|  |
| --- |
| **HW. 6 구조체 배열**  분반: 061  부산대 정보컴퓨터공학부  2015-24402  강나훈  제출일:2019-05-20 |

**1. 구현 내용에 대한 설명**

**(1) 자료 구조** : 최대 50명의 이름과 전화번호를 저장하는 구조체배열을 phone.h에 선언하고 실제로 저장된 구조체의 수를 int형 변수 size에 저장한다. 구조체 배열에 하나씩 추가 할 때마다 size가 1씩증가하고 구조체를 하나 삭제 할 경우 size가 1 감소한다.

**(2) 주요 함수**

* **register.c : 비밀번호를 입력 후 이름과 전화번호 등록**

전역변수 : **extern int size** : phone.h에 선언된 int형 변수, 이름과 전화번호를 추가할 때 마다 하나씩 증가한다.

**extern struct Contact PhoneBook[Max]** : phone.h에 선언된 최대 50명까지 저장 가능한 이름과 전화 번호를 저장하는 Contact형 구조체 배열

**static int count\_ID=0 :** 입력한 패스워드가 “qwerty1234”가 아닐 경우 1씩증가

지역변수 : **char password[13] :** 입력할 패스워드를 저장하는 char형 배열

**char userName[10] :** 추가할 이름을 저장하는 char형 배열

**char phoneNumber[13] :** 추가할 전화번호를 저장하는 char형 배열

구현방법

void registerPhoneData()

{

char password[13], userName[10], phoneNumber[13];

while(1)

{

printf("Password : ");

scanf("%s",password);

if(strcmp(password,"qwerty1234") == 0) **//입력받은 password와 qwerty1234를 비교**

{

printf("New User Name : "); **//password가 일치할 경우**

scanf("%s", userName); **//이름과 전화번호를 입력받는다.**

printf("PhoneNumber : ");

scanf("%s", phoneNumber);

printf("Registered...\n");

strcpy(PhoneBook[size].Name,userName); **//입력받은 이름과 전화번호를 구조체 배열의**

strcpy(PhoneBook[size].PhoneNumber,phoneNumber); **//size위치에 새로 저장하고**

size++; **//size를 1 증가 시킨다**

return; **//원수의 수가 size라면 0~size-1의 위치를 가지기 때문**

}

else **//비밀번호가 일치하지 않는경우**

find\_id(); **//비밀번호 틀린 횟수 검사**

if( count\_ID == 3) **//3번 일치하지 않는 경우**

{

count\_ID = count\_ID - 3; **//다시 count\_ID를 0으로 만들고 함수 종료**

return;

}

}

}

void find\_id(){

if( count\_ID == 0) **//1회 틀렸을 경우**

{

printf("Not Matched!!\n");

count\_ID++;

}

else if( count\_ID == 1) **//2회 틀렸을 경우**

{

printf("Not Matched!!!(twice)\n");

count\_ID++;

}

else if( count\_ID == 2) **//3회 틀렸을 경우**

{

printf("Not Matched!!!(3 times)\n");

printf("You are not allowed to access phoneBook\n");

count\_ID++;

}

}

* **print.c : 등록된 이름과 전화번호 출력**

전역변수 : **extern int size** : phone.h에 선언된 int형 변수, 이름과 전화번호를 추가할 때 마다 하나씩 증가한다.

**extern struct Contact PhoneBook[Max]** : phone.h에 선언된 최대 50명까지 저장 가능한 이름과 전화 번호를 저장하는 Contact형 구조체 배열

void printAll()

{

printf("<<Display all contacts in the PhoneBook>>\n");

for(int i = 0; i < size; i++) **// 0 ~ size-1 즉, 저장된 구조체 배열 원소 탐색**

{

printf("%s \t %15s\n",PhoneBook[i].Name, PhoneBook[i].PhoneNumber);

} **// i위치의 구조체에 저장된 이름과 전화번호를 출력한다.**

}

* **search.c : PhoneBook에 등록된 이름과 전화번호 출력**

전역변수 : **extern int size** : phone.h에 선언된 int형 변수, 이름과 전화번호를 추가할 때 마다 하나씩 증가한다.

**extern struct Contact PhoneBook[Max]** : phone.h에 선언된 최대 50명까지 저장 가능한 이름과 전화 번호를 저장하는 Contact형 구조체 배열

지역변수 : **char searchName[10]** : 찾고자 하는 이름을 저장하는 char형 배열

void searchByName()

{

char searchName[10];

printf("Enter a name to search: ");

scanf("%s", searchName);

for(int i = 0; i<size; i++) **//구조체 배열 탐색**

{

if( strcmp(searchName,PhoneBook[i].Name) == 0) **// i번째 구조체에 저장된 이름과**

{ **//searchName을 비교해서 일치할 경우**

printf("%s \t %15s\n", searchName, PhoneBook[i].PhoneNumber);

return; **//i번째 구조체에 저장된 이름과 전화번호 출력한다.**

}

}

printf("Oops! %s is not in the PhoneBook\n", searchName);

} **//일치하는 이름이 없는경우**

* **delete.c : PhoneBook에 등록된 이름과 전화번호 삭제**

**전역변수** : **extern int size** : phone.h에 선언된 int형 변수, 이름과 전화번호를 추가할 때 마다 하나씩 증가한다.

**extern struct Contact PhoneBook[Max]** : phone.h에 선언된 최대 50명까지 저장 가능한 이름과 전화 번호를 저장하는 Contact형 구조체 배열

지역변수 : **char deleteName[10]** : 삭제할 이름을 저장하는 char형 배열

void deleteByName()

{

char deleteName[10];

printf("Enter a name to delete: ");

scanf("%s", deleteName);

for(int i = 0; i < size; i++) **//전체 PhoneBook 배열 탐색**

{

if(strcmp(deleteName,PhoneBook[i].Name) == 0**)// deleteName과 i번째 구조체의 { //이름이 일치 하는지 체크**

for(int s = i; s < size-1; s++) **//일치한다면 i번째 구조체부터 size-2번째**

{ **//까지 다음 구조체에 저장된 정보로 교체**

PhoneBook[s] = PhoneBook[s+1];

}

size--; **//저장된 구조체 개수 1 감소**

printf("%s is deleted...\n",deleteName);

return;

}

}

printf("Oops! %s is not in the PhoneBook\n", deleteName);**//일치하는 이름이 없는경우**

}

**(4) Makefile**

CC = gcc **//gcc -> CC , 오브젝트파일들을 OBJS로 매크로 설정**

OBJS = phoneBookMain.o register.o print.o delete.o search.o

%.o: %.c **// 현재경로의 .c파일을 엮어서 오브젝트 파일을 만든다.**

$(CC) -c $<

phoneBook: $(OBJS) **// 생성된 오브젝트로 파일을 이용해 phonebook 실행파일을 만든다.**

$(CC) -o phoneBook $(OBJS)

clean:

rm -f main $(OBJS) phonebook **//현재 경로의 모든 오브젝트파일과 실행파일을 삭제한다.**

**2. 실행 방법**

(1) 사용한 운영체제 및 컴파일러의 종류

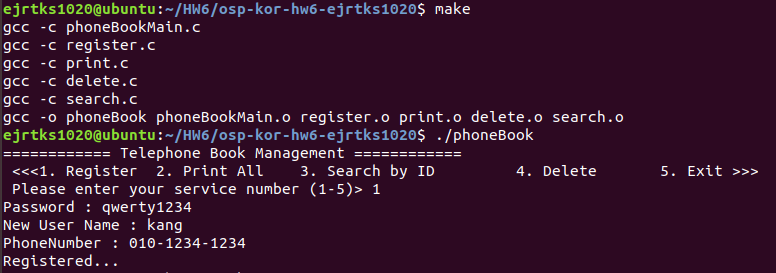
**운영체제** : Ubuntu 64-bit 18.04

**컴파일러** : gcc compiler

(2) 컴파일 방법 및 실행 방법

1. make 입력 -> 실행파일 phoneBook 생성
2. ./phoneBook -> 실행파일 실행

(3) 동작을 확인할 수 있는 실행 화면 캡처



**3. Github 화면**

(1) cloning, adding, committing, push을 위한 github 명령들을 포함

cloning : $git clone <https://github.com/pnusystemprog/osp-kor-hw6-ejrtks1020>

adding : $git add \*.c

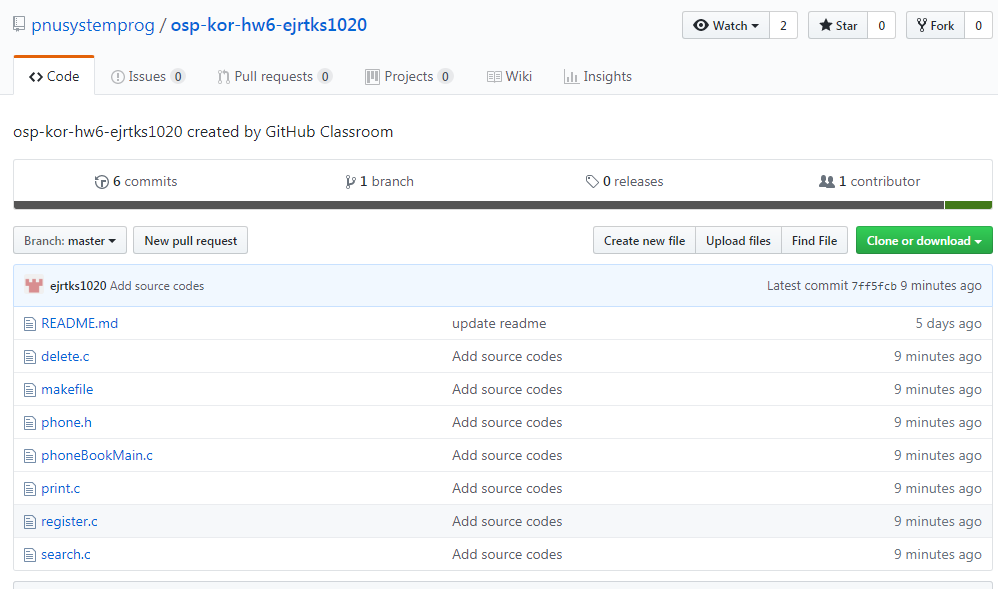
$git add \*.h

$git add makefile

committing : $ git commit -m "Add source codes"

push : $ git push origin master

(2) 소스 코드와 makefile을 push한 후, 본인의 Github repository를 스크린 캡쳐하여 포함



**4. 논의 사항**

register.c에서 구조체 배열에 새로 정보를 저장할 때 원소의 개수를 나타내는 변수 size와 연결하는 방식에 대해 약간 고민을 했었다. 또한 비밀번호를 따로 설정을 해야 하는지, 아니면 소스코드를 짤 때 비밀번호를 설정해두는지에 대해 따로 말이 없어서 일단 그냥 소스코드에 설정한 비밀번호와 입력한 비밀번호를 비교하기로 했다. 그렇기 때문에 비밀번호는 qwerty1234로 고정했다. 3번 비밀번호를 틀리면 비밀번호 틀린 횟수를 기록하는 변수인 static count\_ID가 3이 되는데 이것을 다시 초기화 하지 않아서 이후 register시 처음 비밀번호를 틀려도 바로 함수를 종료되어 수정했다.