

**O’ZBEKISTON RESPUBLIKASI OLIY VA O’RTA MAXSUS**

**TA’LIM VAZIRLIGI**

**SHAROV RASHIDOV NOMIDAGI SAMARQAND DAVLAT**

**UNIVERSITETI**

**Intelektual tizimlar va kompyuter texnalogiyalari fakultiteti**

**Dasturiy injiniring yo’nalishi 203 - guruh talabasi**

**Ilhomjonov Iqbolshohning Obyektga yo’naltirilgan dasturlash fanidan**

**Mustaqil ishi**

**MAVZU: Obyektga mo’jallangan dasturlash asoslari**

**Bajardi: Ilhomjonov Iqbolshoh**

**Tekshirdi: Qodirov.A.A.**

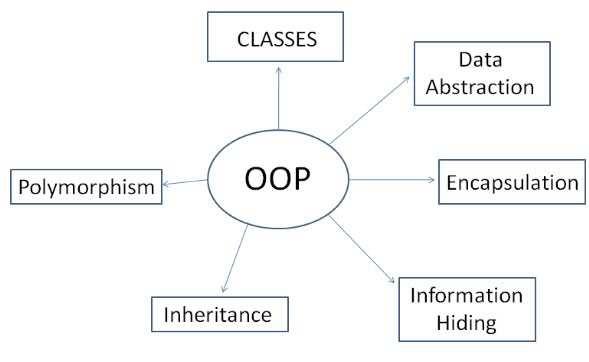
**Samarqand-2024**

***Reja:***

1. **Kirish**
2. **Nazariy qism**
3. Class va obyekt tushunchasi
4. Konstruktorlar va destruktor
5. Meros olish
6. Polimorfizm
7. Obyekt massivlari
8. Do’st class va va do’st funksiyalar
9. Amallarni qayta yuklash
10. **Xulosa**

**Kirish**

Obyektga yo'naltirilgan dasturlash(OYD) haqida

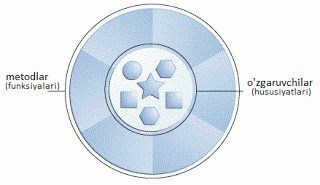


Obyekt nima?

Obyekt Obyektga yo'naltirilgan dasturlash(OYD) texnologiyasining eng asosiy kalit tushunchasidir. Atrofga qarang, haqiqiy hayotdagi bir necha obyektlarni ko'rishingiz mumkin: stol, uy, it, mushuk, televizor va h.k.

Ularning barchasining albatta hususiyatlari va bajaradigan vazifalari (funktsiyalari) bor. Masalan, Mushuk hususiyatlari: rangi, qorni to'qligi, yoshi, vazni; funktsiyalari: ovqat yeyishi, myovlashi, yurishi, sichqon tutishi. Mashina, hususiyatlari: tezligi, rangi, nomi, narxi; funktsiyalari: yurishi, to'xtashi, oyna artgichlarining ishlashi, eshiklarning ochilib yopilishi v.h.k. Bu kabi hayotiy misollarning hususiyatlari va funktsiyalarini aniqlash OYD nuqtai nazaridan fikrlashning eng zo'r ko'rinishidir.

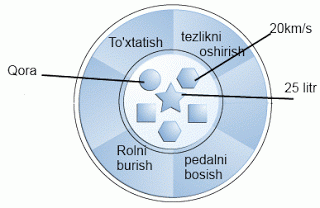
Bir daqiqaga to'xtang va hozirda atrofingizdagi biror narsalarni analiz qiling. Har bir obyekt uchun o'zingizdan so'rang: "Bu obyektning qanday hususiyatlari bor?" , "Qanday vazifalarni bajaradi?" kabi. Va kuzatish natijalaringizni yozib oling, sezgan bo'lsangiz tuziladigan ro'yxat obyektning murakkabligiga qarab ko'payib boradi. Kompyuter indikatorining 2 ta hususiyati bor o'chiq va yoniq; funktsiyalari esa yonish va o'chish. Bu barcha kuzatishlar OYD dunyosiga o'tkazish mumkin.



**Dasturlashdagi obyekt**.

Dasturlashdagi obyekt(bundan keyin oddiygina obyekt deb ketiladi) ham haqiqiy hayotdagi obyektlarga o'xshash: Ular ham qandaydir hususiyatlar va bajaradigan funktsiyalardan iborat bo'ladi. Obyektning hususiyatlari har xil dasturiy o'zgaruvchilardan iborat bo'ladi va ularning o'zgartirish uchun qandaydir funktsiyalar bajariladi. Bunday funktsiyalar bilan o'zgaruvchilarning holatini berkitish mumkin ya'ni aynan o'sha o'zgaruvchini tashqaridan o'zgartirish uchun albatta maxsus funktsiyadan foydalanish kerak bo'ladi. Bu jarayon **"Enkapsulatsiya"** deb atalib, OYDning eng muxim tushunchalaradian biridir. Hech e'tibor berganmisiz dorilarda ham shu termin ishalitladi ya'ni kapsula(ustidan maxsus modda bilan o'ralgan dorilar), buni misolni **Enkapsulatsiya** jarayoni esda yaxshi qolishi va tushunarli bo'lishi uchun keltirdim.

Mashinani tasavvur qiling



**Uni dasturlash obyekit sifatida modellashtiramiz:**

Uning o'zgaruvchilari( hozirgi tezligi, qolgan benzini, va h.k) va uning funktsiyalari(to'xtatish, tezlikni oshirish, rulni burish, va h.k.). Bu yerda uning bakidagi benzini yurishi tufayli kamayib boradi demak uning qiymatining o'zgarishi 0dan bakning sig'imigacha bo'ladi, yoki uning tezligi ham shu kabi aynan qaysidir funktsiyalarning amalga oshirilishi orqali u ham 0 dan maksimal tezligigacha o'zgarishi mumkin. Bulardan tashqari mashinaning ba'zi hususiyatlari borki ular o'zgarmasligi mumkin, masalan, rangi.

Demak, ko'rinib turiptiki mashina ham o'z navbatida bir necha mayda obyeklardan iborat bo'ladi. Va albatta ularni kodda yozganda ham alohida obyekt sifatida ifodalash kerak bu orqali nimalarga erishish mumkin:

Qismlilik: Har bir obyektga tegishli bo'lgan kodlar alohida-alohida, boshqa obyeklarga bog'liq bo'lmagan holda boshqarish imkoniyatiga ega bo'lamiz. Bu hammasi emas, tasavvur qiling mashina obyektini ifodalovchi kodni bo'lmasdan faqat bitta faylda ifodaladik; bu esa murakkabligiga qarab yuzlab hatto minglab qatorli kod bo'lishi mumkin. Undan biror narsani topib-o'zgartrish ancha mashaqqat bo'ladi.

Qayta foydalanish: Yana boshqa plyus tarafi biz bo'laklagan mashinaning detallarini boshqa obyektlarda ham ishlatishimiz mumkin. Masalan, 2 xil mashina ularning shunday qismlari borki aynan bir xil, ana o'shalar uchun ikki marta alohida kod yozmasdan, bitta yozganimizni qayta ishlatishimiz mumkin.

Uzib-ulanuvchanligi: buni tushunish uchun yuqoridagi misoldan foydlanamiz, aytaylik, mashinaning biror qismi ishlamayapti, xo'sh nima qilinadi? yoki ishlab turgan boshqasiga almashtiramiz, yoki tuzatamiz. Mashinaning biror bolti buzilsa uni boshqa ishlab turgani bilan almashtirasiz yoki tuzatamiz lekin mashinani almashtirmaymiz.

**Sinflar (class):**

Classlar - dasturlashda bir qismini yoki barchasini o'z ichiga olgan ma'lumot va funksiyalarni ifodalaydigan "shablon" yoki "to'plam" sifatida xizmat qiladi. Sinflar qo'shimcha ob'ektlarni yaratish uchun qo'llaniladi.

Masalan, quyidagi koddagi Car sinfi bir mashina sinfini ifodalaydi:

class Car {

public:

    string brand;

    string model;

    int year;

};

Bu sinf Carning xususiyatlarini (brand, model, year) ifodalaydi.

**Obyekt (object):**

Obyekt, sinfning ma'lum bir namunasi yoki ma'lum bir kopiysi bo'lib, sinfda ifodalangan ma'lumot va funksiyalarga ega.

Quyidagi koddagi **Car** sinfidan foydalanib, bir mashina obyektini yaratamiz:

Car myCar;

    myCar.brand = "Toyota";

    myCar.model = "Corolla";

    myCar.year = 2020;

Bu kodda **myCar** nomli bir obyekt yaratiladi, va uning **brand**, **model** va **year** xususiyatlari belgilanadi.

1. **Misol** *(Class methodlari va xususiyatlari)*

#include <iostream>

using namespace std;

class Car

{

public:

    string brand;

    string model;

    int year;

    // Sinfning konstruktori

    Car(string brand, string model, int year)

    {

        this->brand = brand;

        this->model = model;

        this->year = year;

    }

    // Method - Obyekt haqida ma'lumot chiqarish

    void displayInfo()

    {

        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << endl;

    }

};

int main()

{

    // Sinfdan obyekt yaratish

    Car myCar("Toyota", "Corolla", 2020);

    // Obyektni ekranga chiqarish

    myCar.displayInfo();

    return 0;

}

// NATIJA



Bu misolda Car sinfi brand, model, va year xususiyatlari bilan va displayInfo metodi bilan ifodalangan. Obyekt yaratilganda, uning ma'lumotlari yuklanadi va displayInfo metodi obyektning ma'lumotlarini ekranga chiqaradi.

Konstruktorlar va Destruktorlar:

**Konstruktorlar:** Konstruktorlar, sinf obyektini yaratish jarayonini boshlash uchun ishlatiladi. Ular sinf nomi bilan bir xil nomga ega bo'lishi kerak va qayta qayta chaqirilishi mumkin emas. Konstruktorlarning asosiy vazifasi, sinf obyektini yaratish va uning barcha xususiyatlari va holatlarini boshlashdir. Konstruktorlar parametrlarga ega bo'lishi mumkin va boshqa sinflarning konstruktorlarini chaqirish orqali sinf tarkibidagi xususiyatlarga murojaat qilishi mumkin.

Misol:

class Car

{

public:

    string brand;

    string model;

    int year;

    // Parametrlar bilan konstruktor

    Car(string b, string m, int y)

    {

        brand = b;

        model = m;

        year = y;

    }

    // Metod - Ma'lumotlarni chiqarish

    void displayInfo()

    {

        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << endl;

    }

};

int main()

{

    // Konstruktor orqali obyekt yaratish

    Car myCar("Toyota", "Corolla", 2020);

    myCar.displayInfo();

    return 0;

}

// NATIJA



**Destruktorlar**: Destruktorlar, sinf obyektini o'chirish jarayonini amalga oshirish uchun ishlatiladi. Ular sinf obyektining to'liqqacha o'chirilishi bo'lgan joyida avtomatik ravishda chaqiriladi. Destruktorlar tarkibiy fayllarni yoki manzillarni qayta ishlatib olish uchun juda qulay bo'lishi mumkin.

Misol:

class Car

{

public:

    string brand;

    string model;

    int year;

    // Parametrlar bilan konstruktor

    Car(string b, string m, int y)

    {

        brand = b;

        model = m;

        year = y;

        cout << "Object is created." << endl;

    }

    // Destruktor

    ~Car()

    {

        cout << "Object is destroyed." << endl;

    }

    // Metod - Ma'lumotlarni chiqarish

    void displayInfo()

    {

        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << endl;

    }

};

int main()

{

    // Konstruktor orqali obyekt yaratish

    Car myCar("Toyota", "Corolla", 2020);

    myCar.displayInfo();

    // Destruktor obyekt o'chirilganda chaqiriladi

    return 0;

}

// NATIJA



**Kapsulatsiya:**

Kapsulatsiya dasturlashning mohiyatini yaxshilash uchun keng ko'lamli o'zgaruvchanlar va funksiyalarni birlashtirish va ularga maxfiylik berishni ta'minlaydi. Buning mazmuni shu: barcha dasturlash tillarida dasturchi o'zgaruvchanlarga doim etkazish mumkin bo'lgan joyga chek kiritishni tavsiya etadi. Shunday qilib, sinfda ifodalangan o'zgaruvchilar o'zgartirishlardan himoyalangan bo'lib, faqat sinfning ichidagi funksiyalar orqali ularga murojaat qilinadi. Bu shakl sinfning ichki ma'lumotlarini boshqa obyektlardan yashiradi va ularga faqat xizmat ko'rsatadi.

Kapsulatsiya, dasturchining kodni yozish va uni boshqa dasturchilar bilan birlashtirish va ishlab chiqishni osonlashtiradi. Buning oqibatida, kodni boshqa funksiyonlar bilan ko'rsatish mumkin bo'lmagan privilant metodlar va o'zgaruvchanlar, asosiy funksiyonlar va interfeyslar yaratiladi.

Misol:

class Car

{

private:

    string brand; // Maxfiy o'zgaruvchan

    string model; // Maxfiy o'zgaruvchan

    int year;     // Maxfiy o'zgaruvchan

public:

    // Konstruktor

    Car(string b, string m, int y)

    {

        brand = b;

        model = m;

        year = y;

    }

    // Metod - Ma'lumotlarni chiqarish

    void displayInfo()

    {

        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << endl;

    }

    // Maxfiy o'zgaruvchanlarga murojaat qilish uchun

    string getBrand()

    {

        return brand;

    }

    string getModel()

    {

        return model;

    }

    int getYear()

    {

        return year;

    }

};

int main()

{

    // Obyekt yaratish va ma'lumotlarni ekranga chiqarish

    Car myCar("Toyota", "Corolla", 2020);

    myCar.displayInfo();

    // Maxfiy o'zgaruvchanlarga murojaat qilish

    cout << "Brand: " << myCar.getBrand() << ", Model: " << myCar.getModel() << ", Year: " << myCar.getYear() << endl;

    return 0;

}

// NATIJA



Bu misolda, Car sinfi kapsulatsiyani amalga oshiradi. brand, model, va year o'zgaruvchanlari maxfiy (private) bo'lib, shuningdek, getBrand(), getModel(), va getYear() metodlari ularga murojaat qilish uchun ishlatiladi. Bu maxfiy o'zgaruvchanlarga to'xta murojaat qilish bilan, sinf ma'lumotlarini boshqa kodlar bilan himoyalash mumkin.

**Meros olish:**

Meros olish - dasturlashda bir sinfdan boshqasini yaratish jarayonidir. Bunda, bir sinfning barcha xususiyatlari va funksiyalari boshqa sinfga meroslashi mumkin. Merosalarda, asosiy sinf "ota sinf" (base class) deb ataladi, uning meroslash uchun kiritilgan sinfi esa "farzand sinf" (derived class) deb ataladi. Farzand sinf, ota sinfning barcha xususiyatlarini va funksiyalarini o'z ichiga oladi, va shuningdek o'z xususiyatlari va funksiyalarini qo'shishi mumkin.

Misol:

// Ota sinf

class Vehicle

{

public:

    string brand;

    int year;

    Vehicle(string b, int y)

    {

        brand = b;

        year = y;

    }

    void displayInfo()

    {

        cout << "Brand: " << brand << ", Year: " << year << endl;

    }

};

// Farzand sinf

class Car : public Vehicle

{

public:

    string model;

    Car(string b, string m, int y) : Vehicle(b, y)

    {

        model = m;

    }

    void displayCarInfo()

    {

        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << endl;

    }

};

int main()

{

    Car myCar("Toyota", "Corolla", 2020);

    myCar.displayCarInfo(); // Brand: Toyota, Model: Corolla, Year: 2020

    myCar.displayInfo();    // Brand: Toyota, Year: 2020

    return 0;

}

// NATIJA



**Polimorfizm:**

Polimorfizm, dasturlashda bitta funksiyani bir nechta shaklda qayta yozish imkonini ta'minlaydi. Bu, o'zgaruvchilarning sinfning nomini ta'kidlab aniqlanishi va dastur tomonidan avtomatik ravishda sinfning mos funksiyasining chaqirilishi mumkinligini anglatadi. Polimorfizm asosan virtual funksiyalar orqali amalga oshiriladi.

Misol:

// Ota sinf

class Animal {

public:

    virtual void sound() {

        cout << "Animal makes a sound" << endl;

    }

};

// Farzand sinf

class Dog : public Animal {

public:

    void sound() {

        cout << "Dog barks" << endl;

    }

};

// Farzand sinf

class Cat : public Animal {

public:

    void sound() {

        cout << "Cat meows" << endl;

    }

};

int main() {

    Animal \*animal1 = new Dog();

    Animal \*animal2 = new Cat();

    animal1->sound(); // Dog barks

    animal2->sound(); // Cat meows

    return 0;

}

// NATIJA



Bu misolda, Animal sinfi ota sinf deb ifodalangan va uning farzand sinflari Dog va Cat sinflari hisoblanadi. Virtual funksiya sound() dan foydalanish orqali, farzand sinflar o'zlarining xususiyati bo'lgan hayvonning tovushini chiqarishadi. Bu, polimorfizmning bir ko'rinishi hisoblanadi.

Animal \*animal1 = new Dog(); deb yaratilgan obyekt Dog sinfi tarkibida, lekin Animal tipi orqali ishlatiladi. Bu, polimorfizmning bir shakli hisoblanadi. Shuningdek, Animal \*animal2 = new Cat(); deb yaratilgan obyekt esa Cat sinfi tarkibida, ammo Animal tipi orqali ishlatiladi.

animal1->sound(); va animal2->sound(); chaqirilganida, obyektning haqiqiy sinfi (Dog yoki Cat) bilan mos keladigan sound() funksiyasi chaqiriladi. Shu sababli, animal1->sound(); chaqirilganda "Dog barks", animal2->sound(); chaqirilganda esa "Cat meows" natijasi olishimiz mumkin.

**Obyekt massivlari:**

Obyekt massivlari, dasturlashda bir nechta obyektlarning to'plamini ifodalaydi. Bu massivlar biror sinf obyektlarini birlashtirish uchun ishlatiladi. Obyekt massivlarining har bir elementi o'zining sinfi (class) bo'lishi kerak.

Misol:

#include <iostream>

#include <string>

using namespace std;

// Mashina sinfi

class Car

{

public:

    string brand;

    string model;

    int year;

    // Konstruktor

    Car(string b, string m, int y)

    {

        brand = b;

        model = m;

        year = y;

    }

    // Ma'lumotlarni chiqarish metodi

    void displayInfo()

    {

        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << endl;

    }

};

int main()

{

    // Obyekt massivlari

    Car cars[] = {

        Car("Toyota", "Corolla", 2020),

        Car("Honda", "Civic", 2019),

        Car("Ford", "Focus", 2018)};

    // Massivdagi har bir mashinaning ma'lumotlarini chiqarish

    for (int i = 0; i < 3; i++)

    {

        cars[i].displayInfo();

    }

    return 0;

}

// NATIJA



Bu kod misolida, Car sinfida mashina obyektini ifodalash uchun parametrlar bilan konstruktor va displayInfo() metodi bor. main() funksiyasida esa, Car sinfi obyekt massivida turli mashinalar yaratiladi va ulardan har biri uchun displayInfo() metodi chaqiriladi.

Obyekt massivlarini ishlatish, dasturchilarga bir qatorda ko'p obyektlar yaratish imkonini beradi va ularga birlashtirilgan amallar (masalan, har bir obyektga murojaat qilish)ni amalga oshirishga imkon beradi.

Friend funksiya:

#include <iostream>

using namespace std;

class MyClass

{

private:

    int num;

public:

    MyClass(int n) : num(n) {}

    // Friend funksiya

    friend void showNum(MyClass obj);

};

// Friend funksiyasiga murojaat

void showNum(MyClass obj)

{

    cout << "The number is: " << obj.num << endl;

}

int main()

{

    MyClass obj(42);

    showNum(obj); // The number is: 42

    return 0;

}

// NATIJA



Bu misolda showNum() funksiyasi MyClass sinfi ichida ekanligi uchun unga murojaat qilish imkonini beradi. Shuningdek, showNum() funksiyasida MyClass obyekti num o'zgaruvchisiga to'g'ridan-to'g'ri murojaat qilinadi.

**Friend class:**

#include <iostream>

using namespace std;

class MyClass

{

private:

    int num;

public:

    MyClass(int n) : num(n) {}

    // Friend klass

    friend class MyFriend;

};

// Friend klassida murojaat qilish

class MyFriend

{

public:

    static void showNum(MyClass obj)

    {

        cout << "The number is: " << obj.num << endl;

    }

};

int main()

{

    MyClass obj(42);

    MyFriend::showNum(obj); // The number is: 42

    return 0;

}

// NATIJA



Bu misolda MyFriend klassi MyClass klassiga dostlikni ta'minlaydi. showNum() funksiyasi MyClass obyekti num o'zgaruvchisiga murojaat qilish uchun MyFriend klassi ichida to'g'ri yaratilgan. MyFriend::showNum() orqali MyClass obyektiga murojaat qilinadi.

**Static:**

Atributlarni va metodlarni statik qilish (static attributes and methods) dasturlashda obyektlar yaratilmagan holda, sinf darajasiga murojaat qilish imkonini beradi. Bu, sinf obyektini yaratishsiz sinf atributlari va metodlariga murojaat qilish uchun foydalaniladi.

Misol:

#include <iostream>

using namespace std;

class MyClass

{

public:

    // Statik atribut

    static int count;

    // Statik metod

    static void showCount()

    {

        cout << "Count is: " << count << endl;

    }

};

// Statik atributni boshlang'ichlash

int MyClass::count = 0;

int main()

{

    // Statik metodga murojaat

    MyClass::showCount(); // Count is: 0

    // Statik atributga murojaat

    MyClass::count = 5;

    // Statik metodga murojaat

    MyClass::showCount(); // Count is: 5

    return 0;

}

// NATIJA



Bu kod misolida, MyClass sinfida count nomli statik atribut va showCount() nomli statik metod mavjud. main() funksiyasida, statik metod va atributlar sinf darajasiga murojaat qilinishi ko'rsatilgan. count atributi 0 ga boshlang'ichlashtiriladi, keyin unga murojaat qilinadi va qiymatini o'zgartiriladi. showCount() metodini chaqirish natijasida count atributi qiymati chiqariladi.

#include <iostream>

using namespace std;

class MyClass

{

public:

    static int count; // Statik atribut

    MyClass()

    {

        count++; // Har bir obyekt yaratilganda count ni oshirish

    }

};

// Statik atributni boshlang'ichlashtirish

int MyClass::count = 0;

int main()

{

    MyClass obj1;

    MyClass obj2;

    MyClass obj3;

    cout << "Number of objects: " << MyClass::count << endl; // 3

    return 0;

}

// NATIJA



Bu misolda, MyClass sinfiga count nomli statik atribut qo'shilgan. Har bir obyekt yaratilganda count o'zgaruvchisi oshiriladi. Natijada, barcha obyektlarning soni MyClass::count orqali hisoblanadi va ekranga chiqariladi.

**Amallarni qayta yuklash**

Operator funksiyalari, C++ dasturlash tilida ma'lum bir operatorning xususiyatlarini va davranişini o'zgartirish uchun yozilgan funksiyalardir. Bu funksiyalar, sinflar orqali obyekt yaratish, to'plamalar ustida amallar bajarish va boshqa muammo yechishda qulaylik yaratish uchun ishlatiladi.

C++ da avtomatik ravishda ko'p qo'llaniladigan operatorlarga (+, -, \*, /, kabi) mos funksiyalar tayinlanishi mumkin. Bu funksiyalar, standart operatorlarni sinflarning obyektlari uchun ham ishlatilmasini ta'minlaydi.

Quyidagi misol operator funksiyalarni ko'rsatadi:

#include <iostream>

using namespace std;

class Complex

{

private:

    double real;

    double imag;

public:

    // Konstruktor

    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

    // Operator + funksiyasi

    Complex operator+(const Complex &other)

    {

        Complex temp;

        temp.real = real + other.real;

        temp.imag = imag + other.imag;

        return temp;

    }

    // Operator - funksiyasi

    Complex operator-(const Complex &other)

    {

        Complex temp;

        temp.real = real - other.real;

        temp.imag = imag - other.imag;

        return temp;

    }

    // Metod - ma'lumotlarni chiqarish

    void display()

    {

        cout << "Real: " << real << ", Imaginary: " << imag << endl;

    }

};

int main()

{

    Complex c1(3.0, 7.0);

    Complex c2(1.0, 2.0);

    Complex sum = c1 + c2;  // operator + ni chaqirish

    Complex diff = c1 - c2; // operator - ni chaqirish

    cout << "Sum: ";

    sum.display();

    cout << "Difference: ";

    diff.display();

    return 0;

}

// NATIJA



Bu kod misolida, Complex sinfiga + va - operatorlari uchun funksiyalar yozilgan. main() funksiyasida, + va - operatorlarini chaqirish orqali ikkita Complex obyektini qo'shish va ayirmoq amalga oshiriladi.

**Sinfni solishtrish:**

Sinf solishtirgichlari, yoki eng mashhur nomi bilan "comparison operators", sinflarning obyektlarini solishtirish uchun ishlatiladi. Bu solishtirgichlar yordamida sinflar bir-biriga teng, katta yoki kichik bo'lganligini aniqlay oladi.

C++ da quyidagi sinflar solishtirish operatorlarini o'zgartirish uchun funksiyalar bilan yaratiladi:

**==:** Bir-biriga tenglikni solishtirish uchun (equal to)

!=: Bir-biriga teng emasligini solishtirish uchun (not equal to)

<: Kichiklikni solishtirish uchun (less than)

>: Kattalikni solishtirish uchun (greater than)

<=: Kichik yoki teng bo'lishni solishtirish uchun (less than or equal to)

>=: Katta yoki teng bo'lishni solishtirish uchun (greater than or equal to)

Sinflar, yuqoridagi solishtirgichlarni o'zgaruvchilari uchun yozilgan funksiyalar orqali qo'llaniladi. Bu solishtirgichlar obyekt ma'lumotlarini solishtirishda yoki shartlarni tekshirishda ishlatiladi.

Quyidagi misol "Rectangle" sinfiga solishtirgichlarni qo'llashni ko'rsatadi:

#include <iostream>

using namespace std;

class Rectangle {

private:

    int width;

    int height;

public:

    Rectangle(int w, int h) : width(w), height(h) {}

    // Tenglikni solishtirish operatori (==)

    bool operator==(const Rectangle& other) const {

        return (width == other.width) && (height == other.height);

    }

    // Kichiklikni solishtirish operatori (<)

    bool operator<(const Rectangle& other) const {

        return (width \* height) < (other.width \* other.height);

    }

    // Katta bo'lishni solishtirish operatori (>)

    bool operator>(const Rectangle& other) const {

        return (width \* height) > (other.width \* other.height);

    }

    // Ma'lumotlarni chiqarish

    void display() {

        cout << "Width: " << width << ", Height: " << height << endl;

    }

};

int main() {

    Rectangle r1(3, 4);

    Rectangle r2(4, 5);

    if (r1 == r2) {

        cout << "The rectangles are equal." << endl;

    } else if (r1 < r2) {

        cout << "Rectangle 1 is smaller than Rectangle 2." << endl;

    } else if (r1 > r2) {

        cout << "Rectangle 1 is larger than Rectangle 2." << endl;

    }

    return 0;

}

// NATIJA



Bu misolda, "Rectangle" sinfiga ==, <, va > solishtirgichlari uchun funksiyalar qo'shildi. main() funksiyasida, Rectangle obyektini solishtirish uchun bu solishtirgichlardan foydalaniladi.

**Move semantikasi:**

Move semantikasi, C++ dasturlash tilida obyektlarni bir joydan boshqasiga ko'chirish uchun ishlatiladi. Bu, obyektlarning kopyalanishini engellemek va dasturni tezroq va samarali qilish maqsadida yaratilgan.

Move semantikasi, ko'chishni "ustunlikli" amalga oshiradi, ya'ni o'zgaruvchini o'zidan boshqa joyga o'chirib yuboradi, ammo kopyalash vaqt va resurslarni sarflash orqali bajarilmaydi. Bu, klonlash protsessini optimallashtiradi va boshqa obyektlarni klonlashda sarflanadigan vaqt va xotirani ortga qoldiradi.

Move semantikasi, asosan, "rvalue" qilib ataladigan obyektlar uchun foydalaniladi. "rvalue", nomidan ham ko'rinadi, boshqa qiymatlarga assign qilish mumkin bo'lgan, ammo mana shu qiymatga murojaat qilish uchun (masalan, bir funksiya natijasidagi vaqtincha obyektlar yoki turli o'zgaruvchilarning qiymatlari) yuqori darajadagi, o'zgaruvchi (variable) yoki qiymatdir.

Move semantikasining operatori std::move() funksiyasi orqali amalga oshiriladi. Bu funksiya, obyektni "ustunlikli" (move) holatiga olib keladi, deb o'ylangan rvalue sifatida yuboradi.

Quyidagi misol, move semantikasining qanday ishlaydiganligini ko'rsatadi:

#include <iostream>

using namespace std;

class MyObject

{

public:

    MyObject() { cout << "Constructor called!" << endl; }

    MyObject(const MyObject &other) { cout << "Copy constructor called!" << endl; }

    MyObject(MyObject &&other) noexcept { cout << "Move constructor called!" << endl; }

};

int main()

{

    MyObject obj1;              // Constructor called!

    MyObject obj2 = obj1;       // Copy constructor called!

    MyObject obj3 = move(obj1); // Move constructor called!

    return 0;

}

// NATIJA



Bu misolda, obj1 dan obj2 ga nusxa olishda kopiya konstruktori (MyObject(const MyObject& other)) chaqiriladi. Keyin, obj1 dan obj3 ga "ustunlikli" (move) konstruktori (MyObject(MyObject&& other)) orqali ko'chiriladi. Move konstruktori noexcept qo'llanib, bundan muhakkak sarflanmaydigan (noexcept) funksiyalar bilan amalga oshiriladi.

**Xulosa**

Men Oyekt haqida umumiy tushuncha oldim, Obyekt va class tushunchasi ular qayerlarda foydalanishi va ularni ishlatish qulayligini bilib oldim. Meros olish va class ichida funcsiyalar ishlatishni misollar orqali tushinib oldim.

**Foydalanilgan adabiyotlar roʻyxati**

1. Клейнберг Дж., Тардос Е. К48 Алгоритмы: разработка и применение. Классика Computers Science / Пер. с англ. Е. Матвеева. — СПб.: Питер, 2016. — 800 с.: ил. — (Серия «Классика computer science»).

2. To‘rayev H.T. Matematik mantiq va diskret matematika.: Oliy ta’lim muassasalarining talabalari uchun darslik: 11 jildlik. H.T. To‘rayev, 1. Azizov; H.T. To'rayevning umumiy tahriri ostida; O‘zR Oliy va o‘rta-maxsus ta’lim vazirligi. - Toshkent: Tafakkur-Bo‘stoni, 2011. - 288 bet

3. Ахо, Альфред, В., Хопкрофт, Джон, Ульман, Джеффри, Д. Структуры данных и алгоритмы.: Пер. с англ.: Уч. Пос. – М. Издательский дом «Вильямс», 2000. – 384 с.: ил. – Парал. Титю англ.

4. Пышкин Е.В. Структуры данных и алгоритмы: реализация на C/C++. - СПб.: ФТК СПБГПУ, 2009.- 200 с., ил.

5. Овсянников, А. В. Алгоритмы и структуры данных : учебно- методический комплекс для специальности 1-31 03 07 «Прикладная информатика (по направлениям)». Ч. 1 / А. В.

6. Домнин Л. Н. Элементы теории графов: учеб. Пособие / Л. Н. Домнин. – Пенза: Изд-во Пенз. Гос. Ун-та, 2007. – 144 с.