2023 September 27

**자료구조와 알고리즘 과제제출**

실습보고서

로고이(가) 표시된 사진

자동 생성된 설명

**담당교수 : 서두옥교수님**

**학과 : 자동차it융합과**

**학번 : 20215229**

**이름 : 이채훈**

**제출일 : 2023년9월27일**

**1. 이진수를 십진수로 변환하는 함수**

**소스코드**

#include <iostream>

#include <string>

using namespace std;

bool validateBinary(int num);

int binaryToDecimal(int num);

int main() {

int num;

int decimal;

cout << "20215229 이채훈" << endl;

while (true) {//이진수 나올때까지 계속 반복

cout << endl;

cout << "이진수(1 또는 0): ";

cin >> num;

if (validateBinary(num)) {

decimal = binaryToDecimal(num);

cout << endl;

cout << "입력한 이진수: " << num << endl;

cout << "변환 된 십진수 : " << decimal;

break;

}

else {

cout << "이진수(1또는 0)만 입력 가능합니다!!!";

}

}

}

bool validateBinary(int num) {

string str = to\_string(num);//입력받은 정수를 문자열로 전환

for (int i = 0; i < str.size(); i++) {

if (str[i] > '1') {

return false;//10진수면 false

}

}

return true;//2진수면 true

}

int binaryToDecimal(int num) {

int sum = 0;

string str = to\_string(num);//입력받은 정수를 문자열로 전환

for (int i = str.size() - 1; i >= 0; i--) {

if (str[i] == '1') {

sum += 1 << (str.size() - 1 - i);//bit shift 연산을 통해 2진수 계산

}

}

return sum;

}

**실행결과**

**텍스트, 스크린샷, 디스플레이, 컴퓨터이(가) 표시된 사진

자동 생성된 설명**

**연구조사**

**알고리즘**

1. num값 입력받기

2. validateBinary(int num)

2-1. 입력 받은 정수를 to\_string을 이용하여 문자열로 변환

2-2. for문을 사용해, 각 index의 값이 1보다 큰지 확인

2-3 . 1보다 크면 false(10진수), 작으면 true(2진수)을 반환한다.

3-1. 반환값이 true일 경우, binaryToDecimal(int num) 실행

3-1-1. 입력받은 정수를 to\_string을 이용하여 문자열로 변환

3-1-2. for문, if문, Bitwise operator

for문 : 문자열로 변환된 이진수의 오른쪽에서 첫째 자리부터 접근

if문 : 자릿수가 1인 경우

bitwise operator : str.size()-1-i 만큼 왼쪽 시프트 연산자를 통해 이진수 계산

3-1-3. 계산된 값을 sum에 누적 합산

3-2. 반환값이 false인 경우 while(true)이므로, 재입력을 받는다.

**\*\*주요 사용 기능\*\***

**string 라이브러리의 주요기능 중 to\_string(num) 사용**

to\_string(num) : 숫자형을 string 타입으로 변환

**string 라이브러리 주요 내장 함수 공부**

1. size() : 해당 문자열의 사이즈 반환

\*length()함수와 동일한 기능

2. capacity() : 문자열 객체에 할당된 메모리 크기 반환

3. resize() : 문자열 객체를 n만큼의 크기로 만듦. 원래 크기가 n보다 크면 남은 부분은 버림. n보다 작으면 남은 공간을 빈공간으로 채움.

4. clear() : string 객체에 있는 문자열을 지우는 함수

\*size, length는 모두 0이 되고 capacity는 그대로 유지됨.

5. empty() : 문자열 객체가 비어있는지 확인하는 함수, bool타입으로 반환

6. at() : 해당 인덱스 문자 반환

7. front(), back() : 문자열의 가장 앞과 뒤를 반환

8. substr(index, len) : 문자열의 index부터 len만큼 잘라서 반환

9. replace(s, len, str) : 문자열(s)의 인덱스 위치에서 len까지 범위를 str로 대체

10. compare() : 매개변수로 들어온 문자열과 비교해서 같으면 0반환, 다르면 -1반환

11. copy(char\*arr, len, index=0) : 문자열을 arr에 index부터 len만큼 복사

12. find(string str, index=0) : 매개변수로 들어온 문자열과 비교하려는 문자열 중에 일치하는게 있는지 확인, 일치하는게 있다면 첫번째 발견 인덱스 반환, index값부터 있는지, 확인, 없다면 쓰레기 값 반환

13. begin() : 문자열의 첫번째 문자를 가리키는 반복자(iterator) 반환

14. end() : 문자의 마지막의 바로 문자를 가리키는 반복자(iterator) 반환

15. swap() : s1문자열과 s2문자열을 바꾸는 것

16. append() : 문자열 끝에 더함

17. assign() : 문자열을 할당하는 함수

18. insert(index, string s) : 문자열을 지정한 위치에 삽입하는 함수

19. erase(index, num) : 문자열을 지우는 함수(인덱스부터 num개의 문자를 지움)

20. to\_string(num) : 숫자형을 string타입으로 변환(char형 숫자이면 아스키 코드로 변환)

21. stoi(str) : 문자열을 숫자로 반환

22. int to char : 숫자를 아스키 코드를 통해 문자로 반환

23. toupper(char), tolower(char) : 해당 문자를 대문자, 소문자로 변환

**bitwise left shift(<<)**

bitwise left shift(<<) : 2진수 계산을 비트 단위 연산자를 통해서 구현함.

예시)

3 = 0011

3 << 1 = 0110 = 6

3 << 2 = 1100 = 12

3 << 3 = 1000 = 8

이진수의 끝에서 벗어난 비트는 손실된다. 여기서는 4bit로 가정하고 있으므로, 8bit일 경우

3 << 3은 11000이 되어 24가 된다. 이 경우 4번째 비트는 5번째 비트로 이동되므로 손실되지 않는다.

**2. 데이터 정렬: 정적 배열과 동적 배열**

**소스코드**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int arr[10];

vector<int> v;

void inputValues() {

cout << "임의의 정수 10개를 입력하세요..." << endl;

for (int i = 0; i < 10; i++) {

cout << "arr[" << i << "] : ";

cin >> arr[i];

v.push\_back(arr[i]);

}

}//값 입력 및 정적, 동적 배열에 값 추가

int main(void) {

cout << "20215229 이채훈" << endl;

cout << endl;

inputValues();

sort(arr, arr + 10);//정적 배열 오름차순 정렬

cout << "정렬(정적 배열) : ";

for (int i = 0; i < 10; i++) {

cout << arr[i] << " ";

}//정적 배열 출력

cout << endl;

sort(v.begin(), v.end());//동적 배열 오름차순 정렬

cout << "정렬(동적 배열) : ";

for (int i = 0; i < v.size(); i++) {

cout << v[i] << " ";

}//동적 배열 출력

return 0;

}

**실행결과**

**텍스트, 스크린샷, 디스플레이, 컴퓨터이(가) 표시된 사진

자동 생성된 설명**

연구조사

알고리즘

1. 정수를 입력받는 inputValues()함수를 호출해서, 정적배열과 동적배열에 값 추가

2. algorithm 라이브러리의 내장함수sort()를 사용해서, 정적배열, 동적배열 오름차순 정렬

3.정적배열에서는 for문의 조건문에 10을, 동적배열에서는 vector라이브러리의 size()내장함수를 사용해서 배열의 크기를 조건문에 추가하여, 배열 출력

**\*\*주요 사용 기능\*\***

**STL(표준 템플릿 라이브러리)**

: STL(Standard Template Library), 표준 템플릿 라이브러리. 벡터, 리스트, 스택, 큐 등의 컨테이너와 이들을 처리하기 위해 사용할 수 있는 여러가지 연산을 포함함.

* 컨테이너(Container) : 데이터 저장
* 반복자(iterartor) : 포인터의 역할
* 알고리즘(algorithm) : 데이터 처리 기능

**컨테이너(Container)**

: 같은 타입의 객체를 저장하는 일종의 집합, 클래스 템플릿으로 컨테이너 변수를 선언할 때 컨테이너에 포함할 요소의 타입을 명시할 수 있음.

* int나 float와 같은 기본자료형 데이터나 사용자 정의 클래스의 객체 등을 저장함.
* 다양한 연산이 제공되어 편리하게 데이터를 활용 가능함.
* 순차 컨테이너 : 동일한 자료형의 객체들을 선형적인 구조로 저장(vetor, list, deque)
* 연상 컨테이너 : 탐색 트리와 같은 인덱스 구조를 이용하는 컨테이너로 키(key)를 이용한 검색 기능을 제공(set, multiset, map, multimap)
* 무순서 연상 컨테이너 : 연상 컨테이너와 같으나 해시함수를 이용하여 검색(unordered-set, unordered-multiset, unordered-map, unordered—multimap)
* 컨테이너 어댑터 : 기본 컨테이너를 기반으로 특정 용도에 맞게 유도된 컨테이너(queue, priority-queue, stack)

**Vector**

: 1차원 배열의 개념을 구현한 순차 컨테이너 유형의 클래스 템플릿, 배열의 일반적인 기능을 포함하면서 여러가지 유용한 멤버함수 및 관리기능이 도입되어 있음. 배열처럼 크기가 고정되어 있지 않고, 필요에 따라 저장 공간을 확장할 수 있다는 특징이 있음.

* 벡터 객체는 요소가 추가되거나 삭제될 때마다 자동으로 메모리를 재할당하여 크기를 동적으로 변경함.
* 해당 컨테이너는 vector 헤더 파일에 저장되어 있으며, 임의 접근을 제공하는 가장 간단한 시퀀스 컨테이너임.

# Vector 객체 선언 구문

· 템플릿 인수로는 벡터 컨테이너에 저장될 요소의 타입을 전달합니다.

· 생성자 인수로는 벡터 컨테이너의 초기 크기를 전달하며, 생략하면 요소를 가지지 않는 빈 벡터를 생성합니다.

vector<템플릿인수> 객체이름(생성자인수);

# Vector 객체 선언 및 초기화

vector<int> vInt = {1, 2, 3};

# vector 멤버 함수

at(): 직접접근을 위한 멤버함수, 지정된 첨자가 범위를 벗어날 경우 예외 발생

size(): 벡터의 논리적인 사이즈 반환

capacity(): 벡터의 물리적인 사이즈 반환, 실제 할당된 메모리의 크기

push\_back(): 벡터의 끝에 데이터를 삽입

pop\_back(): 벡터의 끝에 데이터를 꺼냄

insert(it, value): 지정된 위치에 데이터를 삽입 (반복자 it가 가리키는 위치에 value 삽입)

erase(it): 지정된 위치에 데이터를 삭제 (반복자 it가 가리키는 위치 삭제)

erase(it1, it2): [it1, it2] 범위의 데이터 삭제

resize(n): 논리적 크기 변경

reserve(n): capacity()가 최소한 n을 반환하도록 확장

empty(): 비어 있는 벡터의 경우 true

sort(first, last): 랜덤 액세스 반복자에 의해 지정된 범위의 값들을 정렬, first ~ last 범위의 값들을 정렬

sort(first, last, comp): comp는 정렬 순서를 정하는 함수(callback), 정렬 순서에 대한 기준을 판단할 수 있는 함수

merge(first1, last1, first2, last2, dest)

merge(first1, last1, first2, last2, dest, comp)

- 동일한 기준으로 정렬된 두 개의 데이터 집합을 동일한 기준으로 정렬된 하나의 데이터 집합으로 결합

- first1, last1: 첫 번째 정렬된 데이터 범위

- first2, last2: 두 번째 정렬된 데이터의 범위

- dest: 합병 결과가 저장될 시작 위치

- comp: 합병 순서를 정하는 함수

# 반복자의 값을 구하는 vector의 멤버함수

begin(): 첫 번째 원소를 가리키는 랜덤 액세스 반복자를 반환

end(): 마지막 원소의 다음 위치를 가리키는 랜덤 액세스 반복자를 반환

# 반복자 선언

vector<템플릿인수>::iterator it;

// ex) vector<int>::iterator it = intVec.begin();

# 주의 사항: 첨자의 범위를 벗어날 경우 예외 발생

vector<float fVector(10);

fVector[1] = 1.0f; // 정상

fVector[12] = 12.0f; // 에러 발생

**3. 학생 성적 처리: 동적 메모리 할당과 포인터 참조**

**소스코드**

#include <iostream>

using namespace std;

void stuGrade(const double\* pArr, int num, double& rSum, double& rAve, double& rMax);

int main(void) {

int stuSize;

double\* pArr = nullptr;

double sum, ave, max;

cout << "20215229 이채훈" << endl;

cout << endl;

cout << "입력할 학생(성적) 수: ";

cin >> stuSize;

pArr = new double[stuSize];//입력 받은 학생 수에 따라, 동적으로 배열 할당

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

cin >> \*(pArr + i);

}//입력 받은 학생 수만큼, 성적 입력 받음

stuGrade(pArr, stuSize, sum, ave, max);

cout << endl;

cout << " ##### 성적 결과 출력 ##### " << endl;

cout << endl;

cout << "학생수: " << stuSize << endl;

cout << "총 점: " << sum << endl;

cout << "평 균: " << ave << endl;

cout << "최대값: " << max << endl;

delete[] pArr; //동적으로 할당된 메모리 해체

return 0;

}

void stuGrade(const double\* pArr, int num, double& rSum, double& rAve, double& rMax) {

rSum = 0;

rMax = \*pArr;

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

rSum += \*(pArr + i);//학생들의 성적 총점

if (rMax < \*(pArr + i)) {

rMax = \*(pArr + i);

}//학생들중 성적 최대값

rAve = rSum / num;//학생들의 성적 평균

}

}

**실행결과**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**연구조사**

**알고리즘**

1. 입력할 학생의 수 입력받음

2. 입력 받은 학생 수에 따라, 동적으로 배열 할당

2-1. 포인터 변수 pArr이 배열의 시작 주소를 가리키고, double형식의 배열의 크기가 입력받은 학생의 수만큼 생성됨.

2-2. for문에서 포인터 변수를 통해, 입력 받은 학생의 수만큼, 성적을 입력받아 배열의 메모리 공간에 할당됨.

3. 입력 받은 학생의 수와 학생의 성적을 stuGrade()함수에 전달하여, 총점(rSum), 평균(rAve), 최대값(rMax)을 반환함.

3-1. for문에서 포인터 변수를 통해서 배열에 접근하여, 학생들의 성적 총점를 연산함.

3-2. if문에서 배열의 메모리에 접근하면서, 제일 큰 값을 최대값(rMax)에 대입하여 최대값을 연산함.

3-3. 3-1에서 구한 총점(rSum)과 학생수(stuSize=num)을 활용해 평균(rAve)을 연산함.

4. 학생들의 성적 결과를 출력함.

5. 배열을 사용한 후에, 메모리 누수 방지를 위해서 메모리 해체 명령함.

**\*\*주요 사용 기능\*\***

**동적 메모리 할당**

C++에서는 C언어의 malloc, free 이외에도 new, delete을 이용한 동적 메모리 할당이 가능하다. C++에서 C언어 스타일의 동적 메모리 할당을 위해서는 표준 라이브러리 헤더 파일 ‘<stdlib.h>’를 선언하면 된다. 이 라이브러리 선언 시 malloc, calloc, realloc이라는 다양한 명령어로 메모리를 할당하고 free로 메모리를 해제할 수 있다.

이 문제에서는 new, delete를 사용하여 동적 메모리와 할당과 해제를 구현하였다.

**const&reference**

stuGrade(const double\* pArr, int num, double& rSum, double& rAve, double& rMax) 함수 선언시,

멤버변수들이 const, reference를 활용하여 선언되었다. 참조자로 선언된(double& ~) 변수의 자리에 오는 변수를 참조하여 원본 변수를 변경할 수 있다.

const가 붙은 첫번째 인자인 double \* pArr는 const에 의해 변경할 수 없는 상수가 되었으며, const가 붙은 이상 원본 pArr 포인터가 가리키는 배열을 임의로 stuGrade 함수에서 변경할 수 없다.

**4. 로또 복권 : 동적 메모리 할당과 2차원 배열 자동 생성**

**소스코드**

#include <iostream>

#include <random>//난수 생성을 위한 라이브러리

using namespace std;

void HappyLotto(int num);

int main (void) {

int num;

cout << "20215229 이채훈" << endl;

cout << endl;

cout << "로또 복권을 몇 장 구입 하실래요? "; cin >> num;

cout << endl;

cout << " ### 생성된 로또 번호 ### " << endl;

cout << endl;

HappyLotto(num);

return 0;

}

void HappyLotto(int num) {

const int NUM\_MIN = 1;

const int NUM\_MAX = 45;

//로또 번호 범위 1~45

random\_device rd;//난수 생성기 초기화

mt19937 gen((rd()));//메르센 트위스터 엔진 초기화

uniform\_int\_distribution <int> distribution(NUM\_MIN, NUM\_MAX);//로또번호 1~45 난수 범위 설정

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

int\*\* lotto = new int\* [6];

for (int j = 0; j < 6; j++) {

lotto[j] = new int;

}//6개의 int형 포인터 원소를 갖는 2차원 동적 배열 생성

for (int j = 0; j < 6; j++) {

bool duplicate;//중복 로또 번호를 검사를 위한 bool형 변수

do {

duplicate = false;

\*lotto[j] = distribution(gen);//로또 번호 생성

for (int k = 0; k < j; k++) {

if (\*lotto[k] == \*lotto[j]) {

duplicate = true;

break;

}//중복된 로또 번호 검사

}

} while (duplicate);

}

for (int j = 0; j < 6; j++) {

cout.width(3);

cout << \*lotto[j] << " ";

} //생성된 로또 번호 출력, 간격3 맞추기

cout << endl;

for (int j = 0; j < 6; j++) {

delete lotto[j];

}

delete[] lotto;

//동적 메모리 해체

}

}

**실행결과**

**텍스트, 스크린샷, 디스플레이, 컴퓨터이(가) 표시된 사진

자동 생성된 설명**

**연구조사**

**알고리즘**

1. 생성할 로또 복권수(2차원 배열 개수) 입력받음.

2. HappyLotto()함수를 호출하여, 2차원 배열 생성과, 배열의 요소 출력을 한꺼번에 진행함.

2-1. 난수 생성

2-1-1. random 라이브러리의 기능을 사용하여 난수 생성함.

2-1-2. 로또번호 1-45을 const변수롤 선언하여, 난수 생성하는 내장함수에 전달함.

2-2. 2차원 배열 생성

2-2-1. 입력받은 생성할 로또 복권수의 int형 포인터 원소를 갖는 2차원 동적배열 생성

2-2-2. 6개의 int형 원소를 갖는 1차원 배열 생성

2-3. 중복 로또 번호 검사

2-3-1. do-while문의 실행조건 판단을 위한 bool형 변수 선언

2-3-2. 로또번호를 일단 생성시키기 위해서 do while문을 활용하였고, duplicate=false일 경우 로또 번호를 생성하고, 로또 번호가 중복될 경우 duplicate=true가 되어 break에 의해 로또 번호 생성이 중단되고, while문을 통해 다시 로또 번호 생성이 시작됨.

3. for문을 통해 생성된 로또 번호 출력

3-1. 로또 번호 개수(6)만큼 배열 요소들을 출력

3-2. 로또 번호 출력시 간격을 맞추기 위해, string 라이브러리의 width()내장함수 호출.

4. 동적 메모리 해체

4-1. 1차원 배열 메모리 해체

4-2. 2차원 동적 할당된 메모리 해체

**\*\*주요 사용 기능\*\***

**2차원 배열, 동적 메모리 할당&해제**

이 문제에서 new 명령어를 이용하여 이차원 동적 배열을 할당할때, 이중포인터를 사용하였는데, 일차적으로 포인터를 이용해 동적 메모리를 할당받는 것이 적절하기 때문이고, 2차원 배열이므로 이중 포인터를 사용하는 것이 적합하기 때문이다. int\*\* pArr = new int\* [cnt];

또한, 메모리 해제도 중요하다. do-while문을 수행한 후 각 행에 할당된 메모리부터 해제를 하기 위해 delete lotto[j]를 사용하고 마지막에 delete[] pArr을 하여 메모리 누수를 막는다.

**난수 생성 방법(random 라이브러리)**

C언어에서 난수 발생 관련 함수를 내포하는 라이브러리는 <stdlib.h>이고, 이와 비슷하게 C++에서 제공하는 라이브러리는 <cstlib>이다. 하지만, random 라이브러리를 활용해서 난수생성하는 방법도 있어서, 한번 시도해보고자 위 코드에서 활용하였다.

**random 라이브러리 공부**

난수 생성기 (Random Number Generators - RNGs): 난수 생성기는 무작위 숫자를 생성하는 역할을 함. <random> 라이브러리에서는 여러 종류의 난수 생성기를 제공하는데, 그 중에서도 Mersenne Twister 알고리즘을 사용한 mt19937가 가장 일반적으로 사용됨.

분포 함수 (Distribution Functions): 분포 함수는 생성된 난수의 분포를 제어함. 예를 들어, 균일 분포에서 숫자를 생성하려면 suniform\_int\_distribution 또는 suniform\_real\_distribution을 사용할 수 있다. 또한, 다른 분포 함수를 사용하여 정규 분포, 이항 분포, 포아송 분포 등을 생성할 수 있음.

1. 난수 생성기를 초기화

: random\_device rd;//난수 생성기 초기화

2. 적절한 분포 함수를 선택하고 필요한 매개변수를 설정합니다

:mt19937 gen((rd()));//메르센 트위스터 엔진 초기화

3. 분포 함수를 사용하여 난수를 생성합니다.

: const int NUM\_MIN = 1;const int NUM\_MAX = 45;

uniform\_int\_distribution <int> distribution(NUM\_MIN, NUM\_MAX);//로또번호 1~45 난수

**5. 문자열을 숫자(정수)로 변환 : atoi, stoi**

**소스코드**

#include <iostream>

#include <string>

using namespace std;

int my\_stoi(const string& rStr);

int main(void) {

string str;

cout << "20215229 이채훈" << endl;

cout << endl;

cout << "정수로 변환할 문자열 입력: "; cin >> str;

int result = my\_stoi(str);

cout << "입력 문자열: " << str << ", " << "변환된 숫자(int): " << result;

}

int my\_stoi(const string& rStr) {

const char\* ptr = rStr.c\_str();

//string를 문자열로 변환하는 멤버함수 c\_str()

int result = 0;

int sign = 1;

if (\*ptr == '-') {

sign = -1;

++ptr;

}//첫번째 문자의 음수 판단

else if (\*ptr == '+') {

++ptr;

}//첫번째 문자의 양수 판단

while (\*ptr != NULL) {

if (\*ptr >= '0' && \*ptr <= '9') {

result = result \* 10 + (\*ptr - '0');

}

else {

break;

}

++ptr;

}//0~9이외의 문자가 존재할 경우, 숫자 변환 작업 중단

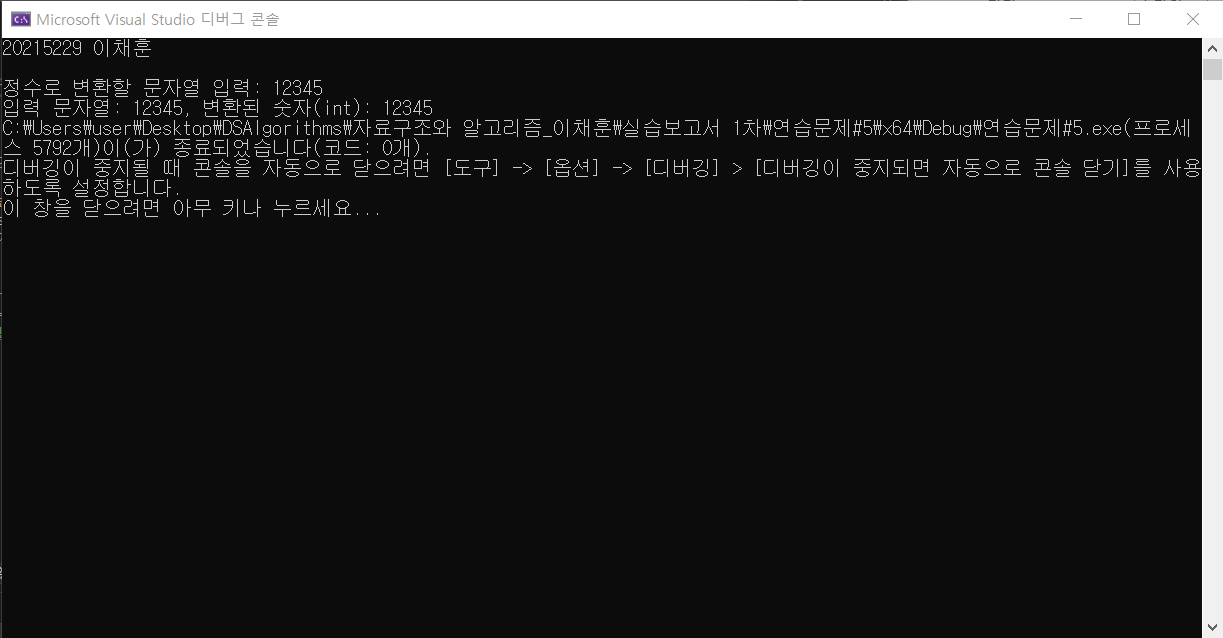
return result \* sign;

//숫자\*부호(sign) 값을 반환

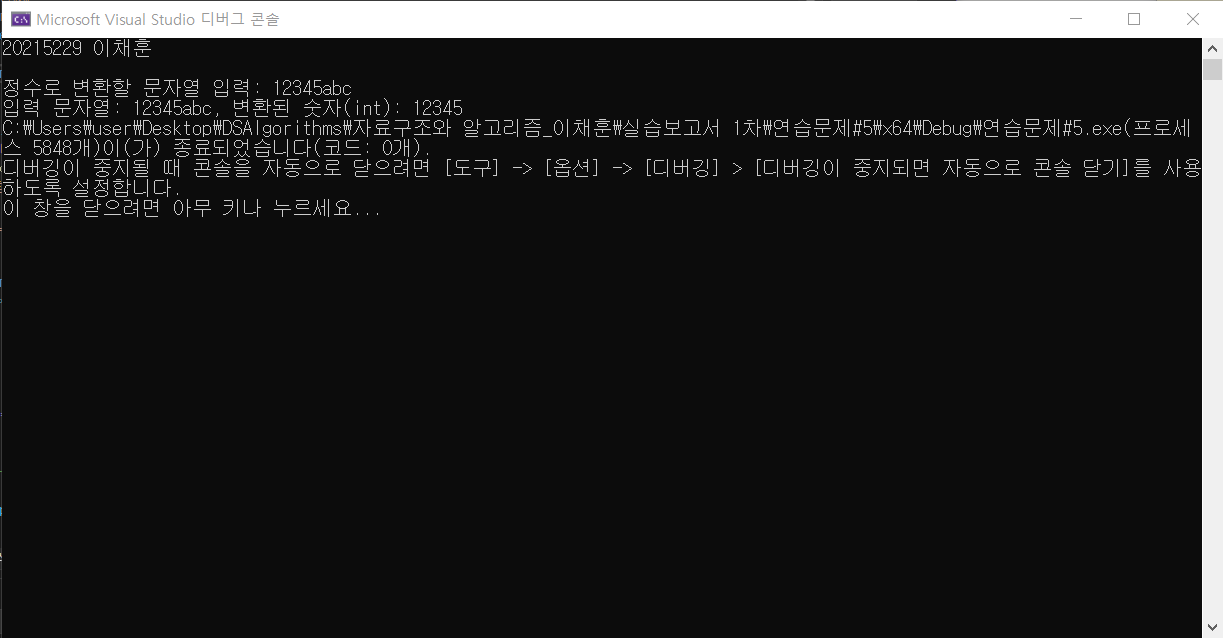
}

**실행결과**

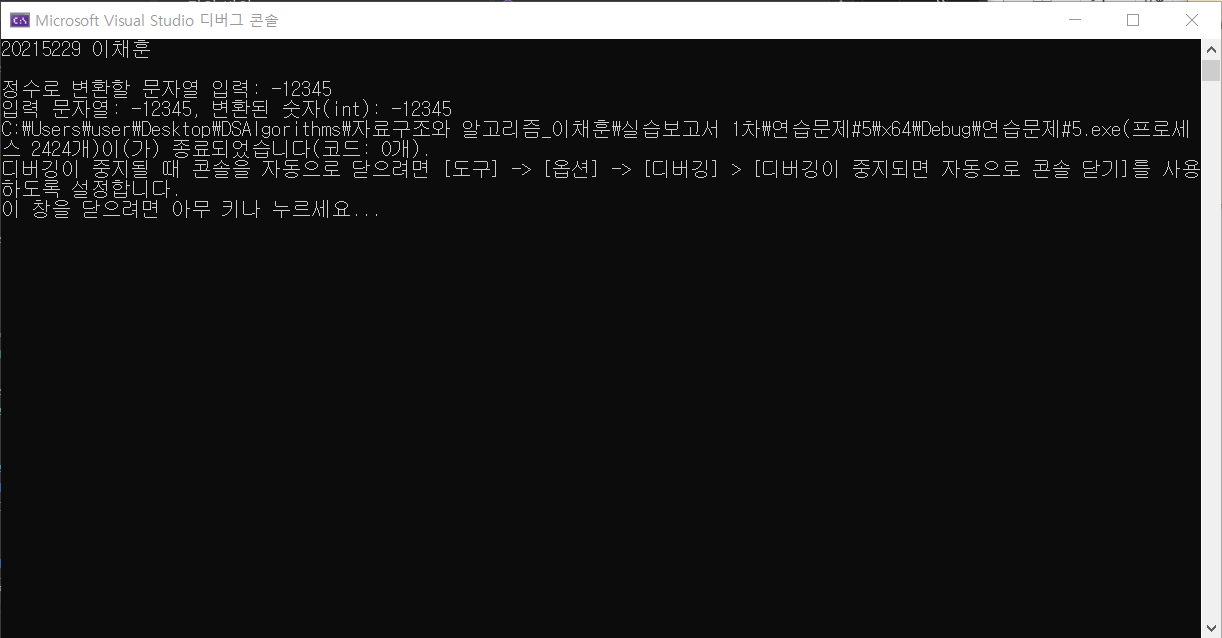
**case1**



**case2**

****

**case3**

****

**연구조사**

**알고리즘**

1. 정수로 변환할 문자열 입력 받음.

2. stoi 함수의 기능인 문자열을 정수로 바꾸는 기능을 대신할 함수 my\_stoi()를 선언하여, 입력받은 문자열을 숫자로 변환

2-1. char형 포인터 변수 ptr에 입력받은 문자열을 string 라이브러리의 멤버함수 c\_str()를 통해서, string에서 char

로 변환함.

2-2. 포인터 변수ptr를 통한 부호 판단을 위해서 int형 sign 변수를 선언&초기화함.

2-3. ‘-‘일 경우, sign에 -1를 대입함.

3. while문에서 문자열의 마지막 요소에 접근할 때까지 and연산을 통해 0~9사이의 문자만 숫자 변환하고, 이외의 문자가 참조될 경우, break를 통해 변환을 중단함.

4. int (result)와 int(sign=1or-1)를 곱한 값을 반환함.

5. my\_stoi()함수에 의해 반환된 값을 출력함.

**6. 은행 계좌의 인증과 출금 : 응용**

**소스코드**

#include <iostream>

#include <stdexcept>

using namespace std;

class Account {

private:

const string name;

const string account;

string pass;

int balance;

public:

Account(string name, string account, string pass = “11111”, int balance = 1)

: name(name), account(account), pass(pass), balance(balance) {}

string getName(void) const {

return name;

};

string getAccount(void) const {

return account;

};

string getPass(void) const {

return pass;

};

int getBalance(void) const {

return balance;

};

void showAccount(void)const {

cout << “이 름: “ << getName() << endl;

cout << “계좌번호: “ << getAccount() << endl;

cout << “패스워드: “ << getPass() << endl;

cout << “현재잔액: “ << getBalance() << endl;

};//계좌관련 정보 출력하는 함수

};

int main(void) {

cout << “20215229 이채훈” << endl;

cout << endl;

Account AC(“홍길동”, “111-2222-3333”, “12345”, 100000);

//임의 객체 생성(과제물의 실행결과와 동일하게 설정함)

AC.showAccount();

cout << endl;

string inAccount, inPass;

int amount, lbalance;

cout << “##### 은행 계좌 인증 #####” << endl;

cout << “계좌번호 입력: “; cin >> inAccount;

cout << “비밀번호 입력: “; cin >> inPass;

try

{

if (inAccount != AC.getAccount() || inPass != AC.getPass())

{

throw 1;//int throw

}

else

{

cout << “출금액 입력: “; cin >> amount;

if (amount > AC.getBalance())

{

cout << “현재 잔액: “ << AC.getBalance() << endl;

throw ‘e’;

//char throw

return 0;

}

else

{

lbalance = AC.getBalance() – amount;

cout << “현재 잔액: “ << lbalance << endl;

cout << AC.getName() << “님의 현재 잔액은: “ << lbalance << “입니다.” << endl;

}

return 0;

}

}

catch (int exception)

{

cout << “\n다음 입력을 다시 한 번 확인하세요!!!” << endl;

Account AC(“”, inAccount, inPass, 0);

//입력받은 계좌번호와 비밀번호로 객체 생성

AC.showAccount();

return 0;

}

//계좌번호와 비밀번호가 다를 경우의 예외처리

catch (char exception)

{

cout << “희망 출금액: (“ << amount << “)이 너무 많네요.” << endl;

return 0;

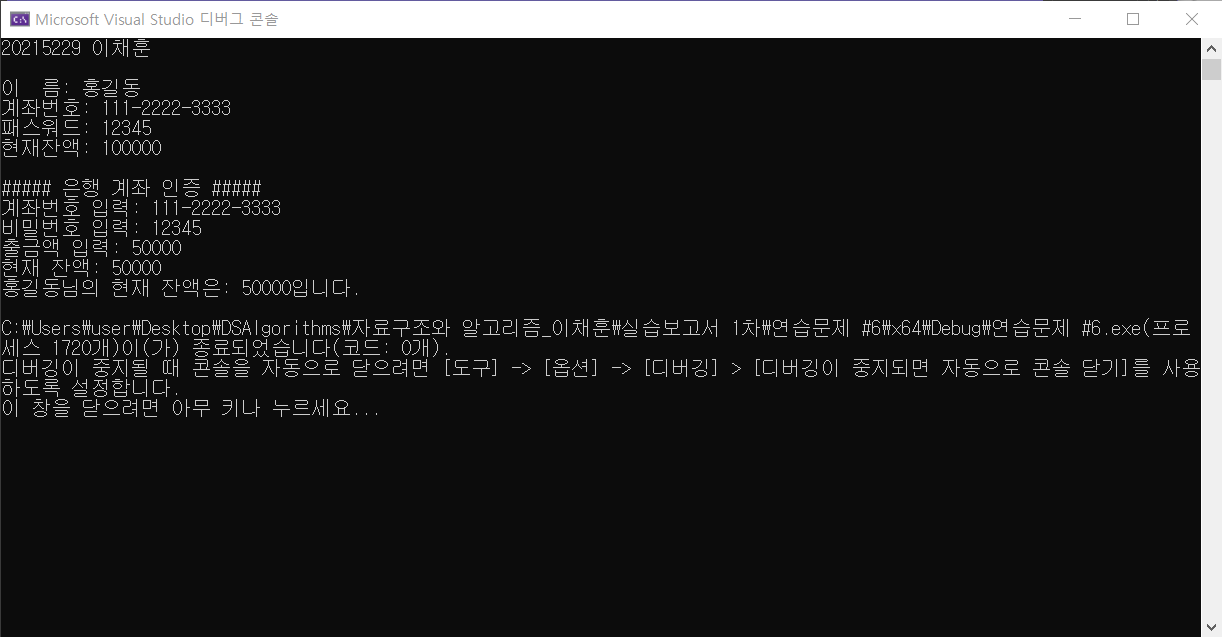
}

//희망출금액이 잔액보다 적을 경우의 예외처리

}

**실행결과**

**case1**

****

**case2**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

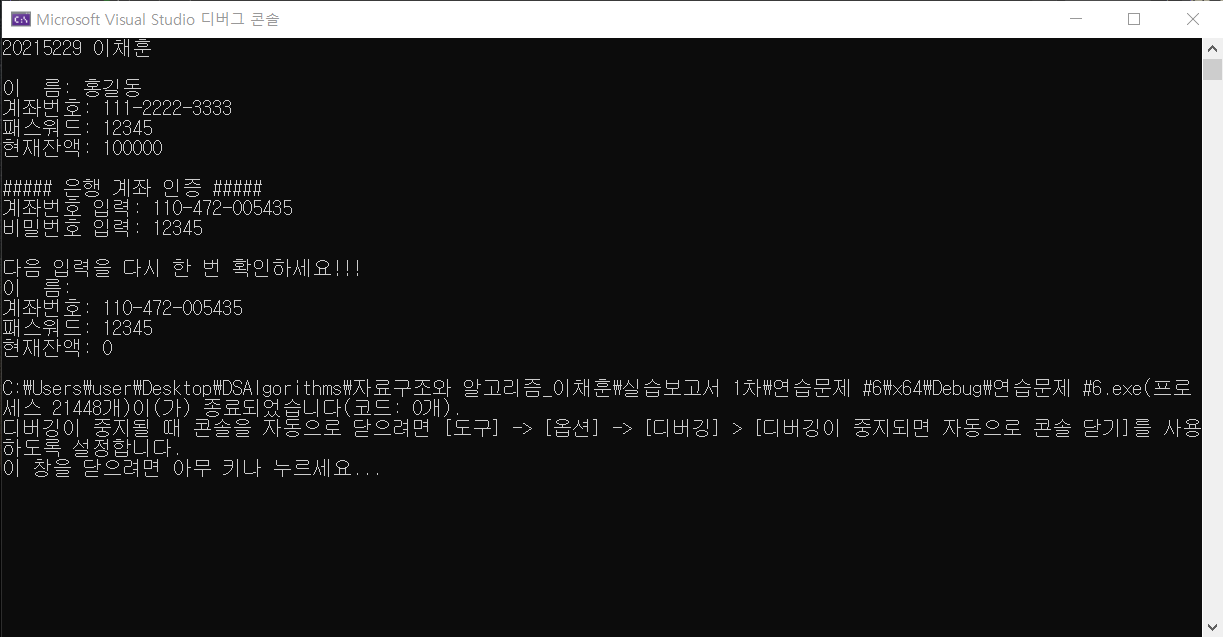
자동 생성된 설명**

**case3**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**case4**

****

**연구조사**

**알고리즘**

1. Account 클래스 정의

1-1. 이름, 계좌번호, 패스워드, 현재잔액을 반환하는 get함수 정의

1-2. 계좌관련 정보를 출력하는 showAccount()함수 정의

2. main 함수

2-1. 과제물의 실행결과와 동일한 임의의 객체 생성

2-2. showAccount()함수를 호출하여 계좌 정보 출력

2-3. 계좌번호와 비밀번호 입력받음.

3. try-catch 예외처리

3-1. if 문에서 throw로 어떤 자료형을 예외처리하냐는 기준으로 예외처리 코드를 구현함.

3-2. main함수에서 입력받은 계좌번호와 비밀번호와 초기에 생성한 객체와의 비교를 통해,

일치하지 않을 경우 int형 예외를 throw함.

3-2-1. 입력받은 계좌번호와 비밀번호로 객체를 생성함.

3-2-2. showAccount()함수를 호출하여 계좌 정보 출력.

3-3. 일치할 경우 출금액을 입력받음.

3-4. 출금액이 현재잔액보다 클 경우, char형 예외를 throw함.

3-4-1. catch(char exception){}; 예외처리

3-5. 출금액이 현재잔액보다 작을 경우, 현재잔액에서 출금액을 빼는 연산을 하고, 출력함.

**\*\*주요 사용 기능\*\***

**try-catch 예외처리**

이 코드에서 예외처리를 위해 try catch를 사용하였습니다. C++에서 try와 catch를 사용하는 예외 처리(Exception Handling)는 프로그램에서 예외 상황을 처리하고 프로그램의 안정성을 향상시키는 중요한 메커니즘 중이기 때문입니다.

if 문등 여러가지 조건등으로, 예외처리를 하는 경우보다 try-catch를 통해서 예외처리를 진행할 경우에는 이와 같은 이점을 가질 수 있습니다.

* **오류 처리의 분리**: 예외 처리를 사용하면 오류 처리 코드를 주 실행 코드에서 분리할 수 있습니다. 이로써 프로그램의 가독성과 유지보수성이 향상되며, 주 코드는 오류 처리 코드로 인한 복잡성에 영향을 받지 않습니다.
* **예외 상황의 전파**: 예외가 발생하면 호출 스택을 거슬러 올라가면서 예외를 처리할 수 있는 가장 가까운 catch 블록을 찾습니다. 이것은 예외를 발생시킨 위치와 예외를 처리하는 위치가 떨어져 있어도 효과적으로 처리할 수 있게 해줍니다.
* **프로그램 안정성 향상**: 예외 처리를 사용하면 예상치 못한 상황에 대비하여 프로그램의 안정성을 높일 수 있습니다. 예외를 처리하지 않으면 프로그램은 비정상적으로 종료될 수 있습니다.

**7. Point 클래스 : 객체간 덧셈과 뺄셈 연산(멤버함수, 일반 함수)**

**소스코드**

#include <iostream>

using namespace std;

class Point {

private:

const int x;

const int y;

public:

Point(int x=0, int y=0) : x(x), y(y) {}

void showPosition() {

cout << “(“<<x<<”, “<<y<<”)” << endl;

}

Point operator+(const Point& obj1) const {

return Point(x + obj1.x, y + obj1.y);

}//연산자+ 함수

friend Point operator-(const Point& obj2, const Point& obj3) {

return Point(obj2.x – obj3.x, obj2.y – obj3.y);

}//friend를 활용한 다중 정의 함수(연산자- 함수)

};

int main() {

cout << “20215229 이채훈” << endl;

cout << endl;

/\*

Point p1; Point p2;

객체 매개변수 초기값 0확인

\*/

Point p1(10, 20);

Point p2(30, 40);

//p1, p2 객체 생성

cout << “p1: “;

p1.showPosition();

cout << “p2: “;

p2.showPosition();

cout << endl;

Point p3 = p1 + p2;

cout << “p3 = p1 + p2” << endl;

cout << “p3: “;p3.showPosition();

cout << endl;

Point p4 = p1 – p2;

cout << “p4 = p1 – p2” << endl;

cout << “p4: “;p4.showPosition();

return 0;

}

**실행결과**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**연구조사**

**알고리즘**

1. Point클래스 정의

1-1. 생성된 객체를 출력하는 showPosition()함수 정의

1-2. 생성된 객체의 호출를 통해 +연산하는 +함수 정의

1-3. 생성된 객체의 호출를 통해 -연산하는 -함수를 frient로 연산자 다중정의 함.

2. main 함수

2-1. p1, p2 객체 생성

2-2. p1, p2 출력

2-3. 연산자 오버로딩을 활용한 p3, p4 객체 생성

2-4. p3, p4 출력

**\*\*주요 사용 기능\*\***

**friend**

함수나 클래스 선언 앞에 선언할 수 있는 friend를 선언한 함수나 class에서는 접근 제어 지시자(private, public, protected)의 영향을 받지 않는다.

하나의 클래스에서 다른 클래스의 내부 데이터에 접근 해야할 경우 프렌드를 써서 권한을 줄 때 사용하면 유용하다. friend를 사용하면 자신의 개체뿐만이 아닌 다른 타입의 개체를 접근하는 것이 가능해지기 때문에 코드의 확장이 수월하게 이루어질 수 있다. 하지만 friend야말로 캡슐화에 지장을 줄 수 있기 때문에 가급적 남발하지 않고 사용해야 한다.

위의 코드에서는 문제에서 제시한 대로 operator-를 구현할 때 friend를 사용하였다. 덕분에 외부에서 일반함수로 operator-를 구현할 때 operator+와 다르게, 접근 제어 지시자의 영향을 받지 않아 private 멤버에 접근이 가능하여, 해당 클래스의 멤버로 간주되어 외부 구현부에서 범위 지정 연산자 ::를 사용하지 않아도 접근할 수 있었다.

**8. Person과 Student 클래스**

**소스코드**

#include <iostream>

#include <string>

using namespace std;

class Person {

public:

Person(const string& name, const string& gender, const string& tel, const string& addr)

: name(name), gender(gender), telI, addr(addr) {}

void showPerson() const {

cout << “이 름: “ << name << endl;

cout << “성 별: “ << gender << endl;

cout << “연락처: “ << tel << endl;

cout << “주 소: “ << addr << endl;

}

private:

string name;

string gender;

string tel;

string addr;

};

class Student : public Person {//Person 클래스를 상속받은 Student 클래스 정의

public:

Student(const string& name, const string& gender, const string& tel, const string& addr, int kor, int eng, int math)

: Person(name, gender, tel, addr), kor(kor), eng(eng), math(math) {

calculateT\_A();

}

void showStudent() const {

Person::showPerson();//Person 클래스의 showPerson함수 선언

cout << “국 어: “ << kor << endl;

cout << “영 어: “ << eng << endl;

cout << “수 학: “ << math << endl;

cout << “총 점: “ << tot << endl;

cout << “평 균: “ << ave << endl;

}

private:

int kor;

int eng;

int math;

int tot;

double ave;

void calculateT\_A() {

tot = kor + eng + math;

ave = static\_cast<double>(tot) / 3.0;

}//총점, 평균 계산하는 함수

};

int main() {

string personName, personGender, personTel, personAddr;

int kor, eng, math;

cout << “20215229 이채훈” << endl;

cout << endl;

cout << “학생의 정보와 성적을 입력하세요...” << endl;

cout << endl;

cout << “이 름: “; cin >> personName;

cout << “성 별: “; cin >> personGender;

cout << “연락처: “; cin >> personTel;

cout << “주 소: “; getline(cin >> ws, personAddr);//공백 입력을 위한 getline()함수

cout << “국 어: “; cin >> kor;

cout << “영 어: “; cin >> eng;

cout << “수 학: “; cin >> math;

cout << endl;

Student student(personName, personGender, personTel, personAddr, kor, eng, math);

//입력값을 통해 객체 생성

cout << “ ### 입력 정보 ###” << endl;

cout << endl;

student.showStudent();

return 0;

}

**실행결과**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**연구조사**

**알고리즘**

1. Person 클래스 정의

1-1. 입력받은 값을 통해 객체를 생성하고, 이 값을 출력하는 showPerson()함수 정의

1-2. 이름, 성별, 연락처, 주소를 멤버변수로 가짐.

2.Person 클래스를 상속받은 Student 클래스 정의

2-1. Person 클래스를 상속받았기 때문에, showPerson()함수 호출을 통해서, 이름, 성별, 연락처, 주소를 출력함.

2-2. 입력받은 값을 통해 객체를 생성하고, 이 값을 출력하는 showStudent()함수 정의

2-3. 국어, 영어, 수학을 멤버변수로 가짐.

3. Student 클래스 내의 calculateT\_A()멤버함수를 정의

3-1. 총점 계산

3-2. 평균 계산시 강제 자료형 변환을 통해, 평균 값 구함.

4. main 함수

4-1. 클래스의 객체 멤버변수들을 입력받음.

4-2. 주소 입력시 공백 입력을 위해 string 라이브러리의 멤버함수 getline()함수 호출

4-3. 입력 받은 값을 통해 객체 생성

4-4. student클래스의 showStudent()함수를 호출하여 학생의 정보와 성적 출력

**\*\*주요 사용 기능\*\***

**getline()**

이름, 주소 문자열을 입력받을 때 공백을 포함한 문자열을 입력받아야 하기 때문에, 기존의 cin으로 해결할 수 없으므로 공백을 포함한 문자열을 입력할 수 있는 getline() 함수를 사용하였다. 이 함수는 string 라이브러리에 내장된 멤버함수이다.

**상속의 멤버 접근 권한**

class Student : public Person 이라는 명령어를 사용해 Student라는 클래스가 Person이라는 클래스에게 public 권한으로 상속받았다는 것을 나타낸다. public, protected, private의 3가지 접근권한으로 상속받을 수 있다. 상속 관계에서 접근 제어 지시어의 사용에 따라 멤버 변수와 함수의 가시성이 결정된다.

**public 상속**: public 상속은 가장 널리 사용되는 상속 방식임.

1. 기반 클래스의 public 멤버는 파생 클래스에서 public으로 상속됩니다.

2. 기반 클래스의 protected 및 private 멤버는 파생 클래스에서 접근할 수 없습니다.

3. 파생 클래스의 객체는 기반 클래스와 동일한 인터페이스를 가지며, 외부 코드에서도 마치 기반 클래스처럼 사용할 수 있습니다.

**protected 상속**: protected 상속은 일반적으로 기반 클래스의 public 및 protected 멤버를 파생 클래스의 protected 멤버로 상속함.

1. 기반 클래스의 private 멤버는 파생 클래스에서 접근할 수 없습니다.

2. 파생 클래스의 객체는 외부 코드에서 기반 클래스의 public 및 protected 멤버에 접근할 수 없지만, 파생 클래스 내부에서는 접근 가능합니다.

**private 상속 :** private 상속은 기반 클래스의 public 및 protected 멤버를 모두 파생 클래스의 private 멤버로 상속함.

1. 기반 클래스의 private 멤버는 파생 클래스에서 접근할 수 없습니다.

2. 파생 클래스의 객체는 외부 코드에서 기반 클래스의 멤버에 접근할 수 없습니다.

**9. Albamon 클래스**

**소스코드**

#include <iostream>

#include <string>

using namespace std;

class Person {

public:

Person(const string& name, int age)

: name(name), age(age) {}

string getName() const {

return name;

}

int getAge() const {

return age;

}

void printPerson(){

cout << “이름: “ << name << “, 나이: “ << age << endl;

}

private:

string name;

int age;

};

class Student : private Person {

// Person 클래스를 상속 받은 Student 클래스 정의

public:

Student(const string& name, int age, int admissionYear, int graduationYear)

: Person(name, age), admissionYear(admissionYear), graduationYear(graduationYear){}

int getad() const {

return admissionYear;

}

int getgY() const {

return graduationYear;

}

void printStudent(){

cout << “이름: “ << getName() << “, 나이: “ << getAge() << endl;

cout << “입학 년도: “ << getad() << endl;

cout << “졸업 년도: “ << getgY() << endl;

}

private:

int admissionYear;

int graduationYear;

};

class Worker : private Person {

// Person 클래스를 상속 받은 Worker 클래스 정의

public:

Worker(const string& name, int age, int workingTime)

: Person(name, age), workingTime(workingTime) {}

int getwT() const {

return workingTime;

}

void printWorker() {

printPerson();

cout << “작업 시간: “ << getwT() << endl;

}

private:

int workingTime;

};

class Albamon : private Student, private Worker {

// Person 클래스를 상속 받은 Worker, Student클래스를 상속받은 Albamon 클래스 정의

public:

Albamon(const string& name, int age, int admissionYear, int graduationYear, int workingTime)

: Student(name, age, admissionYear, graduationYear), Worker(name, age, workingTime) {}

void printAlbamon(){

printStudent();

cout << “작업 시간: “ << getwT() << endl;

}

};

int main() {

Person aPerson(“단군”, 100);

Student bStudent(“이도”, 25, 2020, 2024);

Worker cWorker(“이순신”, 50, 100);

Albamon dAlbamon(“철수”, 20, 2023, 2027, 120);

//4개의 객체 생성

aPerson.printPerson();

bStudent.printStudent();

cWorker.printWorker();

dAlbamon.printAlbamon();

//생성된 객체 값 출력

return 0;

}

**실행결과**

**텍스트, 스크린샷, 컴퓨터, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**연구조사**

**알고리즘**

1. Person 클래스 선언

1-1. name, age get함수 정의

1-2. 이름과 나이를 입력받아, 출력하는 printPerson()함수 정의

2. Person 클래스를 상속 받은 Student 클래스 정의

2-1. admissonYear, graduationYear get 함수 정의

2-2. 입학년도와 졸업년도를 입력받아, 출력하는 printPerson()함수 정의

2-2-1. Person 클래스를 상속받았기 때문에, 이름과 나이를 출력할 때, get함수 사용하여 출력.

2-2-2. 나중에 Albamon클래스에서도 출력을 가능하게 하기 위해서, 입학년도와 졸업년도로 get함수를 사용하여 출력

3. Person 클래스를 상속 받은 Worker 클래스 정의

3-1. workingTime get 함수 정의

3-2. 작업시간을 입력 받아, 출력하는 printWorker()함수 정의

3-2-1. Person 클래스를 상속받았기 때문에, printPerson()함수를 호출하여 이름과 나이 출력.

3-2-2. 작업시간을 get함수를 사용하여 출력

4. Person 클래스를 상속 받은 Worker, Student 클래스를 상속받은 Albamon 클래스 정의

4-1. printAlbamon()함수 정의

4-1-1. student 클래스를 상속받았기 때문에, printStudent()함수를 호출하여 이름, 나이, 입학년도, 졸업년도 출력

4-1-2. Worker 클래스를 상속받았기 때문에, 작업시간을 get함수를 사용하여 출력.

5. main 함수

5-1. 실행결과에 맞는 4개의 객체 생성

5-2. 각 클래스의 print함수를 통해, 생성된 객체 값 출력

**10. 다양한 자료형 출력 : 함수 템플릿**

**소스코드**

#include <iostream>

using namespace std;

template<class T>

void printArr(const T\* pArr, const int num) {

for (int I = 0; I < num; i++) {

cout << \*(pArr + i) << “ “;

}

cout << endl;

}//정의된 배열의 원소들(자료형 상관x)을 출력하는 함수.

char str[] = “Hi ~ Clickseo”;

int iarr[] = { 10, 20 , 30, 40, 50 };

double darr[] = { 10.5, 20.5, 30.5, 40.5, 50.5 };

//char, int, double형 배열 선언 및 초기화;

int main(void) {

cout << “20215229 이채훈” << endl;

cout << endl;

printArr(str, sizeof(str) / sizeof(char));

printArr(iarr, sizeof(iarr) / sizeof(int));

printArr(darr, sizeof(darr) / sizeof(double));

return 0;

}

**실행결과**

**텍스트, 스크린샷, 디스플레이, 컴퓨터이(가) 표시된 사진

자동 생성된 설명**

**연구조사**

**알고리즘**

1. 배열의 자료형과 상관없이 출력하는 함수를 만들기 위해서 template 선언 후, printArr()함수 정의

2. char, int, double형 배열 선언 및 초기화

3. 배열의 길이 계산

sizeof()함수에 배열을 넣을 경우, 배열의 길이가 아닌, 배열의 크기가 반환되기 때문에, 배열의 길이 계산을 위해서, sizeof(배열 이름)/sizeof(배열의 자료형)을 통해, for문의 조건문에 들어갈 num(배열의 길이)을 연산함.

4. printArr()함수의 매개변수로 배열과 배열의 길이를 전달하여, 결과 출력함.

**\*\*주요 사용 기능\*\***

**템플릿(Template) 함수**

템플릿 함수는 템플릿 매개변수 T에 따라 컴파일러가 자동으로 해당 자료형에 맞게 코드를 생성한다. 따라서 T에 어떠한 자료형을 전달하느냐에 따라 printArr 함수는 다양한 자료형의 배열을 출력할 수 있다.

**템플릿 함수를 정의하는 방법**

template <typename T>

T functionName(T parameter1, T parameter2) {

// 함수 본문

// parameter1과 parameter2를 사용한 연산 또는 처리

// 반환값은 T 타입이어야 함

}

typename T는 템플릿 매개변수(Template Parameter)로, 어떤 데이터 타입을 사용할 것인지를 나타냅니다. T는 임의의 이름이며, typename 뿐만 아니라 class 키워드를 사용하여도 동일한 의미를 가집니다.

**템플릿 함수의 주요 특징과 사용법**

* **제네릭한 함수**: 템플릿 함수를 사용하면 동일한 코드를 다양한 데이터 타입에 대해 사용할 수 있습니다. 예를 들어, int, double, string 등의 다양한 데이터 타입에 대해 동일한 함수를 호출할 수 있습니다.
* **컴파일 타임 다형성**: 템플릿 함수는 컴파일 타임에 다형성을 제공하며, 인자로 전달되는 데이터 타입에 따라 컴파일러가 해당 타입에 맞는 코드를 생성합니다.
* **템플릿 인스턴스화**: 템플릿 함수를 사용할 때, 컴파일러는 템플릿을 인스턴스화하여 실제 함수를 생성합니다. 이때, 매개변수의 데이터 타입에 따라 여러 버전의 함수가 생성됩니다.
* **함수 템플릿 호출**: 함수 템플릿을 호출할 때, 컴파일러는 전달된 인자의 데이터 타입을 분석하여 적절한 템플릿 인스턴스를 선택하고 호출합니다.