# Kurs programowania

## Wykład 11

Wojciech Macyna

19 maja 2016

# Przeładowanie nazwy funkcji

Dla C++ cała nazwa funkcji (metody) składa się z właściwej nazwy funkcji i typów argumentów.
Funkcje o tej samej nazwie a różnych listach typów argumentów są dla kompilatora różnymi funkcjami. Kompilator wybiera właściwą funkcję na podstawie listy argumentów.

Przeładowujemy nazwę funkcji a nie samą funkcję.

Różne wersje funkcji muszą różnić się typami argumentów.

# Przeładowywanie operatorów

Można przeładować (zdefiniować dla nowych typów argumentów) prawie wszystkie operatory używane w C++.

## Zasady przeładowywania operatorów

- Można definiować działania tylko dla istniejących operatorów.
- Trzeba zachować konwencjonalną liczbę argumentów (jeśli definiujemy jako metodę w klasie to pierwszym argumentem jest obiekt dla którego wywołujemy operator).
- Nie wolno przedefiniowywać operatorów dla typów systemowych (przeładowanie musi mieć jako argument co najmniej jeden argument typu użytkownika).

Należy nadawać operatorom tylko konwencjonalne znaczenie (zgodne z ich domyślnym działaniem).

# Operatory

## Operatory przypisania =, +=, -=, *=, ...

Operatory te modyfikują obiekt na którym działają.
Dobrze jest zacząć definiowanie operatorów właśnie od nich.
Jako argumenty podajemy referencje do obiektów (stałe jeśli nie zamierzamy ich zmieniać).
Wynik zwracamy jako referencję do lewej strony przypisania (umożliwia to stworzenie sekwencji przypisań).

## Przykład

```
1  Polynomial & Polynomial::operator=(const Polynomial & p)
2  {
3    delete [] coefficients;
4    degree=p.degree;
5    coefficients=new double[degree+1];
6    for(unsigned int i=0; i<=degree; i++)
7      coefficients[i]=p.coefficients[i];
8    return (*this);
9  }
```

# Operatory

## Dwuargumentowe operatory arytmetyczne +,-,*,/,...

Nie modyfikują obiektów które są ich argumentami.
Zwracają obiekt stały aby uniemożliwić błędne podstawienia.

## Przykład

```
1  const Polynomial Polynomial::operator+(const Polynomial & p) const
2  {
3    unsigned int max, min;
4    bool ch = false;
5    if( degree>p.degree ) { max=degree; min=p.degree; ch=true;}
6    else { max=p.degree; min=degree; }
7    Polynomial c(max);
8    for(unsigned int i=0; i<=min; i++ )
9      c.coefficients[i]=coefficients[i]+p.coefficients[i];
10   for(unsigned int i=min+1; i<=max; i++ )
11     if( ch ) c.coefficients[i]=coefficients[i];
12     else c.coefficients[i]=p.coefficients[i];
13   return c;
14 }
```

# Operatory

## Przykład

```
1  const Polynomial Polynomial::operator*(double x) const
2  {
3    Polynomial c(degree);
4    for(unsigned int i=0; i<=degree; i++)
5      c.coefficients[i]=x*coefficients[i];
6    return c;
7  }
```

## Przykład

```
1  friend const Polynomial operator*(double x, const Polynomial & p)
2  {
3    Polynomial c(p.degree);
4    for(unsigned int i=0; i<=p.degree; i++)
5      c.coefficients[i]=x*p.coefficients[i];
6    return c;
7  }
```

# Operatory

## Operatory porównania ==, !=, <, >, <=, >=

Zwracają wartość boolowską.
Operują na stałych argumentach.

## Operatory << i >>

Modyfikują strumienie danych.
Definiowane jako funkcje zaprzyjaźnione.

# Operatory

## Operator dostępu []

Pobiera jeden argumnent - indeks.
Zwraca referencję do danej - umożliwia to użycie po obu stronach podstawienia.

## Przykład

```
1  double & Polynomial::operator[](unsigned int i)
2  {
3    return coefficients[i];
4  }
```

# Operatory

## Operator wywołania ()

Umożliwia zdefiniowanie głównej funkcji w klasie związanej z nazwą obiektu.

## Przykład

```
1  double Polynomial::operator()(double x) const
2  {
3    double v=0.0;
4    for(unsigned int i=degree; i>0; i--)
5      v=x*v+coefficients[i];
6    v=x*v+coefficients[0];
7    return v;
8  }
```

# Przykład użycia

```
1  int main(int argc, char* argv[])
2  {
3    unsigned int n=2;
4    Polynomial a(n), b(n+1), c(0);
5    b[0]=1.0;
6    c = 3 * b;
7    cout << c << endl;
8    cout << c(1.0) << endl;
9    c = b * 2;
10   cout << c << endl;
11   cout << c(1.0) << endl;
12   Polynomial d = a + b + c;
13   cout << d << endl;
14   cout << d(1.0) << endl;
15   cout << d(2.0) << endl;
16   cout << a << endl;
17   cout << a(1.0) << endl;
18   a = b;
19   cout << a << endl;
20   cout << a(1.0) << endl;
21   return 0;
22 }
```

# Podsumowanie

Przeładowanie funkcji umożliwia różne zachowania dla różnych typów argumentów.

Przeładowanie operatorów umożliwia nadanie definiowanym klasom cech ciał matematycznych.
Nie należy jednak przesadzać z przeładowaniem i nadawać operatorom cech działania nezgodnych z ich domyślnym znaczeniem.