민 선 생 코 딩 학 원 훈 련 반

수업노트 Class3

배우는 내용

1. 연산자 오버로딩
2. cin cout 객체
3. string class

복습 : 오버로딩

- 오버로딩이란 같은 이름의 함수를 새로 만드는 것

- 함수이름이 같다면,
  인자값에 따라서 호출되는 함수가 결정 됨

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

class MC
{
    int t;

public:
    void setting()
    {
        t = 10;
    }

    void setting(int value)
    {
        t = value;
    }
};

int main()
{
    MC bt;

    bt.setting(); //필드 t에 10 들어감
    bt.setting(15); //필드 t에 15 들어감

    return 0;
}
```

아래 코드들은 왜 Compile Error 일까?

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

class MINS
{
    int insideValue;

public:
    MINS(int value)
    {
        insideValue = value;
    }

};

int main()
{
    MINS t(100);


    int ret = t + 5;

    cout << ret;

    return 0;
}
```

- Compile Error, 숫자 5를 더할 때
  인스턴스 t 안 어느 맴버변수에다가 더할 지 명확하지 않음

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

class MINS
{
    int insideValue;

public:
    MINS(int value)
    {
        insideValue = value;
    }

};

int main()
{
    MINS t(100);

    int ret = t.insideValue + 5;

    cout << ret;

    return 0;
}
```

- 인스턴스 t에서 private 맴버변수 insideValue를 사용할 수 없음
  따라서 Compile Error

맴버 변수값을 더하는 두 가지 방법

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

class MINS
{
    int insideValue;

public:
    MINS(int value)
    {
        insideValue = value;
    }

    int getValue()
    {
        return insideValue;
    }
};

int main()
{
    MINS t(100);

    int ret = t.getValue() + 5;

    cout << ret;

    return 0;
}
```

- **방법1**
  getValue라는 메서드를 만들어서
  private 맴버변수의 값을 return 해준다
  (많이 쓰는 방식)

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

class MINS
{
public:
    int insideValue;

    MINS(int value)
    {
        insideValue = value;
    }

};

int main()
{
    MINS t(100);

    int ret = t.insideValue + 5;

    cout << ret;

    return 0;
}
```

- **방법2**
  private를 쓰는 것을 포기한다
  전부 public으로 바꾼다.

연산자 (operator) 종류

- 연산자 : **operator** 라고 함
- `+ - * /` 사칙 연산자 뿐만 아니라 비교&대입 연산자 (`= >= <= !=`) `[] () >> <<` 도 연산자 라고 한다.

Common operators

assignment	increment decrement	arithmetic	logical	comparison	member access	other

`a = b`
`a = rvalue`
`a += b`
`a -= b`
`a *= b`
`a /= b`
`a %= b`
`a &= b`
`a |= b`
`a ^= b`
`a <<= b`
`a >>= b`
	`++a`
	`--a`
	`a++`
	`a--`
		`+a`
		`-a`
		`a + b`
		`a - b`
		`a * b`
		`a / b`
		`a % b`
		`~a`
		`a & b`
		`a | b`
		`a ^ b`
		`a << b`
		`a >> b`
			`!a`
			`a && b`
			`a || b`
				`a == b`
				`a != b`
				`a < b`
				`a > b`
				`a <= b`
				`a >= b`
					`a[b]`
					`*a`
					`&a`
					`a->b`
					`a.b`
					`a->*b`
					`a.*b`
						`a(...)`
						`a, b`
						`(type) a`
						`? :`

Special operators

`static_cast` converts one type to another compatible type
`dynamic_cast` converts virtual base class to derived class
`const_cast` converts type to compatible type with different `cv` qualifiers
`reinterpret_cast` converts type to incompatible type
`new` allocates memory
`delete` deallocates memory
`sizeof` queries the size of a type
`sizeof...` queries the size of a parameter pack (since C++11)
`typeid` queries the type information of a type
`noexcept` checks if an expression can throw an exception (since C++11)
`alignof` queries alignment requirements of a type (since C++11)

연산자 종류
(https://ko.cppreference.com/w/cpp/language/operator_precedence)

연산자 (operator) 오버로딩

- 기존 연산자를 다른 역할로 바꿈

  ex) 덧셈 연산자를 쓰면, 곱셈으로 바꿔서 처리 하는 것도 가능

  ex) >> 를 사용하면, 덧셈으로 처리되도록 바꿔서 처리 하는 것도 가능

연산자 오버로딩으로 연산자 역할을 바꾼다

- **operator+** 라는 연산자 오버로딩 메서드를 만들었음
- 앞으로 객체에 **+** 기호를 쓰면
  insideValue의 값을 더해주는 역할을 부여한다.

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

class MINS
{
    int insideValue;

public:
    MINS(int value)
    {
        insideValue = value;
    }

    int operator+(int value)
    {
        return insideValue + value;
    }
};

int main()
{
    MINS t(100);

    int ret = t + 5;

    cout << ret;

    return 0;
}
```

연산자 오버로딩 덕분에
Compile Error가 나지 않는다.

연산자 오버로딩 예제

- **>>** 를 연산자오버로딩 함
- 출력결과
  WOW
  30

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

class HOPE
{
public:

    int operator>>(int value)
    {
        cout << "WOW" << endl;
        return value * 2;
    }
};

int main()
{
    HOPE t;

    int g = t >> 15;

    cout << g;

    return 0;
}
```

연산자 오버로딩 자주 쓰나요?

- 연산자 오버로딩 된 연산자의 동작을 혼동 할 수 있다.

  ex) 내가 알고있는 + 는 덧셈인데, 덧셈으로 동작하지 않는다.

- 직접 연산자 오버로딩을 만들어 사용하는 일은 드물다.

  표준 Class Library 일부에는 연산자 오버로딩이 되어있어서,
  만들어진 연산자오버로딩은 자주 사용한다.

연산자 오버로딩 함수를 만들 일은 드물다.
대신 사용은 자주 한다.

연산자 오버로딩을 쓰는 대표적인 Class / 객체

- cin, cout 인스턴스
- string class

cout / cin

- iostream.h 파일 안

  ostream / istream라는 클래스로 만들어진 인스턴스

  ```
  //int x;
  //ostream cout;
  //istream cin;
  ```

- cout라는 인스턴스에서 << 연산자 오버로딩 되어있는 것을 사용하는 예제

  cout << “HA”;

string class

- 문장을 쉽게 쓰기 위한 class
- `=`, `[]`, `==`, `+` 에 대한 연산자 오버로딩 되어있어서 쉽게 문장을 쓸 수 있다

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

class ABC
{
public:
    int x;
    int y;
};

int main()
{
    ABC t;
    t.x = 10;
    t.y = 15;

    ABC p = 15; //불가능

    string a = "ASDF"; //string class안에 있는 연산자오버로딩 덕분에 가능

    return 0;
}
```

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

int main()
{
    string a = "ABCDEF";

    return 0;
}
```

문장 비교 및 추가하기

- String class 인스턴스에서 여러가지 연산자 오버로딩 사용방법

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

int main()
{
    string a = "ASDF";
    string b = "EWRQ";

    if (a == b) cout << "같은문장";
    else cout << "다른문장";

    return 0;
}
```

문장 비교하기

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

int main()
{
    string a = "AAA";

    string b = a + "BBB";

    cout << b; //출력결과 : AAABBB

    return 0;
}
```

문장 추가하기

string class의 public 메서드

- string class에는 문장 길이를 구해주는 length( ) 라는 public 메서드가 있다.

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

int main( )
{
    string a = "ASDF";

    int len = a.length( );

    cout << len;

    return 0;
}
```

char배열처럼 쓰는 string class

- string class 내부에는 연산자오버로딩 **[ ]** 도 구현되어 있음
- 이덕에 맴버변수를 배열처럼 쓸 수 있음

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

int main()
{
    string a = "ABCDEFG";

    int len = a.length();

    //실행결과 : ABCDEFG
    cout << a << endl;

    //실행결과 : ABCDEFG
    for (int i = 0; i < len; i++)
    {
        cout << a[i];
    }


    return 0;
}
```