Bahasa C++:
Function Template dan Kelas Generik

IF2210 –Semester II 2024/2025

Sumber: Diktat Bahasa C++ oleh Hans Dulimarta

Function Template

Latar belakang

- Seringkali kita membutuhkan suatu operasi yang sejenis terhadap tipe yang berbeda-beda.
- Contoh: fungsi `min()` dapat diterapkan untuk `int` maupun `float`.

```
int min(int a, int b) {
    return a < b ? a : b;
}

float min(float a, float b) {
    return a < b ? a : b;
}
```

- Untuk setiap tipe yang akan dimanipulasi oleh fungsi `min()`, harus ada sebuah fungsi untuk tipe tersebut.

Bagaimana mengatasinya?

- Trik yang biasa digunakan adalah dengan definisi makro:

```
#define mmin(a,b) ((a) < (b) ? (a) : (b))
```

- Namun makro tersebut dapat memberikan efek yang tidak diinginkan pada contoh *statement* berikut:

```
if (mmin (x++, y) == 0) printf ("...");
```

yang akan diekspansi sebagai:

```
if (((x++) < (y) ? (x++) : (y)) == 0) printf ("...");
```

- Subtitusi makro ≠ pemanggilan fungsi.

Solusi: function template

- Deklarasi: menggunakan *prefix* "`template <class XYZ>`" sebelum nama fungsi. Contoh:

```
template <class T>
T min (T a, T b) {
    return a < b ? a : b;
}
```

- Dipanggil dengan cara:

```
int a, b, c;
c = min <int> (a, b);
float x, y, z;
z = min <float> (x, y);
```

Catatan-catatan (1)

- Banyaknya nama tipe (kelas) yang dicantumkan di antara '<' dan '>' dapat lebih dari satu. Setiap nama tipe harus didahului oleh kata kunci `class`. Contoh:

```
template <class T1, class T2>
```

- Nama tipe yang dicantumkan di antara '<' dan '>' harus tercantum sebagai *function signature*. Contoh:

```
template <class T1, class T2, class T3>
T1 myFunc (T2 a, T3 b) {...} /* error: T1 bukan bagian dari
signature */
```

Catatan-catatan (2)

- Definisi template fungsi dapat disertai oleh definisi ”non-template” dari fungsi tersebut.

```
template <class T>
T min_arr (const T x[], int size) {...} /* template */
Complex min_arr (const Complex x[], int size) {...} /* non-
template */
```

  - Apa yang akan terjadi pada pemanggilan berikut?

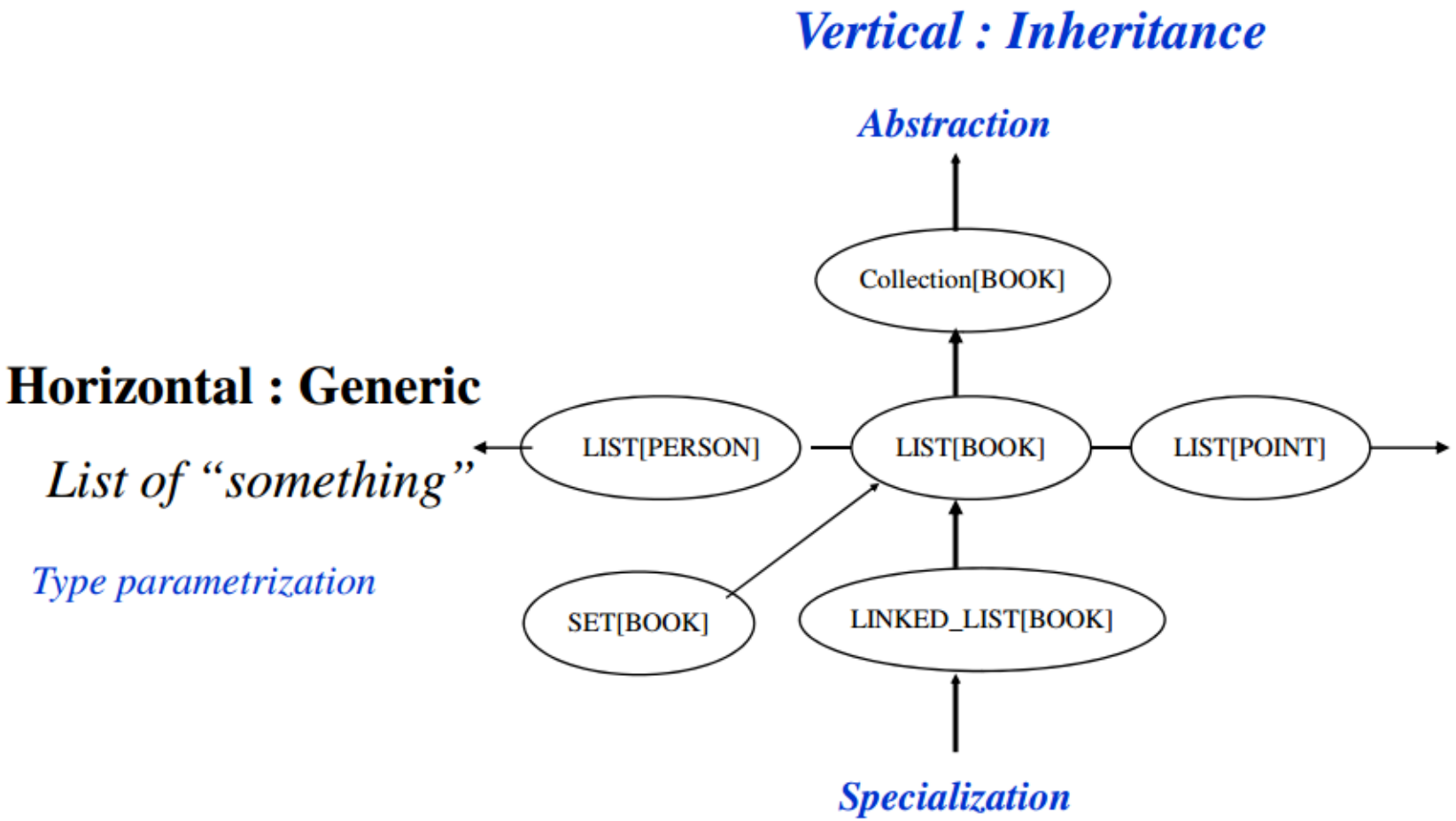
```
Complex c[5], d;
d = min_arr(c, 5);
```

Kelas Generik

Generic Class

- Kelas generic: kelas yang masih “umum”, belum spesifik ketika didefinisikan
- Pada saat deklarasi objek, hal yang umum harus dibuat spesifik
  - Setelah dibuat spesifik, baru bisa dipakai
- Biasanya yang “umum” adalah “type”-nya, dipakai untuk membungkus “operasi” yang sama
- Dalam bahasa C++ menjadi *template*

Generic vs Inheritance



Konsep template pada kelas

- Kelas generik merupakan penerapan konsep template pada kelas.
- Dengan demikian cukup mendefinisikan satu kelas generik (misal kelas `Stack` generik) untuk dapat diinstansiasikan sebagai `Stack` of `int`, `Stack` of `float`, `Stack` of `char`, `Stack` of `Complex`, `Stack` of `String`, dst.
- Untuk menciptakan kelas generik, perancang kelas harus dapat mengidentifikasi parameter-parameter mana yang menentukan sifat kelas.
  - Dalam contoh `Stack`, parameter yang menentukan kelas adalah jenis `int` yang berkaitan dengan data yang disimpan di dalam `Stack`.

Contoh Kelas Generik

- Kelas Point yang masih generic:
  - `GPoint <Numeric>`: kelas Point dengan absis dan ordinat ber-type numerik
  - Saat dideklarasi:
    - `GPoint <integer> P;` maka absis dan ordinat akan ber-type integer
    - `GPoint <float> P;` maka absis dan ordinat akan ber-type float
- Kelas `G_Array <a_type>`: array of elemen bertype `a_type`
  - Pada saat dideklarasikan dapat menjadi: array of `<integer>`, array of `<float>`, array of `<Point>`, ...

Contoh: kelas Stack generik

```
template <class T>
class Stack {
  public:
    // ctor-cctor-dtor
    Stack();                // default ctor
    Stack(int);             // ctor dengan ukuran max stack
    Stack(const Stack&);    // cctor
    ~Stack();

    // services
    void Push (T);          // <=== parameter generik
    void Pop (T&);          // <=== parameter generik
    int isEmpty() const;
    int isFull() const;

    // operator
    Stack& operator= (const Stack&);
    void operator<< (T);    // <=== parameter generik
    void operator>> (T&);   // <=== parameter generik

  private:
    const int defaultStackSize = 500; // ANSI: tidak boleh inisialisasi
    int topStack;
    int size;
    T *data;                // <=== parameter generik
};
```

Catatan-catatan (1)

- Penciptaan objek dilakukan sebagai berikut:

```
Stack<int> a;          // Stack of integer
Stack<double> b(30);   // Stack of double, kapasitas maks = 30
Stack<Complex> c;      // Stack of Complex
```

- Nama `Stack<int>`, `Stack<double>`, ... dapat dipandang sebagai nama tipe baru.
- Definisi fungsi anggota harus dituliskan sebagai fungsi template dan scope yang semula dituliskan sebagai `Stack::` harus dituliskan sebagai `Stack<T>::`.

```
template <class T>
Stack<T>::Stack() {
    size = defaultStackSize;
    topStack = 0;
    data = new T[size];
} /* konstruktor */
```

```
template <class T>
void Stack<T>::Push(T item) {
    // ...
} /* fungsi anggota */
```

Catatan-catatan (2)

- Deklarasi kelas generik maupun definisi fungsi generik dituliskan dalam file header (.h).
  - Ada cara-cara lain, lihat https://stackoverflow.com/questions/495021/why-can-templates-only-be-implemented-in-the-header-file, https://isocpp.org/wiki/faq/templates#templates-defn-vs-decl
- Di luar konteks definisi kelas generik, nama tipe yang dapat digunakan (misalnya oleh fungsi, deklarasi variabel/objek, dsb.) adalah nama tipe hasil instansiasi, seperti `Stack<int>`, `Stack<double>`, atau `Stack<Complex>`. Termasuk di dalam definisi fungsi, contoh:

```
template <class T>
void Stack<T>::Reverse() {
    Stack<T> stemp; // objek lokal yang generik
    // ...algoritma dari Reverse()...
}
```

Spesialisasi kelas generik

- Kita dapat mendefinisikan perilaku spesial untuk kelas generik yang memiliki tipe tertentu sebagai *template parameter*.

```
// class template:
template <class T>
class mycontainer {
    T element;
  public:
    mycontainer (T arg) {element=arg;}
    T increase () {return ++element;}
};

// class template specialization:
template <>
class mycontainer <char> {
    char element;
  public:
    mycontainer (char arg) {element=arg;}
    char uppercase () // perilaku khusus untuk mycontainer <char>
    {
      if ((element>='a')&&(element<='z'))
          element+='A'-'a';
      return element;
    }
};
```