 리스트에 노드가 한개만 있는 경우
			if(list->listSize == 1)
			{
				list->play = NULL;
				list->first = NULL;
			}
			// 커서가 리스트의 첫 노드에 있는 경우
			else if(list->first == list->play)
			{
				list->first = list->play->next;
				list->play = list->play->next;
				list->play->previous = NULL;
			}
			// 커서가 리스트의 마지막 노드에 있는 경우
			else if(list->play->next == NULL)
			{
				list->play->previous->next = NULL;
				list->play = list->play->previous;
			}
			else			// 그 외의 경우
			{
				list->play->previous->next = list->play->next;
				list->play->next->previous = list->play->previous;
				list->play = list->play->next;
			}
			list->listSize--;
		}
}

// 앞 곡으로 변경
void movePrev(LIST *list)
{
		if(isEmpty(list))
		{
```

```
                printf("List is empty! Can't move.₩n");
        }
        else
        {
                // 리스트에 노드가 1개밖에 없거나 커서가 첫 노드에 있는 경우
                if(list->play == list->first || list->listSize == 1)
                {
                        printf("The course is in the first item! Can't move.₩n");
                }
                else
                {
                        // 커서를 왼쪽으로 이동
                        list->play = list->play->previous;
                }
        }
}

// 뒷 곡으로 변경
void moveNext(LIST *list)
{
        if(isEmpty(list))
        {
                printf("List is empty! Can't move.₩n");
        }
        else
        {
                // 리스트에 노드가 1개밖에 없거나 커서가 마지막 노드에 있는 경우
                if(list->play->next == NULL || list->listSize == 1)
                {
                        printf("The course is in the last item! Can't move.₩n");
                }
                else
                {
                        // 커서를 오른쪽으로 이동
                        list->play = list->play->next;
                }
        }
}
```