Милан Чабаркапа

ПРЕКЛАПАЊЕ ОПЕРАЦИЈА У C++

Са малим изузецима већина операција језика C++ може добити специјално значење које се односи на објекте било које класе. За уграђене типове значење операције се не сме променити. На пример, класа која садржи листу може операцији + дати ново значење: додавање елемента листи. Када је преклопљена операција ни једно од њених основних својстава нема утицаја, јер се уводи потпуно нова операција која се односи на објекте нове класе.

Да би се преклопила операција, треба дефинисати шта она значи у односу на класу, над чијим се објектима примењује. Ради тога се креира специјална функција операције (*operator function*), која дефинише дејство операције.

Функција-операције, која је члан класе, има следећу основну форму:

```
tip_rezultata ime_klase :: operator@(spisak parametara)
  { telo funkcije }
```

Овде симбол @ представља знак конкретне операције (на пример, +, −, =, ++, −− итд) која се извршавањем функције реализује другачије него што је уобичајено. Тип резултата који враћа функција је обично исти као и класе којој функција припада, мада се може и разликовати. Операторска функција се може реализовати као функција-члан или пријатељска функција класе.

Анализирајмо, на примеру, како се реализује преклапање операције. Посматраћемо програм којим се креира класа **vector**, у којој су дефинисане операције + и =. Класа **vector** дефинише вектор у Декартовом координатном систему. Операција сабирања вектора се реализује сабирањем одговарајућих координата:

```
#include <iostream.h>
class vector
  {
    int x,y;  // 2 koordinate vektora
  public:
    vector operator+(vector t);     // preklapanje operacije +
    const vector&operator=(const vector &t);    //preklapanje operacije =
    void show();    // prikazuje koordinate tacke
    void assign(int x1, int y1);
  };
vector vector ::  operator+(vector t)
```

```
  {
  vector temp;    // lokalni vektor
  temp.x=x+t.x;  temp.y=y+t.y;
  return temp;
  };
const vector& vector ::  operator=(const vector &t)
  {
  if (&t==this) return *this;    // provera da li se dodeljuje samom sebi
  x=t.x;
  y=t.y;
  return *this;
  };
void vector ::  show()
  {  cout << "x= " << x << ", y= " << y << endl;  };
void vector ::  assign(int x1, int y1)
  {
  x=x1; y=y1;
  };
void main()
  {
  vector a, b, c;
  a.assign(5,5);
  b.assign(10, 10);
  a.show();
  b.show();
  c=a+b;    // koriscenje preklopljenog operatora
  c.show();
  c=a+b+c;
  c.show();
  c=b=a;
  c.show();
  b.show();
  // Niz vektora
  cout << "======================\n";
  vector v[5], vs;
  for (int i=0; i<5; i++)
    {
    v[i].assign(i,i+1);
    cout << "Vektor v[" << i << "]=\n";
    v[i].show();
    }
  vs.assign(0,0);    // inicijalizuje vektor zbir a
  cout << "Zbir vektora ->\n";
  for (i=0; i<5; i++)
    vs=vs+v[i];    // kompajler interpretira kao vs.operator+(v[i])
  vs.show();
  }
```

Извршавањем програма исписује се:

```
x= 5, y= 5
x= 10, y= 10
x= 15, y= 15
```

```
x= 30, y= 30
x= 5, y= 5
x= 5, y= 5
======================
Vektor v[0]=
x= 0, y= 1
Vektor v[1]=
x= 1, y= 2
Vektor v[2]=
x= 2, y= 3
Vektor v[3]=
x= 3, y= 4
Vektor v[4]=
x= 4, y= 5
Zbir vektora ->
x= 10, y= 15
```

Из програма можемо видети да функција:

`vector` **`operator+`**`(vector t)`

иако реализује бинарну операцију +, има само један параметар **t**. То је због тога што када компајлер у току превођења програма наиђе на операцију + над објектима типа **vector**, не реализује (нити може) уобичајено сабирање, већ у случају, на пример израза **a+b**, „погледа“ да ли класа по чијем шаблону је креиран објекат **a** (класа **vector**) има дефинисану операторску функцију за операцију +. Ако има, тада **a+b** интерпретира као позив функције-члана:

`a.`**`operator+`**`(b)`

којим се израчунава **a+b** (слика 1). Овде објекат **a** при позиву функције **operator+(b)** предаје **this**-показивач (показивач на самог себе), па се функција:

```
vector vector ::  operator+(vector t)
  {
  vector temp;    // lokalni vektor
  temp.x=x+t.x;    // x , y su u pozivu a.operator+(b)
  temp.y=y+t.y;    // podaci objekta a, a t.x i t.y podaci objekta b
  return temp;
  };
```

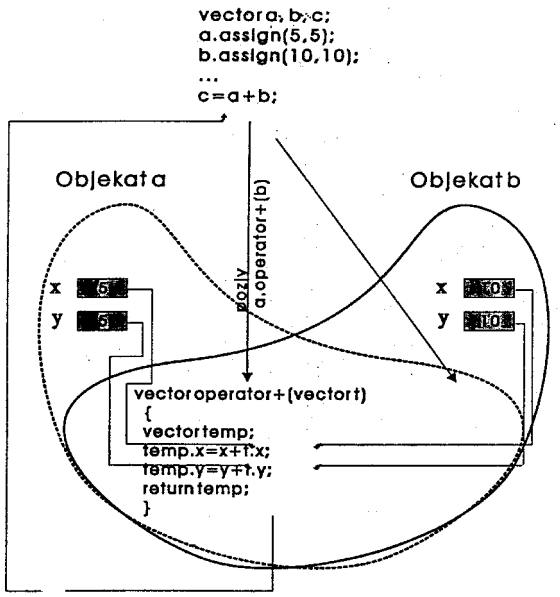
реализује тако што се **temp.x=x+t.x** и **temp.y=y+t.y** извршавају као:

`temp.x=`**`this`**`->x+t.x;`

`temp.y=`**`this`**`->y+t.y`

Овим се обезбеђује да вредности **x** и **y** буду подаци објекта **a** који позива функцију и који је леви операнд операције +.

Дакле, увек леви операнд бинарне операције (овде **a**) активира операторску функцију са једним аргументом – десним операндом (то је овде стварни аргумент **b**), предајући јој показивач **this** (на самог себе) тако да подаци чланови, који се користе у функцији, потичу од објекта (овде **a**) који позива функцију (слика 1).



Сл. 1

Ако је **a+b** израз са десне стране оператора додељивања, на пример:

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

вредност коју враћа **a.operator+(b)** постаје аргумент преклопљеног оператора додељивања =:

```
c.operator=(a.operator(b))
```

којим се објекту **c** додељује вредност аргумента.

У овом примеру је битно да је вредност коју враћа функција **operator+()** типа **vector**. То омогућава коришћење израза типа `a+b+c`. Преклопљена операција `+` не мења вредности својих операнда, док преклопљена операција `=` модификује операнд који се налази са његове леве стране.

Операција додељивања је у свакој класи подразумевајуће дефинисана као поелементно копирање. Она се позива сваки пут, када се једном објекту додељује вредност другог. Ако класа садржи поља, којима се меморија додељује динамички, неопходно је креирати функцију-члан класе која реализује додељивање. Да би се сачувала семантика додељивања C++, функција-операција треба да врати референцу на објекат, који је активирао, и у својству параметра треба да има референцу на објекат који се додељује. У класи vector операција додељивања се реализује на следећи начин:

```
const vector& vector ::  operator=(const vector &t)
  {
  if (&t==this) return *this;    // provera da li se dodeljuje samom sebi
  x=t.x;
```

```
y=t.y;
return *this;
};
```

Префикс **const** на почетку заглавља функције спречава покушај промене вредности објекта на кога реферише функција.

Пошто операција додељивања враћа референцу на објекат дате класе, могуће је коришћење низа таквих операција у једном изразу. Израз

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

се транслира у

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

Преклапање унарних операција ++ и −. Преклапати се могу и унарне операције ++ и −. За реализацију префиксне операције ++ довољно је да се у претходном примеру у опис класе стави декларација:

```
vector &operator++();
```

и опише функција-операција:

```
vector &vector ::  operator++()
  {
  x++; y++;
  return *this;    // vraca vrednost izmenjenog vektora
  };
```

Када се у програму над објектом **c** типа **vector** примени префиксна операција ++:

```
++c;
```

то се интерпретира као позив функције:

```
c.operator++();
```

која објекту **c** додаје јединични вектор и враћа увећани вектор.

Ако се за реализацију унарних операција користе функције-чланови класе, нису потребни јавни параметри функције, јер објекат који активира функцију-операције шаље скривени показивач **this** (на самог себе).

Приметили сте да операторска функција која реализује операцију ++ враћа референцу на објекат који је позвао (у овом примеру **c**). То значи да функцији (оператор − је специјална функција), која враћа референцу може бити додељена вредност (односно вредност се индиректно додељује објекту **c**). Према томе, у складу са стандардом језика C++ може се писати ++c=d. Овакве конструкције нема баш много смисла писати, јер, без обзира на увећање, објекат **c** добија вредност **d**, али стандард ово дозвољава. Може се такође писати низ оператора ++++c, где један оператор ++ враћа референцу на објекат који га је позвао другом оператору ++, и као резултат два пута се реализује увећавање за јединични вектор објекта **c**.

У раним стандардима језика C++ није било могуће разликовати префиксну операцију унарног оператора од постфиксне. Нови стандард даје такву могућност. Функција-члан неке класе **operator@()** одговара унарном префиксном оператору @, док **operator@(int)** одговара постфиксном оператору. При позиву постфиксне верзије функције компајлер јој предаје неку фиктивну целобројну константу, која се унутар функције игнорише – служи само да би се направила разлика у опису префиксне и постфиксне функције-операције.

Да би класа **vector** могла да прави разлику префиксног и постфиксног оператора ++ потребно је да у опис класе ставимо:

```
vector &operator++();  // preklapanje prefiksne operacije ++
vector operator++(int);  // preklapanje postfiksne operacije ++
```

и опишемо функцију-операцију за префиксну операцију:

```
vector &vector ::  operator++()
  {
  x++;
  y++;
  return *this;    // vraca vrednost izmenjenog vektora
  };
```

и опишемо функцију-операцију за постфиксну операцију:

```
vector vector ::  operator++(int)
  {
  // argument funkcije int se ignorise
  vector temp=*this;
  x++; y++;
  return temp;  // vraca vrednost vektora PRE UVECAVANJA
  };
```

Постфиксна операција увећава координате објекта за јединични вектор и враћа вредност објекта пре увећања.

Следећи пример илуструје коришћење префиксног и постфиксног оператора ++. Оператор + се реализује другачије него у претходном примеру коришћењем параметра који је референца са спецификатором **const** – забрањује промену стварног параметра функције. Употреба референце искључује активирање конструктора копирања. **const** иза описа заглавља функције забрањује промену података чланова објекта.

```
#include <iostream.h>
class vector
  {
    int x,y;  // 2 koordinate vektora
  public:
    vector operator+(const vector &t) const;  // preklapanje operacije +
    const vector& operator=(const vector &t)  // preklapanje operacije =
    vector &operator++();  // preklapanje prefiksne operacije ++
    vector operator++(int);  // preklapanje postfiksne operacije ++
```

```
    void show();    // prikazuje koordinate tacke
    void assign(int x1, int y1);
  };
vector vector ::  operator+(const vector &t) const
  {
  vector temp=*this;    // lokalni vektor
  temp.x+=t.x;
  temp.y+=t.y;
  return temp;
  };
const vector& vector ::  operator=(const vector &t)
  {
    if (&t==this) return *this;//  provera da li se dodeljuje samom sebi
  x=t.x;
  y=t.y;
  return *this;
  };
vector &vector ::  operator++()
  {
  x++; y++;
  return *this;    // vraca vrednost izmenjenog vektora
  };
vector vector ::  operator++(int)
  {
  // argument funkcije int se ignorise
  vector temp=*this;
  x++; y++;
  return temp;  // vraca vrednost vektora PRE UVECAVANJA
  };
void vector ::  show()
  {
  cout << "x= " << x << ", y= " << y << endl;
  };
void vector ::  assign(int x1, int y1)
  {  x=x1; y=y1; };
void main()
  {
  vector a, b, c;
  a.assign(5,5);
  b.assign(10, 10);
  a.show();
  b.show();
  c=a+b;  // koriscenje preklopljenih operatora
  c.show();
  c++;    // koriscenje postfiksne operacije
  c.show();
  c++;    // koriscenje postfiksne operacije
  c.show();
  }
```

Извршавањем програма исписује се:

```
x= 5, y= 5
```

```
x= 10, y= 10
x= 15, y= 15
x= 16, y= 16
x= 17, y= 17
```

Преклапање операције индексирања []. Операција индексирања [] се обично преклапа, када класа садржи скуп елемената, за које индексирање има смисла. Ова операција враћа референцу на елемент скупа, обезбеђујући приступ елементу објекта, као да се ради о низу, коришћењем индекса.

Операција индексирања се третира као бинарна операција, где је први операнд – објект класе, а други операнд – целобројни индекс. Операција [] се може дефинисати само као члан класе (касније ћемо видети да се операције преклапања реализују и коришћењем пријатељских функција). Ради илустрације преклапања операције [] у следећем примеру незнатно ће се модификовати класа **vector**:

```
#include <iostream.h>
class vector
  {
    int v[3];    // 3-dim vektor
  public:
    vector() { for (int i=0; i<3; i++) v[i]=0; }   // konstruktor
    vector(const int a[])  // konstruktor
      { for (int i=0;i<3;i++) v[i]=a[i]; };
    vector operator+(vector t);  // preklapanje operacije +
    const vector& operator=(const vector &t);  // preklapanje operacije =
    int& operator[](int i);  // preklapanje operacije []
    void show(void);
  };
vector vector ::  operator+(vector t)
  {
  vector temp=*this;
  for (int i=0; i<3; i++)
    temp.v[i]+=t.v[i];
  return temp;
  };
const vector& vector ::  operator=(const vector &t)
  {
  if (&t==this) return *this;  // provera da li se dodeljuje samom sebi
  for (int i=0; i<3; i++)
    v[i]=t.v[i];
  return *this;
  };
int &vector ::  operator[](int i)
  {
  return v[i];
  }
void vector ::  show(void)
  {
  for (int i=0; i<3; i++)
    cout << v[i] << " ";
  cout << "\n";
```

```
  }
void main()
  {
  int a[3]=1, 2, 3;
  int b[3]=10, 20, 30;
  vector v1(a), v2(b), v3;
  v3=v1+v2;
  // Ispis v3 koriscenjem preklopljene operacije []
  for (int i=0; i<3; i++) cout << v3[i] << " ";
  cout << "\n";
  // Ispis v3 pozivom funkcije-clana
  v3.show();
  // Preklapanje operacije [] uz vracanje reference
  // dozvoljava da se napise:
  v1[0]=100;  // upucuje na v1.v[0] kome se dodeljuje 100
  // v1[i] je poziv funkcije u levom delu operacije dodele:
  // v1.operator[](i)
  // u levom delu se nalaze funkcija i lvalue istovremeno!
  v1[1]=201;
  v1[2]=302;
  v1.show();
  }
```