Code Read if it's required to print address of variable 'x' write -> address of x

```
template<typename T, int offset>
int toInt(T i) {
    return (int) (i * offset);
}

int main() {
    fmt::println("{}", (toInt<double, 2>(1.5)));
}
```

```
int main() {
    int x = 2;
    int y = 2;
    int &r = x;
    int *p = &r;
    int **px = &p;
    p = &y;

    std::cout << px << std::endl;
    std::cout << *px << std::endl;
    std::cout << **px << std::endl;
    std::cout << (*p)-- << std::endl;
    std::cout << r-- << std::endl;
    std::cout << x << std::endl;
    std::cout << y << std::endl;
}
```

```
int main() {
    int tab[] = {1, 10, 100, 1000, 10000};
    int *ptr = tab + 1;
    ptr++;

    std::cout  << ptr << std::endl;
    std::cout  << *ptr << std::endl;
    std::cout  << *(ptr + 1) << std::endl;
    std::cout  << (*ptr) + 1 << std::endl;
}
```

```
int main() {
    const char *napis = "Hope";
    const char *n = napis + 1;
    std::cout << *napis << std::endl;
    std::cout << n << std::endl;
}
```

```
int fun1(int x) {
    x++;
    fmt::println("{}", x);
    return x - 1;
}

int &fun2(int &x) {
    x--;
    return x;
}

int fun3(int *x) {
    int y = *x + 2;
    return y;
}

int main() {
    int a = 1;
    int b = 2;
    fun1(a);
    fmt::println("a -> {}", a);
    b = fun2(b);
    fmt::println("b -> {}", b);
    fun3(&a);
    fmt::println("a -> {}", a);
}
```

```
int main() {
    std::set<int> s;
    s.insert(8);
    s.insert(7);
    s.insert(10);
    s.insert(8);
    std::cout << (*s.begin() + 1) << std::endl;
    std::cout << (*(--s.end())) << std::endl;
    std::cout << s.size() << std::endl;
}
```

```
class A {
public:
    int a;

    A(int a) {
        this->a = a;
    }

    virtual void run() {
        a++;
    }
};

class B : public A {
public:
    int b;

    B(int a) : A(a) {
        this->b = a;
    }

    void run() override {
        A::a += 2;
    }
};

int main() {
    A *b = new B(4);
    b->run();
    std::cout << b->a << std::endl;
}
```

```
class A {
public:
    A() {
        fmt::println("A created");
    }

    A(A &a) {
        fmt::println("A copied");
    }

    virtual ~A() {
        fmt::println("A destroyed");
    }
};

class B : public A {
public:
    B() {
        fmt::println("B created");
    }

    B(B &b) {
        fmt::println("B copied");
    }

    virtual ~B() {
        fmt::println("B destroyed");
    }
};

int main() {
    auto a = B();
    A c = a;
    A &r = a;
}
```