/\*

다음은 단순연결리스트의 기능을 구현하는 프로그램이다.

구현할 기능은

1. makeList – 복수개의 입력되는 데이터를 단순연결리스트로 만드는 기능

2. viewList – 만들어진 단순연결리스트의 데이터를 순서대로 출력하는 기능

3. findLast – 리스트의 맨 마지막 노드를 찾는 기능으로 그 포인터를 리턴함

4. findPosition – 이 기능은 전달인자로 입력되는 데이터를 리스트에서 찾아 만약 리스트에 존재하면 1을 리턴하고 만약 데이터가 리스트에 존재하지 않으면 0을 리턴한다. 또한 리스트의 헤더 포인터외 찾는 데이터를 가르키는 포인터와 그 노드의 이전노드를 가르키는 포인터를 전달인자로 전달하여 만약 존재 한다며 그 포인터 값을 저장하여야 함.

5. insertList – 새로운 데이터를 리스트에 삽입하는 기능으로 전달인자로 리스트 헤더 포인터, 어디 다음에 삽입해야 하는지 알려주는 포인터, 그리고 삽입할 데이터를 전달인자로 전달 받으며 이 기능에서 새로운 노드를 생성하여 삽입하여야 함. 또한, 리스트 생성, 즉 makeList에서 이 기능을 호출하여 리스트를 생성하여야 함.

6. deleteList – 리스트에 원하는 데이터를 삭제하는 기능으로 전달인자로, 리스트 헤더 포인터, 그리고 삭제를 원하는 노드의 포인터 및 이전 포인터가 사용됨.

주의: insertList와 deleteList에서 원하는 노드의 포인터(삽입 혹은 삭제를 원하는 노드 및 그 이전 노드의 포인터)의 값을 구하기 위하여 findPosition기능 사용.

\*/

#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int data;

struct node \*link;

}node; //노드 타입 정의

void makeList(node \*\*); // 리스트 생성 함수. 전달인자: 헤더 포인터의 포인터

void viewList(node \*); // 리스트 출력 함수

node \* findLast(node \*); /\* 리스트 마지막 노드를 찾는 함수로 그 노드의 포인터 리턴

전달인자 : 헤더 포인터 값 \*/

int findPosition(node \*, node \*\*, node \*\*, int);

/\* 입력 데이터와 같은 데이터를 포함하는 노드의 포인터 및 그 이전 노드의 포인터를 찾는 함수.

전달인자: 헤더포인터, 이전노드 포인터, 타깃 포인터, 찾기 원하는 데이터 \*/

void insertList(node \*\*, node \*, int);

/\* 리스트 삽입함수 전달인자: 헤터 포인터의 포인터, 타깃 포인터, 삽입데이터 \*/

void deleteList(node \*\*, node \*, node \*);

/\* 리스트 삭제함수 전달인자: 헤더 포인터의 포인터, 이전포인터, 타깃포인터 \*/

/\* 타깃 포인터는 findPosition에서 원하는 데이터의 위치 이며 이전 포인터는 그 노드의 이전 노드의 포인터임 \*/

void main() {

node \*head = NULL;

node \*pre = NULL;

node \*targetPtr = NULL;

int newData, target;

int found;

// 노드 생성

makeList(&head);

// 생성된 리스트 출력

viewList(head);

fflush(stdin);

// 데이터 삽입

printf("Enter data to insert :\n");

scanf("%d", &newData);

printf("어디 다음에 삽입할까요 : < 첫음에 입력을 원하면 -1 입력>\n");

scanf("%d", &target);

if (target == -1) {

targetPtr == NULL;

insertList(&head, targetPtr, newData);

}

else {

found = 0;

found = findPosition(head, &pre, &targetPtr, target);

if (found == 1)

insertList(&head, targetPtr, newData);

else

printf("There is no %d in the list\n", target);

}

viewList(head);

// delete list

printf("Enter data to delete from list\n");

scanf("%d", &target);

printf("targeted data is %d\n", target);

found = 0;

found = findPosition(head, &pre, &targetPtr, target);

printf("data is %d in the list", found);

printf("found data is %d\n", targetPtr->data);

if (found == 1)

deleteList(&head, pre, targetPtr);

else

printf("There is no %d in the list\n", target);

// View list data

viewList(head);

}

node \* findLast(node \*hptr) {

만약 헤더가 널이 아니면 마지막 노드까지 포인터 이동

리턴 hptr

}

int findPosition(node \*hptr, node \*\* pre, node \*\*targetPtr, int data) {

\*pre = NULL;

\*targetPtr = NULL;

hptr값이 널이 아니고 hptr->data값이 찾는 데이터가 아닐동안

\*pre값 = hptr값

hptr은 다음 노드로 이동

만약 hptr->data와 data의 값이 같으면

\*targetPtr값 = hptr

리턴 1

그렇지 않으면

리턴 0

}

void insertList(node \*\*hptr, node \*pptr, int data) {

node \* new;

새로운 노드 생성

만약 \*hptr 값이 널이면 현재 empty 이므로

\*hptr이 new를 가리키게 함

}

그렇지 않고

만약 pptr값이 널이면 처음에 입력해야 함으로

새로운 노드의링크, 즉 new->link는 \*hptr값으로 함

\*hptr값은 new의 값으로 함.

그렇지 않다면, 즉 리스트 중간에 삽입해야 함으로

new->link 의 값은 pptr의 링크 값이며

pptr->link 는 new의 값으로 함

}

void deleteList(node \*\*hptr, node \*pre, node \*target) {

if (pre == NULL) {

\*hptr = target->link;

}

else {

pre->link = target->link;

}

free(target);

}

void makeList(node \*\*hptr) {

node \*pre = NULL;

node \*new = NULL;

int data;

findLast를 이용하여 리스트 마지막 노드 포인터값을 pre에 저장

printf("입력할 데이터 는 < enter -1 to end>\n");

scanf("%d", &data);

while (data != -1) {

insertList(hptr, pre, data);

리스트 마지막 노드 포인터 확인

printf("Enter data again\n");

scanf("%d", &data);

}

}

void viewList(node \*ptr) {

while (ptr != NULL) {

printf("%d ", ptr->data);

ptr = ptr->link;

}

}