



Napredne tehnike programiranja

Vježba 7



Teme



- Prijatelji klase
- Preopterećenje operatora

Prijatelji klase



- Ako želimo da neka funkcija (koja nije članica klase) ili klasa ima pristup private ili protected elementima klase koju kreiramo, možemo joj to dopustiti tako da ju navedemo kao prijateljsku u definiciji klase

Ključna riječ `friend`

- Primjer friend funkcije:

```
class A {  
    int x,y;  
    ...  
    friend void ispisi( const A& s );  
};  
void ispisi( const A& s ) {  
    cout<< s.x << " " << s.y << endl;  
}
```

friend funkcija može pristupiti
privatnim elementima klase

Ključna riječ `friend`



- Kao prijateljski objekt se može navesti cijela druga klasa, pa npr. sve funkcije klase `stack_manager` imaju pristup privatnim dijelovima klase `stack`

```
class A{  
    ...  
    friend class B;
```

- ili samo neka funkcija druge klase

```
class A{ ...  
    friend void B::ispis();
```

Preopterećenje operatora



- Za svaku klasu možemo definirati operatore koji će biti prilagođeni toj klasi
- Operator se definira kao funkcija sa ključnom riječi `operator` i oznakom tog operatora
- Primjer:

```
Vektor a,b,c;
```

```
c=a+b;
```

Operatori koje smijemo preopteretiti



+	-	*	/	%	^
&		~	!	,	=
<	>	<=	>=	++	—
<<	>>	==	!=	&&	
+=	-=	*=	/=	%=	^=
&=	=	< =	> =	[]	()
→	→*	new	new []	delete	delete []

Operatori koje ne smijemo preopteretiti



::	.*	.	?:
----	----	---	----

Pravila

- Nije moguće kreiranje novih operatora spajanjem simbola (na primjer `**`).
- Možemo preopteretiti samo već postojeće operatore.
- Barem jedan operand mora biti tipa klase (ili enumeracije): drugim riječima, ne možemo promijeniti ponašanje operatora na ugrađenim tipovima:
- `int operator+(int, int); // Greška.`

Primjer

```
class Tocka{
    private:
        float x,y;
    public:
        Tocka () {
            x=0;
            y=0;
        }
        Tocka(float a, float b){
            x=a;
            y=b;
        }
};

int main(){
    Tocka A, B(1,2),C;
    return 0;
}
```

Preopterećenje operatora



- Kako zbrojiti dvije točke?

```
Tocka a(2,3), b(2,2), c;
```

- Možemo napisati funkciju za zbrajanje

```
Tocka zbroji(Tocka a, Tocka b) {  
    Tocka temp( a.x+b.x, a.y+b.y) ;  
    return temp;  
}
```

- Ovako pozivamo zbrajanje:

```
c=zbroji(a,b);
```

Preopterećenje operatora



- Kako zbrojiti dvije točke?

```
Tocka a(2,3), b(2,2), c;
```

- Možemo napisati funkciju za zbrajanje unutar klase

```
Tocka::Tocka zbroji( Tocka b) {  
    Tocka temp( x+b.x, y+b.y);  
    return temp;  
}
```

- Ovako pozivamo zbrajanje:

```
c=a.zbroji(b);
```

Preopterećenje operatora



- Kad želimo za neku klasu definirati operator za tu klasu napišemo funkciju čije se ime sastoji od riječi **operator** i oznake operatora
- Npr: **operator+()**

Preopterećenje operatora

- Kako zbrojiti dvije točke?

```
Tocka a(2,3), b(2,2), c;
```

- Možemo napisati funkcijski član `operator+`

```
Tocka operator+(const Tocka& B) { //globalna funkcija
```

```
    Tocka Tmp;
```

```
    Tmp.x=this->x+B.x;
```

```
    Tmp.y= this->y+B.y;
```

```
    return Tmp;
```

```
}
```

Ovako pozivamo zbrajanje:

```
c=a+b; //može i c=a.operator+(b);
```

Operator može i ne mora biti član klase



- Operator član klase prima implicitni parametar `this` koji je vezan uz prvi operand.
- Aritmetički i relacijski operatori se obično definiraju izvan klase, dok se operatori pridruživanja redovito implementiraju kao funkcije članice.

```
//lijevi operand je dan implicitnim this  
pokazivačem
```

```
Tocka& Tocka::operator+=(const Tocka&);
```

```
// Nije članica klase -> ima dva parametra  
Tocka operator+(const Tocka&, const Tocka&);
```

Operator može i ne mora biti član klase



```
class Tocka{  
friend Tocka operator+(const Tocka& A, const Tocka& B);  
};  
  
Tocka& Tocka::operator+=(const Tocka& B) { //clan klase  
    x=x+B.x;  
    y=y+B.y;  
}  
  
Tocka operator+(const Tocka& A, const Tocka& B){//nije član  
    Tocka Tmp;  
    Tmp.x=A.x+B.x;  
    Tmp.y=A.y+B.y;  
    return Tmp;  
}
```

Preopterećenje operatora <<



Konzistentnost s IO bibliotekom zahtjeva da operator <<

- Uzima `ostream&` kao prvi parametar;
- Uzima referencu na konstantan objekt tipa klase kao drugi parametar;
- Vraća referencu na svoj `ostream` parametar.
- Operator << i >> moraju uvijek biti implementirani izvan klase.

Preopterećenje operatora <<

Operator ispisa dohvaća privatne podatke klase pa stoga mora biti deklariran kao **friend**.

```
class Tocka{
    ...
    friend ostream& operator<<(ostream &izlaz, const Tocka &T);
};

ostream& operator<<(ostream &izlaz, const Tocka &T){
    izlaz<<"x:"<<T.x<<" y:"<<T.y<<endl;
    return izlaz;
}
```

Preopterećenje operatora >>

Operator ispisa dohvaća privatne podatke klase pa stoga mora biti deklariran kao **friend**.

```
class Tocka{
    ...
    friend istream& operator>>(istream &ulaz, const Tocka &T);
};

istream& operator>>(istream &ulaz, const Tocka &T){
    ulaz>>T.x>>T.y;
    return ulaz;
}
```

Preopterećenje operatora <



```
class Brojac{
    int a;
public:
    Brojac(): a(0){}
    int daJ(){return a;}
    int operator++(int) { //stavljjen dummy arg
        return a++
    }
    int operator++(){return ++a;}
    bool operator<(Brojac y){return a<y.a;}
};

int main(){
    Brojac x,y;
    if(x<y) cout<<"tocno!!"<<endl;
```

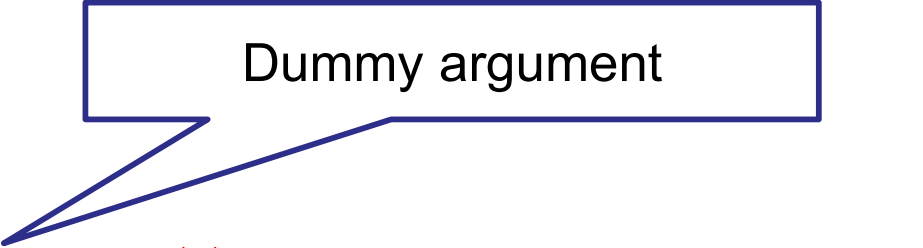
Preopterećenje operatora



- Neki unarni operatori mogu se naći ispred i iza argumenta (npr `++a` i `a++`)
- Za svaki od tih oblika (prefiks i postfix) pišemo posebnu funkciju
- Te funkcije se razlikuju samo po tome što što postfix operator ima nepostojeći argument (dummy argument)

Preopterećenje operatora ++

```
class Brojac{
    int a;
public:
    Brojac(): a(0){}
    int daJ(){return a;}
    Brojac operator++(int) { //postfix ++ - dummy arg
        Brojac temp=*this;
        a++;
        return temp;
    }
    Brojac operator++(){ //prefix operator ++
        ++a;
        return *this;
    } //prefix
```



A diagram consisting of a rectangular box with a purple border containing the text "Dummy argument". A purple line extends from the bottom-left corner of the box, pointing to the `int` parameter in the `operator++(int)` function signature of the `Brojac` class.