2023 11 29

Member Functions Pointers

```
class Nec
{
    public:
        void foo(int);
};

int main()
{
    auto fp1 = &Nec::foo;
    void (Nec::*fp2)(int) = &Nec::foo;

    Nec mynec;
    (mynec.*fp)(12);

    Nec* necptr = &mynec;
    (necptr->*fp)(12);
}
```

```
// Kullanım 1:
class Nec
{
    public:
        void f1(int);
        void f2(int);
        void f3(int);
        void f4(int);
};

void foo(int, void (Nec::*mfp(int))
{
    // hangi üye fonksiyonun işi yapmasını istiyorsak onun adresini göndercez
}

using Mfptr = int (Nec::*)(int);
int main()
{
    Mfptr fpa[] =
    {
        &Nec::f1,
        &Nec::f2,
        &Nec::f3,
        &Nec::f4,
    }

    Nec m;

    for (auto ptr : fpa)
    {
        std::cout << (m.*fptr)(10) << '\n';
        std::invoke(fptr, m, 10) << '\n';
    }
}
```

Data Member Pointer

```cpp
struct Nec
{
    int a {3};
    int b {5};
    int c {7};

};

int main()
{
    Nec mynec;

    auto ptr = &mynec.a; // int*
    auto ptr1 = &Nec::a // int Nec::*

    std::cout << mynec.*x << "\n"; // 3
    std::invoke(ptr, mynec) = 9999;

    std::cout << mynec.a << '\n';

}
```

```cpp
// Kullanım 1:
struct OHLC
{
    double open;
    double high;
    double low;
    double close;
};

double get_moving_average(const std::vector<OHLC>& candles, double OHLC::*fp)
{
    double sum{};
    for (const auto& candle : candles)
    {
        //sum +=candle.*fp;
        sum += std::invoke(fp, candle);
    }

    return sum / candles.size();
}
```

std::bitset

```cpp
#include <bitset>
int main()
{
    using namespace std;
    // bir container değil
    bitset<16> bs1; // 16 bit sağlıyor

    bitset<32> bx{"101010110110101100"};

    bx[4] = true;
    bx[4].flip(); // ters çevirme
    auto n = bx.count(); // kaç tane bit true olduğunu söyler

    auto non = bx.none(); // zero bitse true döner
    auto any = bx.any(); //  herhangi bir true ise true döner
    auto all = bx.all(); // tüm bitler true ise true döner

    auto test = bx.test(2); // 2. bit true mu false mu döner

    auto ulong = bx.to_ulong(); // bit'i ulong'a dönüştürür
    auto to_string = bx.to_string(); // bit'i string'e dönüştürür

    bx.set(5, false); // 5. biti false set ettik
    bx.reset(5);  // 5. biti resetledik ( false yaptık )

    bx.set(4).reset(3).flip(7);


    bitset<16> x {3124u};
    bitset<16> y {3124u};

    cout << x == y << "\n"; // true
    //cout << x < y; // böyle bir operator yok

    auto fless = [](const bitset<32>& b1, const bitset<32>& b2)
    {
        return b1.to_ulong() < b2.to_ulong();
    }

    set<bitset<32>, decltype(fless> bset;


    bitset<16> z{1u};
    cout << ( x << 2) << "\n";
    cout <<  x << 2 << "\n";
}
```

Dinamik Ömürlü Nesneler

global operator new ile neler yapabilirim:

- overload edebilirim
- isimilye çağırabiliriz
- bir sınıf için overload edebiliriz

new T

operator new(sizeof(T))

```
// my operator new overload
void* operator new(std::size_t n)
{
    std::cout << "operator new(size_t) called\n";
    std::cout << "n = " << n << "\n;

    auto vp = std::malloc(n);
    if (!vp)
        throw std::bad_alloc{};

    std::cout << "the address of the allocated block is " << vp << "\n";
}

void operator delete(void *vp)
{
    std::cout << "operator delete called vp = " << vp << "\n";
}

class Neco
{
    unsigned char buffer[2048]{};
};

int main()
{
    std::cout << "sizeof(Neco) = " << sizeof(Neco) << "\n";

    Neco* p = new Neco;

    delete p;
}
```

operator new işlevinin başarısız olması durumunda std::bad_alloc sınıfı türünden bir exception throw eder.

```cpp
#include <new>
void myhandler()
{
    std::cout << "myhandler called!!!";
}

int main()
{
    vector<void*> myvec;

    // exception throw etmek yerine myhandler çağrılacak her döngüde
    set_new_handler(myhandler);

    try
    {
        for (int i = 0; i < 10000; ++i)
        {
            myvec.push_back(operator new(1024 * 1024));
        }
    }
    catch (const std::exception& ex)
    {
        std::cout << "exception caught: " << ex.what() << "\n";
    }
}
```

set_new_handler nasıl kullanabilirim:

1. Daha önce allocate ettiğim belliğin geri verilmesi
2. Daha özel exception sınıfını throw edebiliriz
3. Başka bir handler fonksiyonu çağrabilir.
4. Bir kaç defa deneme yapıp set_new_handler(nullptr) yapabilir.