Data Structure in C
Homework #3
Due 25 (Sun), Midnight, Nov 2018

1. 단순 연결 리스트 (Singly LInked List)를 이용하여 숫자들을 항상 정렬된 상태로 유지하는 리스트 SortedList를 구현한다. 즉, 아래 함수들을 구현한 후, 간단한 테스트 함수 (main function)를 만들어 테스트 하시오 (단, 정렬은 오름차순을 기준함): [30 pt]

void add(List *list, int item) : 정렬된 리스트에 새로운 요소 추가
void delete(List *list, int item) : 정렬된 리스트에서 item 제거
void clear(List *list) : 리스트의 모든 아이템 삭제
int isInList(List *list, int item) : item이 리스트 안에 있는지를 검사하여, 있으면 TRUE (1)를 리턴하고 없으면 FALSE(0)를 리턴
unsigned int getLength(List *list) : 리스트의 길이 (즉, 리스트 내의 아이템 개수)를 구한다
int isEmpty(List *list) : 리스트가 비었는지를 검사하여, 비었으면 TRUE (1)를 리턴하고 아이템이 존재하면 FALSE(0)를 리턴
void display(List *list) : 리스트의 모든 요소를 모니터에 출력

Note 1) 테스트 함수는 무한 루프를 이용하여 위의 함수들을 메뉴를 이용해 테스트하는 프로그램을 작성함.
Note 2) 단순 연결 리스트와 노드는 아래와 같은 구조를 이용함:

```
typedef struct node{
        int data;
        struct node *next;
} Node;

typedef struct list{
        Node *head;
}
```

2. Dummy node 와 이중 연결 리스트를 이용하여 최근 실행한 모바일 앱 이름을 저장하는 app history 관리 프로그램을 작성한다. Dummy node로는 dummyHead와 dummyTail 두 개의 노드를 만들고, 새로운 노드를 추가할 경우 dummyHead 뒤에 추가한다 (즉, 가장 오래된 앱 이름이 dummyTail 바로 앞에 저장되고, 가장 최근에 사용한 앱이 dummyHead 바로 뒤에 저장된다). 이중연결리스트의 기능으로서 다음 함수들을 구현한 후, 간단한 테스트 함수 (main function)를 만들어 테스트 하시오 (테스트 함수는 1번 문제와 같이 무한 루프를 이용하여 위의 함수들을 메뉴를 이용해 테스트하는 프로그램을 작성함.) [40 pt]

unsigned int countList(List *list) // 리스트내의 노드 개수를 반환 (더미 노드 제외)
void printList(List *list) // 리스트내의 노드 데이터들을 헤드 노드 다음부터 출력 (즉, 가장 최근 사용한 앱부터 출력)
Node *createNode(const char *str) // str 문자열 포인터를 데이터로 갖는 이중 연결 리스트 노드에 대해 동적 생성 후, 노드의 주소 값 반환
void insert(List *list) // 리스트에 데이터 추가 (더미 헤드 노드 바로 다음에 추가)
Node *search(List *list, const char *str) // 리스트내의 노드 데이터들을 차례대로 검색해서 str 값과 일치하는 노드가 있으면 해당 노드의 주소를 반환; 일치하는 노드가 없을 경우 NULL 반환
void remove(List *list, Node *p) // p 포인터가 가리키는 노드를 리스트에서 삭제; 오류 상황의 경우 적절한 메시지를 화면에 출력하고 exit() 함수를 이용하여 빠져나감

Note) 이중 연결 리스트와 노드는 아래와 같은 구조를 이용함:

```
typedef struct Dnode{
        char *data;
        struct Dnode *next;
        struct Dnode *prev;
} DNode;
```

```
typedef struct Dlist{
        Node *head;             // 더미 헤드 노드를 가리킴
        Node *tail;             // 더미 테일 노드를 가리킴
} DList;


int main(void){
        ….
        DNode dummyHead;
        DNode dummyTail;
        DList mylist;
        ….

        // 더미 헤드와 더미 테일 노드의 초기화
        dummyHead.data = NULL;    // 초기화하지만 사용하지는 않는 데이터
        dummyHead.next = NULL;
        dummyHead.prev = NULL;

        dummyTail.data = NULL;              // 초기화하지만 사용하지는 않는 데이터
        dummyTail.next = NULL;
        dummyTail.prev = NULL;

        // 리스트의 초기화
        mylist.head = &dummyHead;
        mylist.tail = &dummyTail;

        …
}
```

3. 스택을 배열과 연결 리스트를 이용하여 각각 구현한 후, 아래 코드와 같이 스택에 1,000,000번 정도 push()와 pop()을 반복하는 프로그램을 작성하여 수행시간을 각각 측정한다. 배열로 구현된 스택과 연결 리스트로 구현된 스택에 대하여 수행시간을 기록한 후, 어느 것이 더 빠른지와 그 이유를 간단히 기술하시오. [30pt]

Hint) 다음과 같이 clock() 함수를 이용함.

```
#include <stdio.h>
#include <time.h>

clock_t start, finish;
double duration;
int i, x;

…
initialize(&s);

start = clock();
for (i=0; i<1000000; i++){
    push(&s, i);
    pop(&s, &x);
}
finish = clock();
duration = (double)(finish - start) / CLOCKS_PER_SEC;
printf(“%f 초입니다.\n", duration);
….
```