2019 봄학기 컴퓨터 프로그래밍 [HW_1] Report

소비자아동학부 (소비자학전공) 2015-15356 이준희

<Version #1. C++>

1. 환경 : Visual Studio Code (c++컴파일을 위해 code runner 확장기능 설치 – 자세한 환경설정 내역은 아래 settings.json 파일 스크린샷 첨부) , 윈도우 10

settings.json

```
{
    // Code Runner "C/C++" 설정
    "code-runner.executorMap": {
        "c": "cd $dir && gcc $fileName -o $fileNameWithoutExt && $dir$fileNameWithoutExt",
        "cpp": "cd $dir && g++ $fileName -o $fileNameWithoutExt && $dir$fileNameWithoutExt"
    },

    // Window 10 에서는 Default Shell 이 Powershell이기 때문에 cmd로 변경해야 정상적으로 실행 됨.
    "terminal.integrated.shell.windows": "C:\\Windows\\sysnative\\cmd.exe",

    // 터미널 입력 (출력 사용시 주석 처리)
    "code-runner.runInTerminal": true,
    "files.encoding": "euckr",

    "terminal.integrated.shellArgs.windows": ["/K", "chcp 65001"],

}
```

2. 코드설명 및 프로그래밍 과정 :

```
#include <string>
#include <vector> // for phonebook
#include <iostream>
#include <ctime> // for getting current date to calculate d-day
#include <regex> // to check input for phone number and birthdays
```

먼저 사용한 라이브러리는 다음과 같다. string, vector, iostream은 과제스펙에 따라 포함시켜주었고, ctime은 d-day계산을 위해, regex는 인풋값에 대한 예외처리를 위해 포함시켜주었다.

```
class Person {

    public:
    Person( string &first, string &last, string &phone_num){
        firstName = first;
        //strcpy(firstName,first);
        lastName = last;
        phoneNumber = phone_num;
    }
    void setFirstName( string &first){
        firstName = first;
    }
    string getFirstName(){
        return firstName;
    }
    void setLastName(string &last){
        lastName = last;
    }
    string getLastName(){
        return lastName;
    }
    void setPhoneNumber(string &phone_num){
        phoneNumber = phone_num;
    }
    string getPhoneNumber(){
        return phoneNumber;
    }
    virtual void print(){
        cout<<firstName<<"_"<<lastName<<"_"<<phoneNumber<<endl;
    }

    private:
    string firstName;
    string lastName;
    string phoneNumber;
    };
```

가장 먼저 다른 세 클래스들의 부모 클래스가 되어줄 Person class를 정의해주었다. constructor의 인자로는 첫 번째 이름, 두 번째 이름, 그리고 전화번호의 string reference값을 받는다. 각각 class의 private멤버들에 적절히 값을 할당해준다. set으로 private member값을 간접적으로 얻을 수 있으며, set으로 간접적으로 값을 할당해 줄 수 있다. 마지막으로 print 메소드는 과제스펙에서 명시한 형태를 따라 출력해준다. 또한, 이는 차후에 자식 메소드에서 오버라이드 할 수 있도록 virtual형태로 정의해준다.

```
class Work : public Person {

    public:
    Work(string &first, string &last, string &phone_num, string &team):Person(first, last, phone_num){
        this -> team = team;
    }
    void setTeam(string &team){
        this -> team = team;
    }
    string getTeam(){
        return team;
    }
    void print(){
        cout<<getFirstName()<<"_"<<getLastName()<<"_"<<getPhoneNumber()<<"_"<< team <<endl;
    }

    private:
    string team;
    };
```

Work 클래스는 Person클래스를 상속받는 자식 클래스로 정의했다 (이 뒤에 나올 Friend, Family도 동일한 형태로 선언해주었다.). constructor에는 Person의 constructor에서 받는 인자 외에 team정보를 담고있는 string reference 인자 하나를 추가로 받는다. 따라서, 앞에 Person때와 동일한 세 인자는 Person의 constructor를 활용해 적절한 값을 할당해주고, 마지막으로 Person에 존재하지 않았던 team정보는 따로 코드를 적어준다. 받아온 인자값 team이 Work class의 private member team과 이름이 동일하기 때문에 구분해 주기 위하여 this->team 이라는 문법을 사용해주었다. Person에서는 존재하지 않았던 private멤버 team에 대해서는 따로 get/set 메소드를 정의해주었다. print메소드는 과제요구사항에 맞게 부모클래스의 메소드를 오버라이드 하여 적절한 값을 출력하게끔 하였다.

```
class Friend : public Person {

    public:
    Friend(string &first, string &last, string &phone_num, int &age):Person(first, last, phone_num){
        this -> age = age;
    }
    void setAge(int &age){
        this -> age = age;
    }
    int getAge(){
        return age;
    }
    void print(){
        cout<<getFirstName()<<"_"<<getLastName()<<"_"<<getPhoneNumber()<<"_"<< age <<endl;
    }

    private:
    int age;
    };
```

Friend또한 Person의 자식클래스로 정의해주었다. 다만, constructor에서 받는 앞의 세 인자를 제외한 맨 마지막 인자는 나이값 정보를 저장해줄 age의 int reference로, 그 인자에 대해서는 Person의 constructor를 이용할 수 없으므로 따로 코드를

작성해준다. age 외 앞 세 인자에 대해서는 Person cosntructor를 그대로 재활용하면 된다. 또, Work에서 해주었던것과 같이 Person에 존재하지 않았던 private멤버 age에 대해서 따로 get/set 메소드를 정의해주었다. print또한 과제스펙에서 명시한 형태로 메소드 오버라이딩을 해주었다.

```
class Family : public Person {

    public:
    Family(string &first, string &last, string &phone_num, string &birthday):Person(first, last, phone_num){
        this -> birthday = birthday;
    }
    void setBirthday(string &birthday){  //set birthday (YYMMDD)
        this -> birthday = birthday;
    }
    string getBirthday(){
        return birthday;
    }
    int dDay(){  //calculate the date difference between the birthday and current time
        time_t now = time(0);
        tm *today = localtime(&now);
        // today's date info
        int today_month = 1 + today->tm_mon;
        int today_day = today->tm_mday;
        // parse birthday and get the date info
        int birthday_month = stoi(birthday.substr(2,2));
        int birthday_day = stoi(birthday.substr(4,2));

        // calculate the date difference
        return date_difference(birthday_month, birthday_day, today_month, today_day);
    }
    void print(){
        cout<<getFirstName()<<"_"<<getLastName()<<"_"<<getPhoneNumber()<<"_"<< birthday <<endl;
    }
```

```
private:
string birthday;

int date_difference(int& b_month, int& b_day, int& t_month, int& t_day){
    const int month_days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
    int t_day_nums=0;
    int b_day_nums=0;
    // today는 이번년도 기준 몇일째인지 계산
        for (int i=0; i<t_month-1;i++){
            t_day_nums += month_days[i];
        }
        t_day_nums+=t_day;
    // birth day는 이번년도 기준 몇일째인지 계산
        for (int i=0; i<b_month-1;i++){
            b_day_nums += month_days[i];
        }
        b_day_nums+=b_day;

    if (b_day_nums >= t_day_nums) {  // 역전되지 않는 상황
        return (b_day_nums - t_day_nums);
    }
    else{  // 역전되는 상황
        return (b_day_nums + 365 - t_day_nums);
    }
}
// private member 정의 끝
};
```

Family class 코드는 살짝 길어서 public과 private을 기준으로 잘라서 스크린샷을 첨부했다. public부분을 먼저보면, constructor나 get/set/print메소드 선언해준 부분은 앞의 두 케이스와 상당히 유사하다. (Person과 유일하게 다른 것은 birthday라는 필드가 추가되었다는 것이다) 다만, 앞의 두 케이스와 살짝 다른점이라 하면, 바로 Family class에선 d-day계산 기능을 지닌 메소드가 선언되어있다는 것이다. int D_day라는 public 클래스 메소드 안에서, date_difference()라는 private메소드를 사용하고 있다. D_day 메소드 먼저 봐보면, ctime라이브러리를 통해 오늘 날짜를 계산하고 있다. 라이브러리 작동원리상 월(month)이 0부터 11로 들어오므로 +1을 해 today_month값에 저장해 주었다. 다음으로는, private member birthday에 저장되어 있는 string값에 월, 일이 있는 자리를 각각 따로 뽑아내 int로 만들어주었다. 이렇게 d-day를 계산해야 할 두 날짜의 월, 일을 int형태로 다 뽑아낸 상태가 되었다. 이렇게 뽑아낸 값들을 활용해 private으로 선언해준 date_difference에 인자로 넣어준다. 첫 번째, 두 번째 인자로 생일 월, 일을 넣어주면 되고, 세 번째, 네 번째 인자로 현재 날짜의 월, 일을 넣어주면 d-day계산값을 int형태로 return해준다. date_difference를 자세히 보면, 1-12월까지의 각 달별 마지막 일을 나열하여 month_days에 넣어주었다. t_day_nums는 이번년도에 '오늘'이 되기까지 총 몇 일이 필요했는지, b_day_nums는 같은년도에 '생일'이 되기까지는 총 몇 일이 필요했는지를 각각 계산해준다. 그 계산은 앞서 정의해준 month_days를 활용해 계산해준다. 해당 달 바로 직전달까지는 full로 각 달별 일수를 더해주고, 해당 달은 해당 날짜의 일수를 더해주는 식으로 연산이 이루어진다. 다음으로는, 생일이 오늘을 역전했는지 아닌지에 따라 경우를 나눠준다. 즉, 이번년도에 생일이 이미 지난 상황이라면 역전되었으므로, 365를 더한 상태에서 b_day_nums-t_day_nums를 해준다. 역전되지 않은 상황이라면 365를 더하지 않고 그냥 b_day_nums-t_day_nums를 해주면 된다.

```
class PhoneBook
{
    public:
    PhoneBook(){
        phone_book.reserve(100);
    }
    void add_one(Person* new_member){
        if (phone_book.size()>=phone_book.capacity()){
            phone_book.reserve(phone_book.capacity()*2);
        }
        phone_book.push_back(new_member);
        cout<<"Successfully added new person."<<endl;
    }
    void delete_one(int idx){
        // 먼저 delete가 가능한 요소인지 확인
        if (phone_book.size() <= idx || idx<0){
            cout<<"Person does not exist!"<<endl;
        }
        else{
            phone_book_it = phone_book.begin()+idx;  // Iterator에 벡터의 시작점부터 idx번째 뒤 위치를 알려준다.
            phone_book.erase(phone_book_it); // erase 메소드는 iterator를 인자로 줘야함.
            cout<<"A person is successfully deleted from the Phone Book!"<<endl;
        }
    }
    void print_all(){
        cout<<"Phone Book Print"<<endl;
        for (int i=0; i < phone_book.size(); i++){
            cout<< i+1 << ".";
            phone_book.at(i)->print(); // child class method에 접근하기 위해서 포인터를 사용해줌
        }
    }

    private:
    vector <Person*> phone_book;
    vector <Person*>::iterator phone_book_it;  //  Person 형식의 Vector를 가르키는 Iterator 선언
};
```

main함수를 보다 간소화 하기 위해 PhoneBook 클래스를 따로 정의해주었다. private member로는 Person의 포인터를 구성요소로 하는 phone_book이라는 이름의 벡터를 선언해주었다. 또한, delete_one메소드에서 erase를 쓸 때 필요한 iterator를 위해 phone_book_it도 선언해주었다. constructor에는 처음에 생성될 때 phone_book 벡터에 100만큼의 여유 자리를 예약하도록 .reserve(100)을 해 주었다. Person pointer 요소를 추가하기 위한 add_one 메소드에서는, add를 했을시 phone_book 벡터의 capacity를 넘어선다면, 현재 capacity의 2배 되는 크기를 reserve하도록 하였다(이 조건문에 자주 걸리지 않게 하기 위해-속도문제- 그냥 상수 크기가 아닌, 기존 capacity의 2배 되는 크기를 reserve하게 처리하였다). 앞의 capacity문제가 잘 해결되었다면 vector의 push_back메소드를 통해 요소를 추가해준다. 정상적으로 추가되었다면 해당 메시지를 출력해준다. 다음으로 delete_one메소드를 살펴보면, 먼저 인덱스 상 delete가 가능한 요소인지 확인한다. console 차원에서 가령, 세 번째 사람을 지워달라고 3을 입력할 경우, 거기서 1을 뺀 값을 PhoneBook 클래스의 delete_one 메소드에 인자로 전달해줄 것이기 때문에, 그 부분은 따로 고려하지 않아도 된다. 그 부분을 제외하고 생각했을 때, phone_book 벡터 사이즈 이상의 index 혹은 0 미만의 수가 인자로 들어오면 해당 index의 사람을 삭제할 수 없다는 메시지를 출력한다. 이러한 예외값이 아닌 경우엔, phone_book_it

iterator를 이용해 vetor의 erase메소드를 사용하여 해당 index의 요소를 벡터로부터 삭제해준다. 마지막으로 print_all메소드의 경우, 현재 vector내에 있는 모든 요소들에 루프를 돌며 각 요소들이 지니고 있는 print 메소드를 실행시킨다. Person의 포인터를 요소값으로 지니고 있는 벡터이기에, .이 아닌 ->로 메소드에 접근할 수 있다.

다음으로는 본격적인 콘솔이 구현되어있는 메인함수를 살펴보도록 하자.

```
// interactive console
int main()
{
  PhoneBook my_phone_book = PhoneBook();

  bool exit=false;

  while (!exit){
    cout<<"CP-2015-15356>";
    string num_choice;
    string init_input;
    getline(cin , init_input) ;

    if (init_input=="exit"){
        return 0;
        //exit = true;
    }

    else if (init_input=="1"|init_input=="2"|init_input=="3"){
        num_choice = init_input;
    }

    else if (init_input.empty()) {
        cout<<"Phone Book\n"<<"1. Add person\n"<< "2. Remove person\n" << "3. Print phone book"<<endl;
        cin>>num_choice;
        cin.ignore();
    }
```

처음 main함수를 시작할 때 my_phone_book이라는 이름의 새로운 PhoneBook클래스 인스턴스를 선언한다. 처음에 exit이 false인 동안 계속해서 while루프를 돌며 콘솔을 출력한다. initial state에서의 모든 인풋은 string init_input으로 받는다. exit을 입력했을땐 곧바로 return 0;을 통해 프로그램을 main함수를 끝낸다. 그 외 1,2,3중 하나가 들어오면 실행하고자 하는 명령문 숫자를 저장해 놓을 num_choice에 해당 값을 저장해둔다. 만약 엔터값이 들어올 경우, 실행가능한 명령문 목록을 출력하고, 그 후 실행하고자 하는 명령문 숫자를 string타입으로 num_choice에 받아 저장한다. (입력 후 친 엔터는 무시하게 하기 위해 cin.ignore()문을 써주었다.)

```
if (num_choice=="1"){ // Add
    cout<<"Select Type\n"<<"1. Person\n"<< "2. Work\n" << "3. Family\n" << "4. Friend" <<endl;
    int select_type;
    cin>>select_type;
    cin.ignore();


    string name;
    string phone_num;

    regex right_name_format("^([a-zA-Z]+)(\\s{1})([a-zA-Z]+)"); //알바펫+space+알파벳 only
    while (true){
        cout<<"Name: ";
        getline(cin, name);
        // name 처리 -- format 맞게 들어왔는지 확인
        if (regex_match(name,right_name_format)) break;
        // cout<<"[Error] : Wrong format. Please type again."<<endl;
    }

    // name 처리 -- 통으로 받은 이름을 first name과 last name으로 나누기
    int space = name.find(" ");
    string first_name = name.substr(0, space);
    string last_name = name.substr(space+1);

    regex right_num_format("^(02|010)(-\\d{4}){2}"); //02-XXXX-XXXX or 010-XXXX-XXXX 형태 only
    while (true){
        cout<<"Phone_number: ";
        getline(cin, phone_num);
        // phone_number 처리 -- format 맞게 들어왔는지 확인
        if (regex_match(phone_num,right_num_format)) break;
        // cout<<"[Error] : Wrong format. Please type again."<<endl;
    }
```

1을 선택하면 Add Person기능이 수행된다.

select_type변수에 어느 종류의 요소를 추가하고 싶은지 번호로 선택한 값을 할당한다. 네 요소 모두 name과 phone_num은 공통으로 받아야 하는 부분이니 뒤에 case를 나누기 전에 먼저 입력받는다. name은 알파벳+space+알파벳 형태가 들어올 때까지 while문을 통해 계속해서 입력받는다. 올바른 name 인풋값이 들어오면, space를 기준으로 앞쪽을 first_name, 뒤쪽을 last_name변수에 할당한다. 다음으로 phone_num을 입력받을 때도, 과제스펙에서 명시한 format이 들어올 때까지 while 문을 돈다.

```
if (select_type==1){ // add Person
    Person *p = new Person(first_name, last_name, phone_num);
    my_phone_book.add_one(p);
}
else if (select_type==2){ // add Work
    string team;
    cout<<"Team: ";
    getline(cin, team);

    Work *w = new Work(first_name, last_name, phone_num, team);
    my_phone_book.add_one(w);
}
else if (select_type==3){ // add Family
    string birthday;
    regex right_day_format("^\\d{6}");

    while (true){ // 올바른 입력값이 들어올때까지
        cout<<"Birthday: ";
        getline(cin, birthday);
        // phone_number 처리 -- format 맞게 들어왔는지 확인
        if (regex_match(birthday,right_day_format) && birthday!="000000") break;
        // cout<<"[Error] : Wrong format. Please type again."<<endl;
    }

    Family *fam = new Family(first_name, last_name, phone_num, birthday);
    my_phone_book.add_one(fam);
}
else if (select_type==4){ // add Friend
    int age;
    while (true){ // 올바른 입력값이 들어올때까지
        cout<<"Age: ";
        cin>>age;
        if (age>0) break; // 자연수만 받기
        // cout<<"[Error] : Wrong format. Please type again."<<endl;
    }

    Friend *fr = new Friend(first_name, last_name, phone_num, age);
    my_phone_book.add_one(fr);
}
```

올바른 name, phone_num을 입력받은 후에야 case별로 수행해야 할 코드를 나눠준다. add하고자 하는 게 Person일 경우 새로운 Person 인스턴스를 위한 공간을 만들어주고 해당 Person 포인터를 my_phone_book에 add_one메소드를 통해 추가한다. add하고자 하는 게 Work일 경우 Team을 따로 입력받고, Work 인스턴스 공간을 위한 w 포인터 변수를 선언하고, w를 add_one메소드의 인자로 전달해 준다. Family를 add하고자 하는 경우, birthday를 따로 받아야 한다. 과제 스펙에서 6자리 digit, 자연수만 받는다고 하였으므로 해당 format을 맞춘 인풋이 들어올 때까지 계속 while문을 돈다. Family인스턴스를 만들기 위한 모든 인자값들이 준비되면, 앞의 케이스들에서 했던것과 유사하게 Family포인터를 생성시켜주고, add_one메소드에 인

자로 전달해준다. 마지막으로 Friend를 add하기로 선택했다면, age를 받는다. 자연수만 받는다고 했으니, 0이하의 수가 들어오면 올바른 자연수값이 들어올 때까지 계속해서 입력받는다. 다음엔 앞서했던 것과 비슷한 과정으로 Friend 포인터를 생성해 my_phone_book 의 add_one메소드 인자로 전달해준다.

```
    else if (num_choice=="2"){ //Remove Someone
        int idx;
        cout<<"Enter index of person: ";
        cin>>idx;
        cin.ignore();
        my_phone_book.delete_one(idx-1);
    }
    else if (num_choice=="3"){ //Print Phonebook
        my_phone_book.print_all();
    }
  } // while문 끝

}
```

만약 initial state에서 2를 선택했다면, 삭제할 사람의 인덱스를 입력받는다. 콘솔상의 인덱스와 vector에서의 index는 다르므로 (전자는 1부터 시작, 후자는 0부터 시작), 이를 맞춰주기 위해 delete_one메소드의 인자로는 idx-1을 한 값을 전달해준다.

만약 initial state에서 3을 선택했다면, my_phone_book에 구현된 print_all메소드를 사용해 지금까지의 모든 내용을 출력해준다.

전반적인 코드설명 외 프로그래밍 과정에서 발생했던 트러블슈팅 프로세스는 다음과 같았다.

<Trouble-Shooting Process ①>

```
if (select_type==1){ // add Person
    my_phone_book.add_one(&Person(first_name, last_name, phone_num));
}
else if (select_type==2){ // add Work
    string team;
    cout<<"Team: ";
    getline(cin, team);
    my_phone_book.add_one(&Work(first_name, last_name, phone_num, team));
}
else if (select_type==3){ // add Family
    string birthday;
    regex right_day_format("^\\d{6}");

    while (true){ // 올바른 입력값이 들어올때까지
        cout<<"Birthday: ";
        getline(cin, birthday);
        // phone_number 처리 -- format 맞게 들어왔는지 확인
        if (regex_match(birthday,right_day_format)) break;
        cout<<"[Error] : Wrong format. Please type again."<<endl;
    }
    my_phone_book.add_one(&Family(first_name, last_name, phone_num, birthday));
}
else if (select_type==4){ // add Friend
    int age;
    cout<<"Age: ";
    cin>>age;
    my_phone_book.add_one(&Friend(first_name, last_name, phone_num, age));
}
```

```
HW_1.cpp: In function 'int main()':
HW_1.cpp:252:79: error: taking address of temporary [-fpermissive]
                 my_phone_book.add_one(&Person(first_name, last_name, phone_num));
                                                                               ^
HW_1.cpp:258:83: error: taking address of temporary [-fpermissive]
                 my_phone_book.add_one(&Work(first_name, last_name, phone_num, team));
                                                                                   ^
HW_1.cpp:271:89: error: taking address of temporary [-fpermissive]
                 my_phone_book.add_one(&Family(first_name, last_name, phone_num, birthday));
                                                                                         ^
HW_1.cpp:277:84: error: taking address of temporary [-fpermissive]
                 my_phone_book.add_one(&Friend(first_name, last_name, phone_num, age));
                                                                                    ^
```

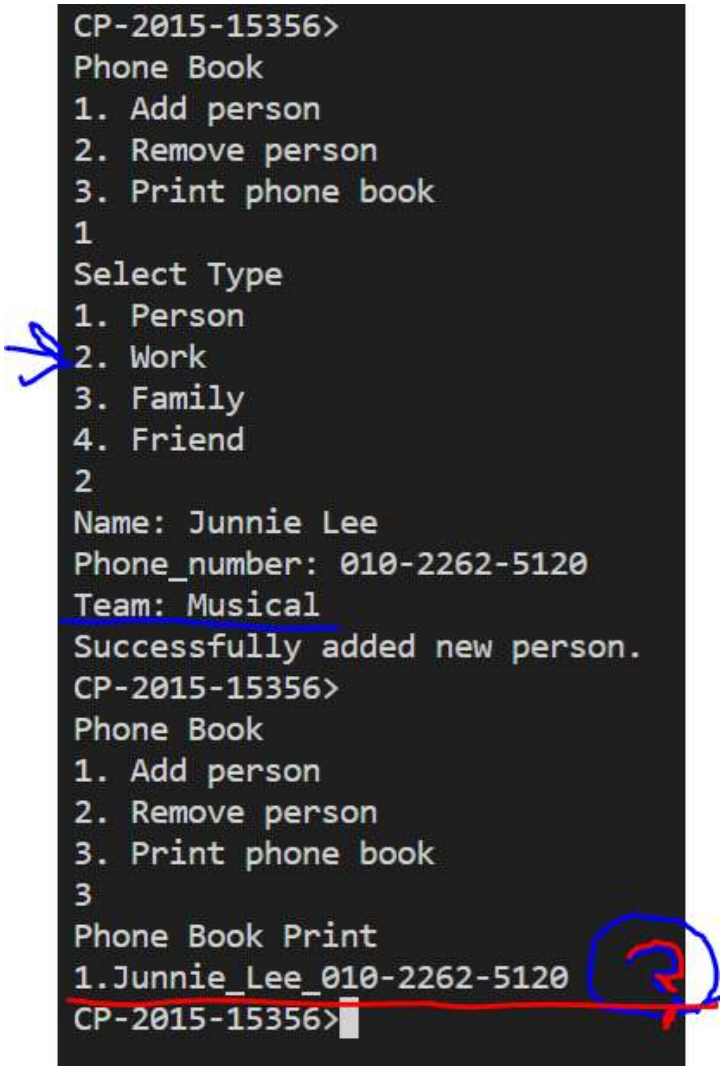
빨간 줄로 표시한 것과 같이 코드를 작성했더니, 위와 같은 오류가 발생해, new키워드를 통한 memory allocation을 해주어 문제를 해결했다. (아래 사진 첨부)

```
    if (select_type==1){ // add Person
        Person *p = new Person(first_name, last_name, phone_num);
        my_phone_book.add_one(p);
    }
    else if (select_type==2){ // add Work
        string team;
        cout<<"Team: ";
        getline(cin, team);

        Work *w = new Work(first_name, last_name, phone_num, team);
        my_phone_book.add_one(w);
    }
    else if (select_type==3){ // add Family
        string birthday;
        regex right_day_format("^\\d{6}");

        while (true){ // 올바른 입력값이 들어올때까지
            cout<<"Birthday: ";
            getline(cin, birthday);
            // phone_number 처리 -- format 맞게 들어왔는지 확인
            if (regex_match(birthday,right_day_format)) break;
            cout<<"[Error] : Wrong format. Please type again."<<endl;
        }

        Family *fam = new Family(first_name, last_name, phone_num, birthday);
        my_phone_book.add_one(fam);
    }
    else if (select_type==4){ // add Friend
        int age;
        cout<<"Age: ";
        cin>>age;

        Friend *fr = new Friend(first_name, last_name, phone_num, age);
        my_phone_book.add_one(fr);
    }
}
else if (num_choice=="2"){ //Remove Someone
```

<Trouble-Shooting Process ②>

두 번째로는 pointer type의 Vector를 선언해줬지만, child class method를 호출하지 못하는 문제가 발생했다. (오로지 parent class method만 호출이 되었다.)

아래 예시를 보면, 분명히 Work class타입의 멤버를 phone_book에 추가해줬는데, print를 명령했을 때 child class인 Work의 print 메소드를 호출하지 못한다. parent class의 print메소드에 묶여있다.



이 문제 또한 12페이지 상단에서처럼 각 class종류별로 각기 따로 dynamic memory allocation을 해 주었더니 문제가 해결되었다.

3. 출력값 확인:

① (정상 입력값의 경우)

```
CP-2015-15356>1
Select Type
1. Person
2. Work
3. Family
4. Friend
1
Name: Annie One
Phone_number: 010-1111-2222
Successfully added new person.
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
1
Select Type
1. Person
2. Work
3. Family
4. Friend
2
Name: Brea Two
Phone_number: 02-1234-1234
Team: Musical
Successfully added new person.
```

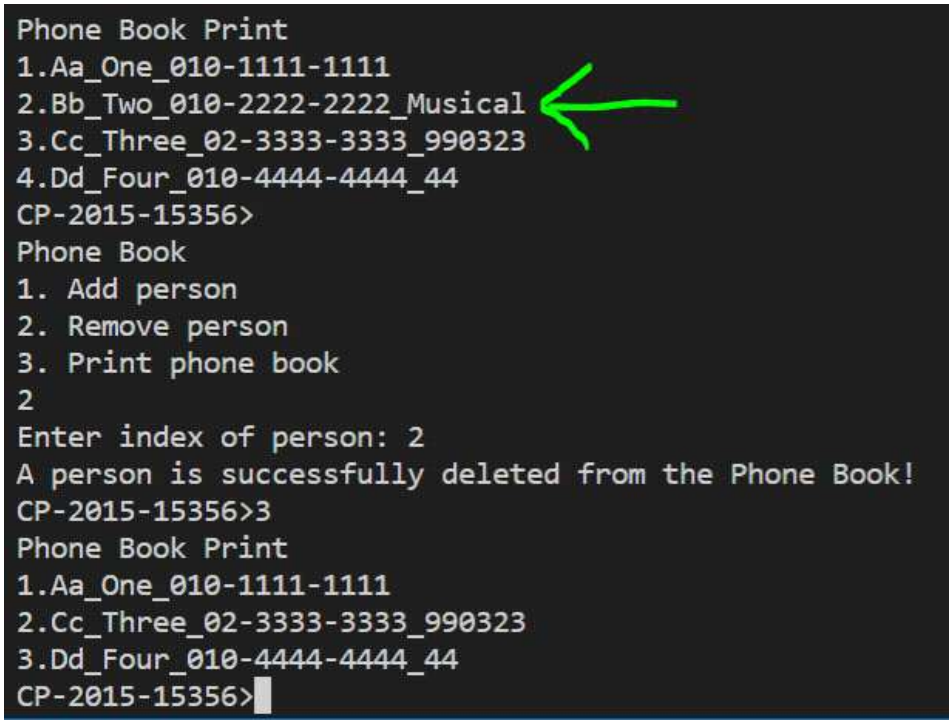
먼저 initial state에서 1을 입력하면 콘솔 출력 없이도 곧바로 Add Person이 가능하다. 정상적으로 1. Person type을 추가해줬다. 두 번째로, initial state에서 엔터를 치면 Phone Book에서 수행할 수 있는 항목 세 가지가 출력된다. 여기서 어떠한 명령을 수행할 것인지 1,2,3중 선택한다. 1을 선택했다. 다음으로 select type은 2를 선택해 work 오브젝트를 생성시켜준다.

```
CP-2015-15356>1
Select Type
1. Person
2. Work
3. Family
4. Friend
3
Name: Carol Three
Phone_number: 010-9999-1111
Birthday: 990323
Successfully added new person.
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
1
Select Type
1. Person
2. Work
3. Family
4. Friend
4
Name: Dylan Four
Phone_number: 010-4444-5555
Age: 99
Successfully added new person.
```

곧이어 각각 select type3 Family와 select type4 Friend를 추가해주었다.

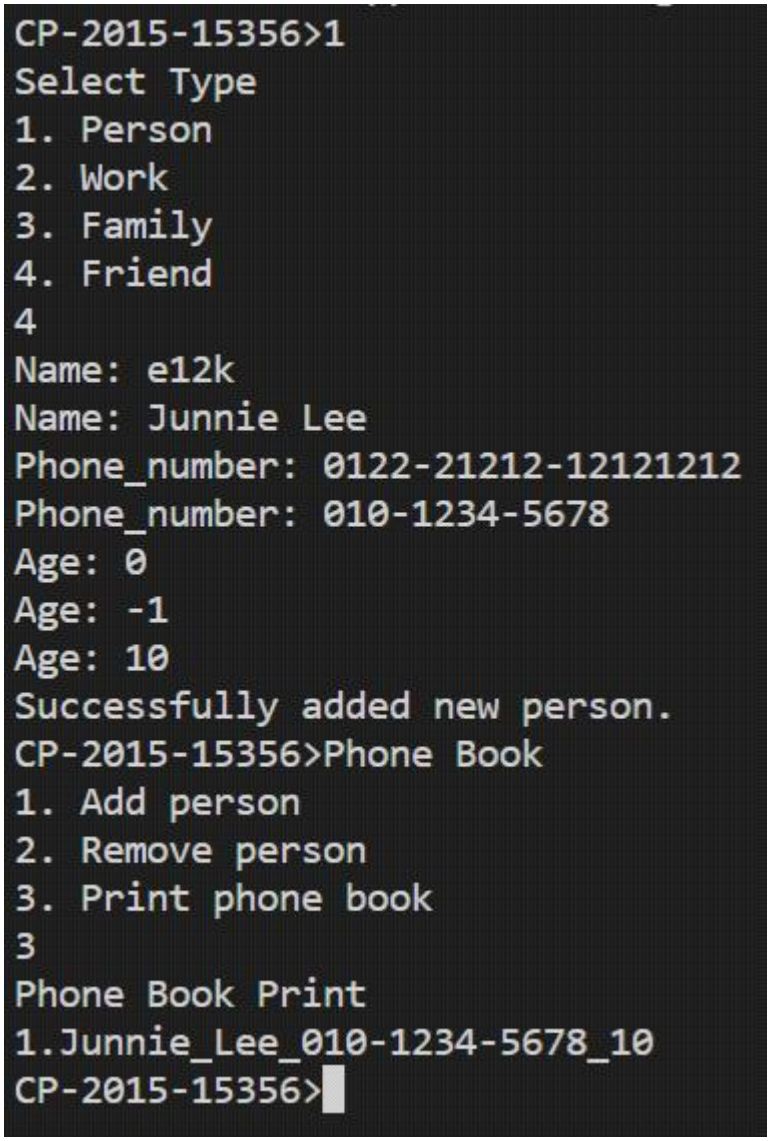
```
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
3
Phone Book Print
1.Annie_One_010-1111-2222
2.Brea_Two_02-1234-1234_Musical
3.Carol_Three_010-9999-1111_990323
4.Dylan_Four_010-4444-5555_99
CP-2015-15356>
```

현재까지 추가한 내역들이 다 정상적으로 PhoneBook 클래스에 들어갔는지 확인하기 위해 3번째 명령을 선택해 PhoneBook을 프린트했다. 앞서 추가해준 네 개 오브젝트들이 각각 정상적으로 잘 추가된 것을 확인해 볼 수 있다.



이번엔 삭제기능이 잘 작동하는지 확인해보기 위해서 세 번째 명령어를 선택해 보았다. 기존에 2번 인덱스에 저장되어 있던 Bb_Two를 삭제했다. index를 입력하면, 해당 사람이 phone book으로부터 정상적으로 삭제되었다는 메시지가 뜬다. 프린트 해보면, 정상적으로 2번이 삭제되고, 기존의 2번자리에 Cc_Three가, 기존의 3번자리에 Dd_Four가 곧이어 새로 자리하고 있음을 확인할 수 있다.

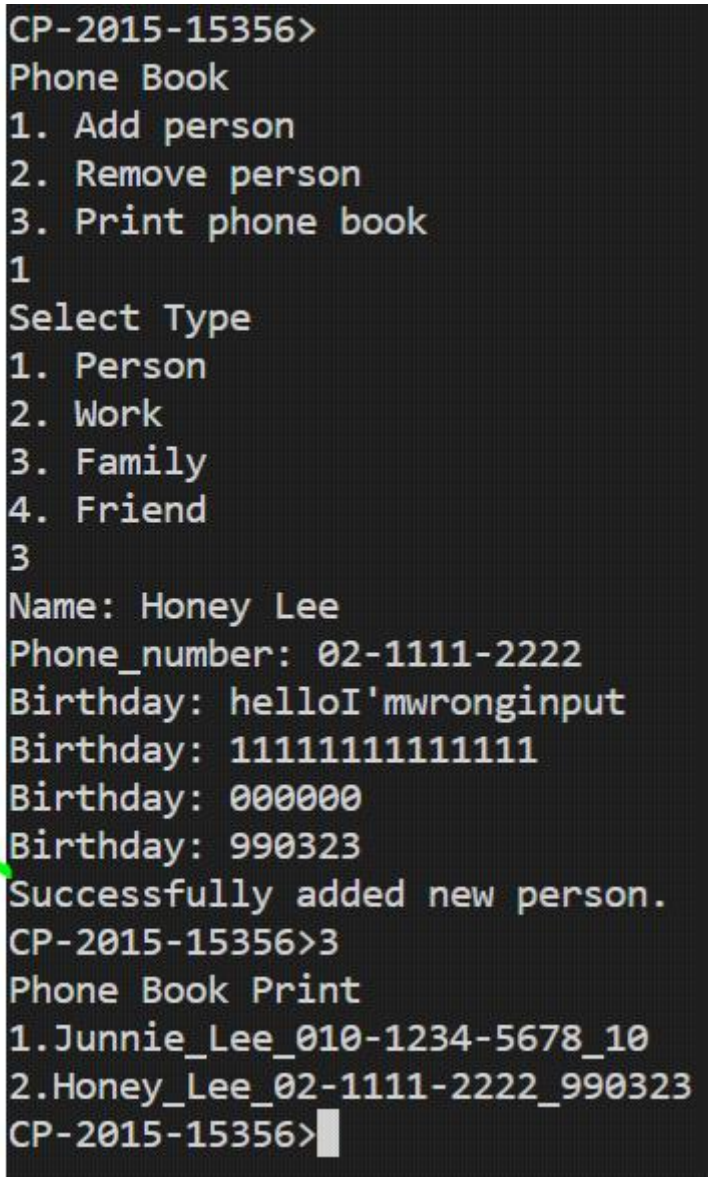
② (잘못된 입력값이 들어올 경우)



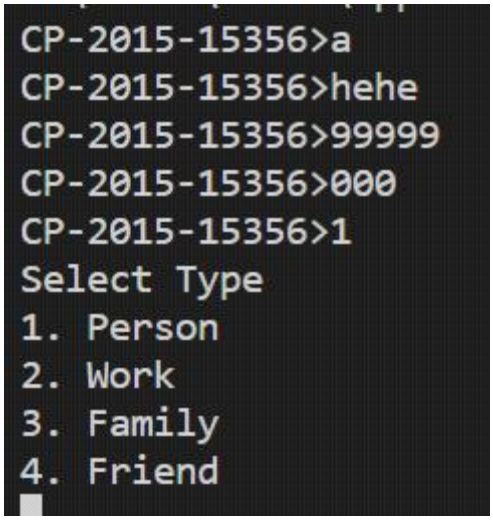
(1) 이름 - (알파벳)+(space 1칸)+(알파벳) 형식만 인풋으로 받는다.

(2) phone_number - (02-xxxx-xxxx) or (010-xxxx-xxxx) 형식이 들어올 때까지 반복해서 계속 재입력 받는다. (x로 표시한 부분은 digit을 의미한다)

(3) age – 자연수가 들어올 때까지 계속해서 다시 입력을 받는다.



(4) 생일 – 과제 q&a에 명시된대로 6자리 자연수만 받도록 예외처리를 해 주었다. 따라서 숫자가 아닌 문자는 받지 않으며, 000000도 결국 0을 의미해 자연수가 아니므로 이 또한 받지 않는다. 6자리 자연수가 들어오면 정상적으로 값을 저장하고 다음 단계로 넘어간다.

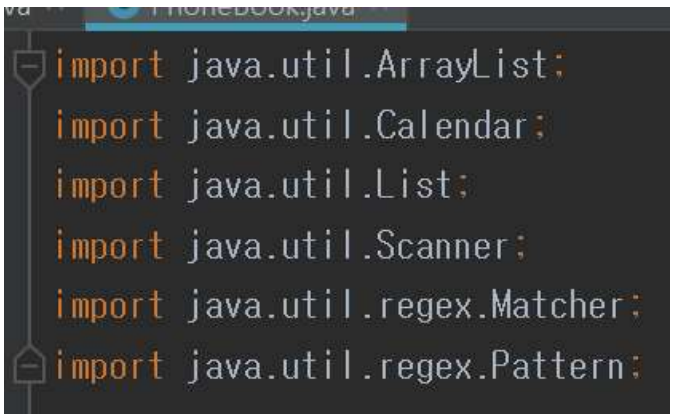


(5) initial state에 엔터, digit 1,2,3 이외의 input이 들어올 경우 인풋을 무시하고 계속 프롬프트를 띄운다. (올바른 입력이 들어올때까지 이 과정을 반복한다)
(위 스크린샷을 보면 1이라는 올바른 input이 들어오면 그제서야 그 다음단계를 올바르게 수행하는 것을 확인해볼 수 있다.)

<Version #2. Java>

1. 환경 : IntelliJ IDEA (JAVA 8), SDK11 (JRE 11.0.2), 윈도우 10

2. 코드설명 및 프로그래밍 과정 :
(전반적인 코드 구조는 앞서 설명한 c++코드와 상당히 유사하니, c++과는 다르게 구현한 부분들 위주로만 간략히 설명하도록 하겠습니다)



먼저 사용한 라이브러리들을 다음과 같다. d-day계산을 위해 Calender 라이브러리를 사용했고, 인풋을 받기 위해 Scanner, 인풋 유효성을 확인하기 위해 regex를 포함해주었다.

PhoneBook이라는 이름으로 자바 클래스 파일을 생성해 주었다.
PhoneBook이 우리 자바 클래스파일의 유일한 public class이다.

```
public class PhoneBook
{
    // public static final String INVALID_INPUT = "[Error] : Wrong format. Please type again.";
    public static final Pattern NAME_PATTERN = Pattern.compile("^([a-zA-Z]+)(₩₩s{1})([a-zA-Z]+)");
    public static final Pattern PHONE_NUM_PATTERN = Pattern.compile("^(02|010)(-₩₩d{4}){2}");
    public static final Pattern BIRTH_DAY_PATTERN = Pattern.compile("^₩₩d{6}");


    public PhoneBook() { phoneBook = new ArrayList<Person>(); }

    public void add_one(Person new_one) {
        this.phoneBook.add(new_one);
        System.out.println("Successfully added new person.");
    }
    public void delete_one(int idx) {
        if (this.phoneBook.size() < idx || idx<=0){
            System.out.println("Person does not exist!");
        }
        else{
            this.phoneBook.remove( index: idx-1);
            System.out.println("A person is successfully deleted from the Phone Book!");
        }
    }
    public void print_all() {
        System.out.println("Phone Book Print");
        for (int i=0; i < phoneBook.size(); i++){
            System.out.print((i+1)+".");
            phoneBook.get(i).print();
        }
    }

    private List<Person> phoneBook;
```

먼저 초반부에 이름, 전화번호, 생일 포맷을 static final로 선언해둔다. 이렇게 선언해준 애들은 추후 콘솔에서 사용한다.
Phone_Book 클래스의 유일한 private member는 Person인스턴트들을 요소로 지니는 phoneBook이라는 이름의 List이다. 자바 코드에서는 c++에서와는 달리 콘솔에서 유저인풋 인덱스와 프로그램 내부 상의 List 인덱스의 차이를 고려해 주지 않고 delete_one 메소드에 idx 인자가 전달되기 때문에, "Person does not exist"를 출력해주는 조건이 이전과는 조금 다르다. 예외적인 index값이 아니라면 List의 remove 메소드를 통해 해당 요소를 삭제해주고, 정상적 삭제내역을 알리는 메시지를 출력해준다.

Person, Friend, Work 클래스는 c++에서 정의한 것과 유사하므로 설명을 생략한다.

```java
class Family extends Person {

    public Family(String first, String last, String phone_num, String birth_day){
        super(first, last, phone_num);
        this.birthday = birth_day;
    }
    public void setBirthday(String birth_day) { this.birthday = birth_day; }
    public String getBirthday() { return this.birthday; }
    public int dDay(){ // d-day 계산하기
        int birthday_month = Integer.parseInt(birthday.substring(2,4));
        int birthday_day = Integer.parseInt(birthday.substring(4));
        return calculate_date_difference(birthday_month, birthday_day);
    }
    public void print(){
        System.out.println(this.getFirstName() + "_" + this.getLastName() + "_" + this.getPhoneNumber() + "_" + this.birthday);
    }

    private String birthday;
}
```

Family 클래스는 c++에서 작성한 것과 조금 다른데, dDay 메소드에선 일단 birthday private멤버에서 월,일만 int형태로 추출해 낸 뒤, 그 인자를 calculate_date_difference 함수에 인자로 전달해 그 결과값을 return한다. calculate_date_difference는 파일 맨 아래쪽에 따로 선언되어있는데, 자세한 내용은 다음과 같다.

```
// 보조 메소드들 모음

public int calculate_date_difference(int birthday_month, int birthday_day){

    long d_day;
    Calendar cal = Calendar.getInstance(); //일단 Calendar 객체를 만들어준다.

    int year = cal.get(Calendar.YEAR); // 현재 연도
    int month = birthday_month;
    int date = birthday_day;

    long now_day = cal.getTimeInMillis(); //현재 시간

    cal.set(year, month: month-1, date); //목표일을 cal에 set

    long event_day = cal.getTimeInMillis(); //목표일에 대한 시간

    if (now_day <= event_day){ // 역전되지 않는상황
        d_day = (event_day - now_day) / (60*60*24*1000);
    }else{ //역전되는 상황
        d_day = (now_day - event_day) / (60*60*24*1000);
        d_day = 365-d_day;
    }
    return (int)d_day;
}
```

생일의 월, 일을 각각 int형태의 인자로 받는다. 자바의 calender 라이브러리를 활용한 Calender 객체를 통해 현재 정보를 받아온다. 먼저 get(Calender.YEAR)를 통해 현재 연도를 가져와 year 변수에 int형으로 값을 할당해준다. 다음으로는 cal.getTimeInMillis()를 통해 오늘에 대한 시간값을 now_day에 받아 저장해준다.
앞서 목표일(생일)을 위해 준비한 인자값 year, month, date를 cal에 해당 값으로 set해준다. 그리고 그 형태로 getTimeInMillis()를 해주어 목표일에 대한 시간을 계산해 event_day에 넣어준다. 그 다음으로는 역전되지 않는 상황/ 역전되는 상황으로 case를 나눠 계산해주는 건 c++에서 구현한 것과 비슷하다. 다만, d_day를 계산할 때 그냥 둘의 차를 구하는 것에서 끝나는게 아니라, 60*60*24*1000으로 나눠주어야 며칠이 차이 나는지에 대한 정상적인 값이 나오는데, 이는 우리가 now_day와 event_day의 값을 cal.getTimeInMillis()를 통해 구했기 때문에 따로 처리 해줘야 하는 부분이다 (getTimeInMillis()은 천분의 일초 단위를 사용해 아웃풋을 내주기 때문에, 1000으로 나누면 초 차이값이 나오고, 거기서 60*60*24를 더 나누면 날짜 차이가 나온다. 따라서 총 1000*60*60*24로 나눠줘야 올바른 날짜 차이값이 나온다.).
여기서, d_day는 long형으로 선언해 계산을 진행해주었지만, return형태는 int이므로 return문 부분에서 int로 캐스팅해주어 반환한다.

클래스들에 대한 설명이 끝났으니, 다음으로는 곧바로 콘솔을 구현한 main함수를 살펴보도록 한다.

```
// interactive console
public static void main(String[] args) {
    PhoneBook my_phoneBook = new PhoneBook();
    boolean exit = false;
    //Scanner scanner = new Scanner(System.in);

    while (!exit) {
        int choice=0;
        System.out.print("CP-2015-15356>");
        Scanner scanner = new Scanner(System.in);
        String initial_input = scanner.nextLine();
        boolean INT_TYPE_INPUT = my_phoneBook.IsInt_ByException(initial_input);

        if (initial_input.equals("exit")) {return;}
        else if (initial_input.equals("")) { // 엔터키를 쳤다면
            System.out.println("Phone Book₩n1. Add person₩n2. Remove person₩n3. Print phone book");
            while (true) {
                choice = scanner.nextInt();
                if (choice==1 | choice==2 | choice==3) break;
                // System.out.println(INVALID_INPUT);
            }
        }
        else if (INT_TYPE_INPUT){
            int num = Integer.parseInt(initial_input);
            if (num==1||num==2||num==3){
                choice = Integer.parseInt(initial_input);
            }
        }
        else{ // 그 외 쌩뚱맞은 input이면 다시 initial state로
            // System.out.println(INVALID_INPUT);
            continue;
        }
```

전반적인 형태는 c++에서와 유사하지만, 부분적으로 다른 부분들이 있다. initial state에서 1,2,3을 입력했을 때 첫 번째, 두 번째, 세 번째 명령문에 해당하는 코드를 실행하게 하기 위해선 1,2,3을 제대로 입력받아 choice 변수에 할당해주어야 하는데, paseInt를 통해 string을 int로 바꿔 1,2,3인지를 확인하려면, 먼저 해당 string이 int로 바꿔질 수 있는 모양인지 사전에 확인해야 하는 작업이 필요했다 (초반엔 이 작업을 해주지 않았다가, 컴파일 에러가 났다). 이 작업은 initial_input을 인자로 넣어주었을 때 return되는 .isInt_ByException메소드의 boolean 반환값을 통해 처리해주었다. .isInt_ByException메소드는 파일 하단에 다음과 같이 정의해주었다.

```java
// (string으로 들어온 input이 int형태로 변환 가능한지 확인)
public boolean IsInt_ByException(String str){
    try
    {
        Integer.parseInt(str);
        return true;
    }
    catch(NumberFormatException nfe)
    {
        return false;
    }
}
```

먼저 들어온 input값에 대해 parseInt를 시도해보고, 성공하면 true를, 도중에 int형태로 변환시킬 수 없어 NumberFormatException 예외가 발생한다면 false를 return 한다.

다시 main함수로 돌아가보면,

```java
        // choice가 제대로 들어왔을 때
        switch (choice) {
            case 1: // Add
                System.out.println("Select Type₩n1. Person₩n2. Work₩n3. Family₩n4. Friend");
                int select_type = scanner.nextInt();
                my_phoneBook.Add_to_PhoneBook(select_type);
                break;
            case 2: // Delete
                my_phoneBook.Delete_from_PhoneBook();
                break;
            case 3: // Print
                my_phoneBook.print_all();
                break;
        }
    }// end of while문
} // end of main 함수
```

choice가 무엇이 들어왔냐(int 타입 인풋)에 따라 각기 다른 코드를 실행한다. c++에서와는 달리 switch문을 사용해주었다. add를 할 경우 Add_to_PhoneBook메소드를, delete할 경우 Delete_from_PhoneBook()메소드를, print 할 경우 print_all메소드를 사용하도록 해주었는데, java에서 코딩할땐, main함수의 코드를 좀더 간결하게 하기 위해 c++에서와는 달리 Add_to_PhoneBook과 Delete_from_PhoneBook() 메소드를 main함수 밖에 따로 선언해주었다. 각각의 자세한 코드 내용은 다음과 같다.

```java
public void Add_to_PhoneBook(int select_type){
    Scanner s = new Scanner(System.in);

    // name 받기
    String name;
    while (true){
        System.out.print("Name: ");
        name = s.nextLine();
        // name 처리 -- format 맞게 들어왔는지 확인
        Matcher m = NAME_PATTERN.matcher(name);
        boolean right_format = m.matches();
        if (right_format) break;

        // System.out.println(INVALID_INPUT);
    }

    String[] splitStr = name.split( regex: "₩₩s+");
    String first_name = splitStr[0];
    String last_name = splitStr[1];

    String phone_number;
    // phone number 받기
    while (true){
        System.out.print("Phone_number: ");
        phone_number = s.nextLine();
        // phone_number 처리 -- format 맞게 들어왔는지 확인
        Matcher m = PHONE_NUM_PATTERN.matcher(phone_number);
        boolean right_format = m.matches();
        if (right_format) break;

        // System.out.println(INVALID_INPUT);
    }
```

PhoneBook에 요소를 add하기 위한 메소드의 코드 앞부분이다. 기본 문법적인 부분들 외엔 c++의 main함수에서 새로운 요소를 전화번호부에 추가하기 위해 해주었던 작업과 동일하다. 정규표현식을 통해 잘못된 인풋형태를 걸러주고, 올바른 인풋이 들어올때까지 반복적으로 입력을 받는다.

어떤 형태의 요소를 추가하든, name과 phone_number를 받아야 하는 것은 동일하니 case를 나눠주기 전에 먼저 그 두 필드를 입력받는다.

```
if (select_type==1){ // add Person
    this.add_one(new Person(first_name, last_name, phone_number));
}
else if (select_type==2){ // add Work
    System.out.print("Team: ");
    String team = s.nextLine();
    this.add_one(new Work(first_name, last_name, phone_number, team));
}
else if (select_type==3){ // add Family
    String birthday;
    while (true){
        System.out.print("Birthday: ");
        birthday = s.nextLine();
        // birthday format 맞게 들어왔는지 확인
        Matcher m2 = BIRTH_DAY_PATTERN.matcher(birthday);
        boolean right_format2 = m2.matches();
        if ((right_format2) && (birthday!="000000")) break; // 생일은 6자리 자연수만 받기
        // System.out.println(INVALID_INPUT);
    } // 입력받기 끝
    this.add_one(new Family(first_name, last_name, phone_number,birthday));
}
else if (select_type==4){ // add Friend
    int age;
    while (true){
        System.out.print("Age: ");
        age = s.nextInt();
        if (age > 0) break; // 나이는 자연수 형태만 받기
    }
    this.add_one(new Friend(first_name, last_name, phone_number,age));
}
```

위 코드는 select_type에 따라 if/else문으로 케이스를 나눠준 것이다. 역시나, 기본적인 문법차이 외에 전반적인 구조는 c++에서 구현한것과 동일하다.

다음으로는, 요소를 삭제하기 위해 따로 main함수 밖에 만들어준 메소드를 살펴본다.

```java
public void Add_to_PhoneBook(int select_type){
    Scanner s = new Scanner(System.in);

    // name 받기
    String name;
    while (true){
        System.out.print("Name: ");
        name = s.nextLine();
        // name 처리 -- format 맞게 들어왔는지 확인
        Matcher m = NAME_PATTERN.matcher(name);
        boolean right_format = m.matches();
        if (right_format) break;

        // System.out.println(INVALID_INPUT);
    }

    String[] splitStr = name.split( regex: "\\s+");
    String first_name = splitStr[0];
    String last_name = splitStr[1];

    String phone_number;
    // phone number 받기
    while (true){
        System.out.print("Phone_number: ");
        phone_number = s.nextLine();
        // phone_number 처리 -- format 맞게 들어왔는지 확인
        Matcher m = PHONE_NUM_PATTERN.matcher(phone_number);
        boolean right_format = m.matches();
        if (right_format) break;

        // System.out.println(INVALID_INPUT);
    }
```

Add_to_PhoneBook 메소드의 코드 앞부분이다.
정규표현식을 이용해 과제스펙에서 명시한 formate의 올바른 값이 들어올 때까지 while문을 돌리는 거나, space를 기준으로 first name, last name을 나누는 거나, case를 나눠 기능을 수행하기 전에 모든 케이스에 공통되는 name과 phone_number 필드를 입력받는 등 전반적인 구조는 c++에서 구현한 것과 거의 동일하다.

```
if (select_type==1){ // add Person
    this.add_one(new Person(first_name, last_name, phone_number));
}
else if (select_type==2){ // add Work
    System.out.print("Team: ");
    String team = s.nextLine();
    this.add_one(new Work(first_name, last_name, phone_number, team));
}
else if (select_type==3){ // add Family
    String birthday;
    while (true){
        System.out.print("Birthday: ");
        birthday = s.nextLine();
        // birthday format 맞게 들어왔는지 확인
        Matcher m2 = BIRTH_DAY_PATTERN.matcher(birthday);
        boolean right_format2 = m2.matches();
        if ((right_format2) && !(birthday.equals("000000"))) break; // 생일은 6자리 자연수만 받기
        // System.out.println(INVALID_INPUT);
    } // 입력받기 끝
    this.add_one(new Family(first_name, last_name, phone_number,birthday));
}
else if (select_type==4){ // add Friend
    int age;
    while (true){
        System.out.print("Age: ");
        age = s.nextInt();
        if (age > 0) break; // 나이는 자연수 형태만 받기
    }
    this.add_one(new Friend(first_name, last_name, phone_number,age));
}
```

Add_to_PhoneBook 메소드의 코드 뒷부분이다.
case에 따라 해야 할 일을 나눠준 부분은 c++의 main함수에서 써주었던 것과 동일한 구조로 작동한다. 단, java에서는 명시적으로 포인터나 주소값을 인자로 전달하지 않는다는 점이 c++에서와 다르다.

다음으로는 전화번호부에 있는 항목을 삭제하기 위해 main함수 밖에 선언해준 메소드 Delete_from_PhoneBook()을 확인해본다.

```
    public void Delete_from_PhoneBook(){
        Scanner s = new Scanner(System.in);
        // 삭제할 index 받기
        System.out.print("Enter index of person: ");
        int idx = s.nextInt();
        this.delete_one(idx);
    }

} // end of PhoneBook classs
```

이또한 c++에서와 구조가 굉장히 유사한데, 유일하게 다른점은 콘솔차원에서 사용자인풋의 index와 list내 index 차이를 고려해 delete_one에 인자전달을 하지 않는다는 점이다. 사용자가 입력한 그대로의 int값을 delete_one메소드의 인자로 전달하고 있기 때문에, 두 index차이는 앞쪽에 이미 서술한 바와 같이 delete_one내에서 처리해준다.

// 이렇게 java코드 전반이 마무리된다.

<Trouble-Shooting Process ①>

```
class Friend extends Person {

    public Friend(String first, String last, String phone_num, int age){
        super(first, last, phone_num);
        this.age = age;
    }
    public void setAge(int age) { this.age = age; }
    public int getAge() { return this.age; }
    public void print(){
        System.out.println(this.firstName + "_" + this.lastName + "_" + this.phoneNumber + "_" + this.age);
    }

    private String firstName;
    private String lastName;
    private String phoneNumber;
    private int age;
}
```

이렇게 private 멤버들을 따로 선언하고 그들을 explicitly 호출했더니 null_null_null_나이 --> 이런 식으로 출력이 되는 문제가 발생했다. 이에 대해서

이렇게 get 메소드를 사용해 간접적으로 접근해주었더니 정상 출력되어 문제가 해결되었다.

3. 출력값 확인:

① (정상 입력값의 경우)

먼저 각 class type별로 정상적인 추가가 가능한지 확인한다.

```
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
1
Select Type
1. Person
2. Work
3. Family
4. Friend
1
Name: Junnie Lee
Phone_number: 010-2262-5120
Successfully added new person.
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
1
Select Type
1. Person
2. Work
3. Family
4. Friend
2
Name: Honey Lee
Phone_number: 010-1234-5677
Team: Musical
Successfully added new person.
```

```
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
1
Select Type
1. Person
2. Work
3. Family
4. Friend
3
Name: Youjin Seo
Phone_number: 010-4444-1111
Birthday: 990323
Successfully added new person.
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
1
Select Type
1. Person
2. Work
3. Family
4. Friend
4
Name: Jihoon Joo
Phone_number: 02-1212-1111
Age: 25
Successfully added new person.
```

4가지 class 타입 모두가 phonebook에 정상적으로 추가된다.

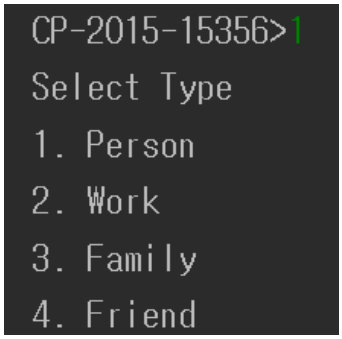
```
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
3
Phone Book Print
1.Junnie_Lee_010-2262-5120
2.Honey_Lee_010-1234-5677_Musical
3.Youjin_Seo_010-4444-1111_990323
4.Jihoon_Joo_02-1212-1111_25
```

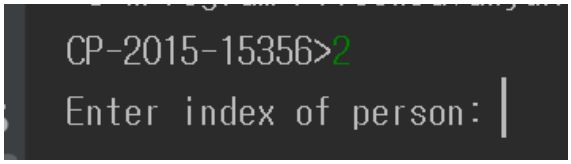
앞서 추가한 항목들을 출력해 보았다.

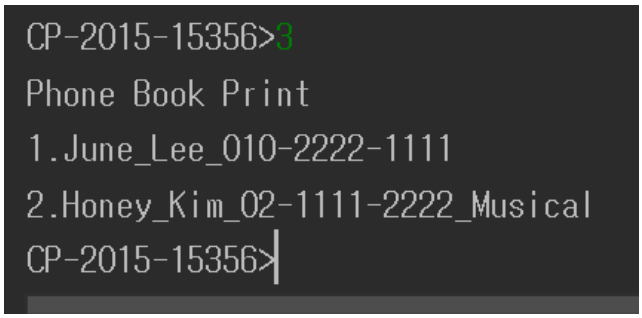
```
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
2
Enter index of person: 3
A person is successfully deleted from the Phone Book!
CP-2015-15356>
Phone Book
1. Add person
2. Remove person
3. Print phone book
3
Phone Book Print
1.Junnie_Lee_010-2262-5120
2.Honey_Lee_010-1234-5677_Musical
3.Jihoon_Joo_02-1212-1111_25
CP-2015-15356>
```

기존에 있던 항목에서 3번째 항목을 삭제 한 뒤, 다시 전화번호부 전체를 출력해 보았다. 기존의 세 번째 항목이 정상적으로 삭제되고, 네 번째 항목이 세 번째를 대

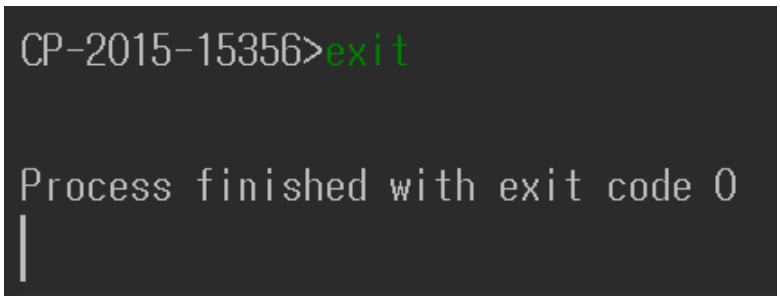
신한 것을 확인할 수 있다.





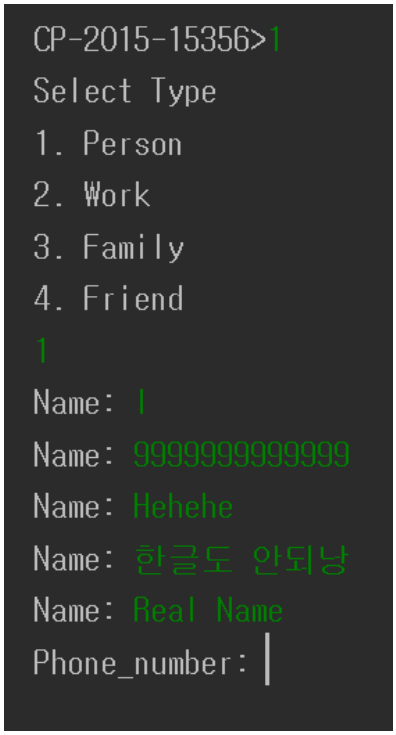


initial state에서 엔터를 쳐서 항목들을 print하지 않고도
곧바로 선택하고자 하는 항목의 번호를 입력하면 해당 기능을 사용할 수 있다.

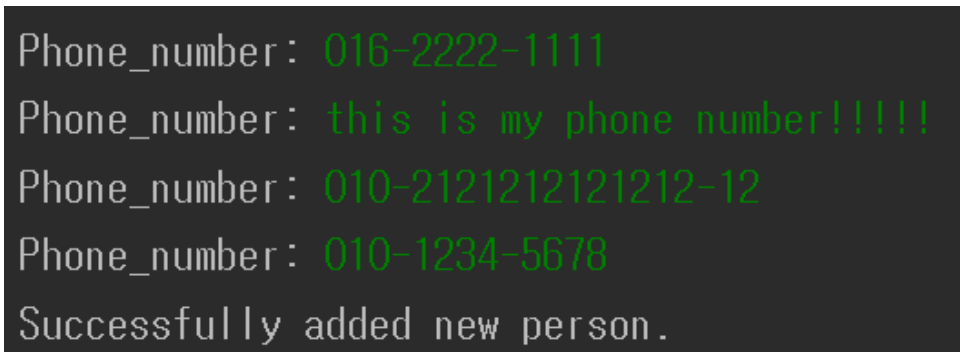


initial state에서 exit을 입력할 경우 정상적으로 프로그램이 종료된다.

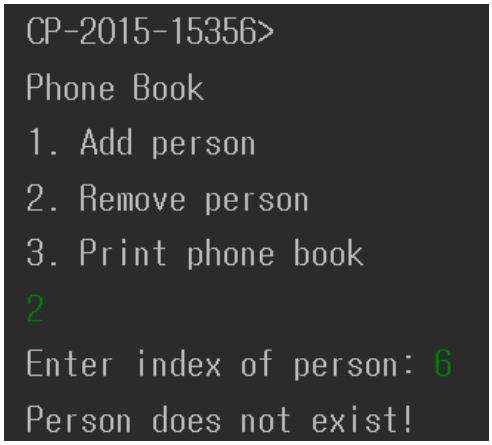
② (잘못된 입력값이 들어올 경우)



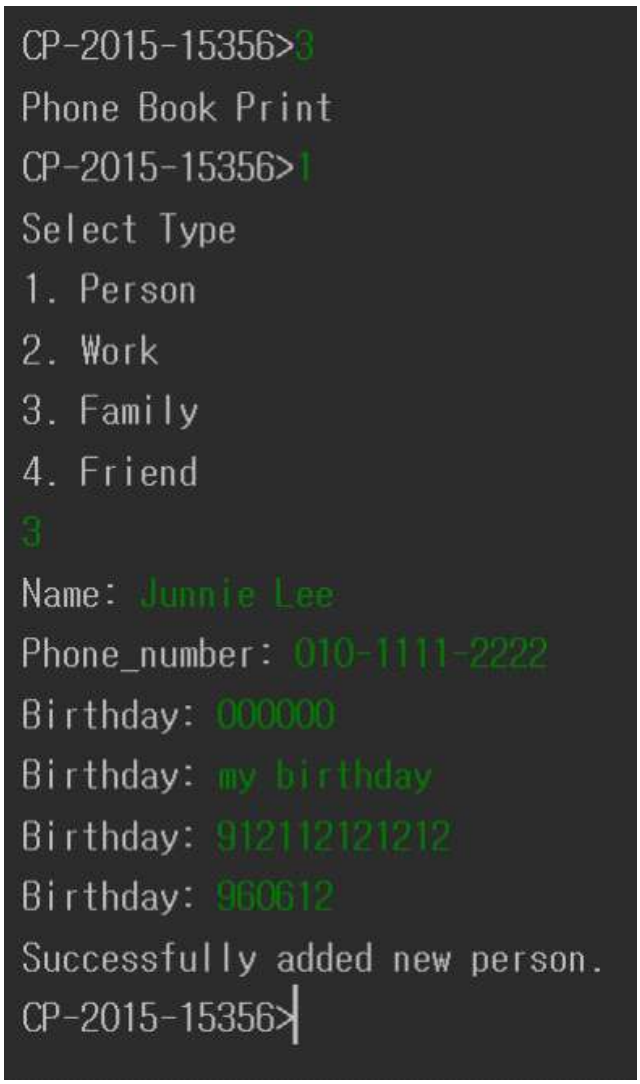
1 or more 알파벳, 띄어쓰기 1칸, 1 or more 알파벳 형식을 맞추기 않은 string은 name input으로 받아들이지 않는다. (즉, 올바른 인풋을 넣을 때까지 반복해 입력을 받는다.)



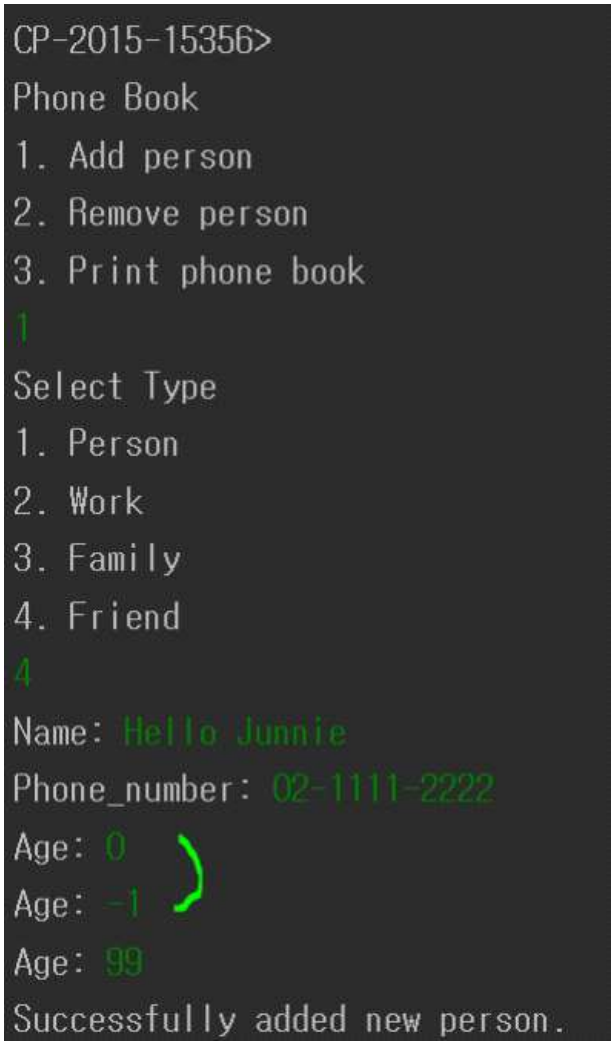
마찬가지로 잘못된 전화번호 형식이 들어올 경우, 올바른 인풋이 들어올때까지 반복해서 다시 입력받는다.



index range가 벗어난 항목을 삭제해달라고 요청할 경우, 존재하지 않는다는 에러 메시지를 띄워준다.



잘못된 생일 형식이 들어올 경우, 다시 입력받는다.
(올바른 입력값이 들어올 때까지 이 과정을 반복한다)



나이 또한 자연수가 아닌 값을 입력할 경우, 자연수인 인풋을 입력할 때까지 그 전 입력값을 무시하고 계속해서 다시 입력받는다.