TEMPLATE FUNGSI
DAN
CLASS

PENDAHULUAN

- Overload fungsi memungkinkan membuat fungsi dengan nama sama tetapi parameter berbeda
- Kelemahan overload adalah semua fungsi yang dibuat harus dibuat body fungsi itu
- Template fungsi adalah kerangka fungsi yang memiliki body yang sama tetapi type data berbeda

Overload Fungsi

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

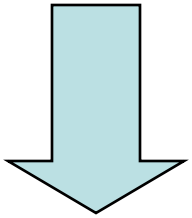
```
int min(float a, float b) {
        return a < b ? a : b;
}
```

TEMPLATE FUNGSI

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

```
template <class T>
void tukar (T &a, T &b) {
        T temp;
        temp = x;
        x = y;
        y = temp;
}
```

- Dengan deklarasi tempate tersebut maka, pendeklarasian fungsi:
  - Void tukar(int &x, int &y);
  - Void tukar(double &x, double &y);
  - Void tukar(long &x, long &y);



DIPERBOLEHKAN

Hal yang harus diperhatikan dalam fungsi template

- Pengenal template (T / TJenis) diawali dengan huruf kapital
- Banyaknya nama tipe (kelas) yang dicantumkan di antara `<' dan `>' dapat lebih dari satu. Dalam penulisannya setiap nama tipe harus didahului oleh kata kunci class.

```
template <class T1, T2> // SALAH: seharusnya
   <class T1, class T2>
   void .....
```

Contoh program

```
template <class T>
void tukar (T &a, T &b) {
        T temp;
        temp = x;
        x = y;
        y = temp;
}

    Void tukar(int &x, int &y);
    Void tukar(double &x, double &y);

    Void main (){
    Double x=17.77;
    Double y =15.83;

    Cout << “x = “ << x << “ y = “  << y <<
    endl;
    Tukar(x, y);
    Cout << “x = “ << x << “ y = “  << y <<
```

```
Int a=55;

Int b =77;


    Cout << “x = “ << x << “ y = “  <<
    y << endl;
    Tukar(x, y);
    Cout << “x = “ << x << “ y = “  <<
    y << endl;


}
```

Overload terhadap fungsi template

```
template <class T>
T maks (T data[], int jml) {
    T n_mak=data[0];
     for (int i=1; i<jml; i++){
         if (data[i] > n_mak){
              n_mak=data[i];
         }
     }
    return(n_mak);
}
template <class T>
T maks (T a, T b) {
              return a > b ? a : b;
}

       Int maks(int, int);
       Double maks(double,
       double);
       Int maks(int [], int);
       Float maks(float [], int);
```

```
       Void main (){
       Int i=5, j=9;

       Cout << maks(I, j) << endl;

       double x=5.55 , y=9,99;

       Cout << maks(I, j) << endl;

       Int a[]={1, 4, 5, 7, 0}

       Cout << maks(a, 5) << endl;

       Double b[]={1.45, 8.34, 4.54, 9.34, 2.34};

       Cout << maks(b, 5) << endl;

       }
```

CLASS TEMPLATE

- konsep template dapat diterapkan juga pada kelas.
- Digunakan untuk mendeklarasikan anggota data dan anggota fungsi class tersebut
- Format template class:

```
Template <class T>
{
        // tubuh fungsi
}
```

- Dengan fasilitas template ini, perancang
- class dapat dengan mudah menciptakan class Stack of integer, Stack of double, Stack of character, Stack of Complex, Stack of Process, Stack of String, dsb
- Untuk menciptakan kelas generik, perancang class harus dapat mengidentikasi parameter-parameter mana yang menentukan sifat kelas.

Contoh program

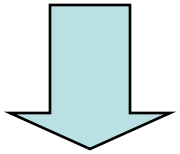
```
template <class Type>
class Stack {
  public:
     Stack(); // default ctor
     Stack (int); // ctor dengan ukuran max stack
     ~Stack();

     void Push (Type); // <=== parameter generik
     void Pop (Type&); // <=== parameter generik
     int isEmpty() const;
     int isFull() const;
     Stack& operator= (const Stack&);
     void operator<< (Type); // <=== parameter generik
     void operator>> (Type&); // <=== parameter generik

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

Deklarasi Objek dari class tersebut

kls-generik < tipe-instansiasi > objek ;



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

Body fungsi anggota

```
template <class Type>
Stack<Type>::Stack ()
{
    size = defaultStackSize;
    topStack = 0;
    data = new TYPE [size];
}
template <class Type>
void Stack<Type>::Push (Type item)
{
// ...
   if (!isFull()) {
      data [topStack] = item;
      topStack++;
    }
// ...
}
```