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)

❖ vector에 저장과 검색

```
//점수를 저장할 정수형 벡터 생성
vector<int> vec;

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

//인덱싱
cout << vec[1] << endl;  //75

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

이터레이터(iterator) – 반복자

❖ 이터레이터(iterator) - 반복자

반복자는 컨테이너에 저장된 자료들을 순회하는 객체이다.

포인터와 유사한 동작을 함.

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

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

//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";
```

이터레이터(iterator) – 반복자

❖ 이터레이터(iterator) - 반복자

반복자는 컨테이너에 저장된 자료들을 순회하는 객체이다.

포인터와 유사한 동작을 함.

```
//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 <string>
#include <vector>
using namespace std;

//Book 클래스 정의
class Book {
private:
    int number;    //책 번호
    string title;  //책 제목
    string author; //저자
public:
    //생성자
    Book(int number, string title, string author);

    //멤버 함수
    int getNumber();
    string getTitle();
    string getAuthor();
    void bookInfo();
};
```

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

❖ 벡터(vector) – 객체 저장

```
//객체 배열
/*Book book[3] = {
    Book(100, "채식주의자", "한강"),
    Book(101, "C++ 완전정복", "조규남"),
    Book(102, "모두의 C언어", "이형우"),
};*/

//vector 자료구조로 객체 생성
vector<Book> books;

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

cout << "********** 책의 정보 **********" << endl;
for (int i = 0; i < 3; i++)
{
    books[i].bookInfo();
    cout << "==============================\n";
}
```

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

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) 자료구조**

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

int main()
{
    //강아지의 종류와 나이를 저장할 map 컨테이너
    map<string, int> dogs;

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

    //map의 크기
    cout << dogs.size() << endl;

    //요소 검색
    cout << dogs["말티즈"] << "세\n";
```

맵(map)

- **맵(map) 자료구조**

```
//요소 삭제
//dogs.erase("불독");

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

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

//향상된 검색
for (auto dog : dogs) {
    cout << dog.first << " " << dog.second << endl;
}
```

```
3
3세
말티즈 3
불독 4
진돗개 1

말티즈 3
불독 4
진돗개 1
```

맵(map)

- **단어를 저장하고 검색하기**

```
찾고 싶은 단어(exit 입력시 종료)>> body
몸
찾고 싶은 단어(exit 입력시 종료)>> korea
대한민국
찾고 싶은 단어(exit 입력시 종료)>> sky
찾는 단어가 없음
찾고 싶은 단어(exit 입력시 종료)>> sea
바다
찾고 싶은 단어(exit 입력시 종료)>> exit
검색을 종료합니다
```

```
//영어 단어와 뜻을 저장할 map 컨테이너
map<string, string> dic;
string eng; //영어 단어(키) 저장 변수

//단어 3개 저장
dic.insert({ "sea", "바다" });
dic.insert({ "korea", "대한민국" });
dic.insert({ "body", "몸" });
dic["smile"] = "미소";  //단어 추가
```

맵(map)

- **단어를 저장하고 검색하기**

```
//저장된 단어 찾기
while (true) {
    cout << "찾고 싶은 단어(exit 입력시 종료)>> ";
    //cin >> eng;
    getline(cin, eng); //공백문자 허용
    if (eng == "exit") break;  // 종료

    if (dic.find(eng) == dic.end()) {
        cout << "찾는 단어가 없음\n";
    }
    else {
        cout << dic[eng] << endl;
    }
}
cout << "검색을 종료합니다\n";
return 0;
```

정렬 – sort()

- **오름차순 정렬**

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

int main()
{
    vector<int> vec = { 7, 6, 3, 5, 4, 1, 2, 0, 8 };
    sort(vec.begin(), vec.end()); //오름차순 정렬

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

정렬 – sort()

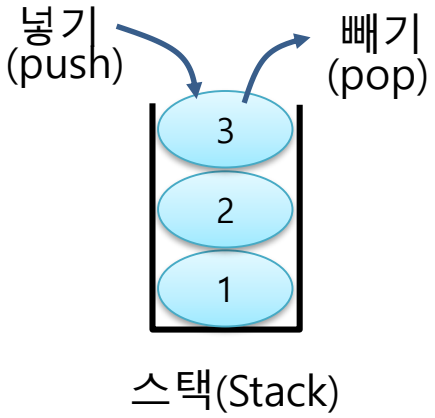
- **내림차순 정렬**

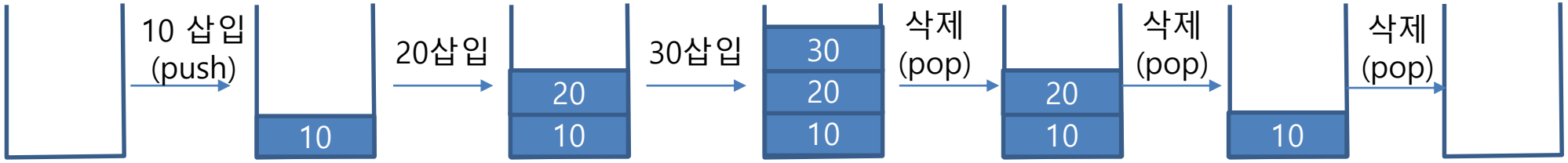
```
//내림차순 정렬
sort(vec.begin(), vec.end(), greater<int>());

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 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;
```

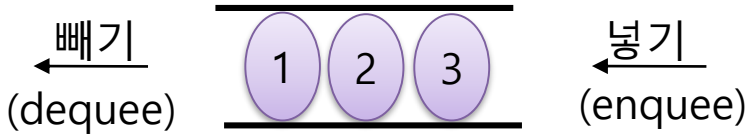
```
문자열 입력: net
뒤집은 문자열: ten
```

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;
```

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