Bahasa C++: Operator Overloading

IF2210 Pemrograman Berorientasi Objek

Sumber: Diktat Bahasa C++ oleh Hans Dulimarta

Operator Overloading

- *Function overloading* adalah fasilitas yang memungkinkan nama fungsi yang sama dapat dipanggil dengan jenis dan jumlah parameter (*function signature*) yang berbeda-beda

```
int a, b, c;
float x, y, z;
c = a + b;  /* "fungsi +" di-overload */
z = x + y;
```

- Dengan fungsi `tambah()`:

```
c = tambah(a, b);
z = tambah(x, y);
```

- Prototipe dua fungsi tambah:

```
int tambah(int, int);
float tambah(float, float);
```

Fungsi Operator

- Fungsi operator = fungsi dengan nama "`operator@`" dengan @ diganti simbol operator yang ada
- Dapat dimanfaatkan dalam manipulasi objek dengan menggunakan simbol

```
Matrix A, B, C;
C = A * B;     /* perkalian matriks */

Complex x, y, z;
x = y / z;     /* pembagian bilangan kompleks */

Process p;
p << SIGHUP;   /* pengiriman sinyal dalam Unix */
```

- Harus didefinisikan oleh perancang kelas sebagai fungsi public

Pendeklarasian Fungsi Operator

- Ada 2 cara/bentuk:
  - Sebagai fungsi non-anggota
    - Jika mengakses anggota yang non-public maka fungsi tersebut harus dideklarasikan sebagai friend
    - Jumlah parameter formal = jumlah operan
  - Sebagai fungsi anggota
    - Dideklarasikan di wilayah public
    - Parameter pertama dari operasi harus bertipe kelas tersebut
    - Jumlah parameter formal = jumlah operan – 1

Contoh Deklarasi Fungsi Operator

```
class Matrix {
  public:
    // fungsi-fungsi operator
    friend Matrix operator* (const Matrix&, const Matrix&);
    // ...
};

class Complex {
  // ...
  public:
    Complex operator/ (const Complex&);
    // ...
};

class Process {
  // ...
  public:
    void operator<< (int);
    // ...
};
```

Implementasi sebagai Anggota

- Deklarasi operator<< dan operator>> pada Stack

```
class Stack {
  // ...
  public:
    void operator<< (int);  // untuk push
    void operator>> (int&); // untuk pop
    // ...
};
```

- Definisi fungsi operator:

```
void Stack::operator<< (int item) {
  push(item);
}

void Stack::operator>> (int& item) {
  pop(item);
}
```

Pemanggilan Fungsi Operator Anggota

- Perbandingan nama “operator@” dengan “nama-biasa”

```
void operator<< (int)   <->  void push(int)
void operator>> (int&)  <->  void pull(int&)
```

- Sebagai fungsi anggota, fungsi operator dapat dipanggil dengan 2 cara: (1) dengan kata kunci operator; (2) hanya simbol

```
Stack s, t;
s.push(100);
t.operator<<(500);
t << 500;
```

Implementasi sebagai Non-Anggota

```
class Stack {
  friend void operator<< (Stack&, int);
  friend void operator>> (Stack&, int&);
};
```

bukan const krn dia mengubah stack

```
void operator<< (Stack& s, int v) {
  s.push(v);
}

void operator>> (Stack& s, int& v) {
  s.pop(v);
}
```

Fungsi Operator Non-Anggota

- Dapat dimanfaatkan **hanya** dengan simbol << atau >>, **tidak** sebagai fungsi dengan nama `operator<<` atau `operator>>`

```
Stack s;
s.operator<<(500);  // ERROR
s << 500;  // OK
```

- Error terdeteksi pada saat kompilasi karena kelas Stack tidak memiliki `operator<<`

Fungsi Anggota atau Non-Anggota?

- Implementasi sebagai anggota: “ruas kiri” operator harus berupa objek kelas tersebut
- Jika “ruas kiri” operator bukan bertipe kelas tersebut, operator harus diimplementasikan sebagai non-anggota
- Operator biner yang ingin dibuat komutatif yang salah satu operannya bukan berasal dari kelas tersebut harus diimplementasikan sebagai dua fungsi dengan nama sama (*overloading*)

```
class Stack {
  void operator+ (int); // sebagai anggota
  friend void operator+ (int, Stack&); // sebagai friend
};

void Stack::operator+ (int m) {
  push(m);
}

void operator+ (int m, Stack& s) {
  s.push(m);
}
```

Anggota atau Non-Anggota?

- Operator (biner) yang ruas kirinya bertipe kelas tersebut dapat diimplementasikan sebagai fungsi non-anggota maupun fungsi anggota
- Operator (biner) yang ruas kirinya bertipe lain **harus** diimplementasikan sebagai fungsi non-anggota
- Operator *assignment* =, *subscript* [ ], pemanggilan ( ), dan selektor ->, harus diimplementasikan sebagai fungsi anggota
- Operator yang (dalam tipe standar) memerlukan operan lvalue seperti *assignment* dan *arithmetic assignment* (=, +=, ++, *=, dst) sebaiknya diimplementasikan sebagai fungsi anggota
- Operator yang (dalam tipe standar) tidak memerlukan operan lvalue (+, -, &&, dst.) sebaiknya diimplementasikan sebagai fungsi non-anggota

Manfaat Fungsi Operator

- Operasi aritmatika terhadap objek-objek matematika lebih terlihat alami dan mudah dipahami oleh pembaca program

```
c = a*b + c/d + e;  // lebih mudah dimengerti
c = tambah(tambah(kali(a,b), bagi(c,d)), e);
```

- Dapat menciptakan operasi input/output yang seragam dengan memanfaatkan *stream* I/O dari C++
- Pengalokasian dinamik dapat dikendalikan perancang kelas melalui operator `new` dan `delete`.
- Batas indeks pengaksesan array dapat dikendalikan lewat operator [].

Perancangan Operator Overloading

- *Assignment* **=** dan *address-of* **&** secara otomatis didefinisikan untuk setiap kelas
- Pilihlah simbol operator yang memiliki makna paling mendekati aslinya
- *Overloading* tidak dapat dilakukan pada **`::`** (scope), **`.*`** (member pointer selector), **`.`** (member selector), **`?:`** (arithmetic if), dan **`sizeof()`**
- Urutan presendensi operator tidak dapat diubah
- Sintaks (*arity*: banyaknya operan) dari operator tidak dapat diubah
- Operator baru tidak dapat diciptakan
- ~~Operator **++** dan **--** tidak dapat dibedakan antara postfix dan prefix~~
  - T& T::operator++(); // prefix
  - T T::operator++(...); // postfix
- Fungsi operator harus merupakan member atau paling sedikit salah satu *argument* berasal dari kelas yang dioperasikan

Operator =

- *Assignment ≠ copy constructor*
  - Pada assignment a = b, objek a dan b sudah tercipta sebelumnya
  - Pada copy constructor Stack s = t, hanya t yang sudah tercipta, objek s sedang dalam proses penciptaan

```
Stack& Stack::operator= (const Stack& s) {
/* assign stack "s" ke stack "*this" */
  int i;
  delete[] data; // bebaskan memori yang digunakan sebelumnya
  size = s.size;
  data = new int[size]; // alokasikan ulang
  topStack = s.topStack;

  for (i=0; i<topStack; i++)
    data[i] = s.data[i];

  return *this;
}
```

Return value operator=()

- Perintah *assignment* berantai berikut:

```
int a, b, c;
a = b = c = 4; // aksi eksekusi: a = (b = (c = 4));
```

- *Assignment* berantai dapat diterapkan pada objek dari kelas jika nilai kembali `operator=` adalah reference
  - Fungsi anggota `operator=` harus mengembalikan nilai kembali berupa objek yang sudah mengalami operasi *assignment* (perintah `return *this`)

Anggota kelas minimal

- Perancang kelas “wajib” mendefinisikan empat fungsi berikut:
  - Constructor
  - Copy constructor
  - Operator assignment
  - Destructor

```
class XC {
  public:
    XC(...);                      // ctor
    XC(const XC&);                // cctor
    XC& operator= (const XC&); // assignment
    ~XC();                        // dtor
    //...member public yang lain
  private:
    //...
};
```

Operator []

- Dapat digunakan untuk melakukan subscripting terhadap objek.
- Jika digunakan sebagai subscripting, batas index dapat diperiksa.
- Parameter kedua (index/subscript) dapat berasal dari tipe data apa pun: integer, float, character, string, maupun tipe/kelas yang didefinisikan user.
- Contoh kasus: membuat Map/Dictionary dengan key berupa string sehingga elemen-elemen `Map m` dapat diakses dengan, e.g., `m["apple"]`.