

## 객체지향프로그래밍 LAB #13

### <기초문제>

1. 아래의 프로그램을 작성하시오. (/\*구현\*/ 부분을 채울 것, 표의 상단: 소스코드, 하단: 실행결과)

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

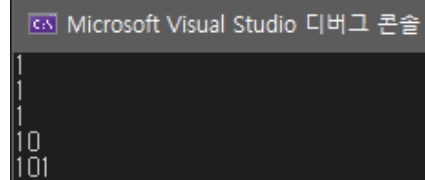
template</* 구현 */>
bool less_than(/* 구현 */) { return a < b; }

template<class T>
T sum(const vector<T>& v) { /* 구현 */ }

int main() {
    cout << less_than(2, 3) << endl;
    cout << less_than(2.1, 2.9) << endl;
    cout << less_than(2, 2.5) << endl;

    vector<int> v1{ 1, 2, 3, 4 };
    vector<double> v2{ 10.1, 20.2, 30.3, 40.4 };
    cout << sum(v1) << endl;
    cout << sum(v2) << endl;

    return 0;
}
```

The screenshot shows the output of the program in the Visual Studio console. The output consists of five lines: the first three lines are the results of the less\_than function calls, and the last two lines are the results of the sum function calls. The output is: 1, 1, 1, 10, 101.

```
1
1
1
10
101
```

2. 아래의 프로그램을 작성하시오. (/\*구현\*/ 부분을 채울 것, 표의 상단: 소스코드, 하단: 실행결과)

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

template<class T>
class Point {
private:
    T x;
    T y;
public:
    Point(T _x, T _y);
    void setXY(T _x, T _y);
    T getX();
    T getY();
    void print();
};
```

```

template<class T>
Point<T>::Point(T _x, T _y) : x(_x), y(_y) {}

template<class T>
void Point<T>::setXY(T _x, T _y) {
    x = _x;
    y = _y;
}

// getX() 구현

// getY() 구현

// print() 구현

int main() {
    Point<int> pt(1, 2);
    Point<double> pt2(1.1, 2.2);
    pt.print();
    pt2.print();
}

```

Microsoft Visual Studio 디버그 콘솔

```

1, 2
1.1, 2.2

```

3. 아래의 프로그램을 작성하시오. (\*구현\* 부분을 채울 것, 표의 상단: 소스코드, 하단: 실행결과)

```

#include <iostream>
#include <vector> // 빠른 search, 느린 pop/push
#include <list>   // 느린 search, 빠른 pop/push
using namespace std;

int main() {
    list<int> myList{ 1, 2, 3, 4 };
    char command;
    int inputVal;
    bool finished = false;
    while (!finished) {
        //command를 입력받음
        cout << "I)input, P)rint, L)ength, E)mpty, Q)uit>>";
        cin >> command;

        //command에 따라 기능 수행
        switch (command) {
            case 'I':
            case 'i':
                cin >> inputVal;
                // push_back 구현
                break;

```

```

        case 'P':
        case 'p':
            // simplified for로 list출력 구현
            break;
        case 'L':
        case 'l':
            cout << "Number of items: " << /* 구현 */ << endl;
            break;
        case 'E':
        case 'e':
            /* 구현 */
            break;
        case 'Q':
        case 'q':
            finished = true;
            cout << "Exit the program" << endl;
            break;
        default:
            cout << "Wrong command" << endl;
            break;
    }

    return 0;
}

```

Microsoft Visual Studio 디버그 콘솔

```

I)input, P)rint, L)ength, E)mpty, Q)uit>>p
1      2      3      4
I)input, P)rint, L)ength, E)mpty, Q)uit>>i
123
I)input, P)rint, L)ength, E)mpty, Q)uit>>p
1      2      3      4      123
I)input, P)rint, L)ength, E)mpty, Q)uit>>l
Number of items: 5
I)input, P)rint, L)ength, E)mpty, Q)uit>>e
I)input, P)rint, L)ength, E)mpty, Q)uit>>p
I)input, P)rint, L)ength, E)mpty, Q)uit>>q
Exit the program

```

4. 아래의 프로그램을 작성하시오. (/\*구현\*/ 부분을 채울 것, 표의 상단: 소스코드, 하단: 실행결과)

```

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

int main() {
    int ary[] = { 1, 2, 3, 4 };
    int* pBegin, *pEnd;
    pBegin = ary;
    pEnd = ary + 4;
    for (/* 구현 */)
        cout << *pIter << "Wt ";
    cout << endl;
}

```

```

        //auto, begin(), end()
        vector<int> v{ 10, 20, 30, 40 };
        auto iter_begin = begin(v);
        auto iter_end = end(v);
        for (/* 구현 */)
            cout << *iter << "Wt";
        cout << endl;

        return 0;
    }

```

Microsoft Visual Studio 디버그 콘솔

1	2	3	4
10	20	30	40

5. 아래의 프로그램을 작성하시오. (/\*구현\*/ 부분을 채울 것, 표의 상단: 소스코드, 하단: 실행결과)

```

#include <iostream>
#include <vector> // 빠른 search, 느린 pop/push
#include <list>   // 느린 search, 빠른 pop/push
using namespace std;

template</* 구현 */>
void print(const Iter& iter_begin, const Iter& iter_end) {
    for (/* 구현 */)
        cout << *iter << "Wt";
    cout << endl;
}

template</* 구현 */>
void print_reverse(const Iter& iter_begin, const Iter& iter_end) {
    Iter iter = iter_end;
    /* 구현 */
    cout << endl;
}

int main() {
    vector<int> v{ 1, 2, 3, 4 };
    list<double> l{ 1.1, 2.2, 3.3 };
    int ary[] = { 100, 200, 300, 400 };

    print(begin(v), end(v));
    print(begin(l), end(l));
    print(begin(ary), end(ary));

    print_reverse(begin(v), end(v));
    print_reverse(begin(l), end(l));
    print_reverse(begin(ary), end(ary));

    return 0;
}

```

C# Microsoft Visual Studio 디버그 콘솔			
1	2	3	4
1.1	2.2	3.3	
100	200	300	400
4	3	2	1
3.3	2.2	1.1	
400	300	200	100

### <응용문제>

1. 아래 코드를 기반으로 다양한 type(int, double, float)을 사용하여 추가, 삭제, 출력 기능을 하는 List class를 구현하고 이를 사용하는 프로그램을 작성하라. 이 때, list를 오름차순으로 정렬하여라.

조건 1. 오름차순으로 정렬을 하기 위해서는 중복된 데이터가 list 안에 있으면 안된다.

조건 2. list에 데이터를 추가하는 순간 정렬이 되어 있어야한다.

나와있는 조건 말고 다른 경우는 예외처리 하지 않아도 됨

Example)

입력 순서 → 3 - 4 - 5 - 1 - 2

list 안 데이터 순서 → 1 - 2 - 3 - 4 - 5

```
int command()
{
    int num;

    cout << "WnWt---- menu ----" << endl;
    cout << "Wt1. 리스트 추가" << endl;
    cout << "Wt2. 리스트 삭제" << endl;
    cout << "Wt3. 리스트 출력" << endl;
    cout << "Wt4. 프로그램 종료" << endl;
    cout << "WnWt입력 --> ";
    cin >> num;
    return num;
}

int main()
{
    CList<type> list; // type형으로 list 선언
    type input; // list에 입력 할 데이터
    int com; // 선택한 기능
    while (1)
    {
        com = command(); // 기능을 선택
        switch (com)
        {
            case 1: // 추가
                cout << "Wn추가할 데이터 : ";
                cin >> input;
                list.Add(input);
                break;
            case 2: // 삭제
                cout << "Wn삭제할 데이터 : ";
                cin >> input;
                list.Delete(input);
                break;
            case 3: // 출력
                list.Print();
        }
    }
}
```

```

        break;
    case 4: // 프로그램 종료
        cout << "WnWt 프로그램을 종료합니다Wn";
        return 0;
        break;
    default:
        break;
    }
}
return 0;}

```

## [참조 1]

```

template <typename T>
class CList
{
public:
    CList();
    ~CList();

    bool IsEmpty(); // list가 비어 있으면 1, 아니면 0
    bool IsFull(); // list가 꽉 차 있으면 1, 아니면 0

    void Add(T data); // list에 데이터 추가
    void Delete(T data); // list에 데이터 삭제
    void Print(); // list에 데이터 출력

private:
    T m_Array[5]; // 데이터를 저장할 공간
    int m_Length; // list에 있는 데이터 수
};

```

### 1 - 출력화면 :

#### <기본 화면>

```

---- menu ----
1. 리스트 추가
2. 리스트 삭제
3. 리스트 출력
4. 프로그램 종료

입력 -->

```

#### <추가>

```

---- menu ----
1. 리스트 추가
2. 리스트 삭제
3. 리스트 출력
4. 프로그램 종료

입력 --> 1
추가할 데이터 : 7

```

#### <추가 할 때 list가 차 있을 때>

```

---- menu ----
1. 리스트 추가
2. 리스트 삭제
3. 리스트 출력
4. 프로그램 종료

입력 --> 1
추가할 데이터 : 9
List is full.

```

#### <중복 된 데이터가 있을 시>

```

---- menu ----
1. 리스트 추가
2. 리스트 삭제
3. 리스트 출력
4. 프로그램 종료

입력 --> 1
추가할 데이터 : 7
중복된 데이터가 존재합니다.

```

#### <삭제>

```

---- menu ----
1. 리스트 추가
2. 리스트 삭제
3. 리스트 출력
4. 프로그램 종료

입력 --> 2
삭제할 데이터 : 7

```

#### <삭제 할 때 list가 비어 있을 때>

```
----- menu -----
1. 리스트 추가
2. 리스트 삭제
3. 리스트 출력
4. 프로그램 종료

입력 --> 2

삭제할 데이터 : 1

List is empty.
```

<출력>

```
----- menu -----
1. 리스트 추가
2. 리스트 삭제
3. 리스트 출력
4. 프로그램 종료

입력 --> 3

※ Current List
1 2 3 4 5
```

<종료>

```
----- menu -----
1. 리스트 추가
2. 리스트 삭제
3. 리스트 출력
4. 프로그램 종료

입력 --> 4

프로그램을 종료합니다
```

2. 아래의 조건을 만족하는 프로그램을 작성하라.

1. 크기가 10인 vector1과 vector2를 만든다.
2. vector1의 범위는 0~10이고 vector2의 범위는 0~20이며 난수로 채워진다.
3. vector1에 있는 어떠한 수와 vector2의 있는 어떠한 수를 곱 했을 때 가장 큰 경우(곱의 최댓값)과 최솟값을 찾는다.
4. 이 때 vector의 데이터에 접근하기 위해서 iterator만을 사용한다.

2 – 출력화면 :

```
<vetor 1>
8 9 7 5 1 7 5 6 3 9
<vetor 2>
8 19 16 0 20 5 6 8 14 18

최댓값 = 180
최솟값 = 0
```

3. 람다 함수를 활용하여 회문을 판별하는 프로그램을 작성하세요.

- 1) 단어를 뒤집어도 똑같은 단어를 회문이라고 정의합니다.(ex. level)
- 2) 람다 함수를 활용하여 회문을 판별하는 프로그램을 작성하세요.
- 3) 'Q' 혹은 'q' 입력 시 반복을 종료합니다.

3 – 출력 예시

```
문자열 하나를 입력해주세요 : LEVEL
입력하신 문자열의 역순 : LEVEL
이 문자는 회문입니다.

문자열 하나를 입력해주세요 : HELLO
입력하신 문자열의 역순 : OLLEH
이 문자는 회문이 아닙니다.

문자열 하나를 입력해주세요 : COMPUTER
입력하신 문자열의 역순 : RETUPMOC
이 문자는 회문이 아닙니다.

문자열 하나를 입력해주세요 : ABCDCBA
입력하신 문자열의 역순 : ABCDCBA
이 문자는 회문입니다.
```



4. 홀수 숫자  $n$ 을 하나 입력받고,  $n \times n$  크기의 마방진을 출력하는 프로그래밍을 작성하세요.

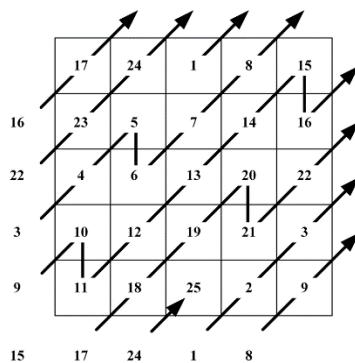
1) 마방진이란,  $n \times n$  행렬에서 가로, 세로, 대각선 방향의 숫자를 더하면 모두 같은 값이 나오는 배열입니다.

- 예시

4	9	2
3	5	7
8	1	6

2) 마방진을 만드는 원리는 1에서부터(보통 1은 첫 번째 줄 가운데에 두고 시작합니다.)

오른쪽 위 대각선 방향으로 숫자를 하나씩 늘려가는 방식을 사용합니다.



4 - 출력예시

```
홀수 숫자를 하나 입력해 주세요 : 3
8   1   6
3   5   7
4   9   2
계속하려면 아무 키나 누르십시오 . . .
```

```
홀수 숫자를 하나 입력해 주세요 : 5
17  24   1   8  15
23   5   7  14  16
 4   6  13  20  22
10  12  19  21   3
11  18  25   2   9
계속하려면 아무 키나 누르십시오 . . .
```

## 5. 다양한 Type을 사용하는 Queue Class를 구현하세요

```
변수(private) :
    T * p_list; // 정수형 변수들을 가지는 배열
    int size; // 현재 저장된 변수들의 개수
    int MAX_SIZE; // 최대 저장할 수 있는 p_list의 크기

함수(public);
    Queue(int _MAX_SIZE = 1000)
        // 생성자: p_list의 크기를 MAX_SIZE만큼 동적 할당.
    ~Queue()
        // 소멸자: p_list의 동적 할당을 해제
    int find_index(T _item)
        // p_list에서 _item과 동일한 값이 있는지 검색 후 발견시 index를 반환한다 만약
        발견하지 못하면 -1을 반환한다
    void Enqueue(T _item)
        // 입력 item을 p_list의 끝에 저장한다. 만약 _item과 동일한 값이 p_list에 존재할 경우
        p_list에 _입력 item을 추가하지 않는다. (힌트: find_index 함수를 사용해서 중복된 값이
        p_list에 있는지 조사후 없는 경우에 입력 item을 p_list에 추가). size가 MAX_SIZE보다 크면
        item을 추가하지 않는다. ("Error: out of memory" 출력)
    T Dequeue()
        // p_list에 있는 첫번째 item을 제거한다음 그 아이템을 return한다 (힌트: size 값을
        줄이면 p_list의 아이템을 제거한 것과 동일한 효과) size가 0일 때는 item을 제거하지 않는다.
        ("Error: No item exists in the list" 출력)
    void print() const
        // Queue 객체의 item들을 출력한다
    int get_size()
        // Queue 객체의 크기를 출력한다
    T get_item(int _index)
        // p_list의 해당 index에 있는 item 값을 리턴한다.
```

<시작코드-변경금지>

```
int main()
{
    Queue<int> int_queue;
    Queue<float> float_queue;
```

```

Queue<char> char_queue;

int_queue.Enqueue(1);
int_queue.Enqueue(2);
int_queue.Enqueue(2);
int_queue.Enqueue(5);

float_queue.Enqueue(4.3);
float_queue.Enqueue(2.5);
float_queue.Enqueue(3.7);
float_queue.Enqueue(3.7);

char_queue.Enqueue('b');
char_queue.Enqueue('b');
char_queue.Enqueue('c');
char_queue.Enqueue('a');

cout << "<Before Dequeue>" << endl;
int_queue.print();
float_queue.print();
char_queue.print();

int_queue.Dequeue();
float_queue.Dequeue();
float_queue.Dequeue();
char_queue.Dequeue();
char_queue.Dequeue();
char_queue.Dequeue();
char_queue.Dequeue();

cout << "<After Dequeue>" << endl;
int_queue.print();
float_queue.print();
char_queue.print();

return 0;
}

```

## 5 - 출력예시

```

<Before Dequeue>
Items in the list : 1, 2, 5,
Items in the list : 4.3, 2.5, 3.7,
Items in the list : b, c, a,
Error: No item exist in the list
<After Dequeue>
Items in the list : 2, 5,
Items in the list : 3.7,
Items in the list :

```

(optional)

※ **Optional** 문제는 성적 산출에 반영되지 않습니다.

## 기반 지식

템플릿 프로그래밍의 장점으로는 서로 다른 자료형이더라도 동일하게 처리를 해줄 수 있다는 점이 있다. 템플릿으로 함수/개체를 정의해주면 컴파일러가 알아서 이를 해당 타입에 맞는 함수/클래스로 변환해서 컴파일해주기 때문이다. (즉, 사용하는 타입 T의 개수가 많아질수록 컴파일할 양이 많아지고 컴파일 시간이 늘어난다. 그렇기 때문에 사용할 타입의 개수가 3개 이상이 아니라면 그냥 두 개의 클래스/함수를 만드는게 낫다)

템플릿은 기본적으로 남용해서는 안된다. 그래서 주로 사용되는 분야가 바로 컨테이너들이다. 대표적인 컨테이너들로는 벡터, 리스트, 스택, 큐 등이 있다.

## 문제

### 문제 개요

쿠팡(Khupang) 개발팀의 프로젝트 리드가 앞으로 C++ STL에서 기본 제공하는 컨테이너들 대신, 자신들만의 컨테이너들을 사용하겠다고 선언했다. 이에 당신에게 할당받은 업무는 바로 "벡터"와 "배열"이다. 벡터는 이미 IntVector를 만들었던 경험이 있으니 그대로 템플릿으로 옮기기만 하면 되지만, 배열은 직접 구현을 해줘야한다. 배열은 말 그대로 "배열"이니 벡터의 기능을 그대로 옮기되 "고정된 크기"의 배열이라는 근본은 바뀌어서는 안된다.

### 문제 설명

\*khupang 네임스페이스 설정

앞으로 쿠팡에서 사용할 모든 클래스/함수들은 khupang 네임스페이스를 사용한다

## \*TVector 구현

계산기 회사에서 구현했던 IntVector의 명세를 그대로 사용하되 메타프로그래밍이 가능하게 템플릿으로 변경해주어야한다.

당연하겠지만 메모리 누수가 나서는 안된다.

다음은 TVector에 추가할 멤버 함수들이다:

$A = \{1, 2, 3\}$ ,  $B = \{2, 3, 4\}$

1. 덧셈연산자 (두 개의 벡터를 중복 여부 무관하게 좌피연산자 - 우피연산자 순서대로 더해준다)
  - a.  $A + B == \{1, 2, 3, 2, 3, 4\}$
2. 뺄셈연산자 (좌피연산자 벡터의 원소들 중 우피연산자의 원소들을 전부 빼준다)
  - b.  $A - B == \{1\}$
3. 덧셈/뺄셈복합할당연산자
  - c.  $A += B \rightarrow A == \{1, 2, 3, 2, 3, 4\}$
  - d.  $A -= B \rightarrow A == \{1\}$
4. 관계연산자 (벡터의 첫번째 원소부터 마지막 원소까지 값을 비교함)
  - e.  $A < B \rightarrow A$ 의 1이  $B$ 의 2보다 작으므로 true
  - f.  $A < \{1, 2, 3, 4\} \rightarrow$  3번째 원소까지는 둘이 같지만 우피연산자가 벡터의 크기가 더 크므로 true
  - g.  $A <= \{1, 2, 3\} \rightarrow$  true
5. 동등연산자 (벡터의 모든 원소가 순서대로 같은지를 판별함)
  - h.  $A == B \rightarrow$  false

i. `A == {3, 2, 1}` ---> false

j. `A == {1, 2, 3}` ---> true

6. TArray를 매개변수로 받는 생성자, 할당연산자

7. Default Constructor를 유저가 사용 가능하게 처리하고, 해당 생성자 불러올 시 크기가 8인 벡터를 생성한다.

## TArray 구현

배열은 고정된 크기를 가지므로 동적 메모리 할당이 없어야한다.

기본적인 멤버함수들은 TVector의 것을 그대로 따라서 사용한다.

다만 배열의 특성을 반영하여 고쳐야한다.

예) 가득찬 배열에 PushBack을 하더라도 배열의 크기가 변경되어서는 안된다

다음은 TVector의 멤버함수 중 TArray에서 구현하지 않아도 되는 함수들 목록이다:

1. Reserve

2. Resize

TVector에서 TArray를 매개변수로 받는 생성자, 할당연산자의 경우 반대로 TVector를 매개변수로 받는 생성자를 작성하되, 현재 TArray의 크기를 벗어나는 원소들은 무시하도록 한다.

예)

```
TArray* pIntArray = new TArray<int, 3>(); // *pIntArray == { ??, ??, ?? }
pIntArray->PushBack(1); // *pIntArray == { 1, ??, ?? }
pIntArray->PushBack(2); // *pIntArray == { 1, 2, ?? }
pIntArray->PushBack(3); // *pIntArray == { 1, 2, 3 }
// pIntArray->PushBack(4); 컴파일 오류
TVector* pIntVector = new TVector<int>(*pIntArray); // *pIntVector == { 1, 2, 3 }
```

```
TArray* pIntArray2 = new TArray<int, 2>(*pIntVector); // *pIntArray2 == { 1,
```

## 문제 규칙

- typename을 쓰는 class를 쓰는 큰 차이는 없으나, 본 문제에서는 typename으로 통일하도록 한다
- [참고 링크](#)
- 입출력, string, cassert를 제외한 그 어떠한 STL 라이브러리도 불허한다.

## 예제 코드

```
#include <cassert>
#include <iostream>

#include "TArray.h"
#include "TVector.h"

int main()
{
    TVector* pIntVector1 = new TVector<int>(); //
    { ??, ??, ??, ??, ??, ??, ??, ?? }
    assert(pIntVector1->GetSize() == 0);
    assert(pIntVector1->GetCapacity() == 8);
    pIntVector1.PushBack(3); // {3}
    pIntVector1.PushBack(5); // {3, 5}

    TVector* pIntVector2 = new TVector<int>(pIntVector1); // {3, 5}

    *pIntVector2 += *pIntVector1; // {3, 5, 3, 5}
    assert(pIntVector2->GetSize() == 4);
    assert(pIntVector2->GetCapacity() == 8);

    TArray* pIntArray1 = new TArray<int, 4>(*pIntVector1); // {3, 5, ??, ??}
    *pIntArray1 += *pIntArray1; // {3, 5, 3, 5}

    pIntArray1->clear();
    assert(pIntArray1->GetSize() == 0);
    assert(pIntArray1->GetCapacity() == 4);

    delete pIntVector1;
```

```
delete pIntVector2;  
delete pIntArray1;
```

```
return 0;  
}
```