/\*

240408 연결리스트

\*/

#include <stdio.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#define MAX\_LIST\_SIZE 100 // 리스트의 최대크기

typedef int element;// 항목의 정의

typedef struct {

element array[MAX\_LIST\_SIZE]; // 배열 정의

int size; // 현재 리스트에 저장된 항목들의 개수

} ArrayListType;

// 오류 처리 함수

void error(const char\* message)

{

fprintf(stderr, "%s\n", message);

exit(1);

}

// 리스트 초기화 함수

void init(ArrayListType\* L)

{

L->size = 0;

}

// 리스트가 비어 있으면 1을 반환

// 그렇지 않으면 0을 반환

int is\_empty(ArrayListType\* L)

{

return L->size == 0;

}

// 리스트가 가득 차 있으면 1을 반환

// 그렇지 많으면 1을 반환

int is\_full(ArrayListType\* L)

{

return L->size == MAX\_LIST\_SIZE;

}

element get\_entry(ArrayListType\* L, int pos)

{

if (pos < 0 || pos >= L->size)

error("위치 오류");

return L->array[pos];

}

// 리스트 출력

void print\_list(ArrayListType\* L)

{

int i;

for (i = 0; i < L->size; i++)

printf("%d->", L->array[i]);

printf("\n");

}

void insert\_last(ArrayListType\* L, element item)

{

if (L->size >= MAX\_LIST\_SIZE) {

error("리스트 오버플로우");

}

L->array[L->size++] = item;

}

void insert(ArrayListType\* L, int pos, element item)

{

if (!is\_full(L) && (pos >= 0) && (pos <= L->size)) {

for (int i = (L->size - 1); i >= pos; i--)

L->array[i + 1] = L->array[i];

L->array[pos] = item;

L->size++;

}

}

element delete1(ArrayListType\* L, int pos)

{

element item;

if (pos < 0 || pos >= L->size) error("위치 오류");

item = L->array[pos];

for (int i = pos; i < (L->size - 1); i++)

L->array[i] = L->array[i + 1];

L->size--;

return item;

}

int main(void)

{

// ArrayListType를 정적으로 생성하고 ArrayListType를

// 가리키는 포인터를 함수의 매개변수로 전달한다.

ArrayListType list;

init(&list);

insert(&list, 0, 10); print\_list(&list);// 0번째 위치에 10 추가

insert(&list, 1, 20); print\_list(&list);// 1번째 위치에 20 추가

insert(&list, 0, 30); print\_list(&list);// 0번째 위치에 30 추가 - 첫번째에 30 추가

insert\_last(&list, 40); print\_list(&list);// 맨 끝에 40 추가

delete1(&list, 0); print\_list(&list);// 0번째 항목 삭제

return 0;

}

**단순 연결 리스트의 연산 #5**

#include <stdio.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

typedef int element;

typedef struct ListNode { // 노드 타입

element data;

struct ListNode\* link;

} ListNode;

// 오류 처리 함수

void error(const char\* message)

{

fprintf(stderr, "%s\n", message);

exit(1);

}

ListNode\* insert\_first(ListNode\* head, int value)

{

ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));//(1)

p->data = value;// (2)

p->link = head;// 헤드 포인터의 값을 복사//(3)

head = p;// 헤드 포인터 변경//(4)

return head;// 변경된 헤드 포인터 반환

}

// 노드 pre 뒤에 새로운 노드 삽입

ListNode\* insert(ListNode\* head, ListNode\* pre, element value)

{

ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));//(1)

p->data = value;//(2)

p->link = pre->link;//(3)

pre->link = p;//(4)

return head;//(5)

}

ListNode\* delete\_first(ListNode\* head)

{

ListNode\* removed;

if (head == NULL) return NULL;

removed = head;// (1)

head = removed->link;// (2)

free(removed);// (3)

return head;// (4)

}

// pre가 가리키는 노드의 다음 노드를 삭제한다.

ListNode\* delete1(ListNode\* head, ListNode\* pre)

{

ListNode\* removed;

removed = pre->link;

pre->link = removed->link;// (2)

free(removed);// (3)

return head;// (4)

}

void print\_list(ListNode\* head)

{

for (ListNode\* p = head; p != NULL; p = p->link)

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

printf("NULL \n");

}

// 테스트 프로그램

int main(void)

{

ListNode\* head = NULL;

for (int i = 0; i < 5; i++) {

head = insert\_first(head, i);

print\_list(head);

}

for (int i = 0; i < 5; i++) {

head = delete\_first(head);

print\_list(head);

}

return 0;

}

// 단어들을 저장하는 리스트

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

typedef struct {

char name[100];

} element;

typedef struct ListNode { // 노드 타입

element data;

struct ListNode\* link;

} ListNode;

// 오류 처리 함수

void error(const char\* message)

{

fprintf(stderr, "%s\n", message);

exit(1);

}

ListNode\* insert\_first(ListNode\* head, element value)

{

ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));//(1)

p->data = value;// (2)

p->link = head;// 헤드 포인터의 값을 복사//(3)

head = p;// 헤드 포인터 변경//(4)

return head;

}

void print\_list(ListNode\* head)

{

for (ListNode\* p = head; p != NULL; p = p->link)

printf("%s->", p->data.name);

printf("NULL \n");

}

// 테스트 프로그램

int main(void)

{

ListNode\* head = NULL;

element data;

strcpy(data.name, "APPLE");

head = insert\_first(head, data);

print\_list(head);

strcpy(data.name, "KIWI");

head = insert\_first(head, data);

print\_list(head);

strcpy(data.name, "BANANA");

head = insert\_first(head, data);

print\_list(head);

return 0;

}

#include <stdio.h>

#include <stdlib.h>

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

typedef int element;

typedef struct ListNode { // 노드 타입

element data;

struct ListNode\* link;

} ListNode;

ListNode\* insert\_first(ListNode\* head, element value)

{

ListNode\* p = (ListNode\*)malloc(sizeof(ListNode));//(1)

p->data = value;// (2)

p->link = head;// 헤드 포인터의 값을 복사//(3)

head = p;// 헤드 포인터 변경//(4)

return head;

}

void print\_list(ListNode\* head)

{

for (ListNode\* p = head; p != NULL; p = p->link)

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

printf("NULL \n");

}

ListNode\* search\_list(ListNode\* head, element x)

{

ListNode\* p = head;

while (p != NULL) {

if (p->data == x) return p;

p = p->link;

}

return NULL;// 탐색 실패

}

// 테스트 프로그램

int main(void)

{

ListNode\* head = NULL;

head = insert\_first(head, 10);

print\_list(head);

head = insert\_first(head, 20);

print\_list(head);

head = insert\_first(head, 30);

print\_list(head);

if (search\_list(head, 30) != NULL)

printf("리스트에서 30을 찾았습니다. \n");

else

printf("리스트에서 30을 찾지 못했습니다. \n");

return 0;

}