Modul Praktikum

Alpro Lanjut

4

DASAR-DASAR FUNGSI & PROSEDUR

DASAR TEORI

Ketika program yang kita buat semakin kompleks, akan semakin sulit untuk dipahami, terlebih lagi disaat kita ingin mengubah atau menambahkan program pada bagian tertentu. Dengan adanya fungsi kita dapat memecah program menjadi sub-sub program yang lebih sederhana sehingga program tidak menumpuk di satu fungsi utama. Penggunaan fungsi juga sangat efisien karena dapat digunakan atau dipanggil kembali, baik di dalam maupun di luar fungsi itu sendiri.

Dalam bahasa C++, fungsi dan prosedur juga digunakan untuk membagi program menjadi bagian-bagian yang lebih kecil yang dapat digunakan kembali (memiliki makna atau arti kenapa dibuat).

Fungsi	Prosedur
Terdapat tipe data dalam deklarasi seperti int, float, string, dll	Tidak terdapat tipe data dalam deklarasi. Dideklarasikan dengan keyword Void
Mengembalikan nilai (return value)	Tidak mengembalikan nilai (void)

1. Deklarasi Fungsi

Saat mendeklarasikan sebuah fungsi sebaiknya berikan nama fungsi yang sesuai menggambarkan isi dari blok kode fungsi tersebut. Pendeklarasian fungsi dapat kita buat dengan cara seperti berikut.

Deklarasi-Fungsi :

```
Tipe_data nama_fungsi(tipe_data_parameter nama_parameter){
      // isi fungsi
}
```

Fungsi akan dikembalikan sebuah nilai (return value) yang merupakan hasil dari prosesnya. Oleh karena itu, kita harus menentukan terlebih dahulu tipe data untuk nilai yang akan kita kembalikan pada fungsi tersebut.

Deklarasi-Prosedur :

```
void nama_fungsi(tipe_data_parameter nama_parameter){
      // isi fungsi
}
```

Prosedur adalah blok kode yang tidak mengembalikan nilai. Prosedur dideklarasikan dengan void.

Untuk penggunaan parameter pada fungsi dan prosedur bersifat opsional, yaitu boleh ada boleh tidak

2. Jenis Penulisan Fungsi & Prosedur

Pada program, fungsi dapat dibuat sebelum atau sesudah penulisan fungsi main. Bila dibuat sesudah penulisan fungsi main, maka harus mendefinisikan (menuliskan prototype fungsi) di bagian atas program (sebelum fungsi main).

Contoh Fungsi **setelah** Fungsi main() :

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

//Deklarasi fungsi
int tambah(int a1, int a2);

int main(){
       //Pemanggilan fungsi
      int angka1, angka2;
      cout << "Angka Pertama : "; cin >> angka1;
      cout << "Angka Kedua : "; cin >> angka2;
      cout << "Hasil Penjumlahan = " << tambah(angka1, angka2);
}

//Definisi fungsi
int tambah(int a1, int a2){
      int hasil;
      //Perlu ditampung pada variabel
      hasil = a1 + a2;
      return hasil;
}
```

Contoh Fungsi **sebelum** Fungsi main() :

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

//Definisi fungsi
int tambah(int a1, int a2){
      int hasil;
      hasil = a1 + a2;
      return hasil;
}

int main(){
      int angka1, angka2;
      cout << "Angka Pertama : "; cin >> angka1;
      cout << "Angka Kedua : "; cin >> angka2;
      //Pemanggilan Fungsi
      cout << "Hasil Penjumlahan = " << tambah(angka1, angka2);
}
```

Contoh Prosedur **setelah** main():

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

// Deklarasi prosedur
void hasilTambah(int a, int b);

int main()
{
    // Pemanggilan prosedur (tidak perlu ditampung pada variabel)
    hasilTambah(5, 3);
    return 0;
}

// Definisi prosedur
void hasilTambah(int a, int b)
{
    cout << "Hasil penjumlahan " << a << " + " << b << " = " << a + b <<
endl;
}
```

Contoh Prosedur **sebelum** main();

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

// Definisi prosedur
void hasilTambah(int a, int b)
{
    cout << "Hasil penjumlahan " << a << " + " << b << " = " << a + b <<
endl;
}

int main()
{
    // Pemanggilan prosedur
    hasilTambah(5, 3);
    return 0;
}
```

3. Keuntungan menggunakan fungsi & prosedur

1. Mempermudah pemeliharaan code
2. Mengurangi kode duplikat
3. Memudahkan debugging dalam kerja sama tim

4. Parameter dan argumen

Parameter adalah variabel yang digunakan dalam deklarasi fungsi atau prosedur untuk menerima input Sedangkan **argumen** adalah nilai yang dikirimkan saat fungsi atau prosedur dipanggil.

Contoh Parameter & Argumen dalam Fungsi Prosedur:

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

// Fungsi prosedur dengan parameter
void sapaUser(string nama)
{
    cout << "Halo, " << nama << "! Selamat datang di C++." << endl;
}
```

```
int main()
{
    sapaUser("Ifnu"); // "Ifnu" adalah argumen
    return 0;
}
```

5. Jenis-jenis parameter dalam C++

1. Parameter berdasarkan nilai (pass by value)

   Parameter berdasarkan nilai artinya argumen hanya disalin ke dalam fungsi dan tidak mempengaruhi variabel aslinya.

   Contoh:

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

void ubahNilai(int x) {
    x = 10;  // Hanya mengubah nilai dalam fungsi
}

int main() {
    int angka = 5;
    ubahNilai(angka);
    cout << "Nilai angka setelah fungsi dipanggil: " << angka << endl;  // Masih 5
    return 0;
}
```

2. Parameter Berdasarkan Referensi (Pass by Reference)

   Berbeda dengan parameter berdasarkan nilai, parameter berdasarkan referensi menggunakan **&**, sehingga perubahan dalam fungsi akan mempengaruhi variabel aslinya.

   Contoh:

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

void ubahNilai(int &x) {
    x = 10;  // Mengubah nilai asli
```

```
}

int main() {
    int angka = 5;
    ubahNilai(angka);
    cout << "Nilai angka setelah fungsi dipanggil: " << angka <<
endl;  // Menjadi 10
    return 0;
}
```

3. Parameter dengan Default Value

Parameter ini memiliki nilai default jika tidak diberikan argumen saat pemanggilan fungsi.

Contoh:

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

void sapaUser(string nama = "User") {
    cout << "Halo, " << nama << "!" << endl;
}

int main() {
    sapaUser();       // Output: "Halo, User!"
    sapaUser("Ifnu"); // Output: "Halo, Ifnu!"
    return 0;
}
```

6. Variabel Global & Lokal

Dalam mendeklarasikan sebuah variabel ketika membuat sebuah fungsi harus memperhatikan scope variabel yang dibuat. Scope variabel dibagi menjadi dua yaitu **variabel global** dan **variabel lokal**.

- **Variabel Global**

  Variabel yang dideklarasikan di luar fungsi dan bisa digunakan di seluruh bagian program.

- **Variabel Lokal**

  Variabel yang dideklarasikan di dalam fungsi dan hanya bisa digunakan dalam fungsi tersebut.

Contoh Variabel Global:

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

int globalVar = 20;  // Variabel global

void contohFungsi() {
    cout << "Nilai globalVar dalam fungsi: " << globalVar << endl;
}

int main() {
    contohFungsi();
    cout << "Nilai globalVar dalam main: " << globalVar << endl;
    return 0;
}
```

Contoh Variabel Lokal:

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

void contohFungsi() {
    int lokalVar = 10;  // Variabel lokal
    cout << "Nilai lokalVar: " << lokalVar << endl;
}

int main() {
    contohFungsi();
    // cout << lokalVar;  // ERROR! lokalVar tidak bisa diakses di sini
    return 0;
}
```

Jika ada variabel lokal dengan nama yang sama dengan variabel global, maka variabel lokal akan menimpa atau memprioritaskan variabel lokal ini disebut **shadowing variabel**.

Contoh Shadowing Variabel:

```cpp
#include <iostream>
```

```cpp
using namespace std;

int angka = 50;  // Variabel global

void contohFungsi() {
    int angka = 10;  // Variabel lokal dengan nama yang sama
    cout << "Nilai angka dalam fungsi: " << angka << endl;  //
Menampilkan 10
}

int main() {
    contohFungsi();
    cout << "Nilai angka dalam main: " << angka << endl;  // Menampilkan
50
    return 0;
}
```