

**LAPORAN PRAKTIKUM
PEMROGRAMAN BERORIENTASI OBJEK**

Judul: Template



**DISUSUN OLEH
ILHAM NUR ROMDONI M0520038**

**PROGRAM INFORMATIKA
FAKULTAS MIPA
UNIVERSITAS SEBELAS MARET
2021**

Template

Menggunakan *template programmer* dapat menerapkan konsep *generic class*. Pada modul kali ini akan ditunjukkan contoh membuat *generic function* menggunakan *template*. Sebuah *generic function* dapat menggunakan parameter dengan tipe data yang berbeda-beda.

baca lebih lanjut di chapter 13 ebook OBJECT-ORIENTED PROGRAMMING C++ SIMPLIFIED

Source code 1

```
#include <iostream>
using namespace std;

template <class FUNC>
void show(FUNC par){
    cout << "isi parameter : "<< par << endl;
}

int main(){
    int x = 234;
    float y = 34.56f;
    double d = 3.444456;
    char ch = 'P';
    string s = "Template";
    show(x);
    show(y);
    show(d);
    show(ch);
    show(s);
    return 0;
}
```

Source code 2

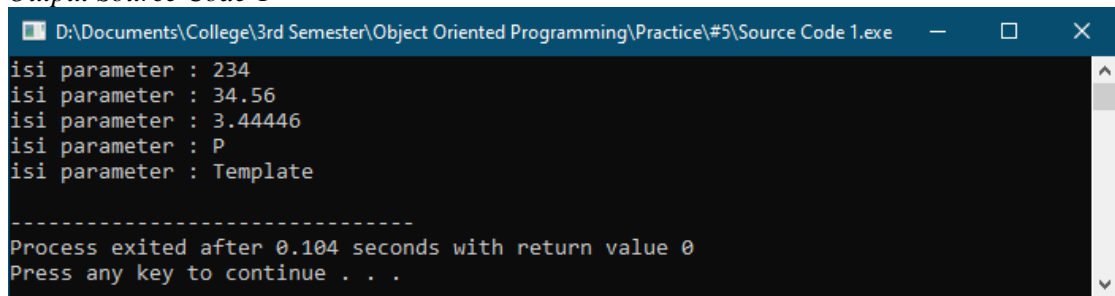
```
#include <iostream>
using namespace std;
#define S 5

template <class T>
T hitungJmlElemenMatrix(T arr[]) {
    T sum = arr[0];
    for(int i=1; i<S; i++) {
        sum = sum + arr[i];
    }
    return sum;
}

int main() {
    int arr[S] = {1, 2, 3, 4, 5};
    cout << hitungJmlElemenMatrix(arr) << endl;
    double arr2[S] = {1.5, 2.1, 3.2, 4.0, 5.3};
    cout << hitungJmlElemenMatrix(arr2) << endl;
    return 0;
}
```

1. Jelaskan apa yang dilakukan pada *Source code 1* dan *Source code 2*! Lampirkan *output* dari kedua *source code* tersebut!

- *Output Source Code 1*



Source code menampilkan tipe-tipe variabel berbeda dengan tipenya menggunakan *function template*. *Function template* dituliskan dengan:

```
void show(FUNC par){  
    cout << "isi parameter : "<< par <<endl;  
}
```

Baris `template<class FUNC>` dituliskan ketika Anda ingin membuat *function template* atau *class template*. Nama `FUNC` adalah nama tipe data *generic*. Ini digantikan oleh tipe data aktual ketika tipe data tertentu digunakan dengan pemanggilan fungsi. Di `main` ketika fungsi-fungsi berikut dipanggil, bergantung pada jenis parameter yang dilewatkan, *function definition* yang terpisah dihasilkan dalam memori.

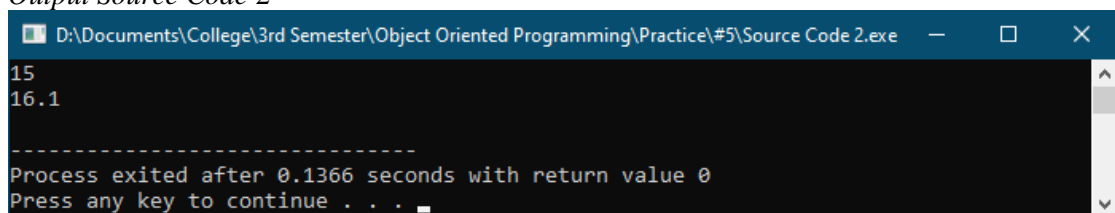
```
show(x);  
show(y);  
show(d);  
show(ch);  
show(s);
```

Misalnya, untuk `show(x)` sebagai tipe `x` adalah `int`, *function definition* dihasilkan sebagai (karena *function template* `void show(FUNC par)`):

```
void show(FUNC par){  
    cout << "isi parameter : "<< par <<endl;  
}
```

Demikian pula untuk *function definition* tipe data lainnya akan dihasilkan. Yaitu untuk 5 tipe data yang berbeda dengan *function template* `show`, lima versi `show` yang berbeda dihasilkan di memori. Jadi *function template* tidak menghemat memori. Ini hanya menyelamatkan kita dari menulis kode berulang yang bekerja untuk berbagai jenis data.

- *Output Source Code 2*



Source code merupakan *function template* untuk menemukan jumlah elemen *array*. *Function template* dibuat untuk mendeklarasikan dan mendefinisikan *array* dari berbagai jenis dan menemukan jumlah elemennya. Misalnya `double arr2[S];` mengeksekusi versi baru *template*, meminta 5 elemen. Angka disimpan dalam *array* dan jumlah ditampilkan oleh panggilan ke fungsi `hitungJmlElemenMatrix(T arr[])` menggunakan `arr2`.

2. Apa manfaat yang bisa didapatkan dengan membuat *generic function* menggunakan *template*?
Memungkinkan kita untuk membuat *function template* yang fungsinya dapat disesuaikan ke lebih dari satu tipe atau *class* tanpa mengulang seluruh kode untuk setiap tipe.