5-MA’RUZA

DO’ST FUNKSIYALAR VA SINFLAR.

ANONIM OBYEKTLAR. LOKAL SINFLAR

DO‘ST FUNKSIYALAR VA SINFLAR

Do‘st sinflar va do‘st funksiyalardan foydalanish. Do‘st funksiya - bu sinf a'zolariga xuddi shu sinf a'zosidek kiradigan funksiya. Boshqa barcha jihatlarda do‘stlik funksiyasi oddiy funksiya kabidir. Bu oddiy funksiya yoki boshqa sinfning metodi bo‘lishi mumkin. Do‘st funksiyani e'lon qilish uchun sinf do‘sti qilmoqchi bo‘lgan funksiya prototipi oldidagi **friend** kalit so‘zidan foydalaning. Buni sinfning public yoki private qismida e'lon qilishingiz muhim emas.

```
class Value
{
private:
    int m_value; public:
    Value() { m_value = 0; }
    void add(int value) { m_value += value; }
```

```
    // Value sinfida do‘st reset() funksiyasini e'lon qilish
    friend void reset(Value &val1);
    };

    // reset() funksiyasi endi Value sinfining do‘stidir
    void reset(Value &val)
    {
        // Endi Value sinfining xususiy a'zolariga kirish huquqiga
egamiz     val.m_value = 0;
    }

    int main()
    {
        Value one;      one.add(4);
    // m_value = 4      reset(one);
    // m_value = 0
        return 0;
    }
```

Bu yerda Value sinfining obyektini qabul qiluvchi va m_value qiymatiga 0 ni o‘rnatuvchi **reset()** funksiyasini e’lon qildik. reset() Value sinfining a’zosi bo‘lmagani uchun reset() funksiyasi odatda kirish huquqiga ega bo‘lmaydi. Biroq, bu funksiya Value sinfi bilan do‘st bo‘lganligi sababli, u Value xususiy (private) a'zolariga kirish huquqiga ega.

DO‘ST SINFLAR VA DO’ST METODLAR

Do‘st sinflar. Bir sinf boshqa sinf bilan do’st bo‘lishi mumkin. Bu birinchi sinfning barcha a'zolariga ikkinchi sinfning xususiy a'zolariga kirish huquqini beradi(-listing).

```
#include <iostream> using namespace std;
class Values
{
private:
```

```
    int m_intValue;
double m_dValue;
public:
    Values(int intValue, double dValue)
    {
        m_intValue = intValue;
        m_dValue = dValue;
    }
    // Display sinfini Values sinfi ishlatadi
friend class Display;
};

class Display
{
private:
    bool m_displayIntFirst;

public:
```

```
        Display(bool      displayIntFirst)    {    m_displayIntFirst =
displayIntFirst; }

        void displayItem(Values &value)
        {
            if (m_displayIntFirst)
                cout << value.m_intValue << " " << value.m_dValue <<
'\n';          else // yoki oldin doubleni chop
    eting
                cout << value.m_dValue << " " << value.m_intValue <<
'\n';
        }
    };
    int main()
    {
        Values      value(7,      8.4);
    Display             display(false);
    display.displayItem(value);
    return 0;  }
```

Do‘st metodlar. Butun sinfni do‘st qilish o‘rniga, faqat sinfning ba'zi metodlari do‘st qilishimiz mumkin. Ularning e’lon qilinishi odatdagi do‘st funksiyalarga o‘xshaydi, faqat boshida **ClassName::** bilan qo‘shilgan metod nomi bundan mustasno (masalan, **Display::displayItem()**).

Display::displayItem() metodini Values sinfiga do’st qilish uchun avvalgi misolimizni qayta tuzamiz. Quyidagi –listingda berilganlarni bajarash mumkin edi.

```
#include <iostream> using
namespace std;
class Display; // Display sinfini e'lon qilish
class Values
{
private:
 int     m_intValue;
double  m_dValue;
public:
 Values(int intValue, double dValue)
```

```
    {
        m_intValue = intValue;
        m_dValue = dValue;
    }

    // Display::displayItem() metodini Values sinfi ishlatadi
    friend void Display::displayItem(Values& value);
    // xato: Values Display sinfining to‘liq ta'rifini ko‘rmaydi
};

class Display
{
private:
    bool m_displayIntFirst;

public:
    Display(bool displayIntFirst) { m_displayIntFirst =
displayIntFirst; }
```

```
    void displayItem(Values &value)
    {
        if (m_displayIntFirst)
            cout << value.m_intValue << " " << value.m_dValue <<
'\n';    else
            cout << value.m_dValue << " " << value.m_intValue <<

'\n';
    }
   };
```

Biroq, bu ishlamaydi. Metodni sinfga do‘st qilish uchun kompilyator do‘st metod aniqlangan sinfning to‘liq ta'rifini (faqat uning prototipini emas) ko‘rishi kerak. Kompilyator ketma-ket kod satrlarini ko‘rib chiqayotib, Display sinfining to‘liq ta'rifini ko‘rmaganligi sababli, lekin uning metodi prototipini ko‘rishga

muvaffaq bo‘lganligi sababli, u do‘st sinfga ushbu metodning ta'rif satrida xato borligini ko’rsatadi.

Display sinfining ta'rifini Values sinfi ta'rifidan yuqoriga ko‘chirishga harakat qilishingiz mumkin:

```
#include <iostream> using
namespace std;
class Display
{
private:
 bool m_displayIntFirst;

public:
 Display(bool displayIntFirst) { m_displayIntFirst = displayIntFirst; }

 void displayItem(Values &value) // Xato: Kompilyator Values sinfini tanimaydi
 {
      if (m_displayIntFirst)
            cout << value.m_intValue << " " << value.m_dValue <<
```

```
'\n';
        else
            cout << value.m_dValue << " " << value.m_intValue <<
'\n';
    } };

    class Values {
    private:
     int      m_intValue;
    double   m_dValue;
    public:
     Values(int intValue, double dValue)
     {
          m_intValue = intValue;
          m_dValue = dValue;
```

```
 }
friend void Display::displayItem(Values& value);
};
```

Biroq, hozir endi boshqa muammo paydo bo‘ladi. Display::displayItem() metodi parametr sifatida Values sinfining obyektiga havolani ishlatgani uchun va faqat Display ta'rifini Values ta'rifidan yuqoriga o‘tkazganimiz uchun, kompilyator Values nima ekanligini bila olmaydi.

Yuqoridagi muammolar juda oson hal qilinishi mumkin. Birinchidan, Values sinfi uchun oldindan e’lon qilishdan foydalanamiz.

Ikkinchidan, Display::displayItem() metodining ta'rifini Display sinfidan tashqariga o‘tkazamiz va Values sinfining to‘liq ta'rifidan keyin joylashtiramiz.

```
#include <iostream> using
namespace std;
class Values; // Values sinfni oldindan e'lon qilish

class Display
{
private:
```

```
    bool m_displayIntFirst;

  public:
    Display(bool displayIntFirst) { m_displayIntFirst = displayIntFirst; }

    void displayItem(Values &value); // bu qator uchun yuqoridagi
e’lon qilish talab qilinadi
  };

  class Values
  {
  private:
```

```
 int     m_intValue;
double  m_dValue;
public:
 Values(int intValue, double dValue)
 {
```

```
        m_intValue = intValue;
        m_dValue = dValue;
 }

 friend void Display::displayItem(Values& value);
};

void Display::displayItem(Values &value)
{
 if (m_displayIntFirst)
        cout << value.m_intValue << " " << value.m_dValue <<
'\n';  else
        cout << value.m_dValue << " " << value.m_intValue << '\n';
}

int main()
```

```
{
    Values value(7, 8.4);
    Display display(false);
    display.displayItem(value);
    return 0;
}
```

ANONIM OBYEKTLAR

Ba'zi hollarda, C ++ tilida o‘zgaruvchi vaqtinchalik kerak bo‘lishi mumkin. Masalan:

```
#include <iostream>
int add(int a, int b)
{
```

```
    int result = a + b;
    return result;
}
int main()
{
    cout << add(4, 2);
    return 0;
}
```

add() funksiyasi result o‘zgaruvchisidan vaqtinchalik o‘zgaruvchi sifatida foydalanadi. Uning alohida roli yo‘q, funksiya faqat qiymatni qaytarish uchun foydalanadi.

add() funksiyasini anonim obyekt orqali yozishning oson yo‘li bor. Anonim obyekt nomi bo‘lmagan qiymatdir. Nom yo‘qligi sababli, bu obyektga u yaratilgan joydan tashqarida murojaat qilishning iloji yo‘q. Shuning uchun anonim obyektlar ko’rinish

sohasiga(-bobga qarang) ega bo‘lib, ular bir xil ifoda doirasida yaratiladi, qayta ishlanadi va yo‘q qilinadi.

Mana, yuqoridagi add () funksiyasi, lekin anonim obyekt yordamida:

```
#include <iostream> using namespace std;
int add(int a, int b)
{
    return a + b; // // a + b ifoda natijasini saqlash va
```

```
qaytarish uchun anonim obyekt yaratiladi
    }
    int main()
    {
        cout << add(4, 2);

        return 0;
    }
```

a + b ifodasini baholashda natija anonim obyektga joylashtiriladi. Keyin anonim obyektning nusxasi murojaat qiluvchiga qaytariladi va anonim obyekt yo‘q qilinadi.

Anonim sinf obyektlari. Garchi biz yuqoridagi misollarda faqat asosiy ma'lumotlar turlaridan foydalangan bo‘lsak-da, anonim

obyektlardan sinflar bilan ham foydalanish mumkin. Obyekt nomini ko‘rsatmaslik kifoya:

```
Dollars dollars(7); // sinfning oddiy obyetki
```

Dollars(8); // sinfning anonim obyekti

Yuqoridagi misolda Dollars (8) satri, Dollars sinfining anonim obyektini yaratadi, unga 8 ni initsializatsiyalaydi va keyin uni yo‘q qiladi. Shu nuqtai nazardan, biz ko‘p foyda olmaymiz. Bu foydali bo‘lishi mumkin bo‘lgan misolni (-listing) ko‘rib chiqaylik:

```
#include <iostream> class Dollars
{
private:
    int m_dollars;

public:
    Dollars(int dollars) { m_dollars = dollars; }      int getDollars()
const { return m_dollars; }
};
```

```
void print(const Dollars &dollars)
{
   cout << dollars.getDollars() << " dollars.";
}
int main()
{
    Dollars dollars(7);
    print(dollars);

    return 0;
}
```

Bu yerda main() funksiyasi dollar obyektini print() funksiyasiga o‘tkazadi. Anonim obyektlar yordamida ushbu dasturni soddalashtira olamiz:

```
#include <iostream>
```

```
class Dollars
{
```

```
private:
    int m_dollars;

public:
    Dollars(int dollars) { m_dollars = dollars; }

    int getDollars() const { return m_dollars; }
};

void print(const Dollars &dollars)
{
   cout << dollars.getDollars() << " dollars.";
}
int main()
{
    print(Dollars(7)); // Dollars sinfining anonim obyekti print()
```

metodiga jo’natilmoqda
return 0; }

SINFLARNING BOSHQA SINFLARDAN TASHKIL TOPISHI

Obyekt maydon sifatida. Murakkab sinflarni hosil qilishda oldin uni tashkil etuvchi oddiyroq sinflarni e'lon qilib, keyin esa ularni birlashtirish orqali sinfni hosil qilish maqsadga muvofiq. Masalan, g‘ildirak sinfi, motor sinfi, uzatish korobkasi sinfi va boshqa sinflarni hosil qilib, keyin esa ularni birlashtirish orqali avtomobil sinfini qurish oldimizga turgan masalani yechishni ancha osonlashtiradi.

Boshqa sinflarni o‘z ichiga olgan sinflar. Ba'zi sinflar a'zo o‘zgaruvchilari sifatida boshqa sinflarni o‘z ichiga olishi mumkin. Jimlik bo’yicha, tashqi sinf yaratishda a'zo o‘zgaruvchilar uchun standart konstruktorlar chaqiriladi. Bu konstruktor tanasi bajarilishidan oldin sodir bo‘ladi. Buni quyidagicha ko‘rsatish mumkin:

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

```
class A
{
public:
    A()
    {
        cout << "A\n";
    }
};

class B
{
private:
   A m_a; // B a'zo o‘zgaruvchi sifatida A ni o‘z ichiga oladi

public:
    B()
```

```
{
    cout << "B\n";
}
```

```
};

int main()
{
    B b;
    return 0;
}
```

Dasturni bajarish natijasi:

```
A
B
```

B o‘zgaruvchisi yaratilganda, B() konstruktori chaqiriladi. Konstruktor tanasi bajarilishidan oldin m_a A sinfining standart konstruktorini chaqirib initsializatsiyalanadi. Bu A qiymatini

beradi, keyin boshqaruv B konstruktoriga qaytadi va B konstruktorining tanasi bajarila boshlaydi.

Bu yerda hammasi mantiqiy jihatdan to'g'ri, chunki B() konstruktori m_a o'zgaruvchisini ishlatishni xohlashi mumkin, shuning uchun avval m_a ni initsializatsiyalash kerak.

LOKAL SINFLAR

Sinf blok ichida, masalan, funksiya tomonida tarifilanishi mumkin. Bunday sinf *lokal sinf* deb ataladi. Lokal sinf komponentalariga shu sinf tariflangan blok yoki funksiya tashqarisida murojaat qilish mumkin emas. Lokal sinf statik komponentlarga ega bo'lishi mumkin emas. Lokal sinf ichida shu sinf aniqlangan soniga tegishli nomlari; statik (static) o'zgaruvchilar, tashqi (extern) o'zgaruvchilar va tashqi funksiyalardan foydalanish mumkin. auto xotira turiga tegishli o'zgaruvchilardan foydalanish mumkin emas. Lokal sinflar komponent funksiyalari faqat joylashuvchi (inline) funksiya bo'lishi mumkin.

Quyidagi misol(-listing)da moddiy nuqta sinfi yaratilib, uning ichida Point sinfiga ta’rif berilgan va Point sinfi obyekt maydoni sifatida kelgan:

```
#include <iostream>
```

```
using namespace std;
class FPoint
{
public: //Nuqta
sinfi
class Point
{
public:
    Point(int x1 =0, int y1 =0)
    {
    x = x1; y=y1;
    }
    int GetX() const {return
x;}     int GetY() const {return
y;}     private:     int x;
    int y;
};
```

private:

```
Point p;
double w;

  FPoint(int x1, int y1, double w1):p(x1, y1), w(wl){};

  void show(){ cout<<"koordinata x =
"<<p.GetX()<<endl; cout<<"koordinata y=
"<<p.GetY()<<endl; cout<<"massa *="<<w;
}

};
int main()
{
cout<<"fizik nuqta"<<endl;
FPoint X(1, 2, 5.5);
```

```
X.show();
```

```
int kk; cin>>kk;
return 0;
}
```

Natija:
fizik nuqta koordinata
x= 1 koordinata y = 2
massa w = 5.5

Bu misolda nuqta sinfining hamma elementlari umumiy, lekin dasturda bu sinfdan foydalanib bo‘lmaydi.