데이터 구조

연결리스트를 이용한

UNIX 사용자 정보 관리 프로그래밍

컴퓨터 공학

2015211373 권정인

목차

1. 개요 및 작업 환경
2. 전체 코드 및 코드의 요약
3. 컴파일 및 결과
4. 평가 및 개선 방안
5. 개요 및 작업 환경

저는 Windows 10 내에 있는 Ubuntu 16.04 기반 Linux 서브 시스템 내에서, 코드 작성은VI(VIM)(CLI도구)으로, 코드 빌드(컴파일)은 G++(CLI도구)로 실행 하였습니다.

코드 작성 시작 전, 우선 scp를 이용하여 우리 학교 서버*(ssh://dalma.dongguk.ac.kr)* 에 접속해, */etc/passwd/* 파일을 제 개인 Linux 서브시스템의 홈 폴더로 가져왔습니다. 그에 대한 명령어는 다음과 같습니다 : *scp –P 11800 xperia196@dalma.dongguk.ac.kr:/etc/passwd ~/passwd*

우리 학교 서버에서 직접 */etc/passwd* 파일에 엑세스하여 작업을한다면, 두번째 문단의 작업은 필요가 없습니다. 하지만, 그렇게 한다면 SuperUser(root) 권한 없이는 사용자를 추가하거나, 삭제를 할 수 없을 것 같습니다. 또한, 위 작업을 하기 위해 */etc/passwd* 파일을 직접 엑세스하는것은, 매우 위험한 작업일 듯 하여 안전하게 제 개인 파일로 복사한 후, 작업을 진행 하였습니다.

코드 파일 경로 : *~/hw5.c*

1. 전체 코드 및 코드의 요약

#include <fstream>

#include <iomanip>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <iostream>

using namespace std;

template<class TYPE>

struct NODE{

TYPE data;

NODE \*link;

}; **//노드를 선언합니다.**

template<class TYPE, class KTYPE>

class List{

private:

NODE<TYPE> \*head;

NODE<TYPE> \*pos;

NODE<TYPE> \*rear;

int count;

bool \_add(NODE<TYPE> \*pPre, TYPE dataIn); **//사용자를 추가하는 함수 선언**

void \_delete (NODE<TYPE> \*pPre, NODE<TYPE> \*pLoc, TYPE \*dataOutPtr); **//사용자를 제거하는 함수 선언**

bool \_search (NODE<TYPE> \*\*pPre, NODE<TYPE> \*\*pLoc, KTYPE identify, int& numder); **//사용자를 검색하는 함수 선언**

public:

List(void);

~List(void);

int addNode(TYPE dataIn, int& numder);

bool removeNode(KTYPE identify, TYPE \*dataOutPtr);

bool searchNode(KTYPE identify, TYPE& dataOut, int& numder);

}; **//리스트를 호환하는 List 클래스를 선언합니다.**

template<class TYPE, class KTYPE>

List<TYPE, KTYPE> :: List(void){

head = NULL;

pos = NULL;

rear = NULL;

count = 0;

} **//List에서 포매팅을 할 수 있도록 합니다.**

template<class TYPE, class KTYPE>

int List<TYPE, KTYPE> :: addNode(TYPE dataIn, int& numder){

bool found;

bool success;

NODE<TYPE> \*pPre;

NODE<TYPE> \*pLoc;

found = \_search(&pPre, &pLoc, dataIn.identify, numder);

if(found)

return(+1); **//키가 중복되었다면 +1을 리턴합니다.**

success = \_add(pPre, dataIn);

if(!success)

return(-1); **//데이터의 오버플로우(Overflow)가 발생하면 -1을 리턴합니다.**

return(0);

} **// 노드 추가 템플릿 정의**

template<class TYPE, class KTYPE>

bool List<TYPE, KTYPE> :: \_add(NODE<TYPE> \*pPre, TYPE dataIn){

NODE<TYPE> \*pNew;

if(!(pNew = new NODE<TYPE>))

return false; **//추가할 노드가 이미 정의화되어있으면 false를 반환합니다.**

pNew->data = dataIn;

pNew->link = NULL;

if(pPre == NULL)

{

pNew->link = head;

head = pNew;

} **//pPre가 없으면 첫 노드의 앞이나 빈 리스트에 삽입합니다.**

else

{

pNew->link = pPre->link;

pPre->link = pNew;

} **//pPre의 존재 하에 중간, 또는 끝에 삽입합니다.**

if(pNew->link == NULL)

rear = pNew; **//리스트의 마지막에 삽입되어 있으면 빈 리스트에 삽입합니다.**

count++;

return true;

}

template<class TYPE, class KTYPE>

bool List<TYPE, KTYPE> :: removeNode(KTYPE dltId, TYPE \*pDataOut){

bool found;

NODE<TYPE> \*pPre;

NODE<TYPE> \*pLoc;

int numder = 0;

found = \_search(&pPre, &pLoc, dltId, numder);

if(found)

\_delete(pPre, pLoc, pDataOut); **//노드가 발견되었다면, 그것을 삭제합니다.**

return found;

} **//노드 제거 템플릿 정의**

template <class TYPE, class KTYPE>

void List<TYPE, KTYPE> :: \_delete(NODE<TYPE> \*pPre, NODE<TYPE> \*pLoc, TYPE \*pDataOut){

\*pDataOut = pLoc->data;

if(pPre == NULL)

head = pLoc->link;

else

pPre->link = pLoc->link;

if(pLoc->link == NULL)

rear = pPre;

count--; **//파일 줄의 수 하나 줄이기**

delete pLoc; **//메모리 반환**

return;

}

template<class TYPE, class KTYPE>

bool List<TYPE, KTYPE> :: searchNode(KTYPE identify, TYPE& dataOut, int& numder){

bool found;

NODE<TYPE> \*pPre;

NODE<TYPE> \*pLoc;

found = \_search(&pPre, &pLoc, identify, numder);

if(found)

dataOut = pLoc->data;

**//노드를 찾았다면, dataOut는 찾은 데이터를 가지게 됩니다.**

return found;

} **//노드 찾기 템플릿 정의**

template<class TYPE, class KTYPE>

bool List<TYPE, KTYPE> :: \_search (NODE<TYPE> \*\*pPre, NODE<TYPE> \*\*pLoc, KTYPE identify, int& numder){

\*pPre = NULL;

\*pLoc = head;

if(count == 0)

return false;

if(strcmp(identify, rear->data.identify) > 0)

{

\*pPre = rear;

\*pLoc = NULL;

numder = 1;

return false;

}

while(strcmp(identify, (\*pLoc)->data.identify) > 0)

{

\*pPre = \*pLoc;

\*pLoc = (\*pLoc)->link;

numder++;

} **//파일을 한번씩 둘러볼때 까지 찾기 시도**

if(strcmp(identify, (\*pLoc)->data.identify) == 0)

return true; **//찾기에 성공했으면 true 반환**

else

return false; **//그게 아니면(찾기에 실패했으면) false 반환**

}

template<class TYPE, class KTYPE>

List<TYPE, KTYPE> :: ~List(void){

NODE<TYPE> \*deletePtr;

if(head)

{

while(count > 0){

deletePtr = head;

head = head->link;

count--;

delete deletePtr;

}

}

} **//리스트를 제거하고, 메모리를 반환하여 효율적으로 리소스 활용**

struct PICTURE{

char identify[15]; **//사용자 ID에 관여한 변수**

char info[140]; **//사용자의 기타 정보에 관여한 변수**

};

void instr (void);

void search (List <PICTURE,char\*>&list);

void printList (List <PICTURE,char\*>&list);

void process (List <PICTURE,char\*>&list);

char getChoice (void);

void init (List <PICTURE,char\*>&list);

void addnode(List <PICTURE,char\*>&list);

void deletenode(List <PICTURE,char\*>&list);

int main(void){

system(“clear”); **//가독성을 향상시키기 위하여 화면을 지우는 유닉스 명령어”*clear*”에 접근합니다.**

List<PICTURE,char\*>list;

init (list);

process (list);

cout <<"\nGoodbye. Program halting..\n";

return 0; **//모든 하위작업들이 함수들로 이루어져 있으므로 마지막(process)에서 끝내는 순간 프로그램을 종료합니다.**

}

void init(List <PICTURE, char\*>& list){

ifstream fileIO;

int addResult;

PICTURE pic;

int user = 0, count = 0, numder = 0;

fileIO.open("passwd"); **//파일을 여세요.**

**//파일을 실제 */etc/passwd* 파일로 작업한다면, *fileIO.open(“/etc/passwd”);* 로 수정합니다.**

if(!fileIO){

cerr<<"\a We can't Open the file. Halting.. \n";

exit(110); **//파일이 없다면 오류메시지를 표시하고, 프로그램을 강제 종료합니다.**

}

cout << "Organizing User Table\n"

<< "Reading file and establishing Table\n";

while(!fileIO.eof()){

fileIO.getline(pic.identify, sizeof(pic.identify), ':');

fileIO.getline(pic.info, sizeof(pic.info));

user += 1; **//':'을 분기로 파일의 한 행씩 파일을 계속 읽습니다.**

addResult =list.addNode(pic, numder);

count++; **//1줄을 읽어들일때마다 count를 1씩 증가합니다.**

if(addResult != 0){

if(addResult == -1){

cout<<"Memory Overflow. Halting..\n";

exit(120);

} **//메모리가 오버플로우 될 경우, 오류메시지를 표시하고 프로그램을 강제 종료합니다.**

else{

cout<<"Duplication User : "

<<pic.identify

<<"\n\a";

} **//파일 내에 중복된 사용자가 있을경우 알려줍니다.**

}

if(count == 50){

cout << ".";

count = 0;

} **//위에 사용된 count가 50이 되었을 경우 '.'을 표시하고 count를 0으로 초기화합니다.**

}

cout <<"Organize User Table completed!\t"<<user<<"Users\n\n";

cout<<endl;

return;

}

char getChoice(void){

int choice;

bool valid;

cout<< "==========================================\n"

<< " 1. Search User\n"

<< " 2. Add User\n"

<< " 3. Delete Uer\n"

<< " 4. Exit\n\n"

<< "Enter your choice : ";

**//사용자에게 입력을 유도합니다.**

do{

cin>>choice;

switch(choice){

case 1:

case 2:

case 3:

case 4:

valid = true;

break; **//사용자의 입력이 1, 2, 3, 4이면 valid을 true로 반환하여 곧 getChoice 함수 종료**

default:

valid=false;

cout<<"\aWrong choice! Retry with enter number\n";

break;

}

}while(valid == true); **//사용자에게 정상적인 입력을 받았을때까지 위의 do 하위 문 반복**

return choice;

}

void process(List <PICTURE,char\*>&list)

{

char choice;

do{

choice = getChoice();

switch(choice){

case 1:

search(list);

break;

case 2:

addnode(list);

break;

case 3:

deletenode(list);

break;

case 4:

break;

}

}

while(choice!=4);

return;

} **//위에서 받은 choice에 따라 각 함수로 연결**

void search(List <PICTURE, char\*>&list){

char id[15]; **//입력할 사용자 ID에 사용될 변수**

bool found;

int numder = 0;

PICTURE pic;

cout << "Search User\n"

<< "Enter User ID : ";

cin >> id; **//ID를 입력받아 id에 마운트**

found = list.searchNode(id, pic, numder);

if(found){

cout << "User : " << pic.identify <<"\n"<<"Info : "

<< pic.identify << ":" << pic.info<<"\n"

<< "Count : "

<< numder

<< "\n\n";

numder = 0; **//결과를 출력하여 찾기횟수(number) 초기화**

}

else{

cout<<"We failed to find : "<<"id"<<"\n"<<"Count : "

<< numder

<< "\n\n";

numder = 0;

}

return;

} **//found함수를 이용하여 검색**

void addnode(List <PICTURE,char\*>&list){

PICTURE pic;

char data[160];

int i = 0, j = 0, numder = 0;

int addResult;

cout << "Add User\n" << "Input User Data : ";

cin >> data;

while(data[i] != '\0'){

pic.info[j] = data[i]; **// data[i]가 0이 아니면 데이터를 pic.info[j]에 적재합니다.**

i++;

j++;

}

pic.info[j] = '\0';

addResult =list.addNode(pic, numder);

if(addResult != 0){

if(addResult == -1){

cout<<"Memory Overflow! Halting..\a\n";

exit(120); **//메모리가 오버플로우됬을경우 오류메시지를 표시하고 강제 종료합니다.**

}

else{

cout<< "We can't add user "

<< pic.identify<<".."

<< "Count : "

<< numder

<< "\n\n";

numder = 0; **//결과가 정상이 아닐 경우(addResult가 0이 아닐경우), 값을 저장하지 않고, 오류 메시지를 표시하고 number을 0으로 초기화**

}

}

else{

cout << "We added User "<<pic.identify

<<". Count : "<< numder

<< "\n\n";

numder = 0; **//결과가 정상일 경우(addResult가 0일 경우), 값을 저장하고 number을 0으로 초기화**

}

}

void deletenode(List <PICTURE,char\*>&list){

char id[15];

bool found;

int numder = 0;

PICTURE pic;

cout << "Remove User\n" << "Enter ID that remove : ";

cin >> id;

found = list.searchNode(id, pic, numder);

if(found){

cout << pic.identify << ":"

<<pic.info

<< " Removed. Count : "

<< numder

<< "\n\n";

found = list.removeNode(id, &pic);

numder = 0;

}

else{

cout<<"User ID : "<<id

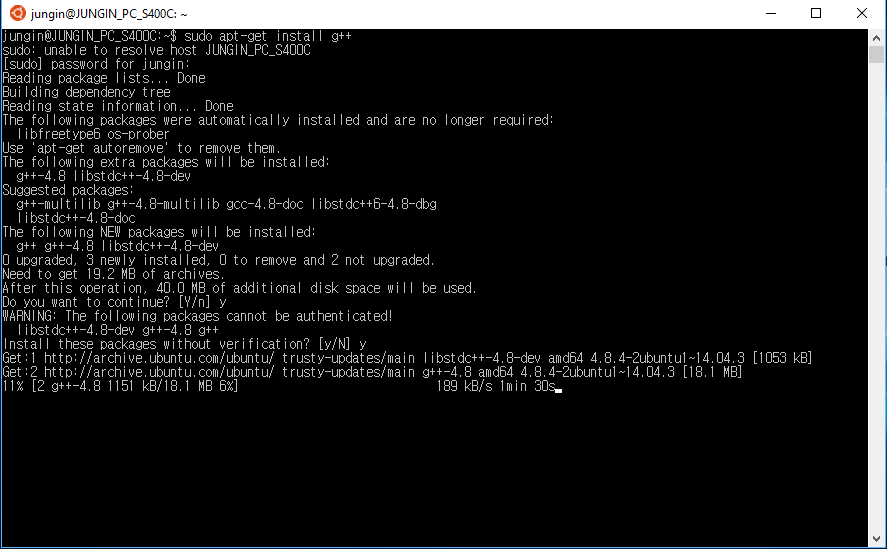
<<" is not found. Cancelling..\n\n";

}

return;

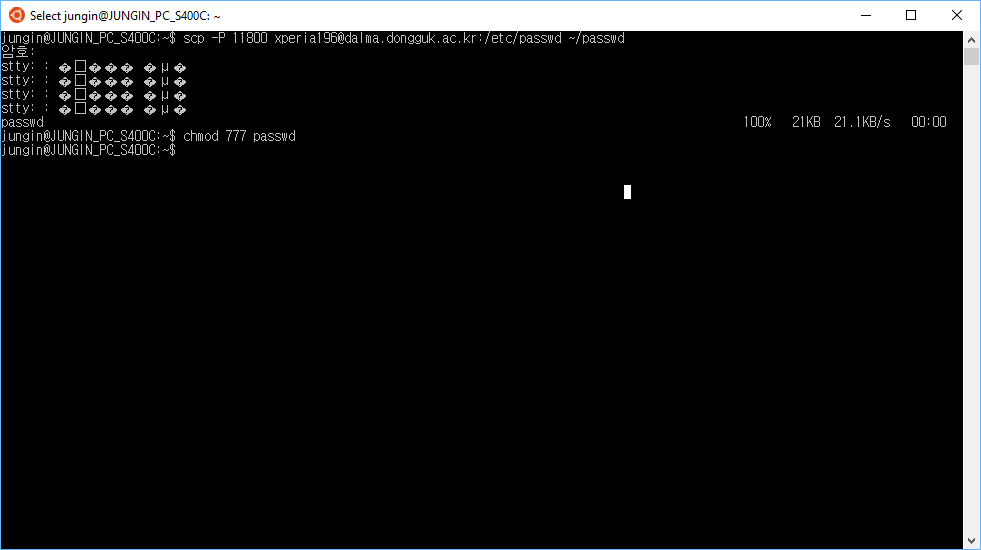
}

1. 컴파일 및 결과



작업 시작 전, C++언어를 컴파일(빌드)하는데 필요한 *g++*명령어를 사용하기 위해 필요한 패키지를 SuperUser 권한으로 설치합니다.

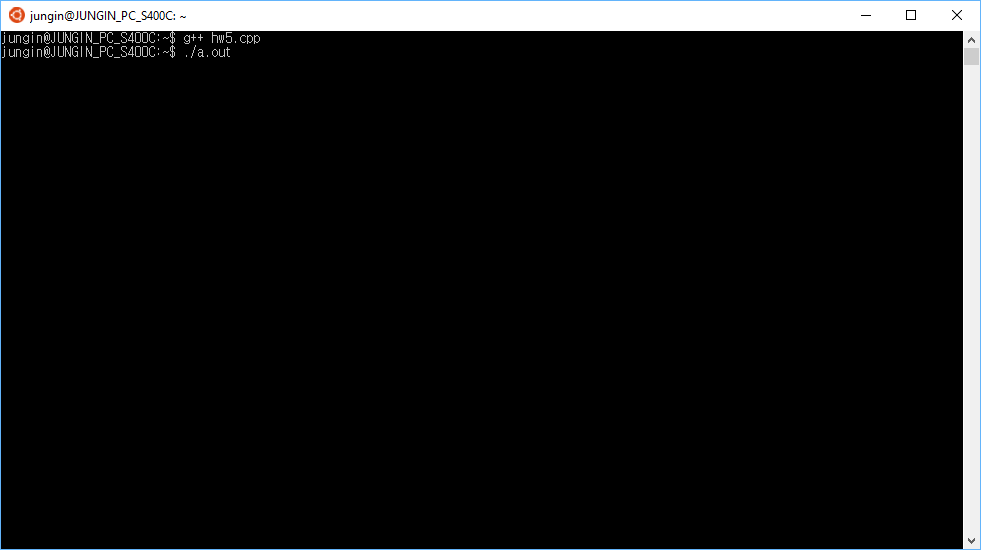
명령어 : sudo apt-get install g++



학과 ssh 서버로 접속하여 */etc/passwd/* 파일을 홈 디렉토리의 *passwd* 파일로 복사한 후, 해당 *passwd* 파일의 권한은 읽기, 쓰기, 실행 모두 허용하여 줍니다.

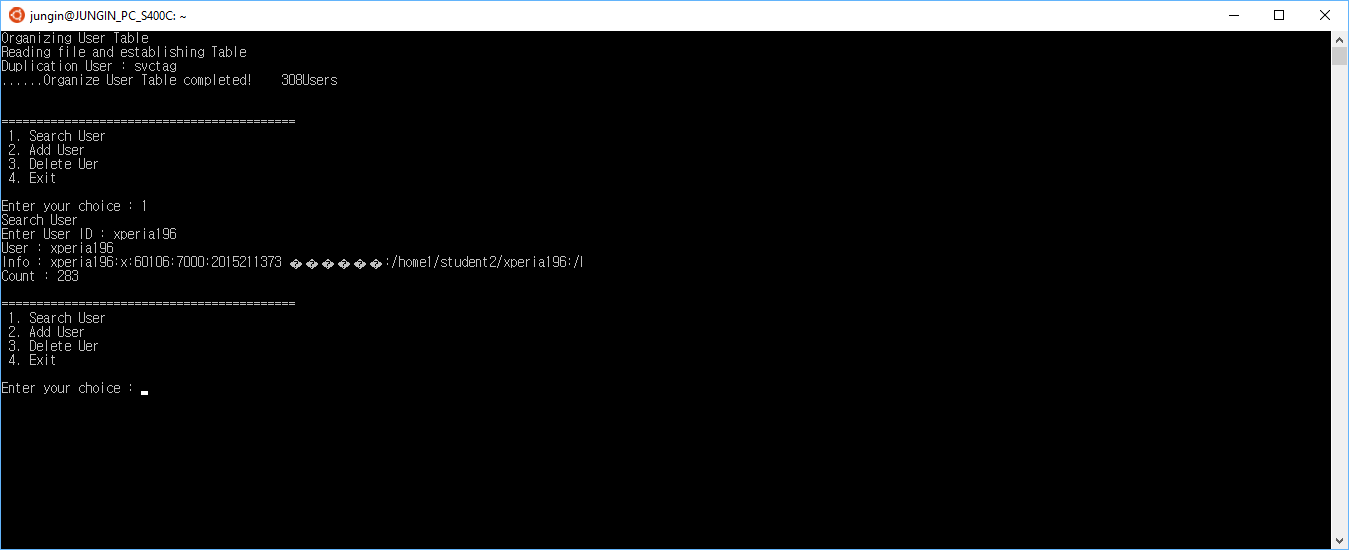
명령어 : *scp –P 11800 xperia196@dalma.dongguk.ac.kr:/etc/passwd ~/passwd*

*chmod 777 passwd*

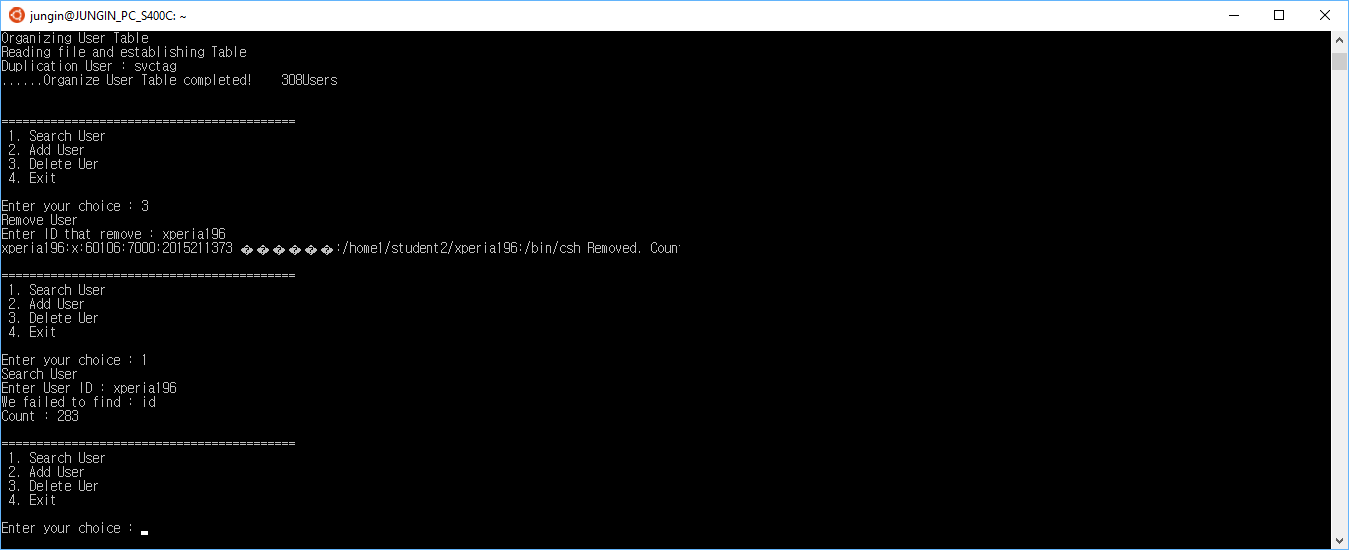


*g++ hw5.cpp* 명령어를 사용하여 *hw5.cpp* 파일을 컴파일(빌드)한 후,

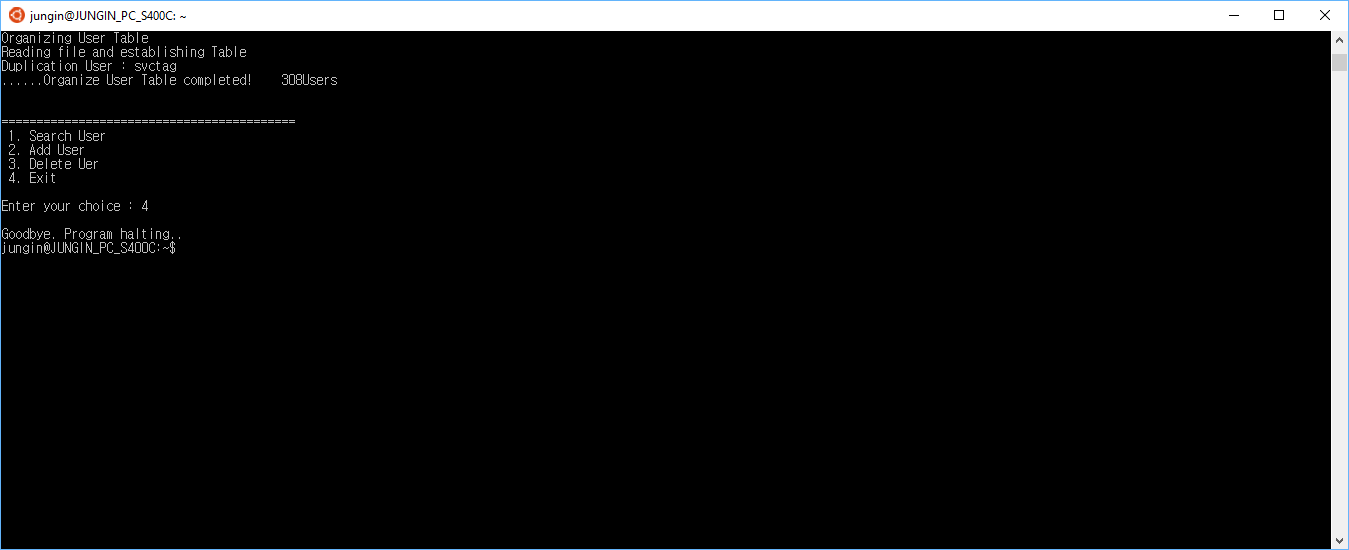
생성된 *a.out* 프로그램을 실행합니다.



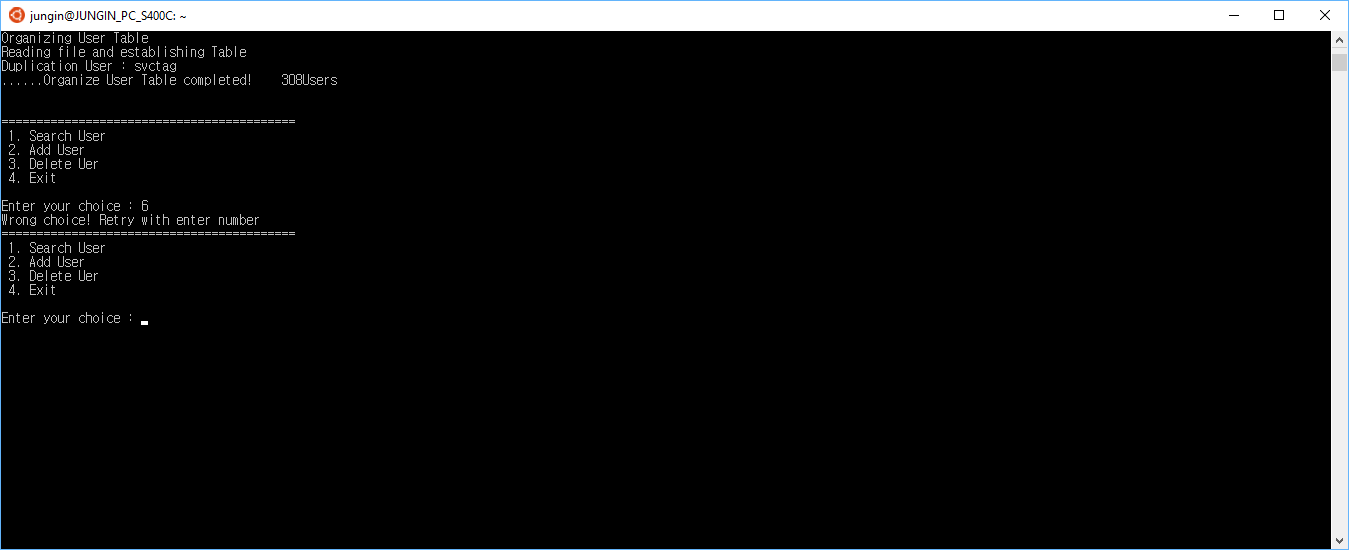
실행한 후, 검색 기능을 이용하여 제 학과서버 ID인 *xperia196* 을 검색해 보았습니다. 결과가 정상적으로 출력됨을 확인할 수 있습니다.(Windows 10 내의 서브시스템 호환성 오류로 한글 이 인코딩이 맞지 않아 깨지지만, 코드 상으로는 아무 문제가 없습니다.)



다음으로, 사용자 삭제 기능을 사용하여 제 학과서버 ID인 *xperia196*을 삭제해 보았습니다. 정상적으로 삭제가 완료됨을 확인할 수 있습니다. 삭제가 완료된 다음, 제대로 삭제되었는지 확인하기 위해 다시 *xperia196*을 검색해 보았습니다. 테이블이 제대로 삭제가 완료되었기때문에, *xperia196*이 검색되지 않음을 확인할 수 있었습니다.



마지막으로, 프로그램이 종료되는 4번 항목에 접근하자, 프로그램이 정상적으로 종료됨을 확인할 수 있었습니다.



번외로, 메뉴 선택 화면에서 , (1, 2, 3, 4)가 아닌 다른 선택 입력 시, 선택을 되물어보는것 역시 정상적으로 작동됨을 확인할 수 있었습니다.

1. 평가 및 개선 방안
2. 대부분의 코드를 *int(main)*에 전부 삽입시키지 않고 함수로 연결하여 코드를 작성하여, 수정 및 보완을 용이하게 하였으며, 코드의 길이 또한 짧아짐을 알 수 있었습니다.
3. C++ 언어를 이용하여 코드를 작성하여 보는것은 처음이었습니다. C++에 대한 문법 등등이 저에게는 낯설어 교재와 인터넷 문서등을 참고하여 코드 작성을 진행하였으나, 많이 힘들었음을 알았습니다. 하지만, 사용자 ID 변수의 길이가 전부 다르기 때문에, 이 프로그램은 C++으로 작성되어야 합니다.
4. 과제의 본 내용은 학과 서버에서 */etc/passwd* 파일에 직접 접근하여 작업하는것이 원칙이였으나, 그렇게 하게되면 *SuperUser(root)* 권한이 필요하며, 해당 파일을 건드리는것은 매우 위험하기 때문에 제 개인 PC로 *passwd* 파일을 복사하여, 안전한 작업 환경에서 작업하였습니다.
5. 아직도 제가 사용하는 시스템에서는 Bash에서 한글 출력이 제대로 되지 않아, 영문으로 작성되었습니다. 학과 서버에 접속하여 작업을 하려하면, 한글 입력이 제대로 되지 않음을 알 수 있었습니다.