# Tips C++ : Penggunaan Template

**I Putu Gede Darmawan**
IPGD_BALI@yahoo.com



## Pendahuluan

Dalam pemrograman , terutama yang sangat tergantung pada tipe variable , sering kali kita direpotkan dengan harus membuat fungsi yang berfungsi sama tapi dengan tipe variable berbeda.Untuk itu pada C++ dikeluarkan lah sebuah keyword baru , yaitu *template*. Dengan penggunaan template kita bisa membuat sebuah fungsi yang bisa mendukung segala macam tipe variable , tidak terbatas pada variable yang di definisikan oleh keyword C/C++ tapi juga mendukung penggunaan untuk tipe variable masa depan. Penggunaan template tidak terbatas hanya pada fungsi tapi juga mencakup class ( termasuk juga struct , union) . Secara garis besar penggunaan template dapat dibagi 2 yaitu template pada fungsi (*function template*)dan template pada Class (*class template*). Oke saya rasa kita langsung saja membahas penggunaan template.

## Function Template

Pertama - tama mari kita membahas tentang function template.Untuk itu marilah kita perhatikan contoh berikut.

```
template <class tipe>
void swap(tipe &a,tipe &b)
{
 tipe c;
 c = a;
 a = b;
 b = c;
}
```

Pada contoh diatas terdapat sebuah fungsi *swap* (menukar 2 buah variable) menggunakan template.. Untuk pemanggilan fungsi dilakukan seperti pemanggilan fungsi tanpa template. Untuk itu mari perhatikan penggalan perintah untuk menggunakan fungsi swap.

Contoh template dengan instance tipe int

```
void main(void)
{
 int a,b;
 a = 10;
 b = 20;
 swap(a,b);
 cout << a << “ “ << b << endl;
}
```

Contoh template dengan instance tipe float

```
void main(void)
{
 float a,b;
 a = 10.0;
 b = 20.0;
 swap(a,b);
 cout << a << “ “ << b << endl;
}
```

Contoh template dengan instance struct tTes

```
struct tTes
{
 int a,b;
};

void main(void)
{
 tTes a,b;

 a.a = 10;
 a.b = 20;
 b.a = 30;
 b.b = 40;

 swap(a,b);
 cout << a.a << " " << a.b << " " << b.a << " " << b.b;
}
```

Dapat dilihat , dengan menggunakan template kita telah menghemat waktu untuk menulis fungsi Pada fungsi swap diatas , juga tidak menutup kemungkinan untuk menukar 2 buah kelas.
Pada contoh diatas fungsi tidak akan bekerja jika variable yang ditukar berbeda tipe . Walaupun kekerabatannya cukup dekat . Anda tidak bisa menukar tipe int (16bit) dengan tipe short (16 bit). Ini terkait dengan pendeklarasian template pada fungsi swap.

```
template <class tipe>
void swap(tipe &a,tipe &b)
```

Pada pendeklarasian fungsi swap dapat dilihat bahwa formal parameter a sama dengan formal parameter b . Oleh sebab itu sudah seharusnya tipe pada aktual

parameter pun harus sama. Untuk contoh fungsi swap diatas sebaiknya anda tidak menggunakan *type-casting* untuk mengatasi masalah diatas. Mengingat fungsi swap diatas dikirim by reference . (Yang mana dalam pengiriman fungsi menggunakan reference berkaitan dengan pointer). Tentu saja ukuran dari variable menjadi sangat penting karena kita telah merubah ukuran dari variable (type casting) bisa saja terjadi hasil yang tak terduga . Walupun kadang – kadang memberikan hasil yang benar. Kecuali jika ukuran kedua tipe variable tersebut benar - benar sama , anda dapat menggunakan *type–casting* dalam hal ini.
Untuk mendefiniskan lebih dari satu type pada template anda dapat menggnakan koma sebagai pemisah
contoh :

Template dengan 2 buah tipe

```
template <class tipe1,class tipe2>
```

# Overloading Template Function

Jikalau suatu saat ada sebuah variable yang harus diperlakukan khusus untuk menukarkannya anda dapat mengoverload sebuah template function. Sebagai contoh : misalkan jika anda ingin menukar 2 buah variable float tanpa perlu memperhatikan negatif atau positif . Anda tinggal membuat sebuah fungsi tambahan dengan variable float. Jadi ketika dijalankan compiler akan memprioritaskan dulu pada fungsi yang memiliki formal parameter dan aktual parameter dengan tipe yang sama. Baru kemudian jika tidak ditemukan maka compiler akan membuat sebuah instance dari template function yang telah anda buat. Untuk lebih jelasnya mari kita lihat contoh source code berikut ini .

```
#include <iostream.h>

template <class Tipe>
void swap(Tipe &a,Tipe &b)
{
 Tipe tmp;
 tmp = a;
 a = b;
 b = tmp;
}

void swap(float &a,float &b)
{
 float c;

 a = (a < 0.00)?-a:a;
 b = (b < 0.00)?-b:b;
 c = a;
 a = b;
 b = c;
}


void main(void)
{
 float a,b;
 a = -10.0;
```

```
 b = 20.0;
 swap(a,b);
 cout << a << " " << b;
}
```

# Class Template

Jenis template yang berikutnya ialah *Class template*. Class Template , tidak hanya berlaku untuk pendefinisian template pada class tapi juga berlaku pada pendefinisian struct dan union.Cara pendefinisan sama seperti Funtion template yang beda hanya pada caramembuat instance dari class template tersebut. Untuk lebih jelasnya mari kita perhatikan contoh pendefinisian Class Template.

Pendefinisian Class Template pada sebuah class CMyTemplate

```
template <class MyTipe>
class CMyTemplate
{
 protected :
  MyTipe m_a,m_b;

public   :
 void Done(void)
{
 memset(&m_a,0,sizeof(MyTipe));            // pendeklarasian method
}                                          // didalam kelas

};

//pendeklarasian methos diluar kelas
template <class MyTipe>
void CMyTemplate<MyTipe>::Init(MyTipe a,MyTipe b)
{
 m_a = a;
 m_b = b;
}
```

Pendefinisian Class Template pada sebuah struct tMyStruct

```
template <class MyTipe>
struct tMyStruct
{
 MyTipe a,b;
};
```

Pendefinisian Class Template pada sebuah union _MyUnion

```
template <class MyTipe>
union _MyUnion
{
 MyTipe a;
 int b;
};
```

Yang perlu diperhatikan dari contoh diatas , antara lain perbedaan antara pendefinisian method didalam atau diluar kelas .
Untuk menggunakan sebuah class template sebelumnya kita harus mendefiniskan instance dari template yang akan dibuat. Untuk mendefinisikan instance dari class template dapt dilakukan dengan cara menuliskan tipe data dari instance yang diapit oleh tanda “<” ,”>” .
Contoh :

```
void main(void)
{
 CMyTemplate<short> a; // membuat instance CMyTemplate dengan tipe short
 tMystruct<short> b;  // membuat instance tMyStruct dengan tipe short
 _MyUnion<float> c;  // membuat instance _MyUnion dengan tipe float
}
```

Pada instance CMyTemplate yang dideklarasikan dengan tipe data short maka MyTipe (pada pendefinisian class ) akan menjadi short .Ini berarti variable m_a dan m_b akan bertipe short begitu pula pada struct dan union. Untuk mendefinisikan lebih dari 1 tipe pada class template dapat anda lakukan sama seperti Function Template yaitu dengan menggunakan tanda koma sebagai pemisah .
Contoh :

```
template <class MyTipe1,class MyTipe2>
class CMyTemplate
{
 protected :
  MyTipe1 m_a
  MyTipe2 m_b;

public   :
  void Init(MyTipe1 a,MyTipe2 b)
  {
   m_a = a;
   m_b = b;
  }
};
```

Untuk membuat instancenya tinggal ditambahkan tanda koma .

Contoh :

```
void main(void)
{
 CMyTemplate<short,float> a;
}
```

Jadi sekarang pada Object a , Variabel m_a akan bertipe short sedangkan pada variable m_b akan bertipe float.

Ya saya rasa sekian saja buat penjelasan template kali ini semoga bisa berguna.
Untuk salam pertanyaan kritik dan lain lain langsung saja kirimkan ke alamat email saya .

[IPGD]