자료구조 보고서

Homework#7

학과 : 소프트웨어학과
학번 : 2016039040
이름 : 윤용진

1. Doubly Linked List 구현에 관한 것이다.

(a) doubly-linked-list.c의 다음 함수를 완성한다.

```
int initialize(headNode** h);
int freeList(headNode* h);
int insertNode(headNode* h, int key);
int insertLast(headNode* h, int key);
int insertFirst(headNode* h, int key);
int deleteNode(headNode* h, int key);
int deleteLast(headNode* h);
int deleteFirst(headNode* h);
int invertList(headNode* h);
```

(b) 이해한 부분을 주석으로 남긴다.

```c
/*
 *  doubly-linked-list.c
 *
 *  Doubly Linked List
 *
 *  Data Structures
 *  Department of Computer Science
 *  at Chungbuk National University
 *
 */



#include<stdio.h>
#include<stdlib.h>
/* 필요한 헤더파일 추가 if necessary */


typedef struct Node {
        int key;
        struct Node* llink;
        struct Node* rlink;
}listNode;



typedef struct Head {
        struct Node* first;
}headNode;
```

```
/* 함수 리스트 */

/* note: initialize는 이중포인터를 매개변수로 받음
         singly-linked-list의 initialize와 차이점을 이해 할것 */
int initialize(headNode** h);

/* note: freeList는 싱글포인터를 매개변수로 받음
        - initialize와 왜 다른지 이해 할것
        - 이중포인터를 매개변수로 받아도 해제할 수 있을 것 */
int freeList(headNode* h);
//int freeList(headNode** h);

int insertNode(headNode* h, int key);
int insertLast(headNode* h, int key);
int insertFirst(headNode* h, int key);
int deleteNode(headNode* h, int key);
int deleteLast(headNode* h);
int deleteFirst(headNode* h);
int invertList(headNode* h);

void printList(headNode* h);


int main()
{
        char command;
        int key;
        headNode* headnode=NULL;

        /* printf("[----- [윤용진]  [2016039040] -----]\n"); */
        printf("[----- [Yoon YongJin]  [2016039040] -----]\n");

        do{

printf("----------------------------------------------------------------\n");
                printf("                                        Doubly  Linked    List
 \n");

printf("----------------------------------------------------------------\n");
                printf(" Initialize    = z          Print          = p \n");
                printf(" Insert Node   = i          Delete Node    = d \n");
                printf(" Insert Last   = n          Delete Last    = e\n");
                printf(" Insert First  = f          Delete First   = t\n");
```

```
                printf(" Invert List    = r              Quit            = q\n");

printf("------------------------------------------------------------------\n");

                printf("Command = ");
                scanf(" %c", &command);

                switch(command) {
                case 'z': case 'Z':
                        initialize(&headnode);
                        break;
                case 'p': case 'P':
                        printList(headnode);
                        break;
                case 'i': case 'I':
                        printf("Your Key = ");
                        scanf("%d", &key);
                        insertNode(headnode, key);
                        break;
                case 'd': case 'D':
                        printf("Your Key = ");
                        scanf("%d", &key);
                        deleteNode(headnode, key);
                        break;
                case 'n': case 'N':
                        printf("Your Key = ");
                        scanf("%d", &key);
                        insertLast(headnode, key);
                        break;
                case 'e': case 'E':
                        deleteLast(headnode);
                        break;
                case 'f': case 'F':
                        printf("Your Key = ");
                        scanf("%d", &key);
                        insertFirst(headnode, key);
                        break;
                case 't': case 'T':
                        deleteFirst(headnode);
                        break;
                case 'r': case 'R':
                        invertList(headnode);
                        break;
```

```
                case 'q': case 'Q':
                        freeList(headnode);
                        break;
                case 'x':
                        freeList(&headnode);
                        break;
                default:
                        printf("\n       >>>>>   Concentration!!   <<<<<      \n");
                        break;
                }
        }while(command != 'q' && command != 'Q');

        return 0;
}


int initialize(headNode** h) {
        /*      전처리
                headNode에 연결된 리스트가 있다면, freeList를 호출하여 할당된 메모리
모두 해제 */
        if (*h != NULL) {
                freeList(h);
        }

        headNode* temp = (headNode*)malloc(sizeof(headNode));
        temp->first = NULL;
        *h = temp;

        return 0;
}

//싱글포인터를 매개변수로 갖는 할당해제 함수
int freeList(headNode* h) {

        listNode* p;
        listNode* prev;

        //이중포인터가 매개변수로 호출되었을 경우
        if (h->first->llink != NULL)
        {
                headNode** x = h;
                listNode** z = *x;
                p = *z;
```

```
                while (p != NULL)
                {
                        prev = p->llink;

                        p = p->rlink;
                        free(prev);
                }
                free(z);
        }
        //싱글포인터가 매개변수로 호출되었을 경우
        else
        {
                p = h->first;

                while (p != NULL)
                {
                        prev = p->llink;

                        p = p->rlink;
                        free(prev);
                }
                free(h);
        }

        return 0;
}

//이중포인터를 매개변수로 갖는 할당해제 함수
//int freeList(headNode** h) {
//      headNode* t = *h;
//      listNode* p;
//      listNode* prev;
//
//      p = t->first;
//
//      while (p != NULL)
//              {
//              prev = p->llink;
//
//              p = p->rlink;
//              free(prev);
//      }
```

```
//
//	free(t);
//	return 0;
//}


void printList(headNode* h) {
	int i = 0;
	listNode* p;

	printf("\n---PRINT\n");

	if(h == NULL) {
		printf("Nothing to print....\n");
		return;
	}

	p = h->first;

	while(p != NULL) {
		printf("[ [%d]=%d ] ", i, p->key);
		p = p->rlink;
		i++;
	}

	printf("  items = %d\n", i);
}




/**
 * list에 key에 대한 노드하나를 추가
 */
int insertLast(headNode* h, int key) {
	/*	전처리	*/
	if (h == NULL) {
		printf("Initialize First\n");
		return 0;
	}

	listNode* p;
	listNode* lastnode = (listNode*)malloc(sizeof(listNode));
```

```
	/* 리스트가 비어있고, 첫 번째 노드 생성일 경우 */
	if (h->first == NULL)
	{
		insertFirst(h, key);
		return 0;
	}

	p = h->first;

	while (p != NULL) {
		if (p->rlink == NULL)
		{
			lastnode->key = key;
			lastnode->llink = p;
			lastnode->rlink = NULL;
			//기존 마지막 노드 오른쪽에 새로운 마지막 노드 연결
			p->rlink = lastnode;
			break;
		}
		p = p->rlink;
	}

	return 0;
}



/**
 * list의 마지막 노드 삭제
 */
int deleteLast(headNode* h) {
	/*	전처리	*/
	if (h == NULL) {
		printf("Initialize First\n");
		return 0;
	}

	if (h->first == NULL)
	{
		printf("There is no Node\n");
		return 0;
	}
```

```
	listNode* p;
	listNode* prev;

	p = h->first;

	while (p != NULL) {
		/* 리스트에 노드가 하나뿐인 경우 */
		if (p->llink == NULL && p->rlink == NULL)
		{
			deleteFirst(h);
			return 0;
		}
		//마지막 노드 삭제
		else if (p->rlink == NULL)
		{
			prev = p->llink;
			prev->rlink = NULL;
			free(p);
			return 0;
		}
		p = p->rlink;
	}

	return 0;
}



/**
 * list 처음에 key에 대한 노드하나를 추가
 */
int insertFirst(headNode* h, int key) {
	/*	전처리	*/
	if (h == NULL) {
		printf("Initialize First\n");
		return 0;
	}

	listNode* firstnode = (listNode*)malloc(sizeof(listNode));

	firstnode->key = key;
	firstnode->llink = NULL;
```

```
        firstnode->rlink = h->first;
        //기존 리스트가 있는 경우, 기존 첫 노드의 왼쪽에 새로운 첫 노드를 연결
        if (h->first != NULL) h->first->llink = firstnode;
        h->first = firstnode;

        return 0;
}

/**
 * list의 첫번째 노드 삭제
 */
int deleteFirst(headNode* h) {
        /*      전처리   */
        if (h == NULL) {
                printf("Initialize First\n");
                return 0;
        }

        if (h->first == NULL)
        {
                printf("There is no Node\n");
                return 0;
        }

        listNode* p;

        p = h->first;

        //노드가 하나뿐인 리스트의 경우
        if (p->rlink == NULL)
        {
                h->first = NULL;
        }
        else
        {
                //삭제할 첫 노드 다음 노드를 기억
                listNode* secondnode;
                //두번째 노드를 첫 노드로 변경
                secondnode = p->rlink;
                secondnode->llink = NULL;
                h->first = secondnode;
        }
        //기존의 첫번째 노드 할당 해제
```

```
        free(p);
        return 0;
}



/**
 * 리스트의 링크를 역순으로 재 배치
 */
int invertList(headNode* h) {
        /*      전처리   */
        if (h == NULL) {
                printf("Initialize First\n");
                return 0;
        }

        if (h->first == NULL)
        {
                printf("Make List First\n");
                return 0;
        }

        //리스트의 순서를 바꾸는 과정에서 기존 리스트의 순서를 기억하기 위한 임시노드
        listNode* nextnode = NULL;
        //리스트의 끝에 도달하면 새롭게 헤드노드의 첫 연결 노드가 될 마지막 노드
        listNode* lastnode = NULL;

        listNode* p;

        p = h->first;

        while (p != NULL)
        {
                //리스트의 마지막 노드 기억
                if (p->rlink == NULL) lastnode = p;
                //리스트의 순서 변경
                nextnode = p->rlink;
                p->rlink = p->llink;
                p->llink = nextnode;
                //기존 리스트의 순서를 따라 다음 노드 탐색
                p = nextnode;
        }
```

```
        h->first = lastnode;

        return 0;
}



/* 리스트를 검색하여, 입력받은 key보다 큰값이 나오는 노드 바로 앞에 삽입 */
int insertNode(headNode* h, int key) {
        /*      전처리   */
        if (h == NULL) {
                printf("Initialize First\n");
                return 0;
        }

        listNode* p;
        listNode* node2insert = (listNode*)malloc(sizeof(listNode));

        /* 리스트가 비어있고, 첫 번째 노드 생성일 경우 */
        if (h->first == NULL)
        {
                insertFirst(h, key);
                return 0;
        }

        p = h->first;

        while (p != NULL) {
                /* 새로운 노드의 삽입 위치가 리스트의 첫 노드 앞인 경우 */
                if (p == h->first && p->key > key)
                {
                        insertFirst(h, key);
                        return 0;
                }
                /* 현재 탐색중인 노드의 키값이 입력받은 키값보다 큰 경우*/
                else if (p->key > key)
                {
                        node2insert->key = key;

                        node2insert->llink = p->llink;
                        p->llink->rlink = node2insert;

                        node2insert->rlink = p;
```

```
                        p->llink = node2insert;

                        return 0;
                }
                p = p->rlink;
        }

        /* 입력받은 키 값보다 큰 키 값을 갖는 노드가 없는 경우 마지막 노드로 새 노드 추
가 */
        insertLast(h, key);

        return 0;
}


/**
 * list에서 key에 대한 노드 삭제
 */
int deleteNode(headNode* h, int key) {
        /*      전처리   */
        if (h == NULL) {
                printf("Initialize First\n");
                return 0;
        }

        if (h->first == NULL)
        {
                printf("There is no Node\n");
                return 0;
        }

        listNode* p;

        p = h->first;

        while (p != NULL) {
                /* 삭제할 노드가 첫 번째 노드인 경우 */
                if (p == h->first && p->key == key)
                {
                        deleteFirst(h);
                        return 0;
                }
                /* 삭제할 노드가 마지막 노드인 경우 */
```

```
                else if (p->rlink == NULL && p->key == key)
                {
                        deleteLast(h);
                        return 0;
                }
                /* 리스트에서 처음으로 마주치는 키 값이 일치하는 노드를 삭제 */
                else if (p->key==key)
                {
                        p->llink->rlink = p->rlink;
                        p->rlink->llink = p->llink;
                        free(p);
                        return 0;
                }
                p = p->rlink;
        }

        //모든 노드 탐색이 끝날 때까지 삭제할 노드를 찾지 못했다면
        printf("There is no node corresponding to the key value.\n");

        return 0;
}
```

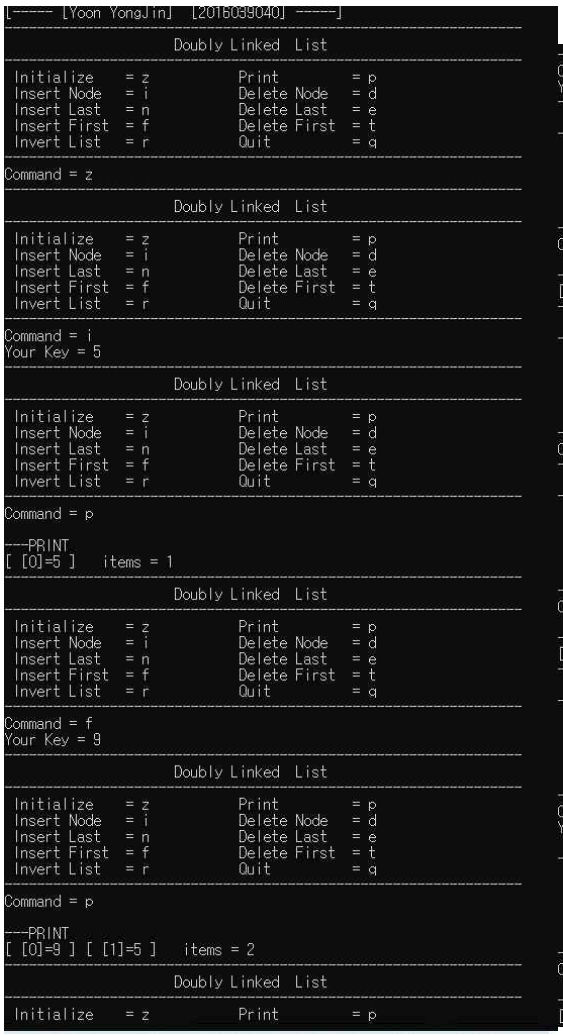
4. GitHub에 hw7 Repository를 생성하고 doubly-linked-list.c를 업로드 한다.

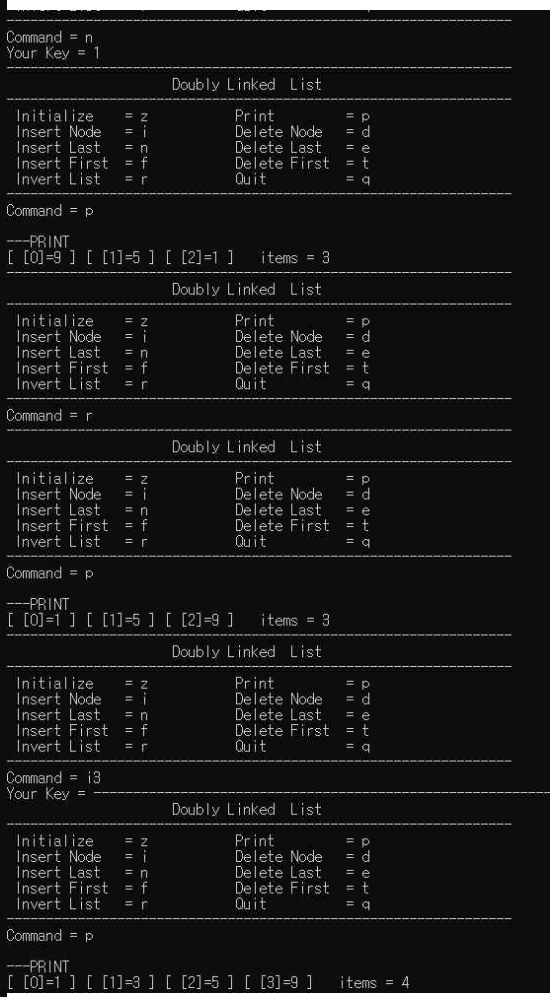
https://github.com/uniz21/DataStructure-HW-7

저만의 문제점인지 모르겠지만 이전부터
예시 소스코드를 브라우저에서 블록지정하여 복사/붙여넣기 하지 말고 오른쪽 클릭-다른 이름으로 저장...을 눌러 저장한다.
이 방법으로 과제의 예시코드를 저장해도 파일의 인코딩 문제인지 정상 작동하던 코드가 한글 주석 작성 이후 작동하지 않습니다.
그래서 문제가 생기는 주석처리 파일, 주석 제거 파일, EUC-KR 인코딩의 c파일을 새로 만들어 제출합니다.

7. 보고서에 실행결과를 Screen Capture하여 첨부한다.

실행결과





```
Command = i
Your Key = 8
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command = p

---PRINT
[ [0]=1 ] [ [1]=3 ] [ [2]=5 ] [ [3]=8 ] [ [4]=9 ]   items = 5
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command = t
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command = p

---PRINT
[ [0]=3 ] [ [1]=5 ] [ [2]=8 ] [ [3]=9 ]   items = 4
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command = e
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command = p

---PRINT
[ [0]=3 ] [ [1]=5 ] [ [2]=8 ]   items = 3
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command = d
Your Key = 5
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command = p

---PRINT
[ [0]=3 ] [ [1]=8 ]   items = 2
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command = z
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command = p

---PRINT
  items = 0
--------------------------------------------------------------------------------
                        Doubly Linked  List
--------------------------------------------------------------------------------
 Initialize    = z          Print         = p
 Insert Node   = i          Delete Node   = d
 Insert Last   = n          Delete Last   = e
 Insert First  = f          Delete First  = t
 Invert List   = r          Quit          = q
--------------------------------------------------------------------------------
Command =
```