**01. CPP 프로그래밍**

**01-1. 개념**

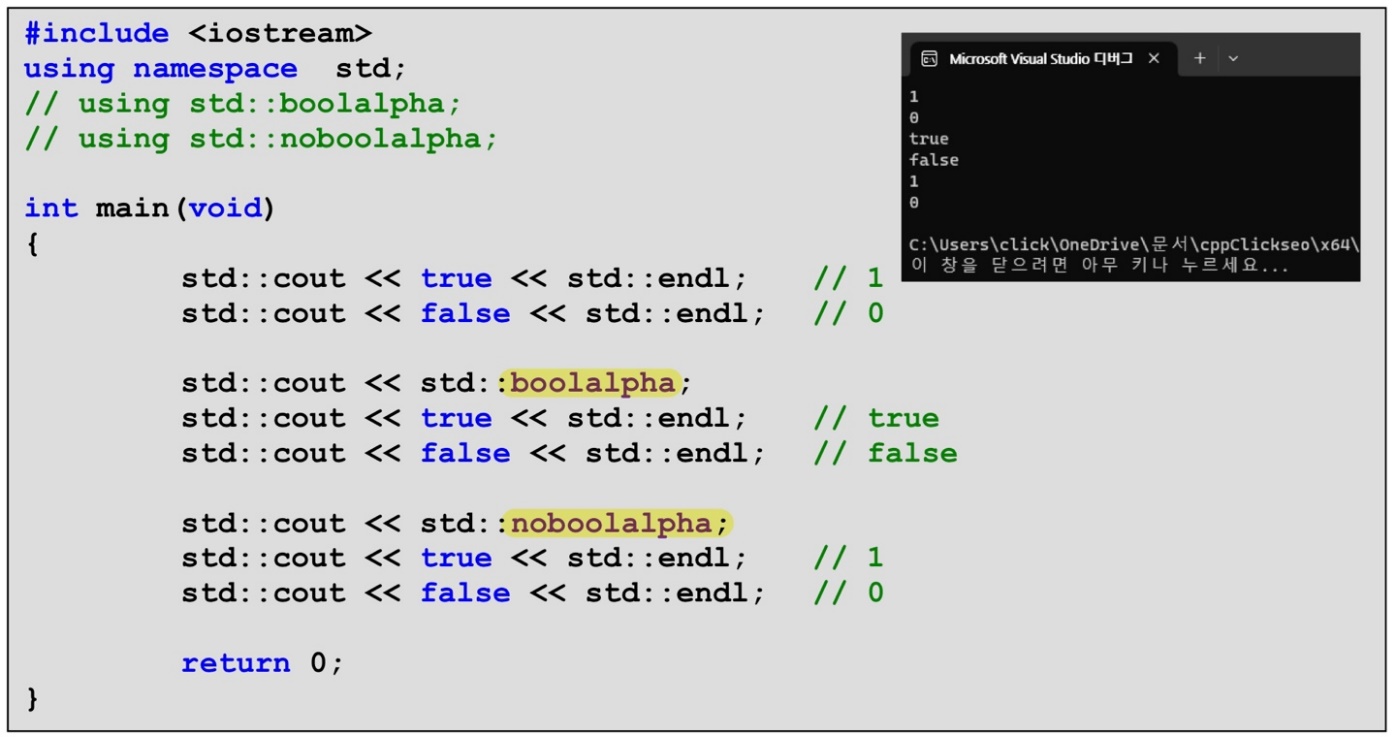
**C++ 프로그래밍 기초**

**데이터 표현 : 변수와 자료형**

sizeof(bool) : 1/sizeof(char) : 1/sizeof(short) : 2/sizeof(int) : 4/sizeof(long) : 4/sizeof(long long) : 8/sizeof(float) : 4/sizeof(double) : 8/sizeof(long double) : 8

**논리형(Boolean Type) : bool**

: 논리형(bool) 변수의 상태는 true와 false 둘 중 하나가 될 수 있다.



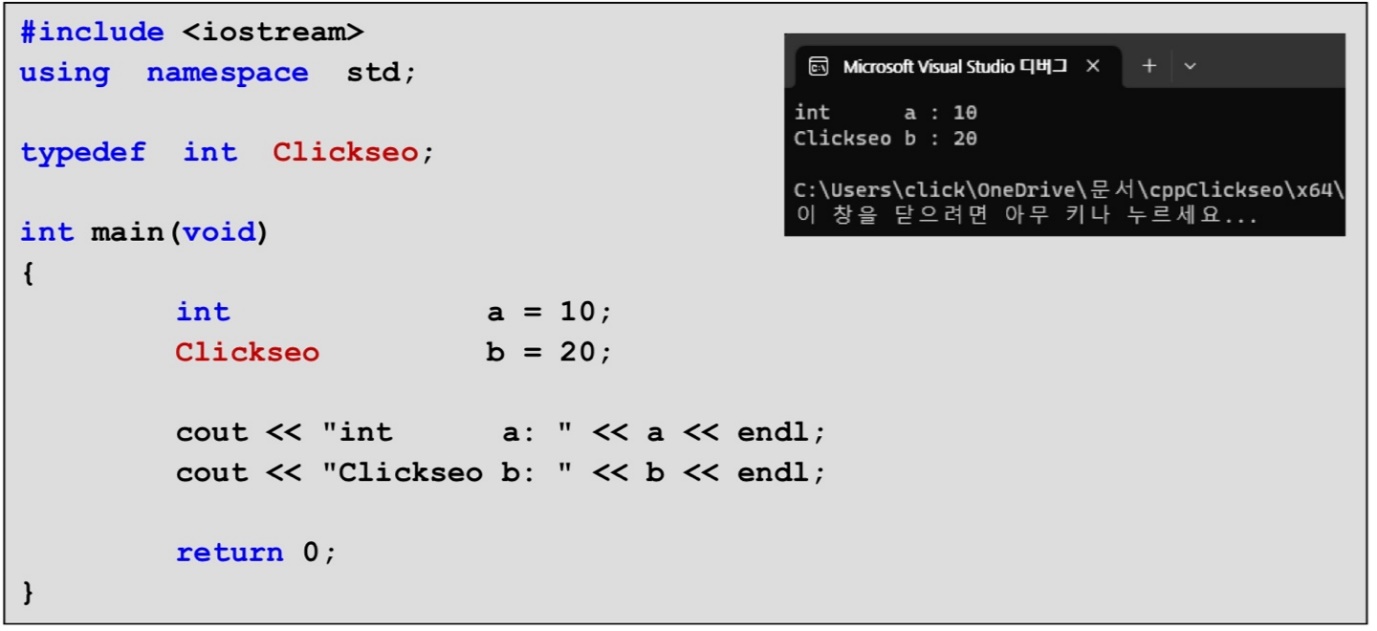
**명시적 형 변환- typeid 연산자**

**텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**typedef 지정자 – 식별자를 별칭 형식으로 선언**

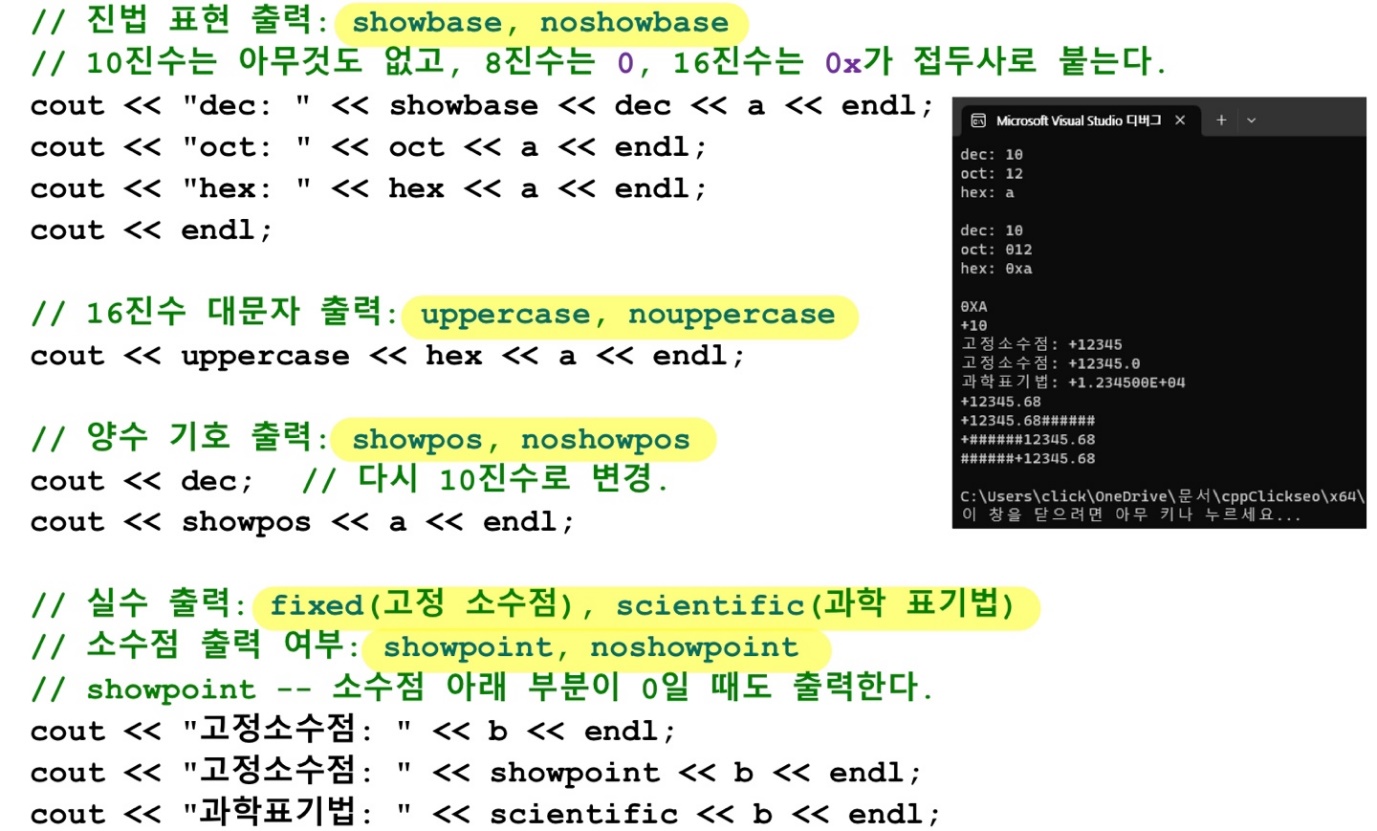
: 주로 복잡한 유형 이름을 대체하는데 사용되는 방법

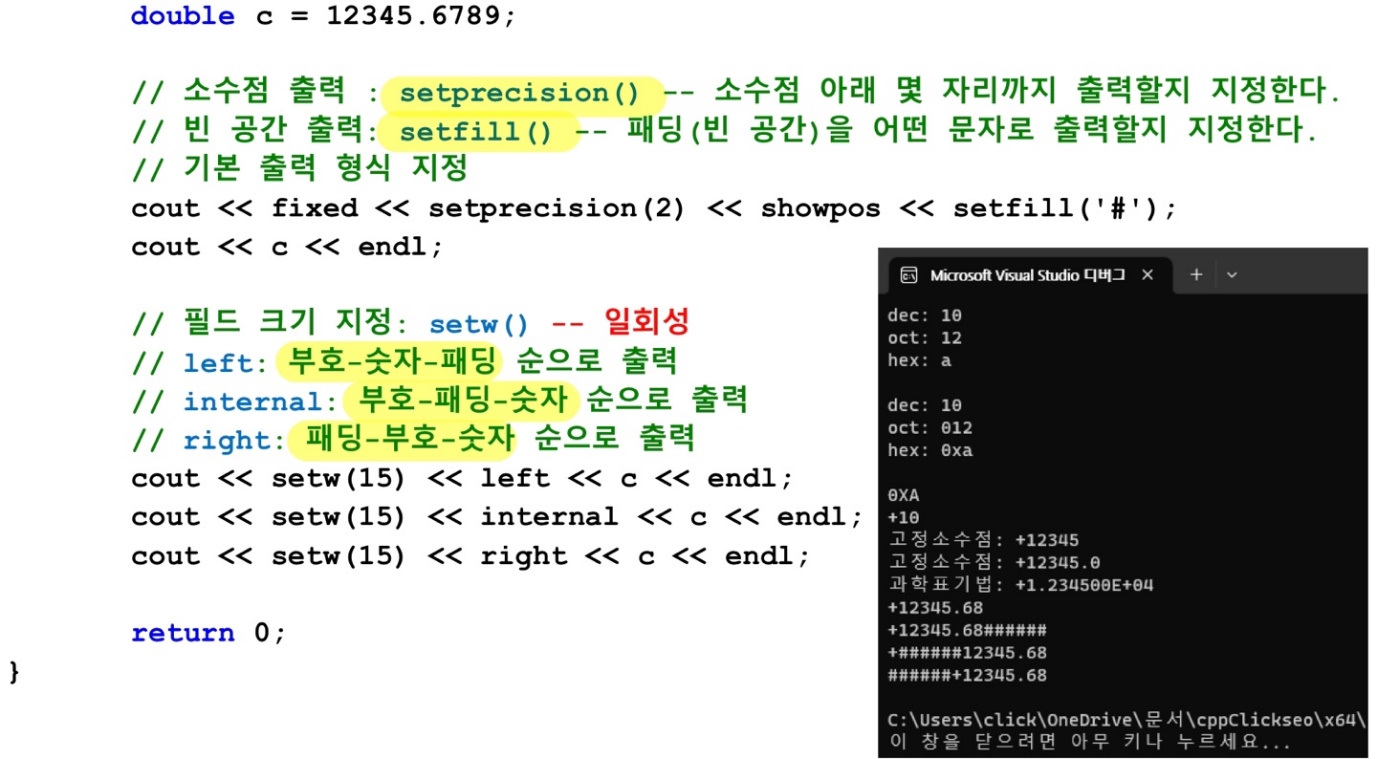


**콘솔입출력 – 입출력 조정자(iomanip)**

텍스트, 전자제품, 스크린샷, 소프트웨어이(가) 표시된 사진

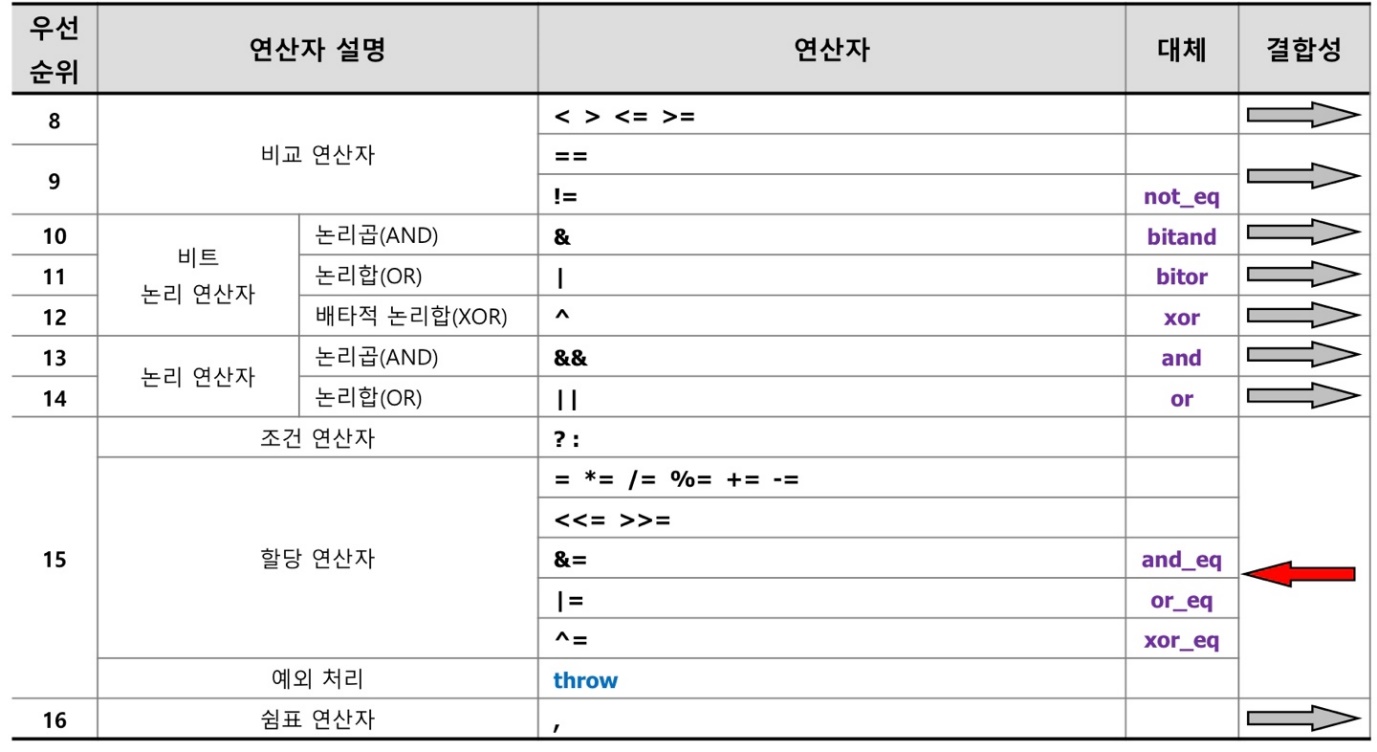
자동 생성된 설명

****

****

**연산자 우선순위와 결합성**

****

****

**제어흐름 : 예외처리**

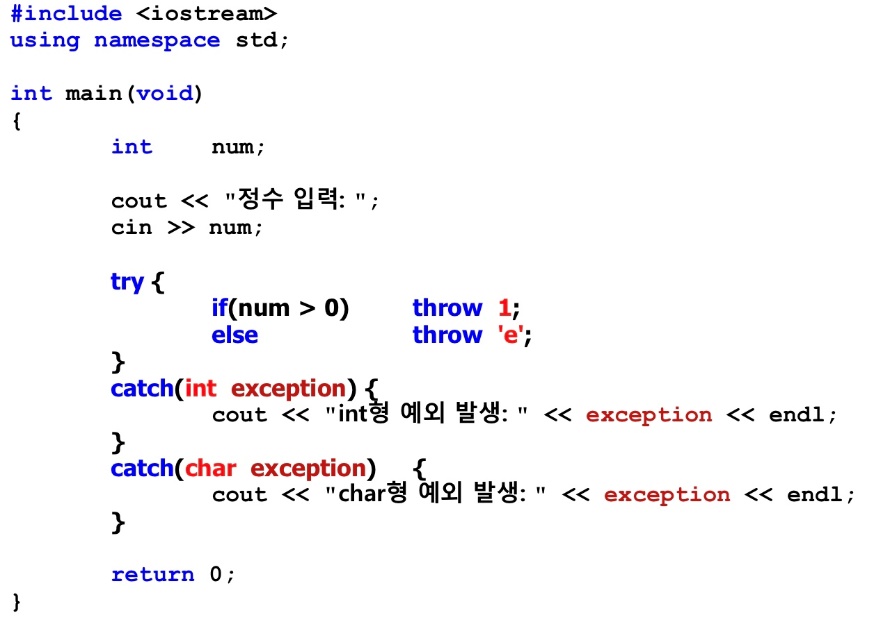
* try…catch

: try(예외 발생에 대한 검사 범위를 설정할 때 사용)

: catch(예외를 처리하는 코드 블록을 선언할 때 사용)

* throw

: 예외 상황이 발생하였음을 알릴 때 사용



**포인터의 이해**

* **포인터형 변수 :** 메모리 공간의 주소를 저장할 수 있는 변수

**“ 포인터형 변수는 메모리 주소 이외에 어떠한 값도 저장하지 않는다는 것을 절대 잊으면 안된다.”**

**# 포인터 연산-증감 연산자**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**# nullptr과 NULL 포인터**

**:** C++11에서 nullptr은 NULL을 대체한다.

**참조의 이해**

**함수 : 다중 정의와 디폴트 인자**

**네임스페이스**

**동적메모리할당**

**배열 : 정적배열과 동적배열**

**문자열**

**문자열 : string 클래스**

**구조체**

**객체지향 프로그래밍**

**클래스와 객체**

**캡슐화와 은닉**

**생성자와 소멸자**

**복사 생성자**

**멤버변수와 함수 제한**

**클래스와 포인터**

**클래스와 배열**

**예외 클래스와 객체**

**예외 클래스와 상속**

**연산자 다중 정의**

**이항 연산자**

**단항 연산자**

**cin, cout 객체**

**string 클래스**

**상속과 다형성**

**상속의 조건**

**멤버 접근 권한**

**객체 생성 및 소멸**

**객체포인터와 참조**

**함수 재정의**

**가상 함수**

**다중 상속**

**파일처리**

**stream 클래스**

**입출력 형식 지정**

**입출력 조정자**

**파일 입출력**

**파일 처리**

**순차 접근과 임의접근**

**일반화 프로그래밍**

**템플릿**

**함수 템플릿**

**클래스 템플릿**

**STL**

**STL 컨테이너**

**STL 알고리즘**

**01-2. 1차 실습보고서 REVIEW**

**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;

}

**교수님 코드**

int validateBinary(int num) {

while (num > 0) {

if (num % 10 != 0 && num % 10 != 1)

return FALSE;//십진수

num /= 10;

}

return TRUE;//이진수

}

int binaryToDecimal(int num) {

int res = 0, pow = 1;

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

res += (num % 10) \* pow;

num /= 10;

pow \*= 2;

}

return res;

}

**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;

}

**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

}

}

**\*\*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

}

}

**교수님 코드**

#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <ctime>

using namespace std;

#define COL 6

int main(void) {

int i, num;

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

//동적 메모리 할당 : 2차원 배열

int\*\* table = new int\* [num + 1];

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

\*(table + i) = new int[COL];

}

\*(table + i) = nullptr;

//로또 복권 번호 생성

bool state;

int temp;

srand(time(nullptr));

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

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

state = TRUE;

temp = rand() % 45 + 1;

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

if (table[i][k] == temp) {

state = false;

break;

}

}

if (state) table[i][j] = temp;

else j--;

}

}

}

**\*\*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

}

**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

}

**교수님 코드**

#include <iostream>

#include <string>

using namespace std;

class Account {

const string \_\_name;

const string \_\_account;

string \_\_pass;

int \_\_balance;

public:

Account(string name, string account, string pass = "11111", int balance = 1);

void setPass(string);

void setBalance(int);

string getName(void)const;

string getAccount(void) const;

string getPass(void)const;

int getBalance(void)const;

void showAccount(void)const;

};

Account::Account(string name, string account, string pass, int balance)

: \_\_name(name), \_\_account(account) {//멤버 초기화 리스트

\_\_pass = pass;

\_\_balance = balance;

}

int main(void) {

Account user("홍길동", "111-2222-3333", "12345", 1000000);

string userAccount, userPass;

user.showAccount;

try {

cout << "\n###### 은행 계좌 인증 #####" << endl;

cout << "계좌번호 입력: "; cin >> userAccount;

cout << "비밀번호 입력: "; cin >> userPass;

if (user.getAccount() != userAccount || user.getPass() != userPass)

throw Account("", userAccount, userPass, 0);

int money;

cout << "출금액 입력: "; cin >> money;

if (user.getBalance() < money)

throw money;

int num = a.getBalance() - money;

user.getBalance(num);

cout << user.getName() << "님의 현재 잔액은: " << user.getBalance() << "입니다." << endl;

}

catch (int money) {//예외처리 : 정수형

cout << "현재 잔액: " << a.getBalance() << endl;

cout << "희망 출금액: (" << money << ")이 너무 많네요." << endl;

}

catch (Account& exception) {//예외처리 : 객체

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

exception.showAccount();

}

return 0;

}

**\*\*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)

**교수님 코드**

Point operator+(const Point& p1, const Point& p2) {

Point temp(p1.x + p2.x, p1.y + p2.y);

return temp;

}

Point operator-(const Point& p1, const Point& p2) {

Point temp(p1.x + p2.x, p1.y + p2.y);

return temp;

}

};

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;

}

**\*\*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;

}

**\*\*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.