**이중연결리스트로 구현한 전화번호부**

**학과 컴퓨터공학과**

**학번 201821405**

**성명 장경원**

**제출일 10/3**

|  |
| --- |
| **Ⅰ. 서론**  **1. 요구분석 및 계획**  **Ⅱ. 본론**  **1. Flowchart 설명**  **2. Flowchart 안의 기능들 설명**  **3. 구현한 코드와 함수 설명**  **Ⅲ. 결론**  **1. 프로그램 실행과정 및 결과**  **2. 결과분석** |

**목차**

|  |
| --- |
| **Ⅰ. 서론** |

**1. 요구분석 및 계획**

**▶ 큰 틀**

전화번호부를 설계할 때 현재 우리가 쓰고 있는 스마트폰에 있는 전화번호부를 참고하여 설계하였고, 그 안에 있는 기능들 또는 그 안에서 다루는 데이터를 저장할 수 있게 했다. 동명이인이나 연락처를 출력했을 때 우선순위(가나다순)에 맞게 출력될 수 있게 했다.

**▶ 저장방식**

이전 학기에 들은 데이터구조라는 수업에서 배열로 저장할 수도 있고, 리스트로 저장할 수도 있다는 걸 배웠다. 그 중에서 리스트를 선택하였고 단순연결리스트보단 이중연결리스트가 주로 쓰인다고 배웠기 또, 이중연결리스트로 구현하면서 포인터 및 동적할당에 대한 개념을 더 확실히 하고 싶었기 때문에 이중연결리스트로 전화번호부의 저장 및 관리를 하기로 했다.

**▶ 기능**

기능은 **메뉴**, **검색**, **출력**, **추가**, **삭제**, **수정**, **종료**, **종료 or 메뉴**로 총 8가지가 있다. 그럼 지금부터 각 기능에 대해 설명하도록 하겠다. **메뉴**, 메뉴라는 기능은 처음 프로그램이 시작되면 실행이 되고, 종료 or 메뉴 기능에서도 쓰이는 기능이다. 메뉴가 실행이 되면 화면에 메뉴를 텍스트로 띄워주고, 숫자를 입력 받아 각 숫자에 해당하는 기능을 수행하도록 한다. **검색**, 검색이라는 기능은 데이터를 입력 받고 해당 데이터를 통해 전화번호부 안에 그 데이터를 가진 항목이 있는지 확인하여 있으면 항목의 주소 값을 리턴하여 그 주소 값 안에 있는 데이터를 출력해주는 기능이다. 스마트폰에서 연락처에 사람이름을 검색하면 바로 뜨는 기능과 같다. **출력**, 출력이라는 기능은 일단 가나다 순으로 출력이 될 수 있도록 버블소팅을 한 뒤에 해당 값을 출력하는 기능이다. 본론 구현 단계에서 자세히 설명하도록 하겠다. **추가**, 추가라는 기능은 데이터를 저장할 공간(노드)를 할당해주고, 그 안에 입력 받은 데이터값을 넣은 후 이중연결리스트에 연결하는 기능이다. **삭제와 수정**, 삭제와 수정이란 기능은 모두 검색 기능을 사용한후에 이루어진다. 데이터를 통해 검색을 하고 해당 데이터가 있는 주소 값을 받아 삭제 또는 수정을 할 수 있는 기능이다. 마지막으로 **종료 or 메뉴**, 종료 or 메뉴의 기능은 메뉴로 간다 종료한다를 고르는 기능이다.

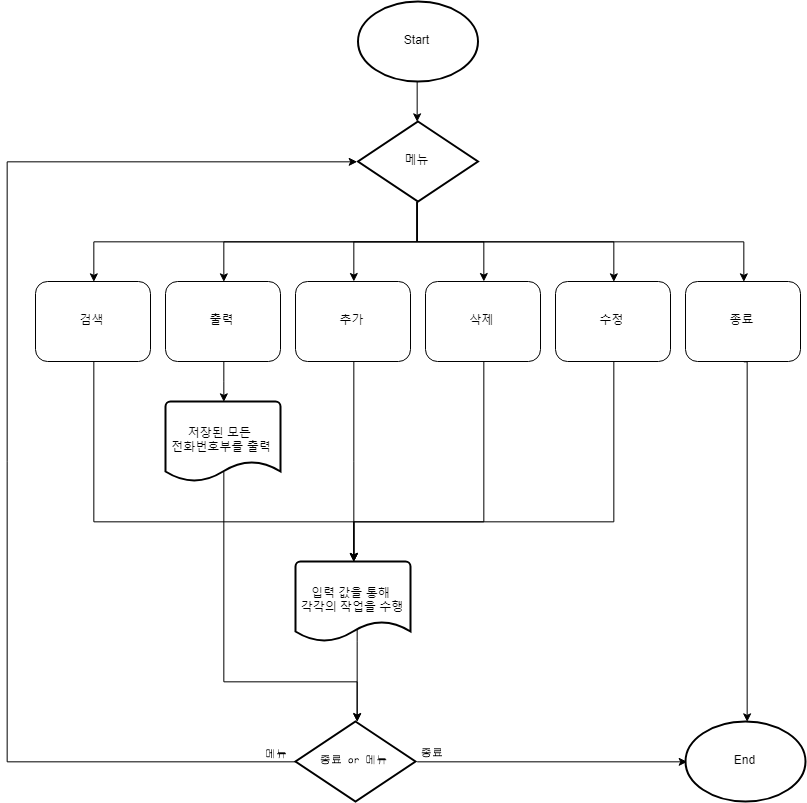
**▶ 각 기능의 필요성과 타당성**

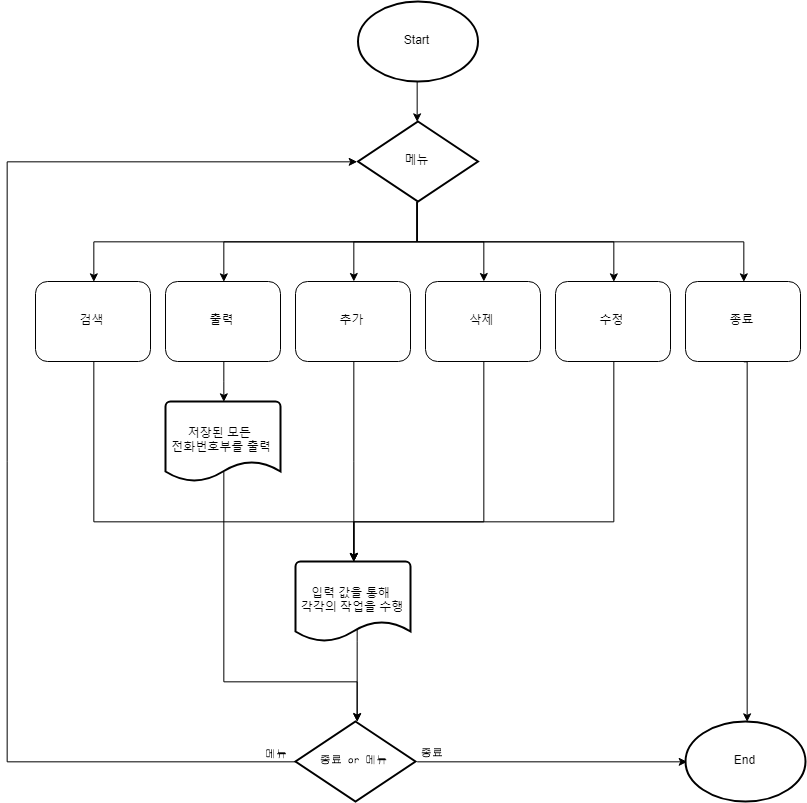
간단하게 말하자면, **메뉴**는 처음 우리가 선택을 할 것에 대한 선택지를 제공하는 것이기에 필요하고, **검색**은 당연히 저장만 되고, 검색이 안된다면 엄청난 데이터를 일일이 읽고 찾아봐야 하지만 검색이란 기능 하나로 한번에 원하는 값을 추출할 수 있기 때문이다. **출력**은 연락처의 데이터는 메모리 상에 존재하기 때문에 저장한 값들을 보기 위해선 출력을 해야 한다. **추가**, **삭제**, **수정**은 데이터를 다루기 위해 있어야하는 기본적인 기능들이다. **종료 or 메뉴**는 사용자들이 하나의 작업을 수행하고 나서 선택지를 주기 위함이다.

|  |
| --- |
| **Ⅱ. 본론** |

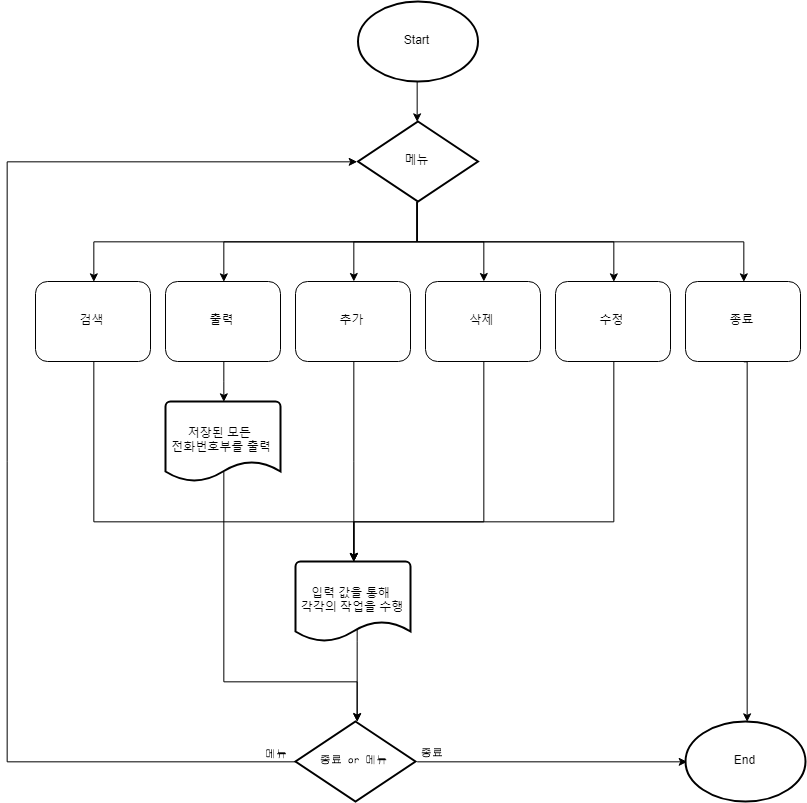
**1. Flowchart 설명**

**▶ 전체적인 Flowchart**

****



▷ 시작이 되고 메뉴에서 각 기능들 중 하나를 선택한다.

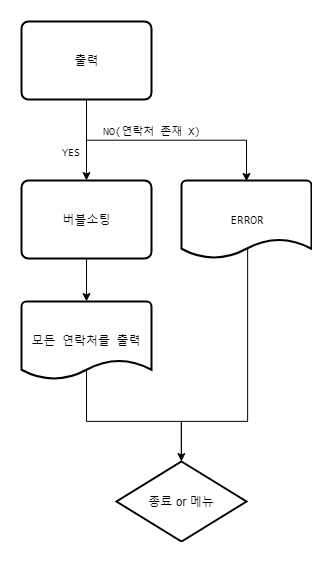


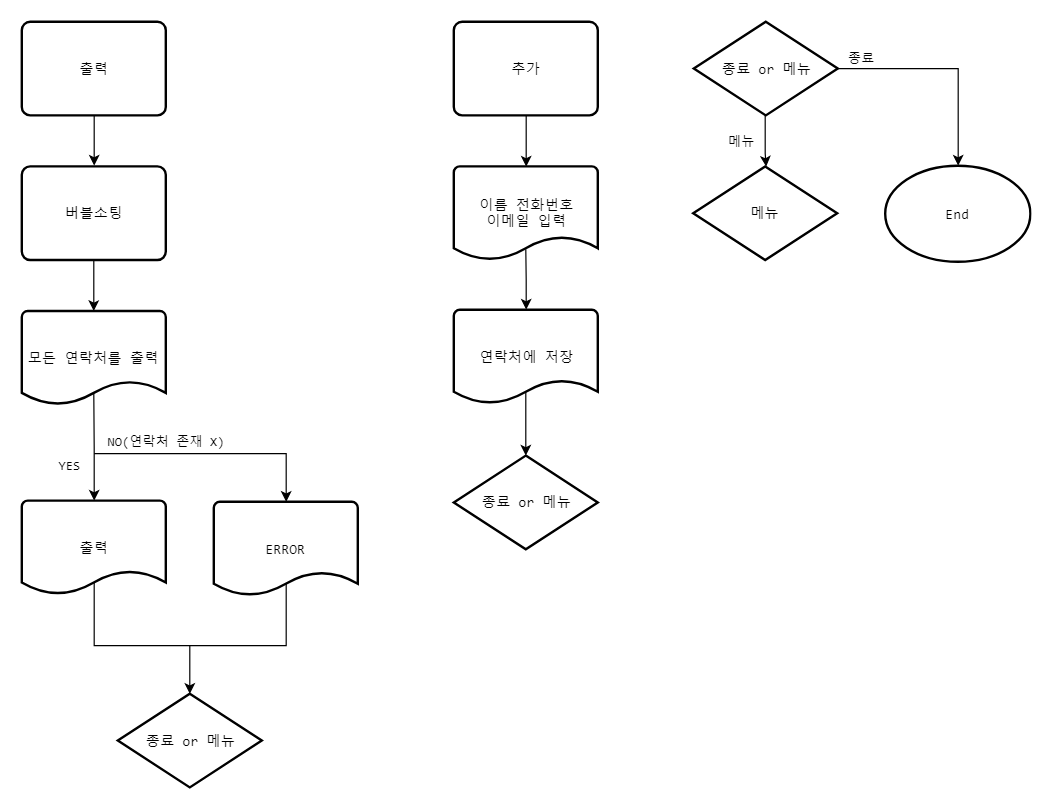
▷ 각 기능에 맞는 작업을 수행하고 종료 or 메뉴 기능으로 가 종료 또는 메뉴로 간다.

**2. Flowchart안의 기능들**

**▶ 각 기능을 구체화하여 설명**

▷출력

****출력 기능은 실행되자 마자 이중연결리스트가 비어있는지 아닌지를 판별하고 비어있으면 ERROR 메시지를 띄우고 안 비어있으면 배열의 주소 값을 받는 포인터 변수를 생성하고, 그곳에 버블소팅이 된 배열의 주소값을 넘겨준 후 이중연결리스트의 노드의 수만큼 반복하여 모든 연락처를 출력합니다.

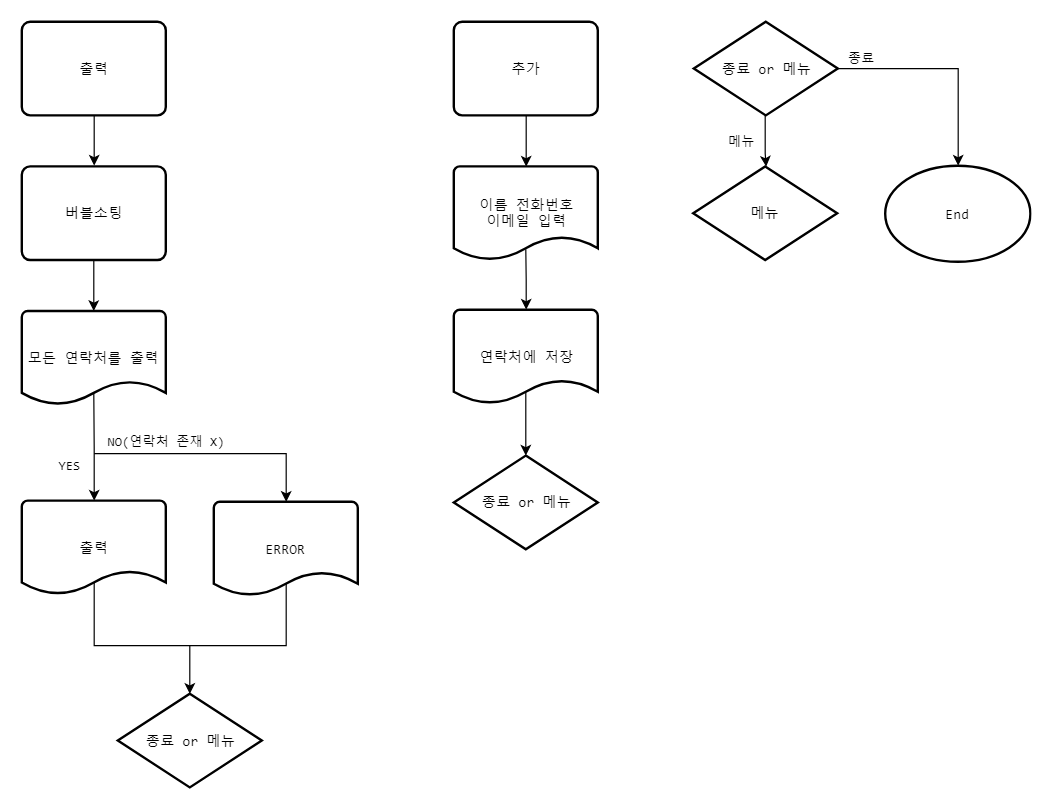
**** ▷ 추가

**추가 기능** 실행 시 노드를 동적할당 하고

그 노드의 데이터에 입력 받은 값을 저장하고,

그 노드를 이중연결리스트에 연결합니다.

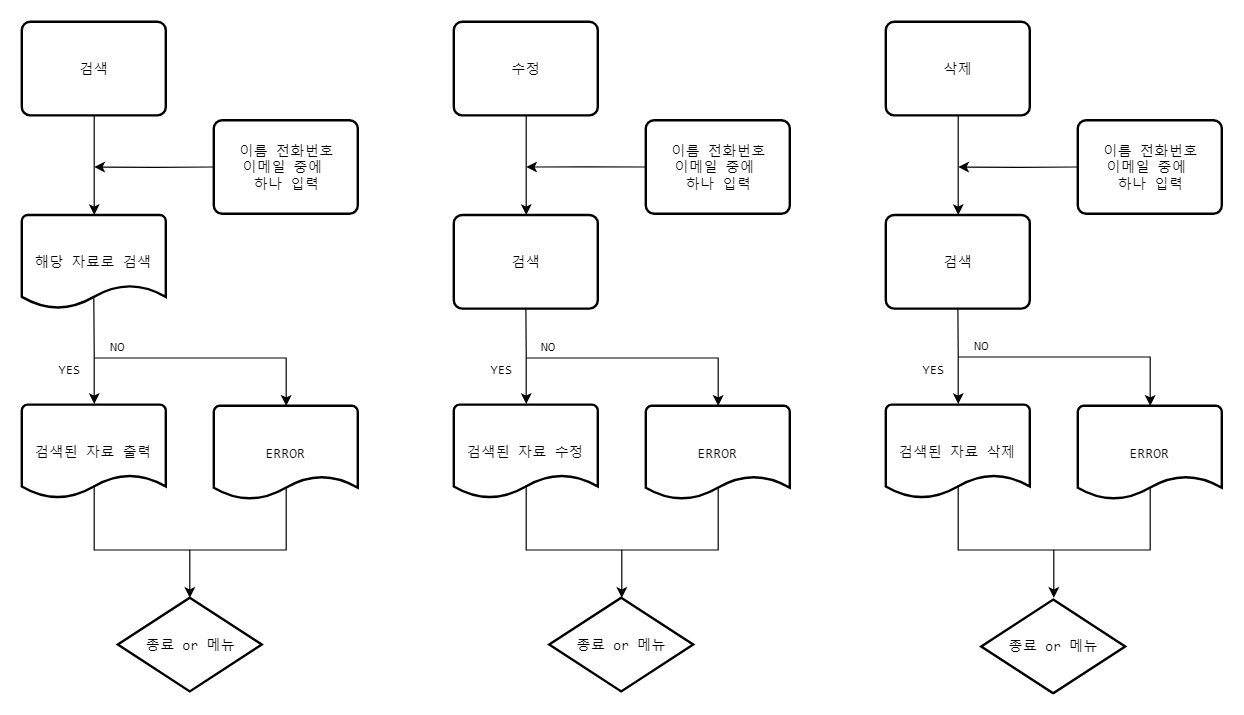
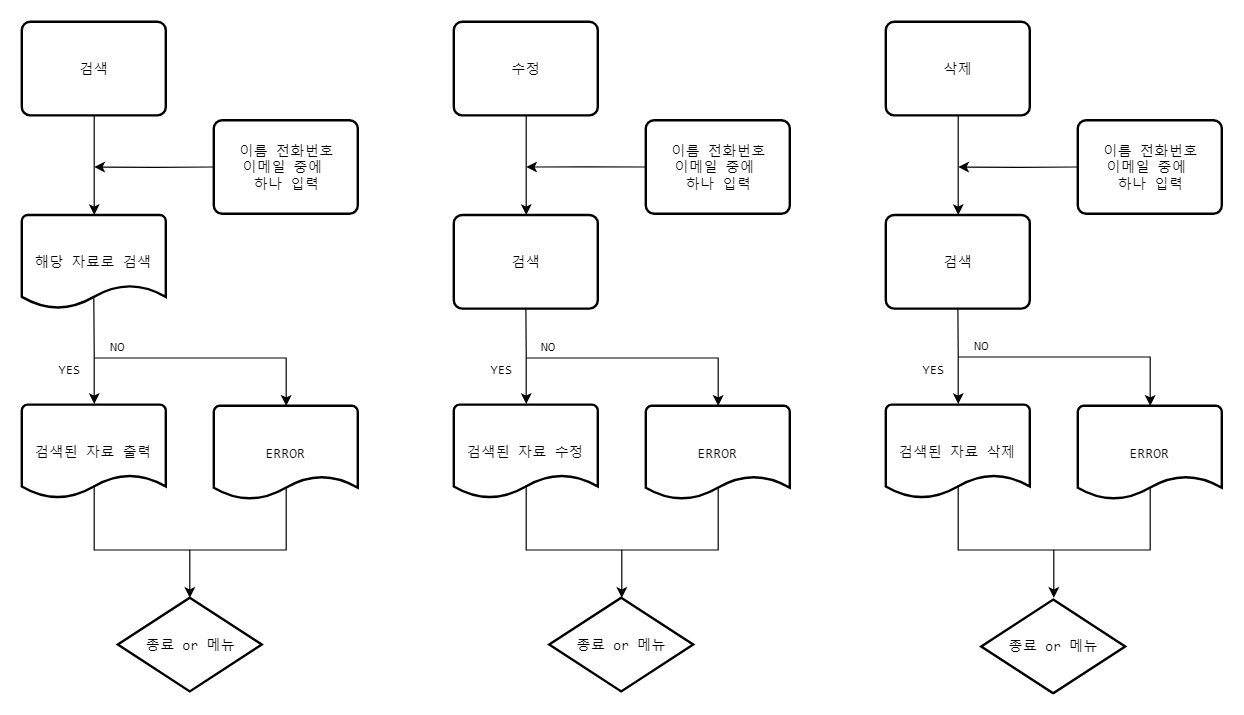
▷ 종료 or 메뉴

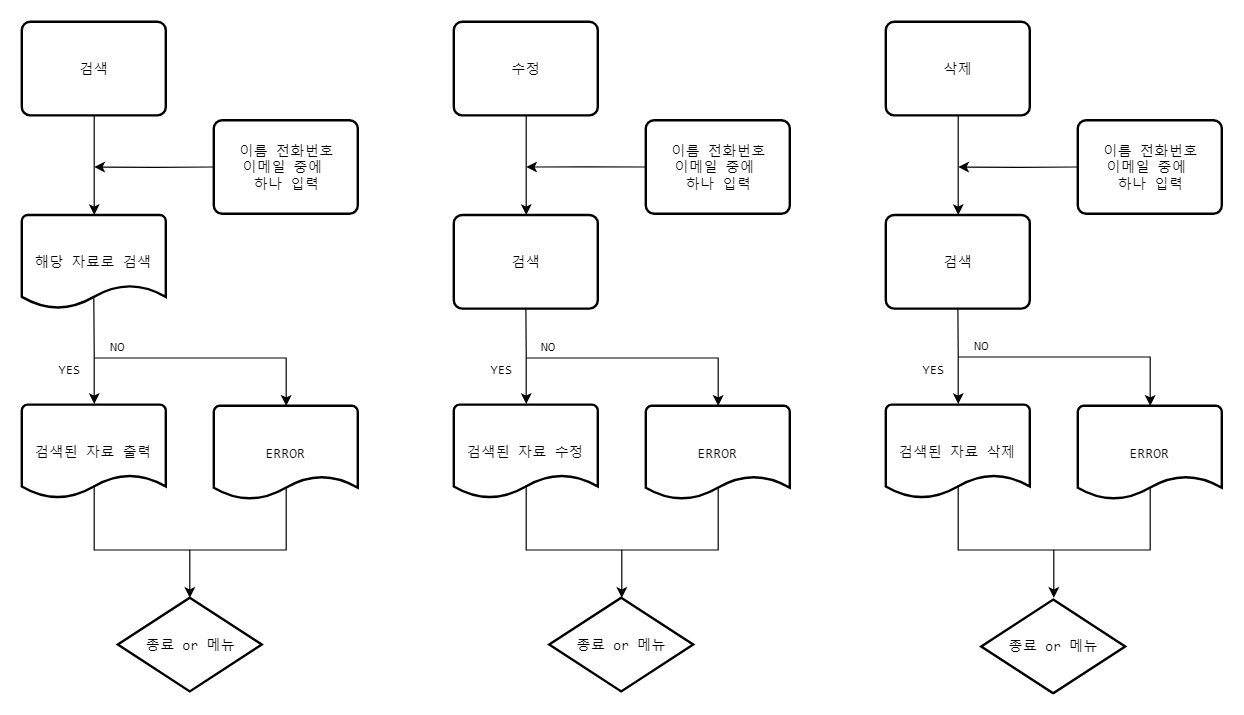
**종료** **또는** **메뉴**를 가리키는

번호를 받아 메뉴 기능을 실행

하거나 종료합니다.

▷ 검색, 수정, 삭제



일단 **검색기능**부터 보면 이름 전화번호 이메일 중 하나를 입력한 뒤에 그 값을 토대로 이중연결리스트에서 검색을 합니다. 그 후에 찾으면 찾은 노드의 값을 리턴하고, 아닐 시 널값을 리턴하도록 되어있습니다.

**수정과 삭제기능**은 검색 기능을 사용하여 만들었습니다. 검색 후에 나온 노드의 주소 값을 통해 수정을 하거나 삭제할 수 있습니다.

**3. 구현한 코드와 함수 설명**

**▶ phoneInfo.h**

글꼴: D2Coding ligature

#pragma once

#include <stdio.h> // 기본 함수들

#include <stdlib.h> // 동적메모리 관리를 위한 함수들 malloc, free

#include <stdbool.h> // bool자료형을 사용하기 위한 헤더파일

#include <string.h> // strcmp함수를 사용하기 위한 헤더파일

#define NAME\_SIZE 10 // NODE 안의 데이터 name문자열의 크기

#define PHONE\_SIZE 12 // NODE 안의 데이터 phoneNum문자열의 크기

#define EMAIL\_SIZE 50 // NODE 안의 데이터 email문자열의 크기

typedef struct node { // 노드의 구조체

struct node \*llink;

struct node \*rlink;

char n\_name[NAME\_SIZE];

char n\_phoneNum[PHONE\_SIZE];

char n\_email[EMAIL\_SIZE];

}NODE;

typedef struct dLinkedList { // 이중연결리스트의 접근 및 관리를 위한 구조체

NODE \*head;

NODE \*tail;

int count;

}DLINKEDLIST;

typedef struct sortArray { // 버블소팅을 위한 구조체

char name[NAME\_SIZE];

NODE \*sData;

}SA;

void dlInit(DLINKEDLIST \*list); // 이중연결리스트 초기화

bool dlIsEmpty(DLINKEDLIST \*list); // 이중연결리스트가 비어있는가

int dlCount(DLINKEDLIST \*list); // dlCount count값을 리턴하는 함수

SA \*dlSort(DLINKEDLIST \*list); // 버블소팅을 하기 위한 함수

void dlPrint(DLINKEDLIST \*list, SA \*Array); // 이중연결리스트를 출력하는 함수

NODE\* dlSearchData(DLINKEDLIST \*list, char \*data); // 검색을 하여 노드의 주소값을 반환하는 함수

void dlInsertLast(DLINKEDLIST \*list, NODE \*dataNode); // 이중연결리스트에 입력된 자료가 있는 노드를 링크하는 함수

void dlDelete(DLINKEDLIST \*list, NODE \*search); // 입력된 노드의 주소값을 링크를 풀고 메모리해제하는 함수

**▶ phoneInfo.c**

#include "phoneInfo.h"

void dlInit(DLINKEDLIST \*list) { // 이중연결리스트 초기화

list->head = (NODE\*)malloc(sizeof(NODE));

list->tail = (NODE\*)malloc(sizeof(NODE));

list->head->rlink = list->head->llink = NULL;

list->tail->rlink = list->tail->llink = NULL;

list->count = 0;

}

bool dlIsEmpty(DLINKEDLIST \*list) { // 비어있느냐 원형이중연결리스트가 아니기 때문에 링크의 핵심은 head->rlink와 tail->llink + count는 추가 삭제시 +-가 이루어짐

if (list->head->rlink == NULL && list->tail->llink == NULL && list->count == 0)

return true;

else

return false;

}

int dlCount(DLINKEDLIST \*list) { // 혹시나 이중연결리스트의 노드의 갯수를 알기 위한 함수

return list->count;

}

SA \*dlSort(DLINKEDLIST \*list) { // 포인터 배열을 생성해 배열로 버블소팅을 하고 해당 배열의 주소값을 반환

int dCount = dlCount(list);

SA \*sArray = (SA\*)malloc(sizeof(SA)\*dCount); // 동적할당을 이중연결리스트의 노드의 갯수만큼

SA temp; // 버블소팅시 데이터를 이동시키기 위한 변수

temp.sData = (NODE\*)malloc(sizeof(NODE)); // 할당이 안되어있기 때문에 동적할당

NODE \*sort = (NODE\*)malloc(sizeof(NODE)); // 이중연결리스트를 탐색하기 위한 sort노드

sort = list->head->rlink;

for (int i = 0; i < dCount; i++) // 이중연결리스트를 처음부터 끝까지 가면서 해당 데이터를 저장

{

strcpy\_s(sArray[i].name, sizeof(sArray[i].name), sort->n\_name); //이름으로 우선순위를 비교하기때문에 이름을 배열안의 이름데이터에 복사함

sArray[i].sData = (NODE\*)malloc(sizeof(NODE));

sArray[i].sData = sort; // 그리고 이름에 맞는 노드의 주소값을 저장해놓음

sort = sort->rlink;

}

sort = NULL; // 검색의 사용한 노드 메모리 해제

free(sort);

for (int j = 0; j < dCount - 1; j++) {

for (int k = 0; k < dCount - 1; k++) {

if (strcmp(sArray[k].name, sArray[k + 1].name) > 0) { // 버블소팅을 통해 우선순위대로 나열함(앞의 것이 아스키코드값이 크다, 앞의 것이 우선순위가 낮다 뒤로)

temp = sArray[k];

sArray[k] = sArray[k + 1];

sArray[k + 1] = temp;

}

}

}

return sArray; // 배열의 주소값 리턴

}

void dlPrint(DLINKEDLIST \*list, SA \*array) { // 버블소팅을 하고 나서 해당 배열의 주소값을 가져와 오름차순으로 출력

int l;

if (dlIsEmpty(list) == true)

printf("전화번호부에 연락처가 존재하지 않습니다.\n");

else {

for (l = 0; l < list->count; l++) { // 출력

if (l == 0) {

printf("\n");

}

printf("이름: %s || 전화번호: %s || 이메일: %s\n", array[l].sData->n\_name, array[l].sData->n\_phoneNum, array[l].sData->n\_email);

}

}

}

NODE\* dlSearchData(DLINKEDLIST \*list, char \*data) { // 데이터를 통해 노드를 탐색하고 찾을 경우 해당 노드의 주소값을 리턴

if (dlIsEmpty(list) == true) {

printf("전화번호부에 연락처가 존재하지 않습니다.\n");

return NULL;

}

else {

NODE \*search = (NODE\*)malloc(sizeof(NODE)); // 검색된 노드의 주소를 저장할 search노드 동적할당

bool searchEd = false; // 검색이 됐는지 안됐는지 확인하는 bool값

search = list->head->rlink;

while (search != list->tail) { // tail 전까지만 데이터 비교

if (strcmp(data, search->n\_name) == 0 || strcmp(data, search->n\_phoneNum) == 0 || strcmp(data, search->n\_email) == 0) { // 데이터 비교하여 같으면

searchEd = true; // 검색표시하고

return search; // 해당 노드의 주소값 리턴

}

search = search->rlink;

}

if (searchEd == false) // 검색이 안됨

{

printf("해당 자료를 가진 연락처는 없습니다.\n");

return NULL;

}

}

return NULL;

}

void dlInsertLast(DLINKEDLIST \*list, NODE \*dataNode) { // 데이터가 입력된 노드를 이중연결리스트에 뒤쪽으로 삽입

if (dlIsEmpty(list) == true) { // 비어있을 경우

list->head->rlink = dataNode; // head와

dataNode->llink = list->head;

dataNode->rlink = list->tail;

list->tail->llink = dataNode; // tail에 링크

list->tail->rlink = list->head;

list->head->llink = list->tail;

list->count += 1;

printf("입력한 연락처가 저장되었습니다.\n");

}

else

{

list->tail->llink->rlink = dataNode; // 앞부분

dataNode->llink = list->tail->llink; // 연결

dataNode->rlink = list->tail;

list->tail->llink = dataNode;

list->count += 1;

printf("입력한 연락처가 저장되었습니다.\n");

}

}

void dlDelete(DLINKEDLIST \*list, NODE \*search) { // 입력된 노드의 주소값의 메모리 해제

NODE \*remove = (NODE\*)malloc(sizeof(NODE));

if (list->count == 1) { // 하나만 있을 경우, 원형 이중연결리스트가 아니기에 절단

list->head->rlink = NULL;

list->tail->llink = NULL;

list->count -= 1;

}

else { // 다른 경우는 삭제된 노드의 앞 노드와 뒷 노드를 연결

remove = search;

remove->llink->rlink = remove->rlink;

remove->rlink->llink = remove->llink;

list->count -= 1;

}

free(remove);

printf("해당 자료를 가진 연락처가 삭제되었습니다.\n");

}

**▶ func.h**

#pragma once // #ifndef \_\_헤더파일이름(대문자)\_H\_ #endif

#include <stdio.h>

#include <windows.h> // 창사이즈 조절되게 하는 헤더파일

#include "phoneInfo.h" // 이중연결리스트 구현하기 위한 함수

void menu(DLINKEDLIST \*DLL); //메뉴

void endOrMenu(DLINKEDLIST \*DLL); // 종료

void printPhoneBook(DLINKEDLIST \*DLL); //출력

void searchInfo(DLINKEDLIST \*DLL); //검색

void insertInfo(DLINKEDLIST \*DLL); // 추가

void modifyInfo(DLINKEDLIST \*DLL); // 수정

void deleteInfo(DLINKEDLIST \*DLL); // 삭제

**▶ func.c**

#include "func.h"

void menu(DLINKEDLIST \*DLL) {

system("mode con cols=40 lines=30"); // cols=칸/행(가로) lines=줄/열(세로)

int num = 0;

bool answer = false;

while (answer != true) {

printf("PhoneBook implemented with C Language\n\n다음 중 수행할 작업을 고르시오.\n\n1. 출력 2. 검색 3. 추가\n4. 삭제 5. 수정 6. 종료\n\n>> ");

scanf\_s("%d", &num);

while (getchar() != '\n'); // 버퍼를 비움

switch (num) {

case 1: printPhoneBook(DLL); answer = true; break; // 출력함수

case 2: searchInfo(DLL); answer = true; break; // 검색함수

case 3: insertInfo(DLL); answer = true; break; // 삽입함수

case 4: deleteInfo(DLL); answer = true; break; // 삭제함수

case 5: modifyInfo(DLL); answer = true; break; // 수정함수

case 6: answer = true; return; // 종료

default: printf("========================================\n"); printf("그런 작업은 없습니다.\n"); printf("========================================\n"); break;

}

}

}

void endOrMenu(DLINKEDLIST \*DLL) { // 메뉴 or 종료

int eNum = 0;

bool eAnswer = false;

while (eAnswer != true) {

printf("========================================\n");

printf("\n다음 중 수행할 작업을 고르시오.\n\n1. 메뉴 2. 종료\n\n>> ");

scanf\_s("%d", &eNum);

while (getchar() != '\n');

switch (eNum) {

case 1: menu(DLL); eAnswer = true; break;

case 2: eAnswer = true; return;

default: printf("========================================\n"); printf("그런 작업은 없습니다.\n"); break;

}

}

}

void printPhoneBook(DLINKEDLIST \*DLL) {

system("mode con cols=40 lines=30");

printf("===============전화번호부===============\n");

dlPrint(DLL, dlSort(DLL)); // 정렬된 배열의 주소를 받아 이중연결리스트의 노드의 갯수만큼 출력하는 함수

endOrMenu(DLL);

}

void searchInfo(DLINKEDLIST \*DLL) {

system("mode con cols=40 lines=30");

char searchData[50]; // 찾는 데이터 값을 받기 위한 변수

printf("검색할 이름 또는 전화번호 또는 이메일 중 하나를 입력하시오.\n\n>> ");

scanf\_s("%s", searchData, sizeof(searchData));

if (dlSearchData(DLL, searchData) != NULL) {

printf("========================================\n");

NODE \*sNode = (NODE\*)malloc(sizeof(NODE));

sNode = dlSearchData(DLL, searchData);

printf("<<검색된 연락처의 정보>>\n이름: %s || 전화번호: %s || 이메일: %s\n", sNode->n\_name, sNode->n\_phoneNum, sNode->n\_email);

}

endOrMenu(DLL);

}

void insertInfo(DLINKEDLIST \*DLL) {

system("mode con cols=40 lines=30");

NODE \*iNode = (NODE\*)malloc(sizeof(NODE));

printf("추가할 이름을 입력하시오.\n\n>> ");

scanf\_s("%s", iNode->n\_name, sizeof(iNode->n\_name));

printf("추가할 전화번호를 입력하시오.\n\n>> ");

scanf\_s("%s", iNode->n\_phoneNum, sizeof(iNode->n\_phoneNum));

printf("추가할 이메일을 입력하시오.\n\n>> ");

scanf\_s("%s", iNode->n\_email, sizeof(iNode->n\_email));

dlInsertLast(DLL, iNode);

endOrMenu(DLL);

}

void modifyInfo(DLINKEDLIST \*DLL) {

system("mode con cols=40 lines=30");

char mSearchData[50];

printf("수정할 이름 또는 전화번호 또는 이메일 중 하나를 입력하시오.\n\n>> ");

scanf\_s("%s", mSearchData, sizeof(mSearchData));

if (dlSearchData(DLL, mSearchData) != NULL) {

printf("========================================\n");

NODE \*mNode = (NODE\*)malloc(sizeof(NODE));

mNode = dlSearchData(DLL, mSearchData);

printf("<<수정할 연락처의 정보>>\n이름: %s || 전화번호: %s || 이메일: %s\n", mNode->n\_name, mNode->n\_phoneNum, mNode->n\_email);

printf("========================================\n");

printf("바꿀 이름을 입력하시오.\n\n>> ");

scanf\_s("%s", mNode->n\_name, sizeof(mNode->n\_name));

printf("바꿀 전화번호를 입력하시오.\n\n>> ");

scanf\_s("%s", mNode->n\_phoneNum, sizeof(mNode->n\_phoneNum));

printf("바꿀 이메일을 입력하시오.\n\n>> ");

scanf\_s("%s", mNode->n\_email, sizeof(mNode->n\_email));

printf("해당 자료로 연락처가 수정되었습니다.\n");

}

endOrMenu(DLL);

}

void deleteInfo(DLINKEDLIST \*DLL) {

system("mode con cols=40 lines=30");

char dSearchData[50];

printf("삭제할 이름 또는 전화번호 또는 이메일 중 하나를 입력하시오.\n\n>> ");

scanf\_s("%s", dSearchData, sizeof(dSearchData));

if (dlSearchData(DLL, dSearchData) != NULL) {

printf("========================================\n");

NODE \*dNode = (NODE\*)malloc(sizeof(NODE));

dNode = dlSearchData(DLL, dSearchData);

printf("<<삭제할 연락처의 정보>>\n이름: %s || 전화번호: %s || 이메일: %s\n", dNode->n\_name, dNode->n\_phoneNum, dNode->n\_email);

dlDelete(DLL, dNode);

}

endOrMenu(DLL);

}

**▶ main.c**

#include "func.h"

#include "phoneInfo.h"

int main(void) {

DLINKEDLIST \*DLL = (DLINKEDLIST\*)malloc(sizeof(DLINKEDLIST));

dlInit(DLL);

menu(DLL);

}

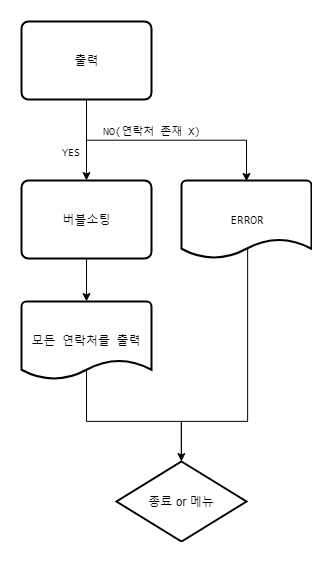
**▶ 주요 함수들과 Flowchart의 해당 부분명시**

▷ 출력

**① func.c의 printPhoneBook**

**② phoneInfo.c의 dlSort**

**③ phoneInfo.c의 dlPrint**

****

**③②**

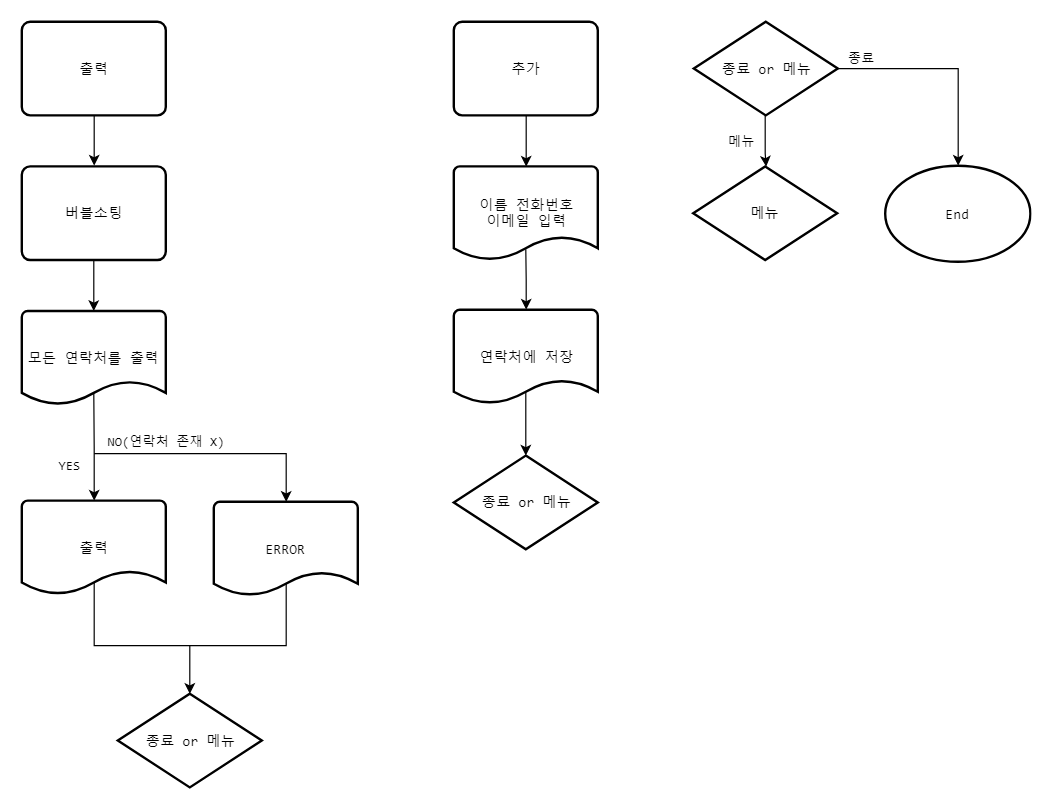
**②**

**①**

▷ 추가

**①** phoneInfo.c의 dlInsertLast

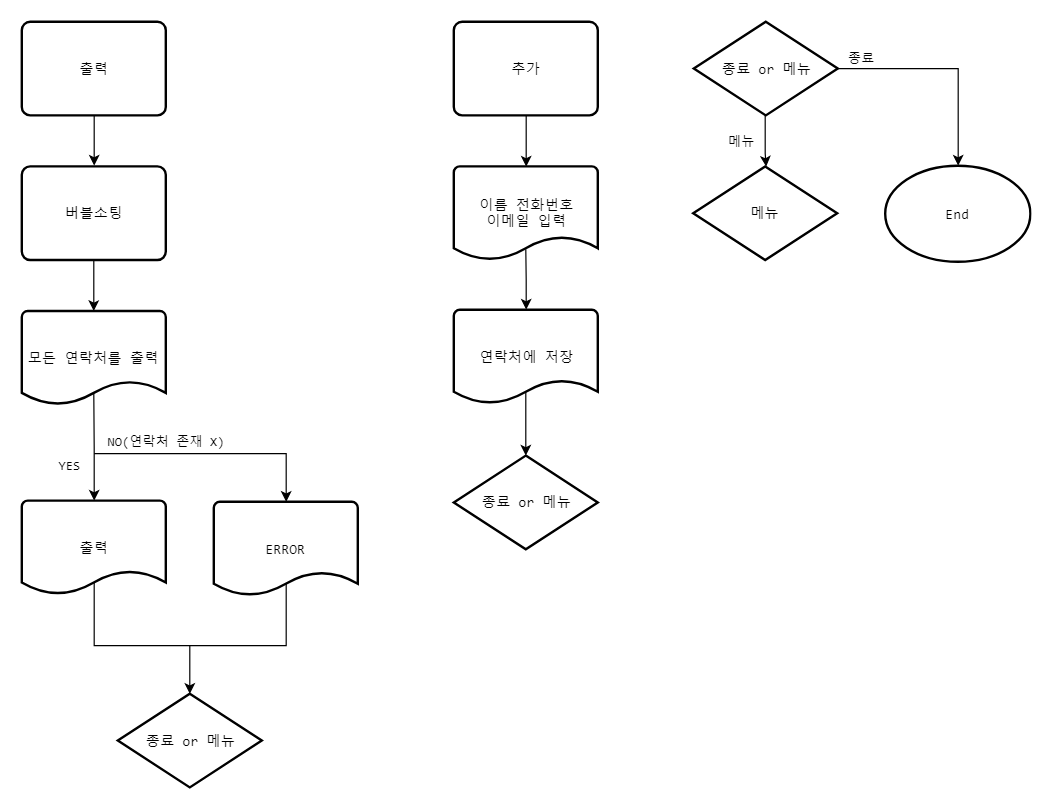
**②** func.c의 insertInfo

****

**②**

**①**

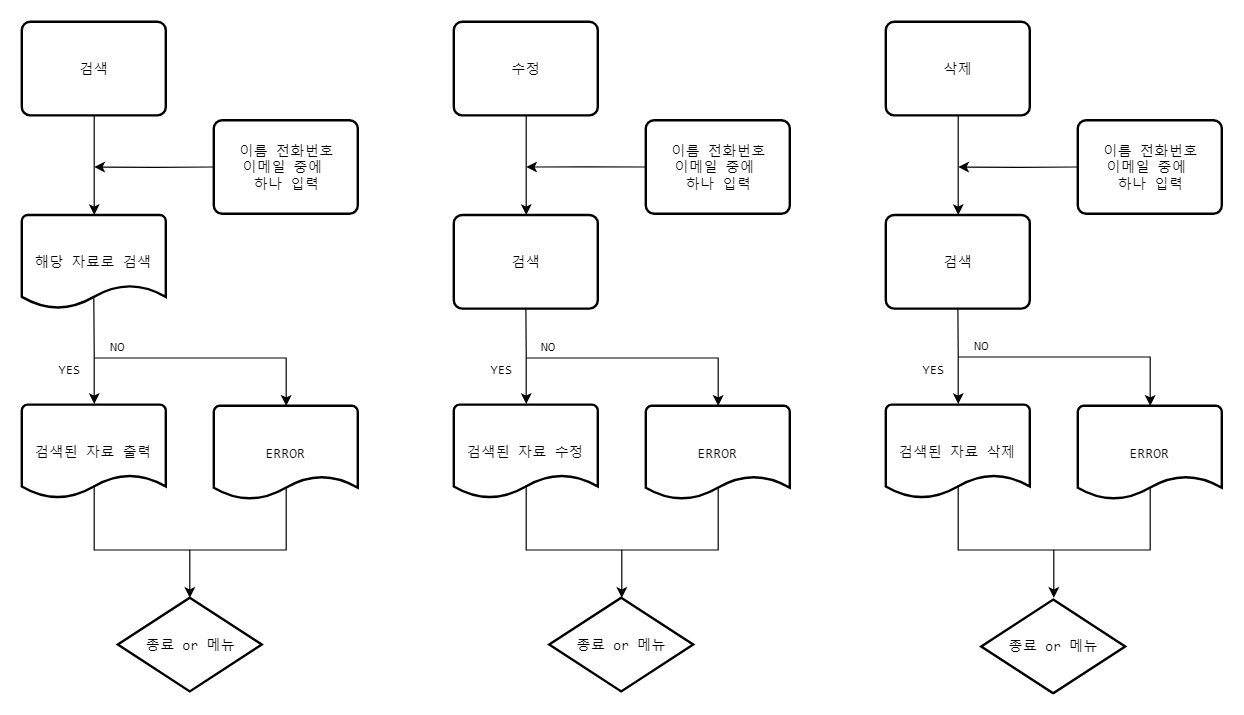
▷ 종료 or 메뉴

 **① phoneInfo.c의 endOrMenu**

**①**

**② phoneInfo.c의 menu**

**②**



▷ 검색, 수정, 삭제

**①**

**①** func.c의 searchInfo

**②** phoneInfo.c의 dlSearchData

**②**

**③** fun.c의 modifyInfo

**④** func.c의 deleteInfo

**⑤** phoneInfo.c의 dlDelete

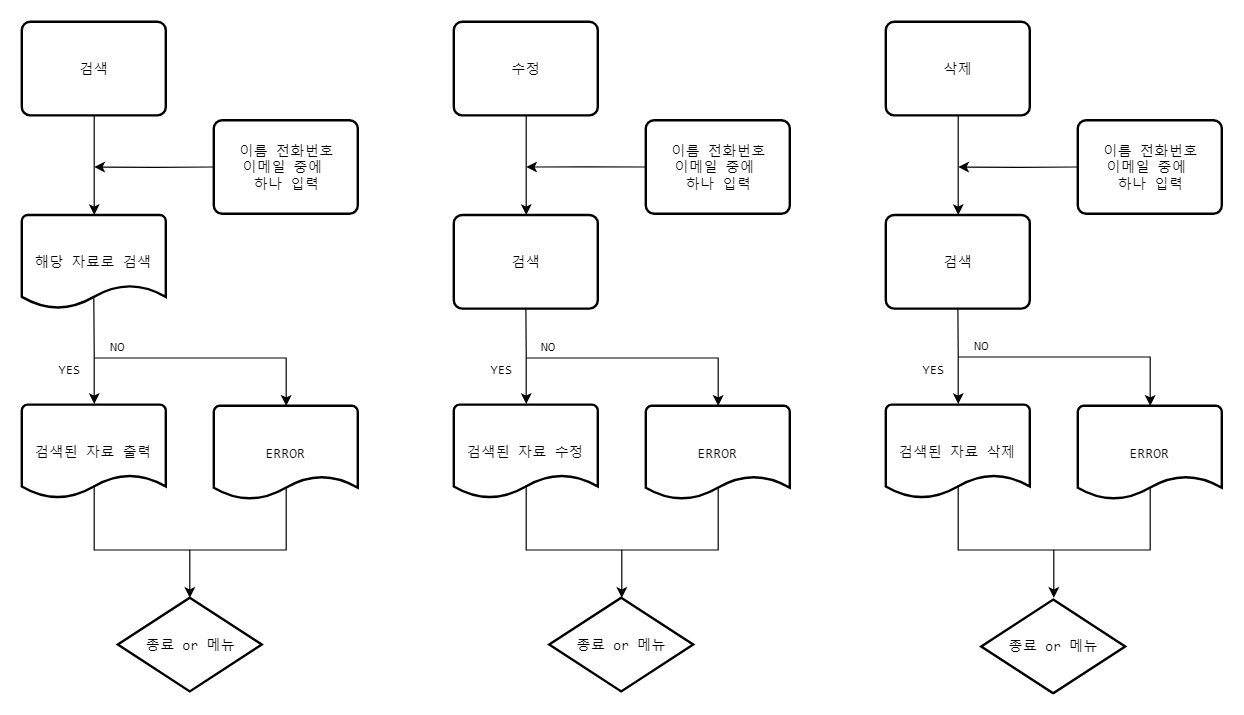
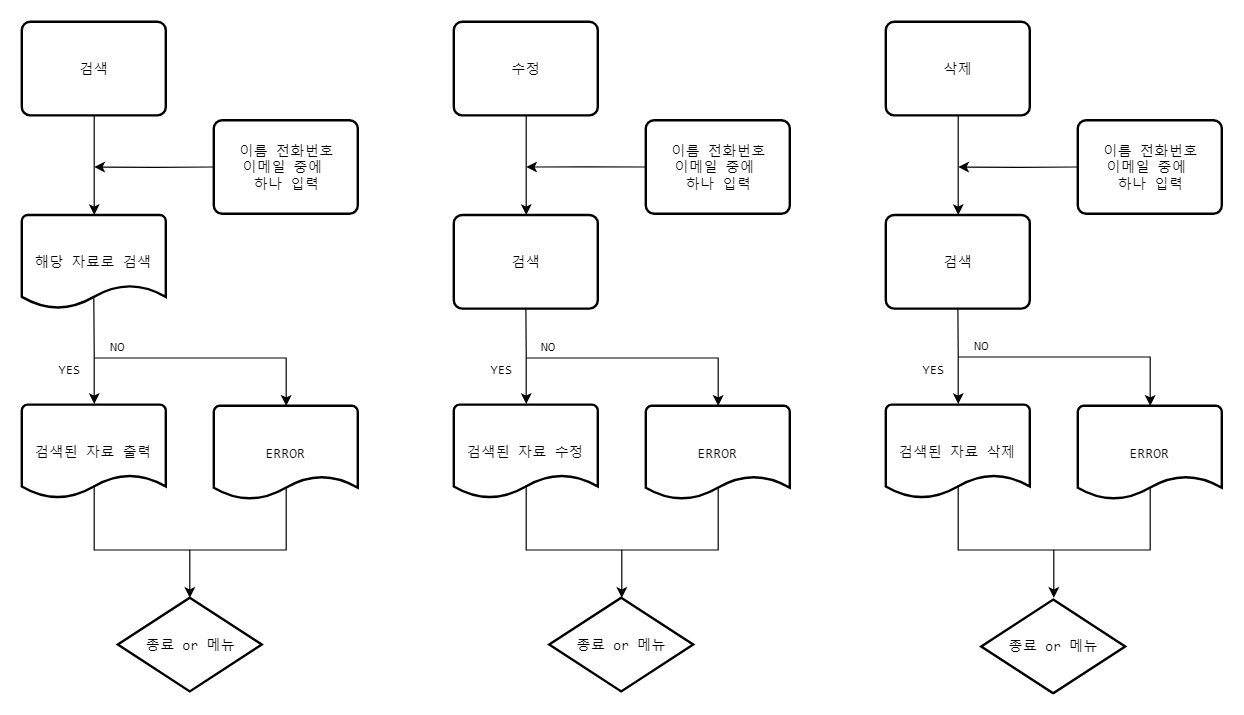
**①**

**①**

**⑤**

**④**

**③**



|  |
| --- |
| **Ⅲ. 결론** |

**1. 프로그램 실행과정 및 결과**

|  |  |
| --- | --- |
| 메뉴 | 출력 |
|  |  |
| 추가 | 검색 |
|  |  |

|  |  |
| --- | --- |
| 삭제 | 수정 |
|  |  |
| 우선순위 확인용 출력 | 우선순위 확인용 추가 |
|  |  |

|  |  |
| --- | --- |
| 우선순위 확인용 출력 | 종료 |
|  |  |

**2. 결과분석**

**▶ 요구분석 및 계획에서 이행하지 못한 점**

동명이인을 구현하지 못했습니다. 대략적인 논리는 만들 수 있었지만 그것을 구현

하기 위해선 너무 많은 변수를 만들어야 했습니다. 논리는 검색기능에서 바로 리턴

값을 주는게 아니라 일부러 쭉 돌리면서 따로 카운트 변수를 만들어 놓고 이름이

같은 사람이 있으면 카운터 변수 +1을 해주어 총 동명이인 수를 안 다음 노드형의

포인터 배열을 만들고, 다시 처음부터 돌리면서 이름이 같을 때 주소 값을 받아

노드형의 포인트 배열에 넣어주어 해당 포인터 배열을 리턴해줘야 합니다. 나머지는

요구분석 및 계획에서 설계한대로 구현하였습니다.

**▶ 구현을 하면서 힘들었던 점**

▷ 이중연결리스트를 구현할 때 처음에 DLINKEDLIST를 선언하고 초기화할 때

DLINKEDLIST안에 있는 head 노드와 tail노드에 바로 NULL을 대입하여 문제가 있었

습니다. 이 문제는 메모리공간이 잡힌 주소 값을 할당하지 않고 NULL값을 대입

하려고 했기에 문제가 있었고 head노드와 tail노드에 동적할당을 하고 초기화를 하여

이 문제를 해결하였습니다.

▷ 버블소팅을 구현할 때 노드 안의 데이터를 비교하고 노드의 링크를 바꾸면서

버블소팅을 구현 할려고 했다가 너무 복잡하고 프로그램이 꼬여 생각을 하고 있다가,

같이 공부하던 친구와 방법을 생각하다. 배열을 통해 버블소팅을 하면 되겠다라는

생각을 하게되었습니다. 어차피 출력할때만 버블소팅된 연락처를 보여주면 되기때문

이였죠. 그리하여 dlSort함수를 재구현하게 되었고, 리턴값을 SA라는 구조체의 주소

값을 리턴하게 되었습니다. 이 함수는 dlPrint함수의 매개변수로 쓰이면서

printPhoneBook함수를 완성하게 됩니다.

**▶ 소감**

설계부터 구현까지 하면서 든 생각은 하나의 프로그램을 효과적이고 빠르게 간결

하게 짜기 위해선 일단 flowchart와 같이 프로그램의 구조를 기능별로 간결하게 나눠

보고, 어떤 작업이 이루어지는가부터 시작해서, 점점 기능을 구체화 시켜야 한다고

깨달았습니다. 차근차근 큰 틀부터 함수 하나하나가 될 때 까지 이 프로그램에 필요한

기능이 무엇인가를 고민하면서 한다면 논리적 오류가 없는 좋은 프로그램이 나올 것입

니다. 그리고 여러 방법의 코딩이 가능한 논리라면, 모든 방법을 한번 다 구현해 본다

면 여러 상황에서 그에 맞는 코딩을 할 수 있을 것 같다고 생각했습니다. 마지막으로

설계하기 이전에 설계 대상을 완벽히 파악하고 있다면, 대중성 있고 편리한 프로그램

이 만들어질 것이라고 생각했습니다.