20203324 게임공학과 최종현
자료구조 2주차 리포트

연습문제 1

```cpp
#include <iostream>
#include <sstream>
#include <algorithm>

using namespace std;
template<typename T>
class dynamic_array
{
    T* data; // 동적 배열 데이터를 저장하는 포인터
    size_t n; // 배열의 크기 원소 개수


public:
    // 생성자 : 배열의 크기를 n으로 설정하고 메모리를 동적으로 할당한다.
    dynamic_array(int n)
    {
        this->n = n;  // 배열 크기 초기화
        data = new T[n]; // 크기에 맞게 메모리 할당
    }

    // 복사 생성자 : 깊은 복사를 수행하여 객체 복사 시 데이터의 독립성을 보장한다.
    dynamic_array(const dynamic_array<T>& other)
    {
        n = other.n; // 배열 크기 복사
        data = new T[n]; // 새 메모리 할당

        for (int i = 0; i < n; i++)
        {
            data[i] = other.data[i]; // 데이터를 개별적으로 복사

        }
    }
```

<T> IntelliSense에 대한 샘플 템플릿 인수를 제공합니다.

```
// 배열 인덱스를 통한 요소 접근을 지원
T& operator[](int index)
{
    return data[index]; // 인덱스에 해당하는 원소 반환
}
// 배열 인덱스를 통한 요소 접근을 지원
const T& operator[](int index) const
{
    return data[index]; // const 객체에 대해서도 읽기 가능
}
// 안전한 접근을 지원하며, 잘못된 인덱스 접근 시 예외를 발생시킴
T& at(int index)
{
    if (index >= n) // 유효한 범위 초과 여부 확인
    {
        throw "Index out of range"; // 예외 발생
    }
    return data[index]; // 범위 내 접근
}


// 배열 크기 반환
size_t size() const
{
    return n; // 배열 원소 개수 반환
}
// 소멸자 : 동적으로 할당된 메모리를 해제하여 메모리 누수를 방지한다.
~dynamic_array()
{
    delete[] data; // 동적 메모리 해제
}
```

```cpp
// begin() 및 end() 함수: 반복자처럼 동작하는 포인터를 반환한다.
T* begin() { return data; } // 시작 포인터 반환
const T* begin() const { return data; } // const 시작 포인터 반환
T* end() { return data + n; } // 끝 포인터 반환
const T* end() const { return data + n; } // const 끝 포인터 반환

// operator+ 함수: 두 dynamic_array 객체를 결합하여 새로운 객체를 생성한다.
// 배열 합치기 기능을 제공한다.
friend dynamic_array<T> operator+(const dynamic_array<T>& arr1, dynamic_array<T>& arr2)
{
    dynamic_array<T> result(arr1.size() + arr2.size()); // 두 배열 크기의 합만큼 새로운 배열 생성
    copy(arr1.begin(), arr1.end(), result.begin()); // 첫 번째 배열의 내용을 복사
    copy(arr2.begin(), arr2.end(), result.begin()+arr1.size()); // 두 번째 배열의 내용을 복사

    return result; // 결합된 배열 반환

}
// to_string() 함수 : 배열의 내용을 문자열로 변환한다.
// sep 파라미터를 사용하여 각 원소 사이의 구분자를 지정할 수 있다.
string to_string(const string& sep = " , ")
{
    if (n == 0)
        return ""; // 배열이 비어 있으면 빈 문자열 반환

    ostringstream os; // 문자열 스트림 객체 생성
    os << data[0]; // 첫 번째 원소 추가

    for (int i = 1; i < n; i++) // 두 번째 원소부터 구분자를 추가하며 문자열 생성
        os << sep << data[i];
    return os.str(); // 생성된 문자열 반환
}
```

```cpp
// student 구조체 : 학생의 이름과 학년을 저장하기 위한 데이터 구조
struct student
{
    string name; // 학생 이름
    int standard; // 학생 학년
};

// operator<< 연산자 오버로딩 : student 객체를 출력할 때 문자열 형식을 지정한다.
ostream& operator<<(ostream& os, const student& s)
{
    return (os << " [ " << s.name << " ," << s.standard << "]");
}
```

```cpp
// 깊은 복사 : 기존 학급 class1을 복사하여 새 학급 calss2 생성
auto class2 = class1;
cout << "1반을 복사하여 2반 생성 : " << class2.to_string() << endl;

// 두 학급을 합쳐서 새로운 큰 학급을 생성
auto class3 = class1 + class2;
cout << "1반과 2반을 합쳐 3반 생성 : " << class3.to_string() << endl;
return 0;
```

```cpp
int main()
{
    int nStudents;
    cout << "1반 학생 수를 입력하세요 : ";
    cin >> nStudents;

    // 첫 번째 학급을 위한 dynamic_array 객체 생성
    dynamic_array<student> class1(nStudents);
    for (int i = 0; i < nStudents; i++)
    {
        string name;
        int standard;
        cout << i + 1 << "번째 학생 이름과 나이를 입력하세요 : ";
        cin >> name >> standard;
        class1[i] = student{ name,standard };
    }

    // 배열 크기보다 큰 인덱스의 학생에 접근
    try
    {
        // 아래 주석을 해제하면 프로그램이 비정상 종료
        // class1[nStudents] = student{ "John",8 }; // 예상할 수 없는 동작
        // 유효하지 않은 인덱스 접근

        class1.at(nStudents) = student{ "John",8 }; // 예외 발생

    }
    catch (...)
    {
        cout << "예외 발생!" << endl;
    }
```

```
1반 학생 수를 입력하세요 : 5
1번째 학생 이름과 나이를 입력하세요 : 최종현
25
2번째 학생 이름과 나이를 입력하세요 : 김명건
25
3번째 학생 이름과 나이를 입력하세요 : 이진재
25
4번째 학생 이름과 나이를 입력하세요 : 배성원
25
5번째 학생 이름과 나이를 입력하세요 : 박철진
25
예외 발생!
1반을 복사하여 2반 생성 : [ 최종현 ,25] , [ 김명건 ,25] , [ 이진재 ,25] , [ 배성원 ,25] , [ 박철진 ,25]
1반과 2반을 합쳐 3반 생성 : [ 최종현 ,25] , [ 김명건 ,25] , [ 이진재 ,25] , [ 배성원 ,25] , [ 박철진 ,25] , [ 최종현 ,25] , [ 김명건 ,25] , [ 이진재 ,25] , [ 배성원 ,25] , [ 박철진 ,25]
```

연습문제 2

```cpp
#include <iostream>
#include <array> // array 사용
#include <type_traits> // 타입 연산 도구(common_type 등)를 사용하기 위해


using namespace std;

// 가변 인자를 받아 std::array를 생성하는 함수 템플릿
// Args : 가변 인자 템플릿을 사용해 다양한 타입의 매개변수를 받을 수 있음


template<typename ... Args>
auto build_array(Args&&...args) -> array<typename common_type<Args...>::type, sizeof...(args)>
{
    // common_type을 사용하여 인자들 간의 공통 타입을 계산
    // 공통 타입 : 모든 인자가 변환될 수 있는 최소 공통 타입
    using commonType = typename common_type<Args...>::type;

    // std::forward를 사용하여 전달받은 인자를 그대로 유지하며 std::array를 초기화
    return { forward<commonType>((Args && )args)... };
}
```

```cpp
int main()
{
    // build_array를 호출하여 다양한 타입의 값을 인자로 전달
    // 1 (int), 0u(unsigned int), 'a'(char) 3.2f(float) , false(bool)
    // 가 공통 타입으로 변환됨
    // 공통 타입은 모든 인자가 변환 가능한 최소한의 공통 타입으로,
    // 여기서는 double(float,int,char 등이 모두 변환 가능)로 결정됨

    auto data = build_array(1, 0u, 'a', 3.2f, false);
    // 생성된 std::array의 각 원소를 출력
    for (auto i : data)
        cout << i << " ";
    cout << endl;

    // 1 -> int에서 double로 변환
    // 0u -> unsigned int에서 double로 변환
    // 'a' -> char에서 double로 변환 ('a'의 ASCII 값 97)
    // 3.2f -> float에서 double로 변환
    // false -> bool에서 double로 변환 (0으로 처리)
}
```