C++_자료구조(컬렉션) & 알고리즘

collection

템플릿– 함수 템플릿

▪ 템플릿(template) 이란?

- 템플릿은 함수나 클래스 코드를 찍어내듯이 생산할 수 있도록 일반화 (generic) 시키는 도구이다.
- 함수 템플릿은 함수 내에서 사용하는 자료형을 일반화된 유형으로 정의하여 그 함수를 호출할때 적절한 자료형을 대입해서 사용

- **템플릿 선언과 제네릭 타입**

  템플릿을 선언할 때는 **template** 키워드를 사용한다.

  template <typename 일반화 유형 이름>

  template<typename T>

  T는 임의의 데이터 형식(자료형)

템플릿– 함수 템플릿

- **템플릿(template) 예제**

```
class Math {
public:
    template <typename T>
    static T abs(T x) {
        return (x < 0) ? -x : x;
    }

    template <typename T>
    static T max(T x, T y) {
        return (x > y) ? x : y;
    }

    template <typename T>
    static T min(T x, T y) {
        return (x < y) ? x : y;
    }
};
```

템플릿- 함수 템플릿

- **템플릿(template) 예제**

```
int main()
{
    // 정적 함수 사용
    cout << Math::abs(-100) << endl;    // 100
    cout << Math::abs(-3.5) << endl;    // 3.5, double 타입 지원

    cout << Math::max(10, 20) << endl;     // 20
    cout << Math::min(5.4, 7.2) << endl;   // 5.4

    return 0;
}
```

STL – 표준 템플릿 라이브러리

C++의 **표준 템플릿 라이브러리(Standard Template Library, STL)**는 다양한 자료구조와 알고리즘들을 미리 만들어서 제공하는 라이브러리를 말한다.

- 컨테이너 : 자료를 저장하는 창고로 **벡터**, **리스트, 큐, 맵** 등
- 알고리즘 : 탐색이나 정렬과 같은 다양한 알고리즘 제공
- 반복자(iterator) : 컨테이너에 저장된 자료들을 순회하는 객체이다. 포인터와 비슷한 동작을 한다.

□ **컨테이너**는 같은 타입의 여러 객체를 저장할 수 있는 묶음 단위의 데이터 구조이다.
쉽게 말해서 화물을 싣는 컨테이너 또는 마트에서 물건을 담는 쇼핑카라고 할 수 있음

벡터(vector)

❖ 벡터(vector)

- vector는 내부에 배열을 가지고 원소를 저장, 삭제, 검색하는 가변 길이 배열을 구현한 클래스이다.
- 정적인 배열의 단점을 보완한 동적 배열로 배열의 크기 변경 및 데이터를 효율적으로 관리.

vector 객체 생성

vector <자료형> 객체 이름

삽입 : push_back()

수정 : vi[0] = 3

벡터(vector)

❖ 벡터(vector)

- **주요 함수**

함수명	기능
push_back(x)	맨 뒤에 x 요소 추가
insert(pos, x)	원하는 위치에 x 삽입
at(x)	x 인덱스의 요소 위치
front()	첫번째 요소 검색
back()	마지막 요소 검색
size()	벡터에 저장된 요소의 개수
empty()	비어있는 지 확인
erase()	특정 요소 삭제
pop_back()	맨 뒤 요소 제거

벡터(vector)

❖ vector에 저장과 검색

```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    vector<int> vec; //정수형 벡터 생성

    //요소 저장
    /*vec.push_back(80);
    vec.push_back(75);
    vec.push_back(90);*/

    vec = { 80, 75, 90 };

    //특정 요소 검색
    cout << "첫 번째: " << vec.front() << endl;
    cout << "마지막: " << vec.back() << endl;
    cout << vec[0] << endl;  //80
    cout << vec.at(1) << endl; //75
```

이터레이터(iterator) – 반복자

❖ 이터레이터(iterator) - 반복자

반복자는 컨테이너에 저장된 자료들을 순회하는 객체이다.
포인터와 유사한 동작을 함.

```
cout << "크기: " << vec.size() << endl;

//전체 조회
for (int i = 0; i < vec.size(); i++) {
    cout << vec[i] << endl;
}
cout << "----------------------\n";

//반복자 설정
vector<int>::iterator it = vec.begin();

//인덱싱
cout << *it << endl;  //80
cout << *(it + 1) << endl;  //75
cout << *(it + 2) << endl;  //90
cout << "----------------------\n";
```

이터레이터(iterator) – 반복자

❖ 이터레이터(iterator) - 반복자

```
//반복자로 전체 조회
cout << *vec.begin() << endl;  //첫번째 요소
cout << *(vec.begin() + 1) << endl;  //두번째 요소
cout << *(vec.end() - 1) << endl;  //마지막 요소
cout << "----------------------\n";

for (it = vec.begin(); it != vec.end(); it++) {
    cout << *it << endl;
}
cout << "----------------------\n";
```

이터레이터(iterator) – 반복자

❖ 벡터 요소 수정 및 삭제

```
//요소 수정
//vec[1] = 100;
vec.at(1) = 100;

//마지막 요소 삭제
vec.pop_back();

//특정 요소 삭제
/*for (it = vec.begin(); it != vec.end(); it++) {
    if (*it == 90) {
        vec.erase(it);
        break;
    }
}*/

//범위 기반 for
for (auto v : vec) {
    cout << v << endl;
}
```

vector를 활용한 도서 관리

❖ 벡터(vector) – 객체 저장

```
#include <iostream>
#include <vector>
using namespace std;

class Book {
private:
    int number;    //책번호
    string title;  //책제목
    string author; //저자

public:
    //생성자
    Book(int number, string title, string author);

    int getNumber();
    string getTitle();
    string getAuthor();
    void showBookInfo(); //책 정보 출력
};
```

vector를 활용한 도서 관리

❖ 벡터(vector) – 객체 저장

```
Book::Book(int number, string title, string author) {
    this->number = number;
    this->title = title;
    this->author = author;
}

int Book::getNumber() {
    return number;
}

string Book::getTitle() {
    return title;
}

string Book::getAuthor() {
    return author;
}

void Book::bookInfo() {
    cout << "책 번호 : " << getNumber() << endl;
    cout << "책 제목 : " << getTitle() << endl;
    cout << "책 저자 : " << getAuthor() << endl;
}
```

vector를 활용한 도서 관리

❖ Book Test

```
//Book 인스턴스를 저장할 vector 생성
vector<Book> books;

//도서 생성
books.push_back(Book(100, "채식주의자", "한강"));
books.push_back(Book(101, "C++ 완전정복", "조규남"));
books.push_back(Book(102, "모두의 C언어", "이형우"));

//특정 요소 검색 - 인덱싱
//books[1].showBookInfo(); //1번 요소 검색
//books.at(1).showBookInfo();

//특정 요소 검색 - 반복자 or auto 중 1개만 사용할 것
//vector<Book>::iterator it = books.begin(); //0번 요소
//it->showBookInfo();

//특정 요소 검색 - auto
auto it = books.begin() + 1; //1번 요소
it->showBookInfo();
```

vector를 활용한 도서 관리

❖ Book Test

```
cout << "********** 책의 정보 **********\n";
for (int i = 0; i < books.size(); i++) {
    books[i].showBookInfo();
    cout << "================================\n";
}
cout << endl;

//도서 삭제 - 2번 요소
books.erase(books.begin() + 2);

cout << "***** 삭제후 책의 정보 *****\n";
for (auto book : books) {
    book.showBookInfo();
    cout << "================================\n";
}
```

```
********** 책의 정보 **********
책 번호: 100
책 제목: 채식주의자
저자: 한강
================================
책 번호: 101
책 제목: C++ 완전정복
저자: 조규남
================================
책 번호: 102
책 제목: 모두의 C언어
저자: 이형우
================================

***** 삭제후 책의 정보 *****
책 번호: 100
책 제목: 채식주의자
저자: 한강
================================
책 번호: 101
책 제목: C++ 완전정복
저자: 조규남
================================
```

맵(map)

❖ 맵(map)

- 키(key)와 값(value)의 쌍을 원소로 저장하고 '키'를 이용하여 값을 검색하는 컨테이너이다.
- 순서 없이 저장되고 출력됨

map 객체 생성

map <키 자료형, 값 자료형> 객체 이름

map <string, int> dogs

삽입 : dogs.insert({"진돗개", 1})

수정 : dogs["진돗개"] = 2

맵(map)

❖ 맵(map)

- **주요 함수**

함수명	기능
insert({key, value})	요소 추가(키, 값)
size()	저장된 요소의 개수
at(key)	key로 값에 접근
find(key)	키를 가진 요소의 반복자 반환
empty()	비어 있는지 확인
erase(key)	key에 해당하는 요소 삭제

맵(map)

- **맵(map) 자료구조**

```cpp
#include <iostream>
#include <map>
using namespace std;

int main()
{
    map<string, int> dogs; //map 자료구조 생성

    //요소 추가
    dogs.insert({ "말티즈", 3 });
    dogs.insert({ "진돗개", 2 });
    dogs.insert({ "불독", 4 });
    dogs["푸들"] = 1;  //요소 추가

    cout << dogs.size() << endl; //저장된 요소의 개수

    //요소 검색
    cout << dogs["말티즈"] << "세\n";
    cout << dogs.at("말티즈") << "세\n";
```

```
3
3세
말티즈, 3
불독, 4
진돗개, 2
====================
말티즈, 3
불독, 4
진돗개, 2
====================
말티즈, 3
진돗개, 2
```

맵(map)

- **맵(map) 자료구조**

```
//전체 검색 - 반복자 사용
map<string, int>::iterator it;
for (it = dogs.begin(); it != dogs.end(); it++) {
    cout << it->first << ", " << it->second << endl;
}
cout << "====================\n";

//전체 검색 - auto
for (auto it = dogs.begin(); it != dogs.end(); it++) {
    cout << it->first << ", " << it->second << endl;
}
cout << "====================\n";

//요소 삭제
dogs.erase("불독");

for (auto dog : dogs)
    cout << dog.first << ", " << dog.second << endl;
```

맵(map)

- **컴퓨터 용어 사전**

```
검색할 단어 입력(exit - 종료)>> 비트
정보 기술에서 데이터의 가장 작은 단위로, 0 또는 1의 값을 가진다
검색할 단어 입력(exit - 종료)>> 컴파일
프로그래밍 언어로 작성된 소스 코드를 컴퓨터가
이해하고 실행할 수 있는 기계어로 변환하는 과정을 말한다.
검색할 단어 입력(exit - 종료)>> 알고리즘
찾는 단어가 없습니다.
검색할 단어 입력(exit - 종료)>> exit
검색을 종료합니다
```

- 프로그램 설명
  - 컴퓨터 용어 사전 프로그램을 구현한다.
  - Map을 사용하여 단어와 그 정의를 저장한다.
  - 사용자 입력을 통해 단어를 검색할 수 있는 기능을 제공한다.

맵(map)

- **컴퓨터 용어 사전**

```
#include <iostream>
#include <map>
#include <string> //getline()
using namespace std;

int main()
{
    map<string, string> dict; //map 자료구조 생성
    string eng; //검색할 단어

    //단어 저장
    dict.insert({ "이진수", "컴퓨터가 사용하는 0과 1로 이루어진 수" });
    dict.insert({ "비트", "정보 기술에서 데이터의 가장 작은 단위로, "
        "0 또는 1의 값을 가진다" });
    dict.insert({ "컴파일", "프로그래밍 언어로 작성된 소스 코드를 컴퓨터가 "
        "이해하고 실행할 수 있는 기계어로 변환하는 과정을 말한다."});
```

맵(map)

- **컴퓨터 용어 사전**

```
/* //특정 단어 검색
cout << dict["비트"] << endl;

auto it = dict.find("비트");
cout << it->first << ": " << it->second << endl;

//전체 검색
for (auto& dic : dict)
    cout << dic.first << ": " << dic.second << endl;*/
```

맵(map)

- **컴퓨터 용어 사전**

```
//단어 검색
while (true) {
    cout << "검색할 단어 입력(exit - 종료)>> ";
    getline(cin, eng); //공백 문자 허용

    if (eng == "exit") {
        break; //반복 종료
    }
    else if (dict.find(eng) == dict.end()) {
        cout << "찾는 단어가 없습니다.\n";
    }
    else {
        cout << dict[eng] << endl;
    }
}
cout << "검색을 종료합니다\n";
```

정렬 – sort()

- **오름차순 정렬**

```
int a[5] = { 3, 2, 5, 4, 1 };
int temp;

//cout << size(a) << endl;

//오름차순 정렬
for (int i = 0; i < size(a) - 1; i++) { //4
    for (int j = i + 1; j < size(a); j++) { //5
        if (a[i] > a[j]) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        }
    }
}
```

정렬 – sort()

- **오름차순 정렬**

```
/*
    { 3, 2, 5, 4, 1 }
    i=0, j=1, 3>2, 2,3,5,4,1
          j=2, 2>5, 교환없음
          j=3, 2>4,
          j=4, 2>1, 1,3,5,4,2
    i=1, j=2, 3>5,
          j=3, 3>4,
          j=4, 3>2, 1,2,5,4,3
    i=2, j=3, 5>4, 1,2,4,5,3
          j=4, 4>3, 1,2,3,5,4
    i=3, j=4, 5>4, 1,2,3,4,5 정렬 완료!
*/

for (int i = 0; i < 5; i++) {
    cout << a[i] << " ";
}
```

```
1 2 3 4 5
```

정렬 – sort()

- 오름차순 정렬 - <algorithm> 헤더 파일 필요

```
#include <iostream>
#include <vector>
#include <algorithm>  //sort() 사용
using namespace std;

int main()
{
    vector<int> vec = { 3, 2, 5, 4, 1 };
    sort(vec.begin(), vec.end()); //오름차순 정렬

    //반복자(iterator)
    /*vector<int>::iterator it;
    for (it = vec.begin(); it != vec.end(); it++) {
        cout << *it << " ";
    }*/
```

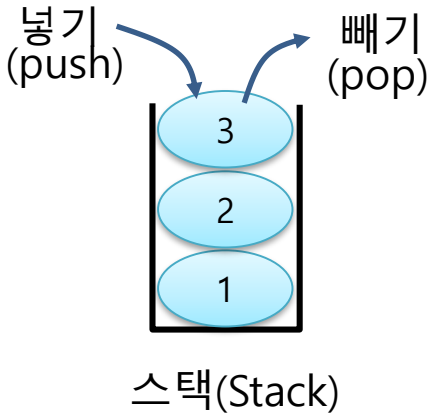
정렬 – sort()

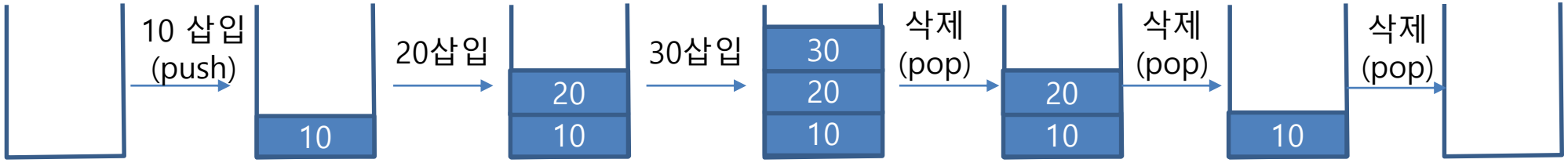
- **내림차순 정렬**

```
for (auto v : vec)
    cout << v << " ";
cout << endl;

//내림차순 정렬
sort(vec.begin(), vec.end(), greater<int>());

for (auto it = vec.begin(); it != vec.end(); it++) {
    cout << *it << " ";
}
```

스택(Stack)

- **스택(Stack)**
  - **후입선출(LIFO : Last in First Out)** 구조
    나중에 들어간 자료를 먼저 꺼냄
    (응용 예: 접시 닦이, 스택 메모리, 게임 무르기)





스택(Stack)

● 스택(Stack)

- **주요 함수**

함수명	기능
push(x)	맨 위에 x 추가.
pop()	맨 위 요소 제거.
top()	맨 앞(위) 요소 반환
empty()	스택이 비었는지 확인
size()	스택에 들어있는 요소의 개수

스택(Stack)

- **스택(Stack)**

```cpp
#include <iostream>
#include <stack>
using namespace std;

int main()
{
    //정수를 저장할 stack 컨테이너 생성
    stack<int> stack;

    //요소 저장
    stack.push(1);
    stack.push(2);
    stack.push(3);

    //스택의 크기
    cout << stack.size() << endl; //3
```

스택(Stack)

- **스택(Stack)**

```
//스택의 맨 위 요소
cout << stack.top() << endl;  //3

stack.pop(); //스택에서 데이터 제거
cout << stack.top() << endl; //2

stack.pop();
cout << stack.top() << endl; //1

stack.pop();

//스택이 비었는지 확인
if (stack.empty()) {
    cout << "스택이 비었습니다.\n";
}
else {
    cout << "스택이 비어 있지 않습니다.\n";
}
```

```
3
3
2
1
스택이 비었습니다.
```

스택(Stack)

- **스택 – 문자열 뒤집기**

```
stack<char> stk;

//요소 추가 : a - b - c
stk.push('a');
stk.push('b');
stk.push('c');

while (!stk.empty()) {
    cout << stk.top() << " ";
    stk.pop();  //요소 삭제 : c - b - a
}
```

스택(Stack)

- **스택 – 문자열 뒤집기**

```
//문자열 뒤집기
string str;
cout << "문자열 입력: ";
cin >> str;

stack<char> stk;
for (char c : str) {
    stk.push(c);  //문자 1개 저장
}

cout << "뒤집은 문자열: ";
while (!stk.empty()) {
    cout << stk.top();
    stk.pop();
}
cout << endl;
```

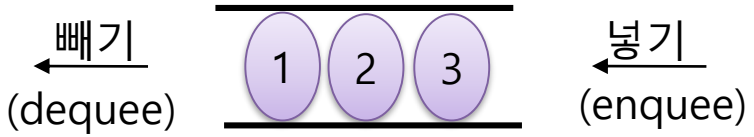
```
문자열 입력: net
뒤집은 문자열: ten
```

큐(Queue)

- **큐(Queue)**
  - 선입선출(FIFO : First in First Out) 구조

    배열에서 먼저 들어간 자료를 먼저 꺼냄

    (응용 예: 버스정류장 줄서기, 운영체제 작업큐)



큐(Queue)

- **큐(Queue)**
  - 주요 함수

함수명	기능
push(x)	맨 뒤에 x 추가.
pop()	맨 앞 요소 제거.
front()	맨 앞 요소 반환
back()	맨 뒤 요소 반환
empty()	큐가 비었는지 확인
size()	큐에 들어있는 요소의 개수

큐(Queue)

- 큐(Queue)

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main() {
    queue<int> que;

    // 요소 추가 : 10 - 20 - 30
    que.push(10);
    que.push(20);
    que.push(30);

    cout << "큐의 크기: " << que.size() << endl;
    cout << "첫 번째 요소: " << que.front() << endl; //10
    cout << "맨 뒤 요소: " << que.back() << endl;    //30
```

큐(Queue)

- 큐(Queue)

```
// 요소 제거 (dequeue)
que.pop(); // 10 제거
cout << "다음 요소: " << que.front() << endl;

// 큐 출력
while (!que.empty()) {
    cout << que.front() << " ";
    que.pop(); //요소 제거: 10 - 20 - 30
}
```

```
큐의 크기: 3
첫 번째 요소: 10
맨 뒤 요소: 30
다음 요소: 20
20 30
```

큐(Queue)

- **은행 대기줄**

```
queue<string> q;

// 고객 대기열
q.push("고객A");
q.push("고객B");
q.push("고객C");

while (!q.empty()) {
    cout << q.front() << "님 업무 처리 중..." << endl;
    q.pop();
}

cout << "모든 고객의 업무가 완료되었습니다.\n";
```

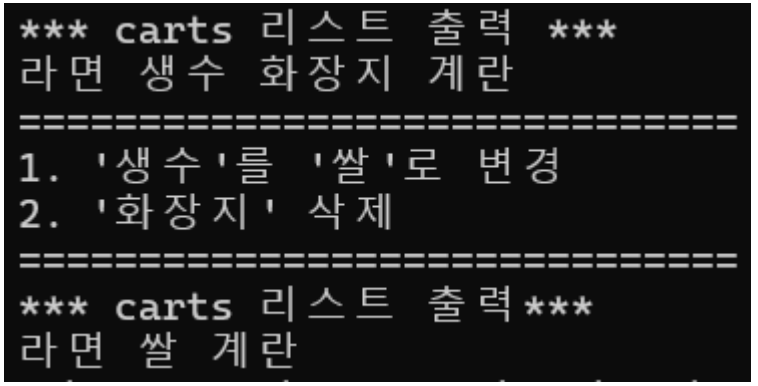
실습 문제 1 - vector

---

carts 리스트를 벡터를 사용하여 아래와 같이 구현하세요

[파일이름: cartList.cpp]

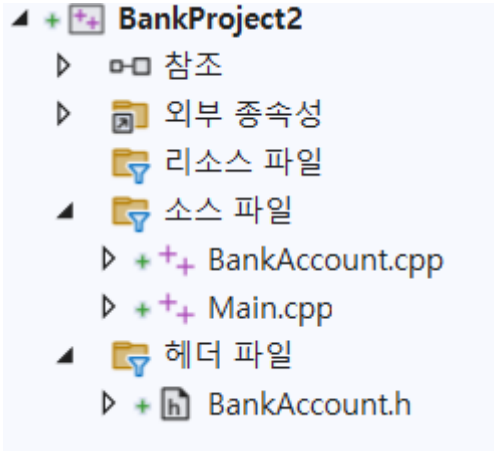
☞ 실행 결과



은행 거래 프로젝트

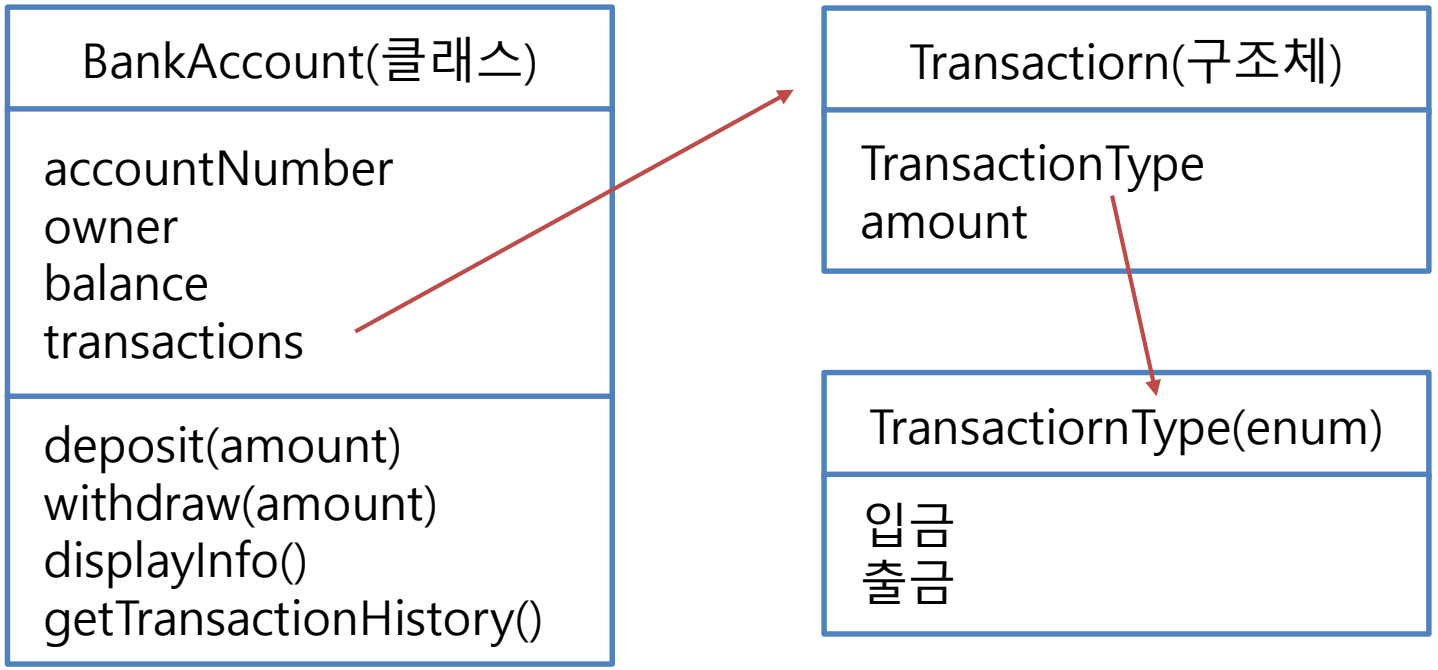
❖ 프로젝트 구조

1. BankAccount.h – 헤더 파일(클래스, 구조체)
2. BankAccount.cpp – 함수 구현부
3. Main.cpp – 실행 파일(객체 생성)



은행 거래 프로젝트

❖ 은행 거래 내역



은행 거래 프로젝트

❖ 은행 거래 내역

입력 화면

```
=====================================================
1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료
=====================================================
```

1. createAccount()
2. deposit()
3. withdraw()
4. displayAccount()

은행 거래 프로젝트

계좌 생성>

```
============================================================
1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료
============================================================
선택> 1
계좌주를 입력하세요: 나저축
계좌가 성공적으로 생성되었습니다! (계좌번호: 1000)
```

예금>

```
============================================================
1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료
============================================================
선택> 2
입금할 계좌번호를 입력하세요: 1000
입금할 금액을 입력하세요: 20000
20000원이 입금되었습니다. 현재 잔액: 20000원
```

출금>

```
============================================================
1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료
============================================================
선택> 3
출금할 계좌번호를 입력하세요: 1000
출금할 금액을 입력하세요: 5000
5000원이 출금되었습니다. 현재 잔액: 15000원
```

은행 거래 프로젝트

계좌 검색>

```
=============================================================
1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료
=============================================================
선택> 4
조회할 계좌번호를 입력하세요: 1000

*계좌 정보
    계좌 번호: 1000
    계좌주: 나저축
    잔고: 15000
[나저축] 계좌 거래 내역(최근 2건)
1 | 입금 | 20000원
2 | 출금 | 5000원
```

은행 거래 프로젝트

❖ BankAccount 클래스 – BankAccount.h

```
#include <iostream>
#include <vector>
using namespace std;

//enum 자료형
enum TransactionType {
    입금,
    출금
};

//구조체
struct Transaction {
    TransactionType type;
    int amount;
};
```

은행 거래 프로젝트

❖ BankAccount 클래스

```
class BankAccount {
private:
    int accountNumber;
    string owner;
    int balance;
    vector<Transaction> transactions;

public:
    BankAccount(int accountNumber, string owner, int balance = 0) :
        accountNumber(accountNumber), owner(owner), balance(balance) {
    }

    int getAccountNumber();
    void deposit(int amount);
    void withdraw(int amount);
    void displayInfo();
    void getTransactionHistory();

private:
    void addTransaction(TransactionType type, int amount);
};
```

은행 거래 프로젝트

❖ 입금 – BankAccount.cpp

```
#include "BankAccount.h"

//계좌 번호 반환
int BankAccount::getAccountNumber() {
    return accountNumber;
}

//입금
void BankAccount::deposit(int amount) {
    if (amount < 0) {
        cout << "유효한 금액을 입력하세요.\n";
    }
    else {
        balance += amount;
        cout << amount << "원이 입금되었습니다. 현재 잔액: " <<
            balance << "원\n";
        addTransaction(TransactionType::입금, amount);
    }
}
```

은행 거래 프로젝트

❖ 출금

```
void BankAccount::withdraw(int amount) {
    if (amount < 0) {
        cout << "유효한 금액을 입력하세요.\n";
    }
    else if (amount > balance) {
        cout << "잔액이 부족합니다. 다시 입력하세요.\n";
    }
    else {
        balance -= amount;
        cout << amount << "원이 출금되었습니다. 현재 잔액: " <<
            balance << "원\n";
        addTransaction(TransactionType::출금, amount);
    }
}
```

은행 거래 프로젝트

❖ 거래 내역

```
void BankAccount::addTransaction(TransactionType type, int amount) {
    Transaction trans; //거래 1건 생성
    trans.type = type;
    trans.amount = amount;
    //벡터에 거래 1건씩 저장
    transactions.push_back(trans);
}

void BankAccount::getTransactionHistory() {  //거래 내역 조회
    cout << "[" << owner << "] 계좌 거래 내역(최근 " << transactions.size() << "건)\n";
    if (transactions.empty()) {
        cout << "거래 내역이 없습니다.\n";
        return;
    }

    int i = 1;  //거래 내역 번호
    for (const auto& trans : transactions) {
        cout << i++ << " | " << (trans.type == TransactionType::입금 ? "입금" : "출금");
        cout << " | " << trans.amount << "원\n";
    }
}
```

은행 거래 프로젝트

❖ 계좌 정보 출력

```
//계좌 정보
void BankAccount::displayInfo() {
    cout << "\n*계좌 정보\n";
    cout << "    계좌 번호: " << accountNumber << endl;
    cout << "    계좌주: " << owner << endl;
    cout << "    잔고: " << balance << endl;
}
```

은행 거래 프로젝트

❖ **Main.cpp**

```
#include "BankAccount.h"

vector<BankAccount> accounts; //계좌를 저장한 벡터 생성
int nextAccountNumber = 1000;  //1000번 부터 생성

//계좌 생성하다
void createAccount() {
    string name;
    cout << "계좌주를 입력하세요: ";
    cin >> name;

    BankAccount newAccount(nextAccountNumber, name); //신규 계좌 생성
    accounts.push_back(newAccount); //벡터에 계좌 저장
    cout << "계좌가 성공적으로 생성되었습니다! (계좌번호: " <<
        nextAccountNumber << ")\n";
    nextAccountNumber++;  //계좌번호 1증가
}
```

은행 거래 프로젝트

❖ **Main.cpp**

```
//계좌 검색
BankAccount* searchAccount(int accNum) {
    for (auto& account : accounts) {
        //이미 등록된 계좌와 입력한 계좌가 일치하면
        if (account.getAccountNumber() == accNum) {
            return &account; //계좌 주소 반환
        }
    }
    return nullptr;
}
```

은행 거래 프로젝트

❖ **Main.cpp**

```
void deposit() {
    int accNum;
    int amount;

    cout << "입금할 계좌번호를 입력하세요: ";
    cin >> accNum;

    BankAccount* account = searchAccount(accNum); //객체 생성
    if (account) {
        cout << "입금할 금액을 입력하세요: ";
        cin >> amount;
        if (cin.fail()) { //문자를 입력한 경우 오류 처리
            cin.clear(); //초기화
            //최대 1000개의 문자를 읽어서 '\n' 까지 무시함
            cin.ignore(1000, '\n');
            cout << "숫자를 입력하세요.\n";
            return;
        }
        account->deposit(amount);
    }
    else
        cout << "계좌번호를 찾을 수 없습니다.\n";
}
```

은행 거래 프로젝트

❖ **Main.cpp**

```
void withdraw() {
    int accNum;
    int amount;

    cout << "출금할 계좌번호를 입력하세요: ";
    cin >> accNum;

    BankAccount* account = searchAccount(accNum);
    if (account) {
        cout << "출금할 금액을 입력하세요: ";
        cin >> amount;
        if (cin.fail()) {
            cin.clear();
            cin.ignore(1000, '\n');
            cout << "숫자를 입력하세요.\n";
            return;
        }
        account->withdraw(amount);
    }
    else
        cout << "계좌번호를 찾을 수 없습니다.\n";
}
```

은행 거래 프로젝트

❖ **Main.cpp**

```
void displayAccount() {
    int accNum;
    cout << "조회할 계좌번호를 입력하세요: ";
    cin >> accNum;

    BankAccount* account = searchAccount(accNum);
    if (account) {
        account->displayInfo();  //계좌 정보 출력
        account->getTransactionHistory(); //거래 내역 출력
    }
    else
        cout << "계좌번호를 찾을 수 없습니다.\n";
}
```

은행 거래 프로젝트

❖ Main.cpp

```
int main()
{
    int choice;  //메뉴
    bool run = true; //상태 변수

    while (run) {
        cout << "=============================================================" << endl;
        cout << "1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료    " << endl;
        cout << "=============================================================" << endl;
        cout << "선택> ";
        cin >> choice;

        switch (choice) {
        case 1:
            createAccount();
            break;
        case 2:
            deposit();
            break;
```

은행 거래 프로젝트

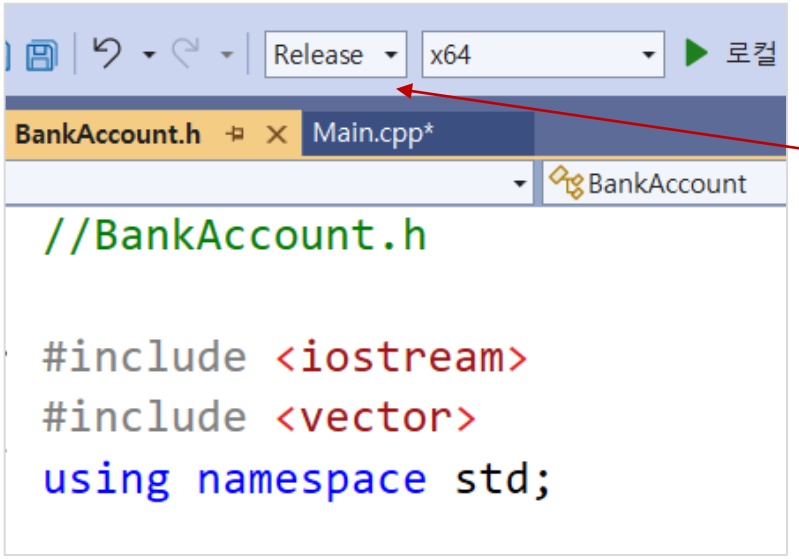
❖ **Main.cpp**

```
            case 3:
                withdraw();
                break;
            case 4:
                displayAccount();
                break;
            case 5:
                run = false;
                cout << "프로그램을 종료합니다.";
                break;
            default:
                cout << "지원되지 않는 기능입니다.\n";
                break;
            }
        } //while 종료

        system("pause"); //exe 파일에서 윈도우 꺼짐 방지
        return 0;
}
```

파일 배포

◈ 파일 배포

파일 배포란 c++언어 소스파일을 .exe 실행 파일로 만들어 공개 및 서비스 하는 것을 말한다.



Debug 모드를
Release 모드로 바꾼다.

Ctrl + F5로 실행

파일 배포

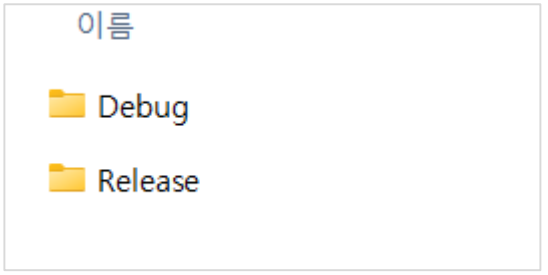
◈ exe 파일에서 윈도우 꺼짐 문제 해결

system("pause") 를 명시함

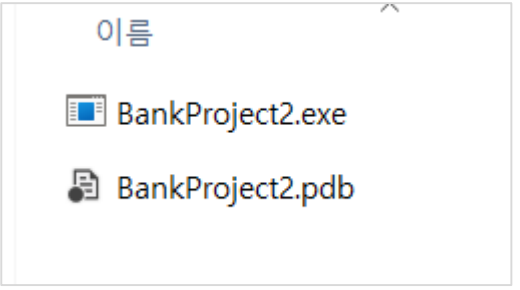
```
            case 5:
                run = false;
                cout << "프로그램을 종료합니다.";
                break;
            default:
                cout << "지원되지 않는 기능입니다.\n";
                break;
            }
        } //while 종료

        system("pause"); //exe 파일에서 윈도우 꺼짐 방지
        return 0;
}
```

파일 배포

◈ 파일 실행하기



Release 폴더 생성됨



BankProject2.exe 파일 생성

파일 배포

◈ 파일 실행하기

```
1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료
=========================================================
선택> 1
계좌주를 입력하세요: 김기용
계좌가 성공적으로 생성되었습니다! (계좌번호: 1000)
=========================================================
1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료
=========================================================
선택> 2
입금할 계좌번호를 입력하세요: 1000
입금할 금액을 입력하세요: 20000
20000원이 입금되었습니다. 현재 잔액: 20000원
=========================================================
1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료
=========================================================
선택> 3
출금할 계좌번호를 입력하세요: 1000
출금할 금액을 입력하세요: 5000
5000원이 출금되었습니다. 현재 잔액: 15000원
=========================================================
1. 계좌생성 | 2. 예금 | 3. 출금 | 4. 계좌검색 | 5. 종료
=========================================================
선택> 4
조회할 계좌번호를 입력하세요: 1000

*계좌 정보
    계좌 번호: 1000
    계좌주: 김기용
    잔고: 15000
[김기용] 계좌 거래 내역(최근 2건)
1 | 입금 | 20000원
2 | 출금 | 5000원
```