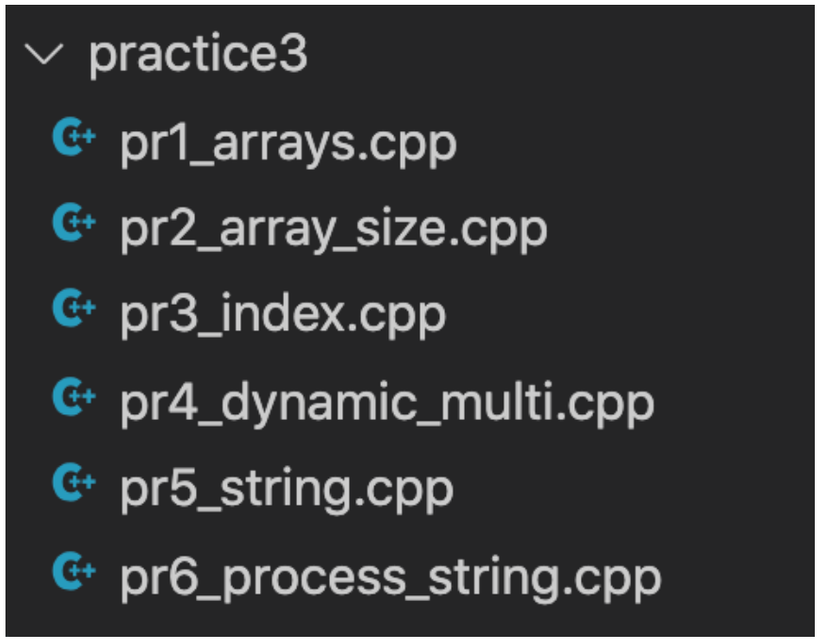
C++ 실습 3

실습 내용

- 이번 주 실습은 이론시간에 배웠던 배열과 문자열에 관련된 내용을 직접 실행하며 확인하는 것입니다.

- 총 6개의 cpp파일이 있고 하나씩 실행하며 확인하면 됩니다.



새 실습파일을 받는 방법

- 이미 지난주에 Git을 설치하고 GitHub의 강의 저장소를 가져왔다는 가정하에 설명하겠습니다.

- 기본적으로 새 실습파일을 받기 위해서는 새롭게 업데이트된 강의 저장소의 내용을 내 PC로 가져오는 명령을 실행하면 됩니다.

- Git에서 이 명령은 'git pull'로, 원격 저장소의 내용을 내가 복제한 저장소로 당겨(pull)오라는 의미입니다.

새 내용 가져오기

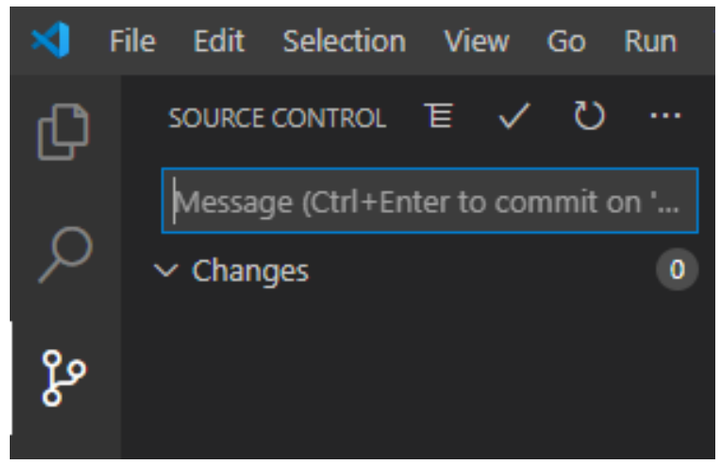
- Git Pull을 실행하는 법은 간단합니다.

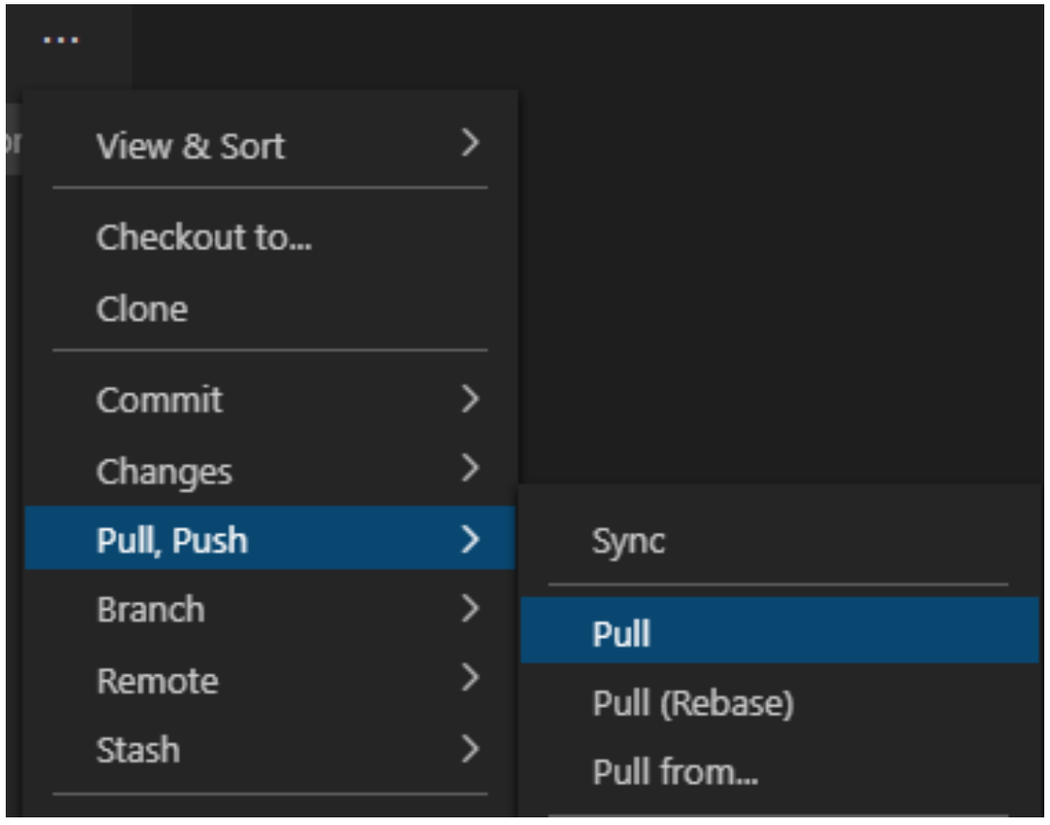
- 우선 왼쪽에서 세번째 탭을 선택합니다.

- SOURCE CONTROL이라는 탭 이름이 보입니다.

- 이름 옆에서 ...으로 된 부분을 누르면, 오른쪽과 같은 메뉴가 나타납니다.

- Pull을 선택하면 자동으로 업데이트 된 내용을 가져옵니다.





가져온 내용 확인

- 다시 왼쪽에서 가장 위의 버튼을 눌러 EXPLORER로 돌아갑니다.

- 오른쪽처럼 practice3 폴더가 추가된 것을 확인할 수 있습니다.

첫번째 실습

- pr1_arrays.cpp 파일을 실행합니다.
- 매우 간단한 실습으로 강의에서 설명한 다양한 배열 생성 및 초기화 방법을 보여주고 있습니다.
- 실행하여 배열에 저장된 값들이 어떻게 표시되는지 확인하세요.

```cpp
//기본적인 배열 생성 및 초기화
int arr[3]; //초기화가 없는 경우.
cout << "arr[0] = " << arr[0] << endl;
cout << "arr[1] = " << arr[1] << endl;
cout << "arr[2] = " << arr[2] << endl;

int new_arr[3] = {};   //모두 0으로 초기화.
cout << "new_arr[0] = " << new_arr[0] << endl;
cout << "new_arr[1] = " << new_arr[1] << endl;
cout << "new_arr[2] = " << new_arr[2] << endl;

double new_double_arr[3] = { 1.2e3, 0.023e2, 0.13 };    //값을 지정.
cout << "new_double_arr[0] = " << new_double_arr[0] << endl;
cout << "new_double_arr[1] = " << new_double_arr[1] << endl;
cout << "new_double_arr[2] = " << new_double_arr[2] << endl;

//초기화시 배열의 크기보다 초기값의 원소수가 작은 경우.
int a[3] = { 1, 2 };
cout << "a[0] = " << a[0] << endl;
cout << "a[1] = " << a[1] << endl;
cout << "a[2] = " << a[2] << endl;
```

두번째 실습

```
//실행시간에 배열 크기가 정해지는 경우
int len;
cout << "Input array size: ";
cin >> len; //len에 배열의 크기를 입력 받음.
int array[len*2]; //길이가 컴파일 시간에 정해지지 않으므로, { }로 초기화하면 에러.

//또는 new 키워드 사용.
int *a = new int[len];
for(int i=0; i<len; i++)
    cout << i << " = " << a[i] << endl;
delete[] a;
```

- 두번째 실습은 배열의 크기와 관련된 실습입니다.
- 첫 부분은 배열의 크기를 입력받아 생성합니다.
- 크기를 입력하려면 아래의 터미널 부분을 클릭하여 값을 넣으면 됩니다.

```
Input array size:
```

두번째 실습

```
//실행시간에 배열 크기가 정해지는 경우
int len;
cout << "Input array size: ";
cin >> len; //len에 배열의 크기를 입력 받음.
int array[len*2]; //길이가 컴파일 시간에 정해지지 않으므로, { }로 초기화하면 에러.

//또는 new 키워드 사용.
int *a = new int[len];
for(int i=0; i<len; i++)
    cout << i << " = " << a[i] << endl;
delete[] a;
```

- 첫번째 표시된 부분의 경우, 배열 크기가 실행시간에 정해지므로 {} 를 사용하여 초기화할 수 없습니다.

- 실제 = { };를 추가하여 빌드가 정상적으로 되는지 확인해 보세요.

두번째 실습

```
//실행시간에 배열 크기가 정해지는 경우
int len;
cout << "Input array size: ";
cin >> len; //len에 배열의 크기를 입력 받음.
int array[len*2]; //길이가 컴파일 시간에 정해지지 않으므로, { }로 초기화하면 에러.

//또는 new 키워드 사용.
int *a = new int[len];
for(int i=0; i<len; i++)
    cout << i << " = " << a[i] << endl;
delete[] a;
```

- 아래 부분은 new 키워드를 사용하여 동적 할당 배열을 만드는 부분입니다.
- 마지막 delete[]로 메모리를 해제해 주는 부분을 잊어서는 안됩니다.
- 중간의 for문은 아직 배우지 않았지만 i값을 0~len까지 1씩 증가시키며 반복하여 출력하기 위한 부분으로 이해하시면 됩니다.

두번째 실습

- 마찬가지로 윗부분의 for문 두 개는 모두 array배열의 원소를 출력해주는 부분입니다.

- before에 해당하는 부분에서는 배열이 아직 초기화 되지 않았으므로, 이상한 값이 출력됩니다.

- after에 해당하는 부분에서는 모두 0으로 초기화되어 값이 출력됩니다.

- 마지막의 배열크기는 전체 arr배열이 차지한 byte수를 int형 원소가 차지하는 byte로 나누어 원소의 개수를 구합니다.

```
//array를 0으로 초기화.
for(int i=0; i<len; i++) {
    printf("before: array[%d] = %d\n", i, array[i]);
    array[i] = 0;
}

for(int i=0; i<len; i++) {
    printf("after: array[%d] = %d\n", i, array[i]);
    array[i] = 0;
}

//배열의 크기를 입력하지 않는 경우.
int arr[] = { 1, 3, 2 };
len = sizeof(arr) / sizeof(int);   //배열크기 구하기
cout << "[" << arr[0] << ", " << arr[1] << ", " << arr[2
cout << "size of arr: " << len << endl;
```

세번째 실습

```cpp
//인덱스 범위가 벗어나도 컴파일 됨.
int a[3] = {};
int b[5] = {};
cout << "a[100] = " << a[100] << endl;

//배열 인덱스를 벗어난 범위에 값을 변경하는 경우.
int diff = b - a; //배열 a와 b의 주소차이를 저장. b[0]에 해당하는 a의 인덱스가 됨.
a[diff] = 123; //값을 대입.
cout << "a[diff] = " << a[diff] << endl; //메모리의 값이 바뀌게 됨.

//실제로 변경된 값이 b[0]에 영향.
for(int i=0; i<5; i++)
    cout << b[i] << ", ";
```

- 세번째 실습은 배열 인덱스가 범위를 벗어날 때 생기는 문제를 보여줍니다.

- a, b 두 개의 배열을 선언하여 메모리가 할당되고 나면, diff변수에 두 배열의 주소차이를 계산하여 넣습니다.

- 이 diff를 인덱스로 하여 a배열의 값을 변경하였는데, 실제로는 b라는 배열에 할당된 메모리를 접근하게 되기 때문에 b배열의 원소가 변경되는 문제가 생깁니다.

네번째 실습

- 네번째 실습은 우선 고정 배열과 동적 할당 배열의 성능을 비교해봅니다.

- 똑같이 크기가 1,000인 배열을 고정크기와 동적 할당으로 생성하였습니다.

- 네모로 표시된 부분은 반복적으로 이 배열의 값을 읽고 1을 더해 다시 배열에 쓰는 작업을 합니다.

- 고정 크기 배열과 동적 할당 배열에서 동일 작업을 수행할 때 시간 차이를 출력합니다.

```
//고정 배열과 동적 할당 배열 성능비교
int size = 1000;
int fixed_arr[1000] = {};
int* dynamic_arr = new int[size];  //int*에 주의

//write and read 1,000,000 times.
int iter = 1000000;
clock_t s_time = clock();
for(int n=0; n<iter; n++) {
    for(int i=0; i<size; i++) {
        fixed_arr[i] = fixed_arr[i] + 1;
    }
}
clock_t e_time = clock();
cout << "Fixed 1,000,000 read and write: " << (do

s_time = clock();
for(int n=0; n<iter; n++) {
    for(int i=0; i<size; i++) {
        dynamic_arr[i] = dynamic_arr[i] + 1;
    }
}
e_time = clock();
cout << "Dynamic 1,000,000 read and write: " << (
```

네번째 실습

- 네번째 실습은 우선 고정 배열과 동적 할당 배열의 성능을 비교해봅니다.

- 똑같이 크기가 1,000인 배열을 고정크기와 동적 할당으로 생성하였습니다.

- 네모로 표시된 부분은 반복적으로 이 배열의 값을 읽고 1을 더해 다시 배열에 쓰는 작업을 합니다.

- 고정 크기 배열과 동적 할당 배열에서 동일 작업을 수행할 때 시간 차이를 출력합니다.

```
//고정 배열과 동적 할당 배열 성능비교
int size = 1000;
int fixed_arr[1000] = {};
int* dynamic_arr = new int[size];  //int*에 주의

//write and read 1,000,000 times.
int iter = 1000000;
clock_t s_time = clock();
for(int n=0; n<iter; n++) {
    for(int i=0; i<size; i++) {
        fixed_arr[i] = fixed_arr[i] + 1;
    }
}
clock_t e_time = clock();
cout << "Fixed 1,000,000 read and write: " << (do

s_time = clock();
for(int n=0; n<iter; n++) {
    for(int i=0; i<size; i++) {
        dynamic_arr[i] = dynamic_arr[i] + 1;
    }
}
e_time = clock();
cout << "Dynamic 1,000,000 read and write: " << (
```

네번째 실습

- 제 컴퓨터에서 1백만번의 읽기/쓰기 작업을 반복수행한 결과입니다.

- 고정 배열을 사용하면 약 1초, 동적 할당 배열을 사용하면 약 1.3초가 걸리게 됩니다.

- 시간상으로는 0.3초이지만, 비율상으로는 약 30%만큼이 느려졌습니다.

- 배열의 size (빨간 네모 두 곳)을 증가시키거나, 반복횟수(주황색 네모)를 증가시켜 이 경우에도 성능차이가 유사한지 확인해 보세요.

```
Fixed 1,000,000 read and write: 1.02277s
Dynamic 1,000,000 read and write: 1.33178s
```

```
int size = 1000;
int fixed_arr[1000] = {};
int* dynamic_arr = new int[size];
```

```
//write and read 1,000,000 times.
int iter = 1000000;
```

네번째 실습

- 네번째 실습 후반부는 다차원 배열을 사용하는 실습입니다.

- 어떤 형태로 다차원 배열을 정의하고, 초기화할 수 있는지 다양한 경우를 확인합니다.

- 접근을 위해 어떤 식으로 인덱스를 사용하는지 확인해보고, 실제 출력 결과를 예측해보세요.

```cpp
//다차원 배열
int array2d[3][2] = {
    {1, 2 },
    {2, 3 },
    {3, 4 }
};

//2차원 배열에서 i, j가 각각 행,열번호로 작용
for(int i=0; i<3; i++) {
    for(int j=0; j<2; j++){
        cout << array2d[i][j] << ",";
    }
    cout << endl;
}

int arr2d_0[2][2] = {};   //모두 0으로 초기화
int arr2d_1[2][2] = { {1, }, {2, }}; // {{1, 0}, {2, 0}}
cout << "arr2d_0" << endl;
for(int i=0; i<2; i++)
    cout << "{" << arr2d_0[i][0] << ", " << arr2d_0[i][1] << "}" << endl;
cout << "arr2d_1" << endl;
for(int i=0; i<2; i++)
    cout << "{" << arr2d_1[i][0] << ", " << arr2d_1[i][1] << "}" << endl;
```

다섯번째 실습

- 다섯번째 실습은 문자열을 char[] 와 string 클래스 두 가지로 선언하는 예시가 들어있습니다.

- 간단한 부분이니 실행하여 결과를 확인해 보면 됩니다.

```
char str0[10] = {'p', 'r', 'o', 'g', 'r', 'a', 'm', '\0'};
char str1[8] = "program";
char str2[] = "program";

cout << "str0 = " << str0 << endl;
cout << "str1 = " << str1 << endl;
cout << "str2 = " << str2 << endl;

//문자열 자르기
str0[3] = '\0';
cout << "str0 = " << str0 << endl;

//string클래스
string string1 = "string ";
string string2 = "class";
string str = string1 + string2;
cout << "string1 + string2 = " << str << endl;
```

여섯번째 실습

```cpp
//문자열 대입
string1 = string2;
cout << "string1 = " << string1 << endl;
cout << "str = " << str << endl;

//문자열 길이
int len1 = str.size();
int len2 = strlen(str0);
cout << "length of \"" << str << "\" = " << len1 << endl;
cout << "length of \"" << str0 << "\" = " << len2 << endl;

//특정위치 문자 얻기 at(index)
cout << "3rd character of str: " << str.at(2) << endl; //인덱스에 주의.

//부분 문자열 얻기 substr(start index, length)
cout << "First 6 characters: \"" << str.substr(0, 6) << "\"" << endl;
cout << "From 8th character to the end: \"" << str.substr(7) << "\"" << endl;   //인덱스 차이에 주의.
cout << "From 4th to 6th: \"" << str.substr(4, 3) << "\"" << endl;    //인덱스+길이에 주의.
```

- 여섯번째 실습은 C++의 다양한 문자열 처리 기능을 실습합니다.

여섯번째 실습

```
//문자열 검색 str.find(query, start_index)
str = "programming introduction";
int index = str.find("ro");
cout << "index of \"ro\" = " << index << endl;
index = str.find("ro", index+1);  //이미 찾은 곳 이후부터 다시 검색
cout << "index of 2nd \"ro\" = " << index << endl;


//문자열 바꾸기 str.replace(start_index, length, new_string)
str = "I like dog";
string q = "dog";
cout << "original str = " << str << endl;
str.replace(str.find(q), q.size(), "dragon!");   //dog를 찾아 dragon!으로 변경.
cout << "replaced str = " << str << endl;
```

- 초반부는 강의시간에 다룬 여러 문자열 처리 방법을 복습하게 되어 있으니, 결과를 예측하고 실제 실행결과와 비교해 보세요.

여섯번째 실습

```
//문자열 입력 받기1
char your_char[] = {};
string your_input;
cout << "Input something with space(e.g. string class): ";
cin >> your_char;
cout << "My input string is " << your_char << endl;      //띄어쓰기 전까지만 인식.

cout << "Input something with space again: ";
cin >> your_input;
cout << "My input string is " << your_input << endl;     //이전에 입력받은 부분이 그대로 출력됨.
```

- 여섯번째 실습 후반부는 문자열 사용자 입력을 받을 때 발생할 수 있는 문제를 보여줍니다.

- 입력 문자열로 띄어쓰기가 포함된 문자열을 입력하고, 어떻게 동작하는지 확인해 봅니다.

- 입력을 두 번 받도록 되어있는데, 두 번 모두 제대로 입력이 되는지 확인합시다.

여섯번째 실습

```
cin.ignore(32767, '\n');
```

```
//문자열 입력 받기2 getline()
cout << "Input something with space(e.g. string class): ";
cin.getline(your_char, 20);
cout << "My input string is " << your_char << endl;

cout << "Input something with space(e.g. string class): ";
getline(cin, your_input);
cout << "My input string is " << your_input << endl;
```

- 문자열 입력 받기2 부분은 이런 문제를 해결하여 제대로 처리하기 위한 방법을 보여줍니다.

- 위의 cin.ignore()부분은 이전 슬라이드에서 입력했던 내용들을 무시하기 위해 추가된 부분입니다.

- 띄어쓰기가 있는 문자열을 입력하고 제대로 저장이 되는지 확인해 봅니다.

실습 제출물

```
Input something with space(e.g. string class): My Input!
My input string is My
Input something with space again: My input string is Input!
Input something with space(e.g. string class): this is test input
My input string is this is test input
Input something with space(e.g. string class): Another test input!
My input string is Another test input!
```

- 출석인정을 위해서, 마지막 여섯번째 실습 파일의 코드를 실행합니다.

- 마지막 부분의 사용자 입력 문자열을 저장하는 예제 부분에서, 자신이 입력한 문자열이 어떻게 출력되는지 나오는 부분을 캡쳐하여 제출합니다.

실습 정리

- 총 6개의 파일로 실습을 진행하였습니다.
- 각각의 실습에서 배열을 초기화하고, 어떻게 문자열을 처리하는지 확인해 보았습니다.
- 배열과 문자열 처리에서는 인덱스에 꼭 주의해야 한다는 점을 기억하세요.
- 특히 마지막 여섯번째 실습의 문자열 처리는 앞으로 다양하게 사용될 확률이 높으니 눈여겨 보도록 합니다.