## 자료구조 보고서

Homework#6

학과 : 소프트웨어학과

학번 : 2016039040

이름 : 윤용진

1. Singly Linked List 구현에 관한 것이다.

```
/*
 * singly linked list
* Data Structures
 * Department of Computer Science
 * at Chungbuk National University
 */
/* 필요한 헤더파일 추가 */
#include<stdio.h>
#include<stdlib.h>
typedef struct Node {
        int key;
        struct Node* link;
} listNode;
typedef struct Head {
        struct Node* first;
}headNode;
/* 함수 리스트 */
headNode* initialize(headNode* h);
int freeList(headNode* h);
int insertFirst(headNode* h, int key);
int insertNode(headNode* h, int key);
int insertLast(headNode* h, int key);
int deleteFirst(headNode* h);
int deleteNode(headNode* h, int key);
int deleteLast(headNode* h);
int invertList(headNode* h);
void printList(headNode* h);
int main()
        char command;
```

```
int key;
     headNode* headnode=NULL;
     /* printf("[----- [윤용진] [2016039040] -----]\n"); */ /* 내려받은 소스의 인코
딩 문제로 한글 깨짐 */
     printf("[---- [Yoon YongJin] [2016039040] ----]\n");
     do{
printf("-----\n");
          printf("
                                           Singly Linked List
\n");
printf("-----\n");
           printf(" Invert List = r
                                 Quit
                                           = q n''
printf("-----\n");
           printf("Command = ");
           scanf(" %c", &command);
switch(command) {
           case 'z': case 'Z':
                headnode = 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;
                default:
                        printf("\n
                                       >>>> Concentration!! <<<<
                                                                           \n");
                        break;
                }
        }while(command != 'q' && command != 'Q');
        return 1;
headNode* initialize(headNode* h) {
        /* headNode가 NULL이 아니면, freeNode를 호출하여 할당된 메모리 모두 해제 */
        if(h != NULL)
                freeList(h);
        /* headNode에 대한 메모리를 할당하여 리턴 */
        headNode* temp = (headNode*)malloc(sizeof(headNode));
        temp->first = NULL;
        return temp;
```

```
int freeList(headNode* h){
       /* h와 연결된 listNode 메모리 해제
        * headNode도 해제되어야 함.
        */
       listNode* p = h->first;
       listNode* prev = NULL;
       while(p != NULL) {
               prev = p;
               p = p - \sinh;
               free(prev);
       free(h);
       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* node = (listNode*)malloc(sizeof(listNode));
       /* 빈 노드에 key값 저장 */
       node->key = key;
       /* 노드를 리스트 첫부분에 연결 */
       node->link = h->first;
       h->first = node;
       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;
       }
       int didInsert = 0;
       listNode* p;
       listNode* newNode = (listNode*)malloc(sizeof(listNode));
       /* 리스트의 끝에 도달한 이후 NULL값을 갖는 링크의 key 혹은 link를 탐색하지 않
도록 이전 노드를 기억한다.*/
       listNode* prev;
       /* 리스트가 비어있고, 첫 번째 노드 생성일 경우 */
       if (h->first == NULL)
               insertFirst(h, key);
               didInsert = 1;
              return 0;
       }
       p = h \rightarrow first;
       prev = h->first;
       while (p != NULL) {
               /* 새로운 노드의 삽입 위치가 리스트의 첫 노드 앞인 경우 */
               if (h-\text{sfirst} == p \&\& \text{key} < p-\text{key})
                      newNode->key = key;
                      newNode->link = p;
                      h->first = newNode;
                      didInsert = 1;
                      break;
               /* 현재 검색중인 노드의 키값이 입력받은 키값보다 큰 경우*/
               else if (key < p->key)
                      newNode->key = key;
                      newNode->link = p;
```

```
prev->link = newNode;
                       didInsert = 1;
                       break;
               prev = p;
               p = p->link;
       /* 입력받은 키 값보다 같거나 큰 키 값을 갖는 노드가 없을 경우 가장 뒤에 노드 추
가 */
       if (!didInsert)
               insertLast(h, key);
               return 0;
       }
       return 0;
/**
* list 끝에 key에 대한 노드하나를 추가
int insertLast(headNode* h, int key)
               전처리 */
       if (h == NULL)
               printf("Initialize First\n");
               return 0;
       }
       listNode* p;
       listNode* newNode = (listNode*)malloc(sizeof(listNode));
       /* 리스트가 비어있고, 첫 번째 노드 생성일 경우 */
       if (h->first == NULL)
       {
               insertFirst(h, key);
               return 0;
       }
       p = h \rightarrow first;
```

```
while (p != NULL) {
                if (p->link==NULL)
                        newNode->key = key;
                        newNode->link = NULL;
                        p->link = newNode;
                        break;
               }
               p = p - \sinh;
       }
       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* node = (listNode*)malloc(sizeof(listNode));
       node = h->first;
        /* 두번째 노드를 첫 노드로 변경 */
       h->first = node->link;
       /* 메모리 해제 */
       free(node);
       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;
       }
       int i = 0;
       listNode* p;
       listNode* prev;
       p = h \rightarrow first;
       while (p != NULL) {
               /* 삭제할 노드가 첫 번째 노드인 경우 */
               if (i == 0 && key == p->key)
               {
                       deleteFirst(h);
                       return 0;
               /* 리스트에서 처음으로 마주치는 키 값이 일치하는 노드를 삭제 */
               else if (key == p->key)
               {
                       prev->link = p->link;
                       free(p);
                       break;
               prev = p;
               p = p->link;
               į++;
       }
```

```
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;
        }
        int i = 0;
        listNode* p;
        listNode* prev;
        p = h->first;
        while (p != NULL) {
                /* 삭제할 노드가 첫 번째 노드인 경우 */
                if (i == 0&& p->link == NULL)
                       deleteFirst(h);
                       return 0;
                /* 리스트의 마지막 노드 삭제*/
                if (p->link == NULL)
                       prev->link = NULL;
                       free(p);
                       break;
                prev = p;
                p = p->link;
                į++;
```

```
return 0;
/**
* 리스트의 링크를 역순으로 재 배치
int invertList(headNode* h) {
       int listIndex = 0;
       listNode* p;
       /* 역순으로 재 배치할 리스트의 노드 */
       listNode* reverse = NULL;
       listNode* nextNode;
       p = h \rightarrow first;
       while (p != NULL)
              /* 리스트의 다음노드부터 임시 저장 */
              nextNode = p->link;
              /* 재 배치 */
               p->link = reverse;
              /* 재 배치된 리스트를 역순 리스트에 저장 */
              reverse = p;
              /* 역순 배치되지 않은 기존 리스트를 불러온다 */
              p = nextNode;
       }
       h->first = reverse;
       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->link;
    i++;
}

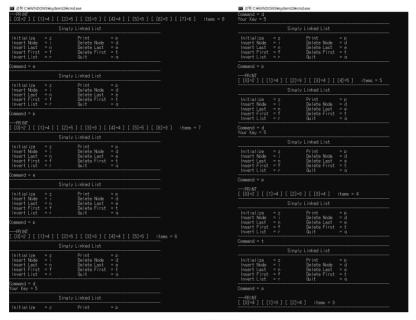
printf(" items = %d\n", i);
}
```

4. GitHub에 hw6 Repository를 생성하고 singly-linked-list.c를 업로드 한다.

## https://github.com/uniz21/DataStructure-HW-6

7. 보고서에 실행결과를 Screen Capture하여 첨부한다. 실행결과





■ 선택 C:₩WINDOWS₩syst	em32#cmd.exe	
PRINT [ [0]=4 ] [ [1]=3 ]	[[2]=4] items = 3	
	Singly Linked List	
Initialize = z Insert Node = i Insert Last = n Insert First = f Invert List = r	Print Delete Node Delete Last Delete First Quit	= p = d = e = t = a
Command = r		
	Singly Linked List	
Initialize = z Insert Node = i Insert Last = n Insert First = f Invert List = r	Print Delete Node Delete Last Delete First Ouit	= p = d = e = t = q
Command = p		
PRINT [ [0]=4 ] [ [1]=3 ]	[[2]=4] items = 3	
	Singly Linked List	
Initialize = z Insert Node = i Insert Last = n Insert First = f Invert List = r	Print Delete Node Delete Last Delete First Quit	= P = d = e = t = q
Command = z		_
	Singly Linked List	
Initialize = z Insert Node = i Insert Last = n Insert First = f Invert List = r	Print Delete Node Delete Last Delete First Quit	
Command = p		
PRINT items = 0		
	Singly Linked List	
Initialize = z Insert Node = i Insert Last = n Insert First = f Invert List = r	Print Delete Node Delete Last Delete First Quit	= p = d = e = t = a
Command = f Your Key = 2		
	Singly Linked List	
Initialize = z	Pr int	* p

	0111913	EIIIKGG EIGE	
Initialize Insert Node Insert Last Insert First Invert List	= z = i = n = f = r	Print Delete Node Delete Last Delete First Quit	= A
Command = n Your Key = 6			
	Singly	Linked List	
Initialize Insert Node Insert Last Insert First Invert List	= z = i = n = f = r	Print Delete Node Delete Last Delete First Quit	= p = d = e = t = a
Command = n Your Key = 7			
	Singly	Linked List	
Initialize Insert Node Insert Last Insert First Invert List		Print Delete Node Delete Last Delete First Ouit	= e
Command = n Your Key = 9			
	Singly	Linked List	
Initialize Insert Node Insert Last Insert First Invert List		Print Delete Node Delete Last Delete First Quit	
Command = p			
PRINT [ [0]+2 ] [ [1	]=6 ] [ [2]=7	] [ [8]=9 ]	items = 4
	Singly	Linked List	
Initialize Insert Node Insert Last Insert First Invert List	= z = i = n = f = r	Print Delete Node Delete Last Delete First Quit	# p d # e t a
Command = r			
	Singly	Linked List	
Initialize Insert Node	= z + i	Print Delete Node	= p

Command = r			
	Sir	ngly Linked List	
Initialize			
Insert Node		Delete Node	
Insert Last		Delete Last	
Insert First		Delete First	
Invert List Command = pPRINT		Delete First Quit 2]=6 ] [ [3]=2 ]	
Invert List Command = pPRINT	9=7 ] [ [		
Invert List Command = pPRINT	9=7 ] [ [	Ouit 2]=6][[3]=2]	
Invert List Command = p PRINT [[0]=9] [[1	* r  ]=7 ] [ []  Sii	Ouit 2]=6 ] [ [3]=2 ] ngly Linked List	= q items = 4
Invert List  Command = p PRINT [[0]=9] [[1]  Initialize Insert Node Insert Last	* r  ]=7 ] [ [:   Sii   = z   = i	Ouit 2]=6 ] [ [3]=2 ] ngly Linked List Print	= q items = 4 = p = d = e
Invert List  Command = p PRINT [[0]=9] [[1]  Initialize Insert Node	* r  ]=7 ] [ []  Sii  = z  = i	Ouit  2]=6][[3]=2]  ngly Linked List  Print Delete Node	= q items = 4 = p = d