

# C++ 실습 6

# 실습 내용

- 이번 주 실습은 포인터와 참조변수에 관련된 다양한 코드를 확인해 보는 실습입니다.
- 총 5개의 cpp파일이 있고 하나씩 실행하며 확인하면 됩니다.
- Git Pull 명령을 이용해 새로운 파일들을 내려받습니다.
- 이번주도 지난주와 마찬가지로 C++11 표준을 사용하는 코드들이 있습니다.

```
▼ practice6
  C++ pr1_pointer_basics.cpp
  C++ pr2_pointer_arithmetic.cpp
  C++ pr3_pointer_arr_func.cpp
  C++ pr4_dynamic_allocation.cpp
  C++ pr5_reference.cpp
```

# 첫번째 실습

- pr1\_pointer\_basics.cpp 파일을 실행합니다.
- 포인터 변수가 가지고 있는 메모리 값과, 역참조를 통해 얻을 수 있는 값 등을 확인해보는 예제입니다.
- 실행하여 출력되는 값을 비교해 보면 됩니다.

```
int s = 10;  
int t = 3;  
short u = 1;
```

```
int* ptr = &s;  
int* ptr2 = &t;  
short* ptr3 = &u;
```

```
cout << "s = " << s << endl;  
cout << "&s = " << &s << endl; //변  
cout << "ptr = " << ptr << endl;  
cout << "*ptr = " << *ptr << endl;
```

```
cout << "ptr2 = " << ptr2 << endl;  
cout << "ptr3 = " << ptr3 << endl;
```

//포인터가 가리키는 메모리에 있는 값을 변경.

```
*ptr = 5;  
cout << "*ptr = " << *ptr << endl;  
cout << "s = " << s << endl; //변
```

# 두번째 실습

- pr2\_pointer\_arithmetic.c  
p 파일은 포인터 변수에 값을 더하거나 뺀을 경우, 주소값이 어떻게 변화하는지 확인하는 예제입니다.
- 첫번째 부분에서는 우선 포인터 변수들을 선언하고 이를 이용해 값을 계산하는 예제가 있습니다.
- y의 초기화 부분에서 \*연산자를 지워 ptr, ptr2 자체의 값으로 계산하면 어떻게 되는지 확인해 보세요.

```
int a = 10;
int b = 2;
int* ptr = &a;
int* ptr2 = &b;

int x = a / b;
int y = *ptr / *ptr2;    //*연산자를 지우면 어떻게 될까요?

//a, b변수로 계산한 x의 값과 포인터를 이용해 계산한 y값의 비교.
cout << "x = " << x << endl;
cout << "y = " << y << endl;

//두 포인터 변수의 주소값 확인
cout << "ptr = " << ptr << endl;
cout << "ptr2 = " << ptr2 << endl;
```

# 두번째 실습

- 이후에는 포인터 주소값에 값을 더하거나 빼면 어떻게 주소가 변화하는지 확인해 봅니다.
- 실제로 16진수 주소값이 출력되는데, 이 값들의 차이가 몇인지 확인해 보세요.
  - 데이터형에 따라 값이 달라 질 수 있습니다.
  - 포인터를 int형이 아닌 다른 데이터형으로 바꾸어 차이가 무엇인지 확인하는 것도 좋습니다.

```
//포인터 주소값에 1을 더하거나 빼는 경우.
```

```
cout << "ptr2+1 = " << ptr2+1 << endl;  
cout << "ptr-1 = " << ptr-1 << endl;
```

```
//포인터 주소값에 변수를 더하거나 뺌.
```

```
int k = 1;  
cout << "ptr2+k = " << ptr2+k << endl;  
cout << "ptr-k = " << ptr-k << endl;
```

# 세번째 실습

- 세번째 실습은 포인터와 배열, 매개 변수에 관련된 실습입니다.
- 첫 부분에서는 포인터 변수도 배열 처럼 사용할 수 있고,
- 배열 또한 포인터처럼 사용할 수 있다는 것을 확인해 봅니다.

```
int a[3] = {1, 2, 3};  
int* x = a;
```

//index를 이용한 참조 가능.

```
cout << "a[1] = " << a[1] << endl;  
cout << "x[1] = " << x[1] << endl;
```

//역참조 연산자\*를 이용한 참조 가능.

```
cout << "*(a+1) = " << *(a+1) << endl;  
cout << "*(x+1) = " << *(x+1) << endl;
```

//index와 복합 사용.

```
cout << "(a+1)[1] = " << (a+1)[1] << endl;  
cout << "(x+1)[1] = " << (x+1)[1] << endl;
```

# 세번째 실습

- 다음 부분에서는 2차원 배열과 포인터의 차이에 대해서 확인해 보고,
- 포인터를 매개변수로 사용하는 방법에 대해 확인해 봅니다.
- func()와 func2()를 호출했을 때 예상되는 동작을 잘 생각해보고, 실행 결과와 비교해 보세요.

```
void func(int* x) {  
    *x = 5;  
}  
  
void func2(int (*x)[2], int size) {  
    x[1][2] = 2;  
}
```

//2차원 배열

```
int b[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

//int\*\* y = b; //이 부분 주석을 해제하면 b를 대입하는 부분에서 에러가 발생.

//func(1); //매개변수가 포인터이므로 정수값을 넣으면 에러.

```
int y = 1;
```

```
func(&y); //포인터형을 인자로 하기위해 &연산자 사용.
```

```
cout << "y = " << y << endl;
```

//2차원 배열 인자.

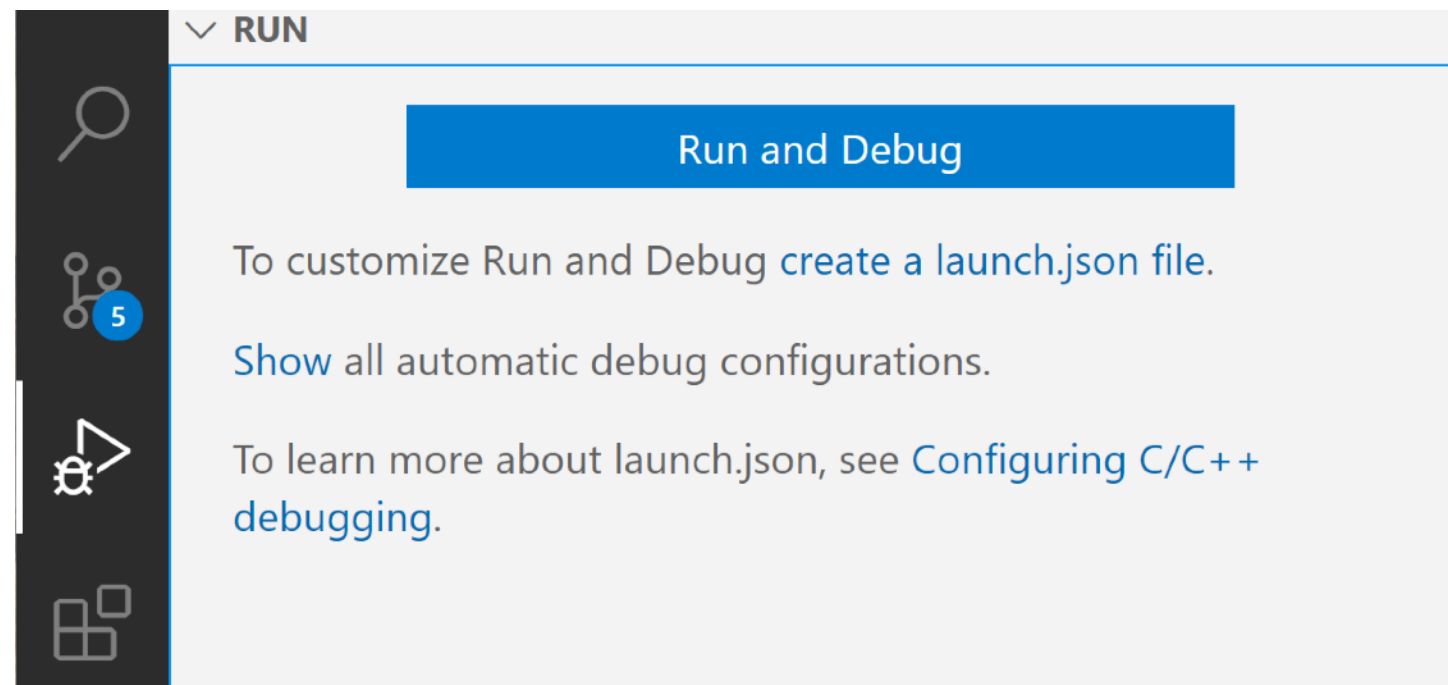
```
int z[3][2] = {};
```

```
func2(z, 3);
```

```
cout << "z[1][2] = " << z[1][2] << endl;
```

# 네번째 실습

- 네번째 실습을 위해 VSCode의 디버깅(Debugging) 기능을 사용할 것입니다.
- 이를 위해 추가적인 설정이 필요합니다.
- 먼저 실습 파일을 열고, **VSCode의 왼쪽 탭에서 4번째 디버그 탭을 선택하세요.**
- 설정을 위해 create a launch.json file이라고 된 부분을 클릭합니다.

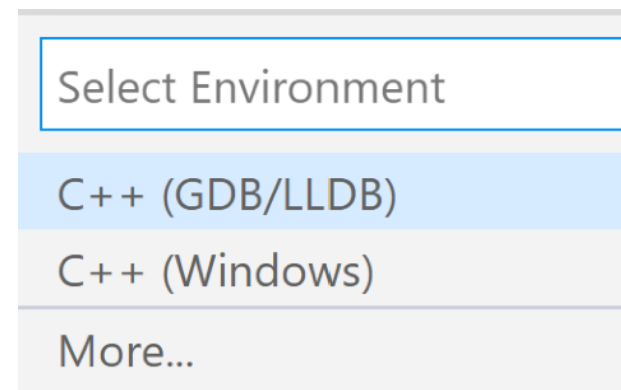




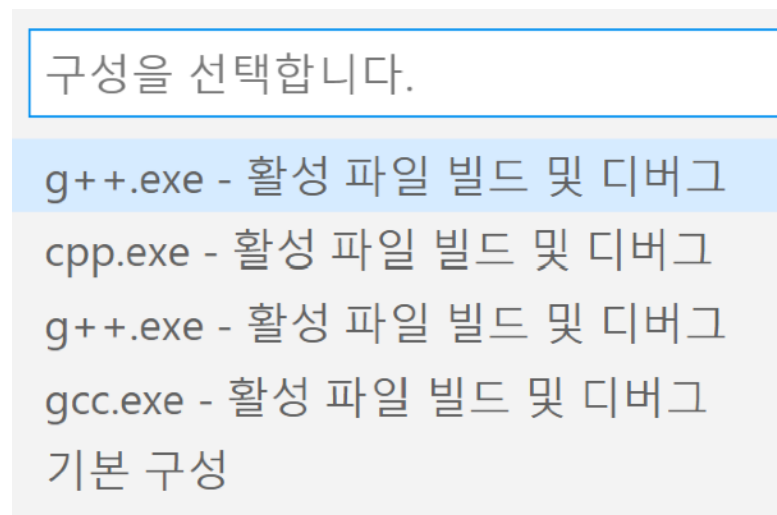
# 네번째 실습

- 1번과 같은 화면이 나오면 첫번째 GDB/LLDB를 선택합니다.
- 다시 2번과 같은 화면이 나오면 첫번째 “g++.exe ...” 를 선택합니다.
- 자동으로 launch.json 파일이 생성되고 디버그 창이 열리게 됩니다.

1

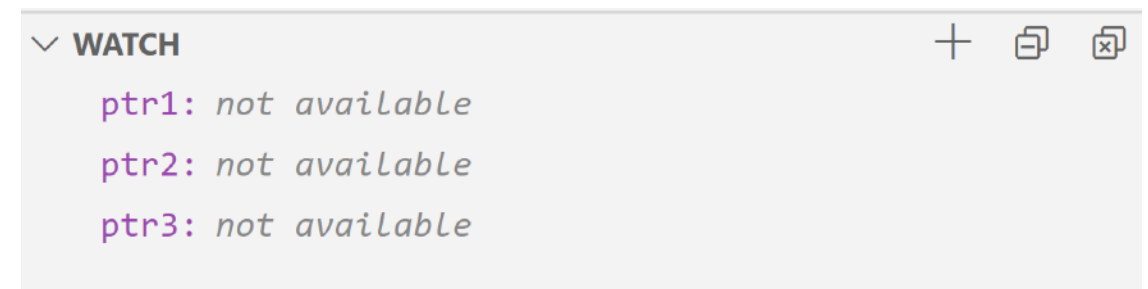
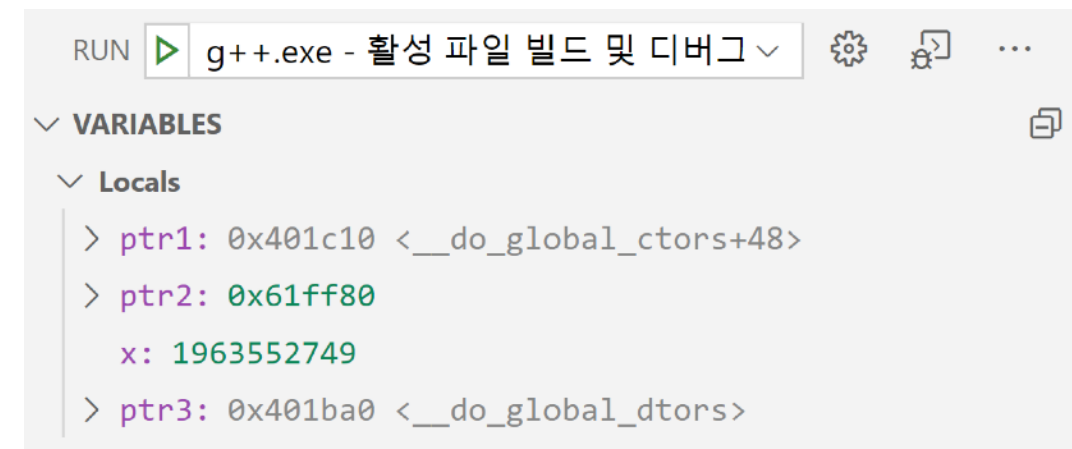


2



# 네번째 실습

- 창이 열리면 오른쪽과 같은 부분들을 화면에서 볼 수 있습니다.
- 첫번째 부분은 디버그 모드로 파일을 실행하고, 파일에 포함된 변수들을 보여주는 부분입니다.
- 실제 네번째 실습 코드에 변수들이 표시됩니다.
- 아직 제대로 실행되기 전이므로 값은 이상한 값이 들어가 있습니다.
- 두번째는 감시할 표현식을 입력하는 부분입니다.
  - +버튼을 누르고 ptr1~ptr3을 입력하세요.
  - \*ptr1~\*ptr3를 추가로 넣어도 됩니다.



# 네번째 실습

- 코드가 있는 창에서는 윗부분에 오른 쪽과 같이 컨트롤러가 표시됩니다.
- 왼쪽의 줄 번호 옆에 마우스 커서를 가져가면 클릭하여 빨간색 점을 표시 할 수 있습니다.
- 이 점이 있는 곳이 Break Point로, 디버그 모드로 실행을 했을 경우 실행이 중단되는 지점입니다.
- 실행버튼을 누르게 되면 6번째 줄에서 실행이 중단되는 것을 확인할 수 있습니다.

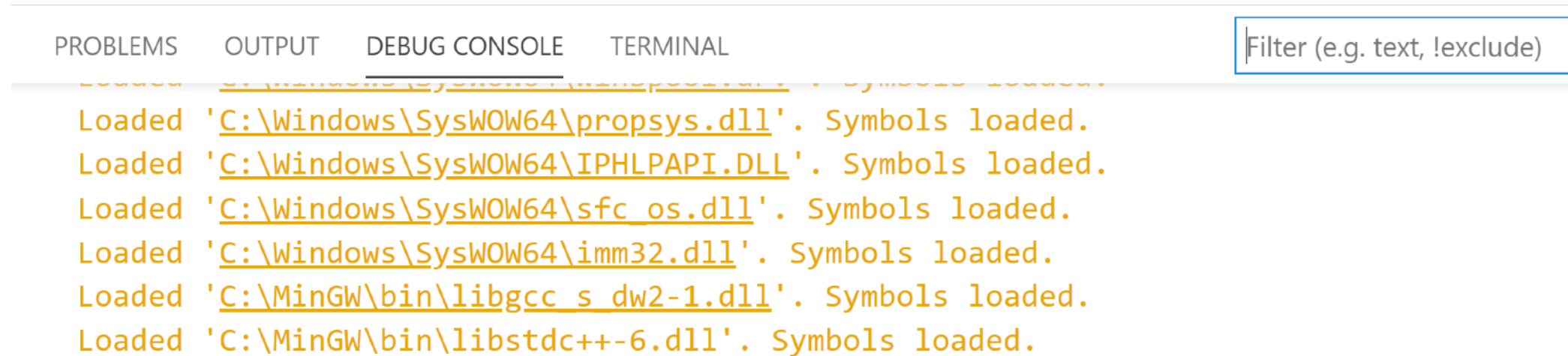
한단계만 실행      처음부터 다시

실행      중단

```
pr4_dynamic_alloc...  
practices > practice6 > pr4_dynamic_allocation.cpp > main()  
1  #include<iostream>  
2  using namespace std;  
3  
4  int main() {  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15
```

```
int* ptr1 = new int;  
int* ptr2 = new int;  
int x = 5;  
int* ptr3 = &x;  
  
cout << "*ptr2 = " <<  
cout << "ptr1 = " <<  
cout << "ptr2 = " <<  
cout << "ptr3 = " <<
```

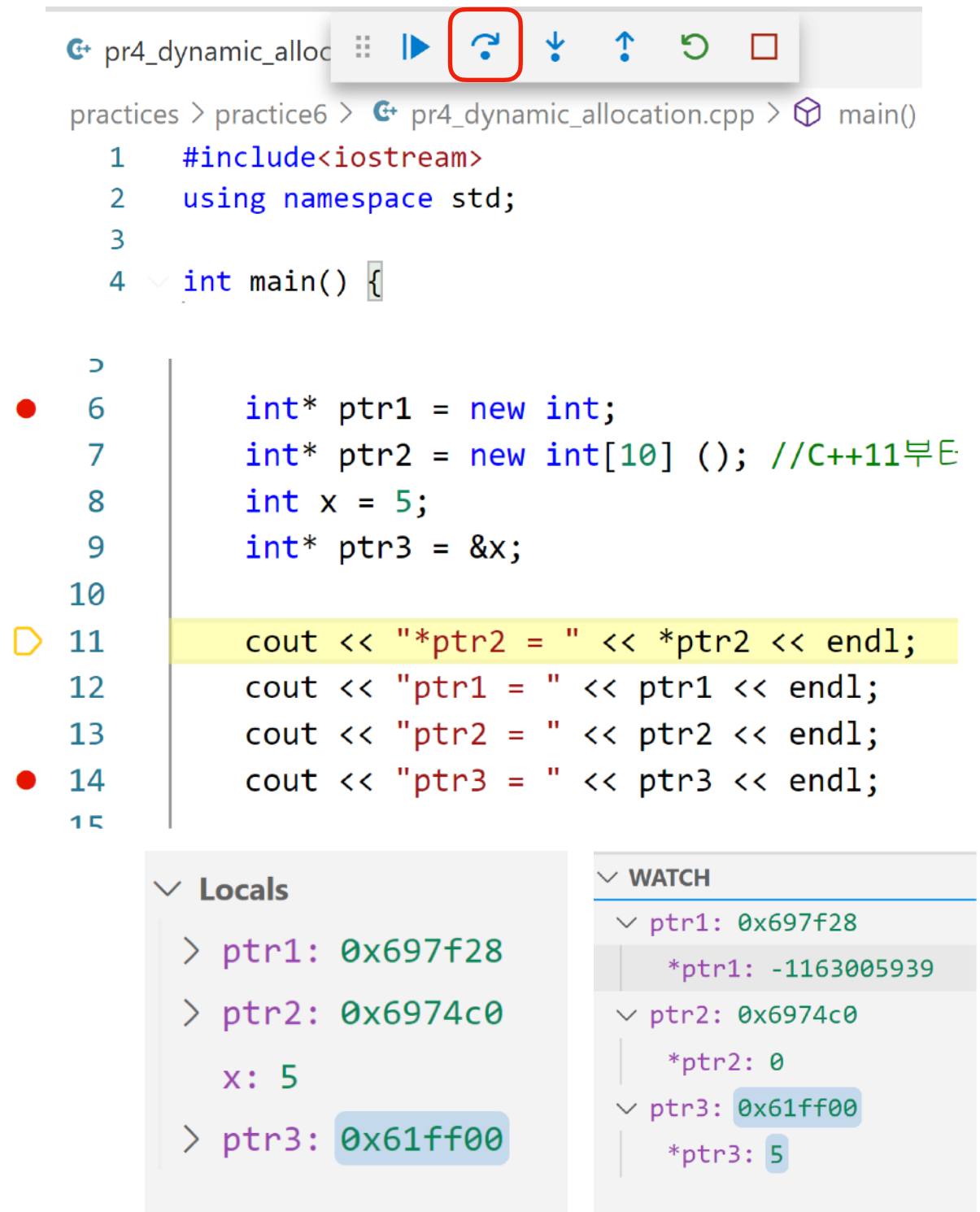
# 네번째 실습



- 디버그 모드에서 출력되는 내용은 Debug Console에서 볼 수 있습니다.
- 터미널의 경우에는 디버그 모드의 출력은 나타나지 않으니 주의하세요.
- 여러가지 출력이 많이 나오므로, 오른쪽의 Filter에 단어를 입력하여 정해진 출력만 볼 수 있습니다.
- ptr을 입력하면 실습 파일이 실제로 출력하는 부분만 필터링하여 볼 수 있습니다.

# 네번째 실습

- 컨트롤러에서 실행을 눌러 실행이 되고 나면, 6번째 줄에서 실행이 멈춥니다.
- 그 상태에서 두번째 버튼을 한 번씩 클릭하여 한 줄씩 프로그램을 실행합니다.
- 코드가 실행되면서 프로그램의 상태가 변화하는 것을 왼쪽 부분의 Locals와 WATCH에서 확인할 수 있습니다.



```
pr4_dynamic_alloc
practices > practice6 > pr4_dynamic_allocation.cpp > main()
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5
6      int* ptr1 = new int;
7      int* ptr2 = new int[10] (); //C++11부터
8      int x = 5;
9      int* ptr3 = &x;
10
11     cout << "*ptr2 = " << *ptr2 << endl;
12     cout << "ptr1 = " << ptr1 << endl;
13     cout << "ptr2 = " << ptr2 << endl;
14     cout << "ptr3 = " << ptr3 << endl;
15 }
```

**Locals**

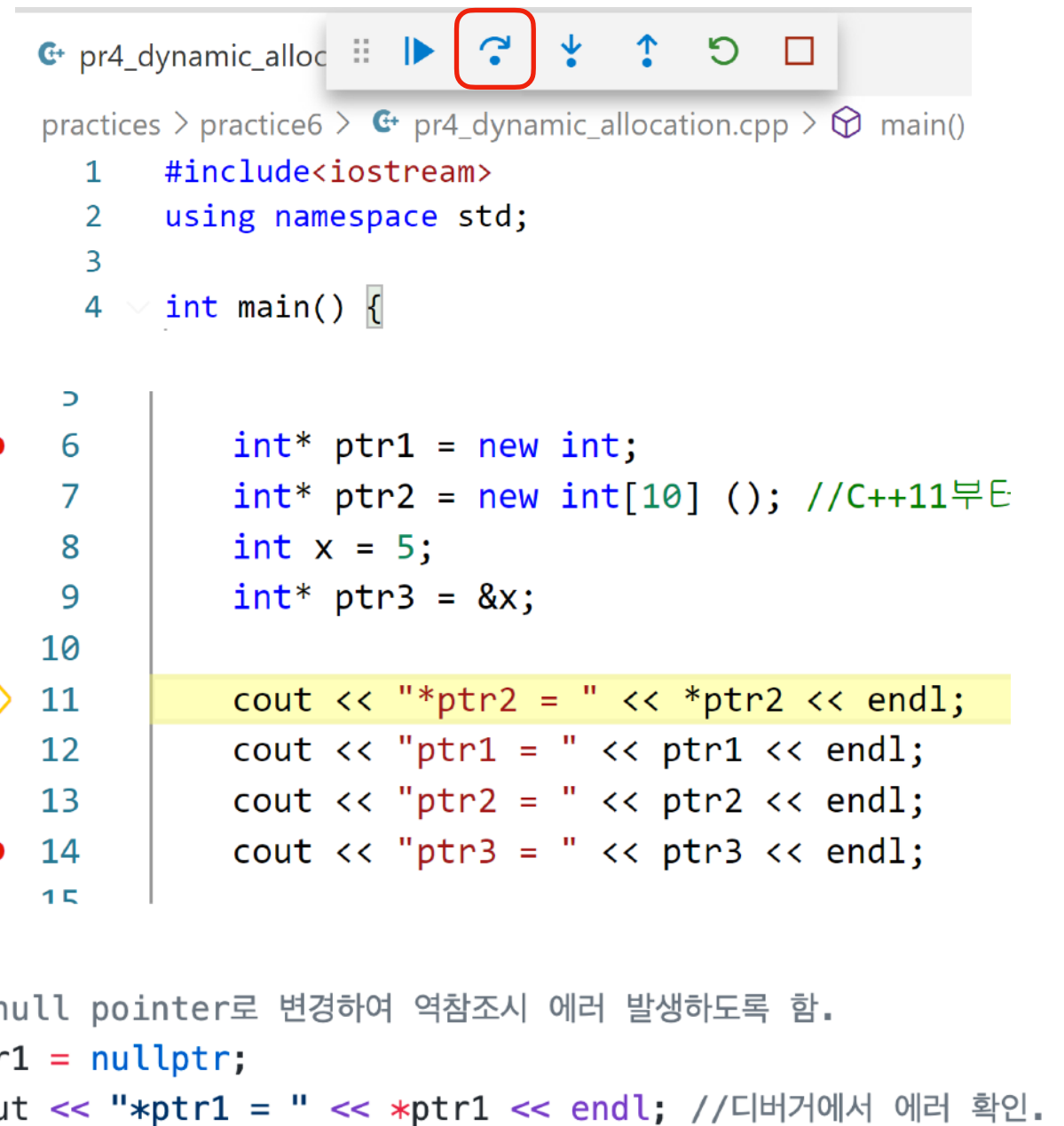
- > ptr1: 0x697f28
- > ptr2: 0x6974c0
- x: 5
- > ptr3: 0x61ff00

**WATCH**

- > ptr1: 0x697f28
  - \*ptr1: -1163005939
- > ptr2: 0x6974c0
  - \*ptr2: 0
- > ptr3: 0x61ff00
  - \*ptr3: 5

# 네번째 실습

- 네번째 실습은 이렇게 하나씩 코드를 실행하여 프로그램 상태 변화를 확인하는 것입니다.
- 최종적으로 코드가 아래 나온 부분에 도달하면 에러가 발생하게 됩니다.
- F5키를 이용하여 디버그 모드에서 다시 파일을 실행해 볼 수 있습니다.
- 기존에 실행하던 방식으로 실행하여 디버그 모드와 차이를 확인해 보는 것도 좋습니다.



The screenshot shows a C++ IDE with a file named `pr4_dynamic_allocation.cpp`. The code is as follows:

```
practices > practice6 > pr4_dynamic_allocation.cpp > main()
1  #include<iostream>
2  using namespace std;
3
4  int main() {
5
6      int* ptr1 = new int;
7      int* ptr2 = new int[10] (); //C++11부터
8      int x = 5;
9      int* ptr3 = &x;
10
11      cout << "*ptr2 = " << *ptr2 << endl;
12      cout << "ptr1 = " << ptr1 << endl;
13      cout << "ptr2 = " << ptr2 << endl;
14      cout << "ptr3 = " << ptr3 << endl;
15
16      //null pointer로 변경하여 역참조시 에러 발생하도록 함.
17      ptr1 = nullptr;
18      cout << "*ptr1 = " << *ptr1 << endl; //디버거에서 에러 확인.
19  }
```

The IDE interface includes a toolbar with icons for running, stepping through code, and other debugging functions. The code is color-coded, and line 11 is highlighted in yellow. The comments at the bottom of the code indicate the purpose of the final lines: to cause a runtime error by dereferencing a null pointer.

# 네번째 실습

- launch.json이 자동 생성되지 않았을 경우를 대비한 파일 예제입니다.
- 에러가 발생한다면 네모로 되어있는 부분이 자신의 tasks.json 파일에서 빌드로 생성하는 이름과 맞는지, MinGW 경로가 맞는지 확인하세요.

```
"configurations": [  
  {  
    "name": "g++.exe - 활성 파일 빌드 및 디버그",  
    "type": "cppdbg",  
    "request": "launch",  
    "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",  
    "args": [],  
    "stopAtEntry": false,  
    "cwd": "${workspaceFolder}",  
    "environment": [],  
    "externalConsole": false,  
    "MIMode": "gdb",  
    "miDebuggerPath": "C:\\\\MinGW\\\\bin\\\\gdb.exe",  
    "setupCommands": [  
      {  
        "description": "gdb에 자동 서식 지정 사용",  
        "text": "-enable-pretty-printing",  
        "ignoreFailures": true  
      }  
    ],  
    "preLaunchTask": "C/C++: g++.exe build active file"  
  }  
]
```

# 다섯번째 실습

- 다섯번째 실습은 참조변수 사용 및 const와 참조, 포인터에 관련된 실습입니다.
- 첫번째 부분은 참조 변수를 선언, 초기화하고 사용하는 방식을 확인하는 부분입니다.
- 변수 a와 n이 같은 변수처럼 동작하는 것을 볼 수 있습니다.
- 밑 부분에서는 참조로 매개변수를 사용해 Call by Reference로 동작하는 것을 확인할 수 있습니다.

```
void print(int (&arr)[5]) {  
    cout << "Array:" << endl;  
    //Ranged For 사용가능.  
    for(int x : arr)  
        cout << x << ", ";  
}  
  
int a = 5, b = 3;  
int& n = a;  
cout << "a = " << a << endl;  
cout << "n = " << n << endl;  
n = 10;  
cout << "a = " << a << endl;  
cout << "n = " << n << endl;  
n = b;  
cout << "a = " << a << endl;  
cout << "n = " << n << endl;  
  
//배열 출력.  
int arr[] = { 1, 2, 3, 4, 5 };  
print(arr);
```



# 다섯번째 실습

- 두번째 부분은 const 키워드 사용 예제를 보여주는 실습입니다.
- 주석으로 처리된 부분들을 해제해서 컴파일러 에러가 어떻게 표시되는지 확인해 보세요.
- const 키워드 사용에 따라 어떤 값을 변경하는 것이 불가능한지 꼭 익혀두셔야 합니다.

```
//상수를 가리키는 포인터.  
const int SIZE = 10;  
const int* ptr = &SIZE; //const가 반드시 필요.  
const int& ref = SIZE; //const가 반드시 필요.  
//이 부분은 허용되지 않음 - 주석을 해제하면 컴파일러에러 표시됨.  
//*ptr = 1;  
//ref = 1;  
  
//변수를 참조하면?  
const int* ptr2 = &a;  
const int& ref2 = a;  
//여전히 허용되지 않음.  
//*ptr = 2;  
//ref2 = 2;  
  
//상수 포인터(const pointer)  
int* const ptr3 = &a;  
*ptr3 = 10; //가리키는 변수를 바꾸는 것은 허용됨.  
cout << "a = " << a << endl;  
//포인터가 가리키는 주소는 변경 불가능.  
//ptr3 = ptr2;  
  
//메모리, 포인터 값 둘 모두 변경 불가능.  
const int* const ptr4 = &a;  
// *ptr4 = 3;  
// ptr4 = ptr;
```

# 실습 제출물

```
26 //null pointer로 변경하여 역참조시 에러 발생하도록 함.  
27 ptr1 = nullptr;  
28 cout << "*ptr1 = " << *ptr1 << endl; //디버거에서
```

Exception has occurred.  
EXC\_BAD\_ACCESS (code=1, address=0x0)

## Locals

```
> ptr1: 0x0000000000000000  
> ptr2: 0x0000000100300010  
x: 5  
> ptr3: 0x00007ffefbffd34
```

- 네번째 실습에서 디버그 모드를 사용했을 때 에러가 출력되는 부분을 캡처하여 제출하시면 됩니다.
- VSCode가 아닌 다른 IDE를 사용하신다면 그 IDE의 디버그 모드 화면을 캡처하세요.
- 설정 등의 문제나, 다른 IDE 사용에 따른 문제로 에러 화면을 캡처하기 어렵다면 오른쪽처럼 ptr1~ptr3의 값을 디버그 모드에서 확인하는 화면을 올리셔도 됩니다.

# 실습 정리

- 이번주는 총 5개의 파일로 실습을 진행하였습니다.
- 포인터와 참조 변수 사용법에 대해 익혔습니다.
- 이들과 메모리와의 관계에 대해서도 알아보았습니다.
- 네번째 실습에서는 처음으로 디버그 모드를 사용하는 방법에 대해 간단하게 알아보았습니다.
- 디버그 모드는 앞으로 복잡한 데이터형인 구조체(struct)나 클래스(class)를 다루는 경우 프로그램 상태를 확인하기 위해 유용하게 쓸 수 있으니 꼭 사용법을 익히는 것이 좋습니다.