

PENGENALAN PEMROGRAMAN MODULAR

Dalam sebuah program, seringkali terdapat program yang memiliki persoalan yang kompleks. Sehingga perlu dilakukan pemecahan persoalan yang kompleks menjadi beberapa bagian yang lebih mudah diselesaikan. Ide inilah yang mencetuskan struktur pemrograman modular, yaitu memecah persoalan menjadi sub-sub persoalan yang biasa disebut sub program.

Contoh 1

Terdapat sebuah program yang bertugas untuk **menghitung luas persegi panjang** dan **mencetak hasil perhitungannya ke dalam layar** seperti berikut:

```
int main(){
    int panjang = 10; int lebar = 6;
    int luas = panjang * lebar;
    cout <<Luas : <<luas;
}
```

Lalu, bagaimana jika program tersebut menghitung dua buah persegi panjang dengan ukuran yang berbeda-beda seperti berikut:

```
int main(){
    // Luas Persegi Panjang 1
    int panjang1 = 10; int lebar1 = 6;
    int luas1 = panjang1 * lebar1;
    cout <<"Luas " <<luas1;
}

// Luas Persegi Panjang 2
int panjang2 = 4; int lebar2 = 9;
int luas2 = panjang2 * lebar2;
cout <<"Luas " <<luas2;
}
```

Kode program untuk **menghitung luas** dan **mencetak perhitungan** dituliskan secara berulang

Jika program tersebut dibagi ke dalam sub program berdasarkan tugasnya:

1. Sub program yang bertugas untuk melakukan perhitungan persegi panjang

```
void hitungLuas () {  
    luas = panjang * lebar;  
}
```

2. Sub program yang bertugas untuk mencetak hasil perhitungan ke dalam layar

```
void hitungKeliling () {  
    keliling = 2*(panjang+lebar;  
}
```

Sehingga, pada program utama ketika kode program tersebut akan digunakan kembali hanya perlu dilakukan pemanggilan atau menuliskan nama sub program tempat kode program tersebut dituliskan ketika dibutuhkan. Sehingga program utama sekarang:

```
int main() {  
    // Luas Persegi Panjang 1  
    int panjang1 = 10; int lebar1 = 6;  
    hitungLuas ();  
    cetak ();  
}  
  
// Luas Persegi Panjang 2  
int panjang2 = 4; int lebar2 = 9;  
hitungLuas ();  
cetak ();  
}
```

Cukup menuliskan nama sub program yang akan digunakan

Berdasarkan contoh tersebut dalam program komputer seringkali terdapat bagian program yang ingin digunakan kembali sehingga perlu dituliskan secara berulang. Pada contoh program yang bertugas menghitung luas persegi panjang dan mencetak hasil perhitungan dituliskan secara berulang padahal melakukan aktivitas yang sama.

Dibandingkan menuliskan kode yang sama secara berulang, program dapat dikelompokkan ke dalam beberapa sub program berdasarkan tugasnya. Pada program suatu program utama yang kompleks dapat dibagi ke dalam beberapa sub program berdasarkan tugasnya atau dikenal dengan teknik pemrograman modular. Dengan pemrograman modular memungkinkan pemrogram memanggil kembali subprogram yang telah didefinisikan setiap kali diperlukan dalam program tersebut. Pemrogram tidak perlu berulang kali mendefinisikan sekumpulan instruksi yang diperlukan.

Dikenal dua tipe subprogram yang biasa digunakan untuk memecah persoalan kompleks menjadi lebih sederhana, yaitu fungsi (*function*) dan prosedur (*procedure*). Kedua tipe subprogram ini dapat digunakan bersamaan maupun salah satunya saja dalam sebuah program. Masing-masing tipe subprogram memiliki karakteristik dan perilaku yang berbeda sehingga penggunaannya dalam program juga berbeda-beda.

KEGUNAAN PEMROGRAMAN MODULAR

Kegunaan modularisasi program dengan fungsi atau prosedur adalah sebagai berikut:

1. Menghindari duplikasi kode

Untuk aktivitas yang harus dilakukan lebih dari satu kali, modularisasi menghindari penulisan teks program yang sama secara berulang kali. Di sini sub program cukup ditulis sekali saja dalam bentuk fungsi atau prosedur, lalu dapat dipanggil dari bagian lain di dalam program sesuai kebutuhan.

2. Kemudahan dalam mencari dan menemukan kesalahan (*debug*) program

Karena setiap fungsi melakukan aktivitas spesifik maka apabila terdapat kesalahan di dalam program kesalahan tersebut cukup ditelusuri di dalam fungsi atau prosedur yang bersangkutan.

3. Program menjadi *reusable* atau dapat digunakan kembali sesuai kebutuhan.

Dengan menggunakan fungsi, program dapat digunakan secara berulang kali dengan cara memanggilnya sesuai kebutuhan

4. Program menjadi terstruktur, sehingga mudah dipahami dan mudah dikembangkan.

Dengan memisahkan langkah-langkah detail ke satu atau lebih sub program (fungsi dan prosedur), maka program utama menjadi lebih terstruktur, jelas dan mudah dimengerti.

PROSEDUR



Prosedur merupakan modul atau sub program yang merupakan bagian dari program utama dan bertugas mengerjakan tugas tertentu dan **tidak mengembalikan suatu nilai.**

Ciri-ciri dari prosedur adalah sebagai berikut:

1. Tidak memiliki nilai kembalian
2. Menggunakan keyword `void` pada saat pendefinisian prosedur.
3. Karena prosedur tidak mengembalikan suatu nilai maka tidak adanya keyword *return* pada badan prosedur
4. Tidak dapat langsung ditampilkan hasilnya.

Umumnya prosedur digunakan untuk melakukan proses-proses yang tidak menghasilkan nilai, seperti melakukan pengulangan, proses pengesetan nilai ataupun yang lainnya. Untuk membuat prosedur pada program maka perlu dilakukan **pendefinisian prosedur** agar prosedur dapat dikenali oleh program, lalu dilakukan **pemanggilan prosedur** yang telah didefinisikan pada saat dibutuhkan.

PARAMETER

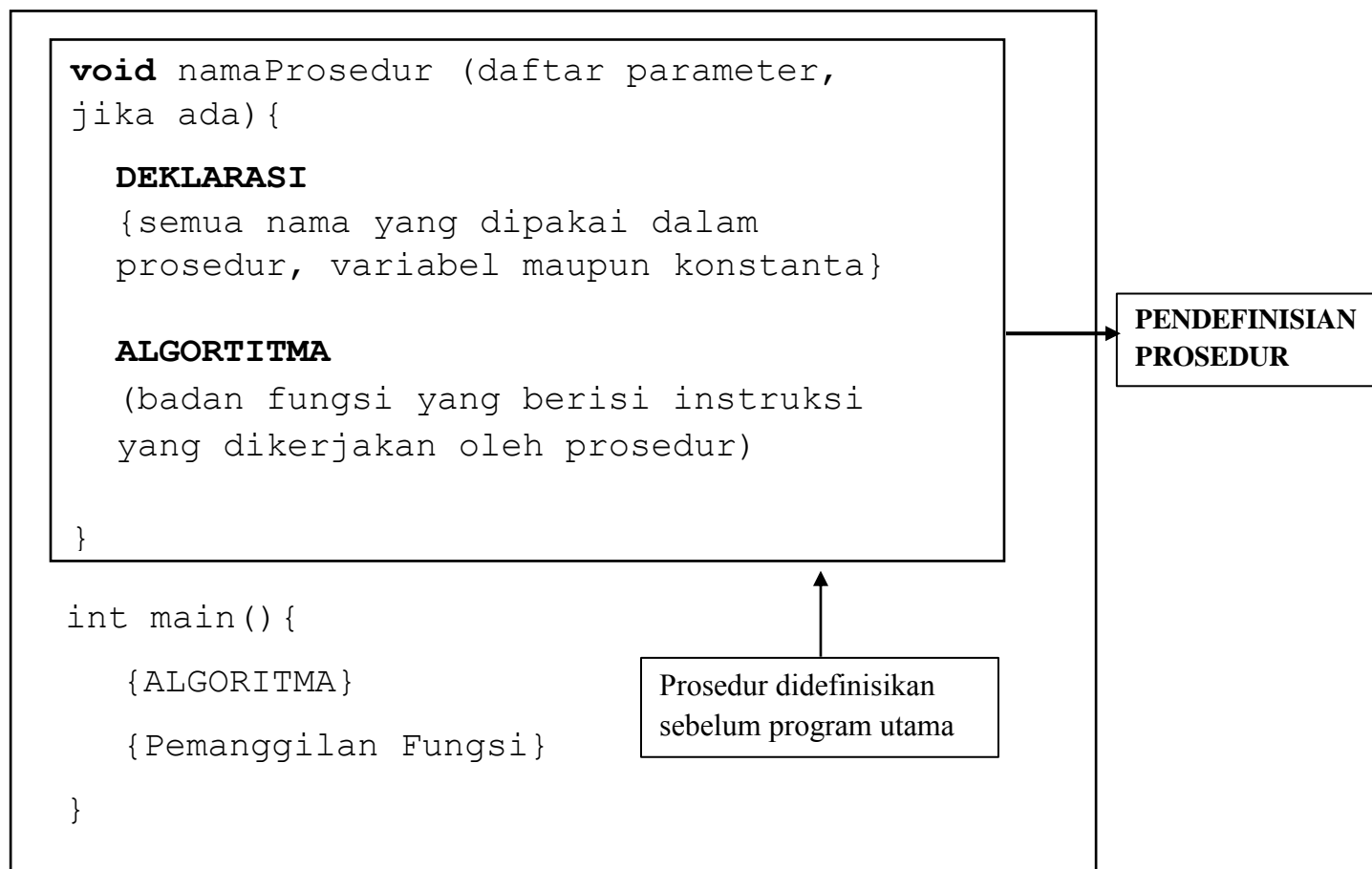
Biasanya prosedur maupun fungsi membutuhkan parameter untuk melakukan pertukaran informasi dengan sub program lain atau program pemanggil untuk mendapatkan data. Parameter ditulis pada saat pendefinisian dan pemanggilan prosedur maupun fungsi. (Penjelasan mengenai parameter dilihat di sub bab selanjutnya). Parameter adalah nama-nama peubah yang dideklarasikan pada bagian *header* pada saat pendefinisian prosedur dan disertakan pada saat pemanggilan prosedur untuk mengirimkan data atau informasi ke dalam prosedur.

A. PENDEFINISIAN PROSEDUR

Agar prosedur dikenali oleh program maka perlu dilakukan **pendefinisian prosedur**. Prosedur didefinisikan pada bagian atas program sebelum program utama. Dalam pembuatan sebuah prosedur, pemrogram harus mendefinisikan:

- Keyword *void* (karena prosedur tidak mengembalikan suatu nilai maka ditulis dengan void yang artinya kosong)
- Nama prosedur
- Satu atau lebih instruksi pada badan prosedur
- Daftar parameter prosedur (jika ada, penggunaan parameter pada prosedur bukanlah suatu keharusan)

Adapun sintaks pendefinisian prosedur adalah sebagai berikut:



Keterangan:

- **void**: menspesifikasikan tipe data nilai kembalian prosedur, karena prosedur tidak memiliki nilai kembalian maka ditulis dengan keyword **void** yang berarti kosong.
- **nama prosedur**: aturan pemberian nama pada prosedur secara bebas, dengan mengikuti aturan penamaan variabel
- **parameter**: variabel yang bertindak untuk menerima nilai masukan pada prosedur, daftar parameter diletakan alam tanda kurung " () ", dibelakang nama fungsi. Parameter pada suatu prosedur boleh ada ataupun tidak.
- **badan prosedur**: berisi deklarasi dan instruksi yang dikerjakan oleh prosedur. Badan prosedur ditulis dalam tanda kurung kurawal " {} "

CONTOH 1

Membuat sebuah program yang menampilkan menu pada ATM seperti berikut:

```
1. CEK SALDO
2. PENARIKAN TUNAI
3. SETOR TUNAI
4. TRANSFER
5. PEMBAYARAN
```

Menu tersebut ditampilkan pada halaman awal dan halaman pemilihan menu.

Pendefinisian Prosedur

```
void menu() {

    cout<< " 1. CEK SALDO"<<endl;
    cout<< " 2. PENARIKAN TUNAI"<<endl;
    cout<< " 3. SETOR TUNAI"<<endl;
    cout<< " 4. KELUAR"<<endl;

}
```

Pada program tersebut didefinisikan prosedur bernama prosedur menu yang bertugas untuk mencetak menu pada program. Pernyataan void pada awal pendefinisian prosedur menyatakan bahwa prosedur tidak mengembalikan suatu nilai. Sehingga tipe data yang diberikan pada nilai kembalian prosedur adalah **void**. Prosedur tidak memiliki parameter sehingga daftar parameter pada badan prosedur tidak diberi nilai atau dikosongkan

Namun meskipun sudah dilakukan pendefinisian prosedur, ketika program dijalankan prosedur tersebut tidak dapat langsung dieksekusi. Prosedur dapat dieksekusi ketika dilakukan **pemanggilan prosedur**.

B. PEMANGGILAN PROSEDUR

Prosedur bukan merupakan suatu program yang dapat berdiri sendiri jadi fungsi tidak dapat dieksekusi secara langsung. Ini berarti, instruksi-instruksi di dalam badan prosedur baru dapat dilaksanakan ketika prosedur tersebut diakses. Prosedur dapat diakses dengan melakukan pemanggilan prosedur dari program pemanggil (dapat dari program utama atau sub program lainnya).

Beberapa hal yang perlu diperhatikan pada saat pemanggilan prosedur antara lain adalah sebagai berikut:

- Nama prosedur yang dituliskan pada saat pemanggilan harus sama dengan nama prosedur yang dipanggil.
- Penulisan daftar parameter pada saat pemanggilan prosedur harus sama baik jumlah, urutan, dan typenya dengan daftar parameter pada pendefinisian prosedur
- Saat pemanggilan terjadi korespondensi antara parameter yang disertakan pada saat pemanggilan fungsi serta parameter yang dideklarasikan dalam fungsi

Adapun sintaks pemanggilan prosedur adalah sebagai berikut:

`namaProsedur (daftar parameter, jika ada);`

Pemanggilan prosedur dilakukan dengan menuliskan nama prosedur, dan daftar parameter yang ada pada prosedur. Parameter adalah variabel yang terdapat pada prosedur.

Pemanggilan prosedur menu()

```
#include<iostream>
using namespace std;
void menu() {
    cout<<"1. CEK SALDO"<<endl;
    cout<<"2. PENARIKAN TUNAI"<<endl;
    cout<<"3. SETOR TUNAI"<<endl;
    cout<<"4. KELUAR"<<endl;
}

int main() {
    //menampilkan menu pada halaman awal
    menu();
    //menampilkan menu pada halaman menu
    menu();
}
```

PENDEFINISIAN
PROSEDUR

PEMANGGILAN
PROSEDUR

Prosedur menu dipanggil pada pernyataan

```
menu();
```

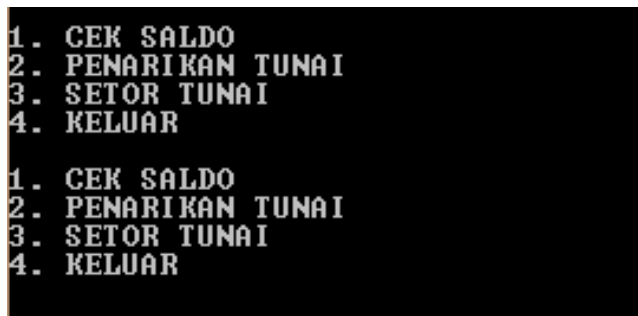
karena prosedur tidak memiliki parameter maka cukup menuliskan nama prosedur saja, daftar parameter dalam tanda kurung dikosongkan. Ketika NamaProsedur dipanggil, kendali program berpindah secara otomatis ke prosedur tersebut. Seluruh instruksi di dalam badan prosedur dilaksanakan. Setelah semua instruksi selesai dilaksanakan,

kendali program berpindah secara otomatis kembali ke instruksi sesudah pemanggilan prosedur. Prosedur dapat dipanggil berulang kali sesuai dengan kebutuhan. Prosedur akan dieksekusi sesuai dengan jumlah prosedur tersebut dipanggil.

Ketika prosedur dipanggil, prosedur akan bekerja dalam mekanisme berikut:

1. Yang pertama kali dieksekusi adalah program utama.
2. Ketika prosedur dipanggil, kendali program kini berpindah ke prosedur menu()
3. Ketika dipanggil semua instruksi dalam badan prosedur dieksekusi.
4. Setelah semua instruksi selesai dilaksanakan, kendali program berpindah secara otomatis kembali ke instruksi sesudah pemanggilan prosedur.

HASIL EKSEKUSI:



```
1. CEK SALDO
2. PENARIKAN TUNAI
3. SETOR TUNAI
4. KELUAR

1. CEK SALDO
2. PENARIKAN TUNAI
3. SETOR TUNAI
4. KELUAR
```

Pada program dengan prosedur, jika terdapat instruksi atau kode program yang sama yang akan digunakan kembali maka tidak perlu dituliskan secara berulang. Cukup dilakukan **pemanggilan pada prosedur tempat kode program tersebut dituliskan secara berulang kali** sesuai dengan kebutuhan.

Program di atas sekaligus menjelaskan bahwa suatu prosedur cukup **didefinisikan satu kali tetapi bisa digunakan beberapa kali**. Sehingga pemakaian prosedur dapat menghindari duplikasi kode dan menghemat penulisan program maupun kode dalam memori.

CONTOH 2 (Prosedur dengan Parameter)

Membuat program yang mencetak kalimat *greeting message* berikut!

```
Selamat Datang Tiara
Terima Kasih Telah Mendaftar

Selamat Datang Nabila
Terima Kasih Telah Mendaftar

-----
```

Jika diperhatikan program tersebut mencetak kalimat yang sama secara berulang, hanya dengan nilai yang berbeda. Maka dapat kita gunakan **parameter** agar prosedur dapat menerima nilai masukan yang nilainya dapat kita atur. Parameter ditulis pada pendefinisian prosedur dengan menuliskan nama dan tipe data parameter, serta nilainya dituliskan pada saat pemanggilan.

```
#include<iostream>
```

```
using namespace std;
```

PARAMETER

```
void sapa(string nama){
```

```
    cout<<"Selamat Datang "<<nama<<endl;
```

```
    cout<<"Terima Kasih Telah
Mendaftar"<<endl;
```

```
}
```

```
int main()
```

```
    sapa("Tiara");
```

```
    sapa("Nabila");
```

```
}
```

PENDEFINISIAN
PROSEDUR

Pemanggilan Prosedur
dengan Parameter

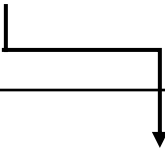
Berdasarkan program tersebut fungsi `hitungLuas()` dipanggil pada pernyataan:

```
sapa("Tiara");  
sapa("Nabila");
```

Prosedur dipanggil dengan menyertakan nilai dari parameter, nilai tersebut akan diterima oleh prosedur sebagai nilai masukan. Ketika prosedur dipanggil, prosedur akan bekerja dalam mekanisme berikut:

1. Yang pertama kali dieksekusi adalah program utama.
2. Prosedur `sapa()` dipanggil dengan mengirimkan nilai parameter "Tiara" dan "Nabil" yang nantinya akan diisikan ke parameter nama dan digunakan prosedur untuk melakukan sejumlah instruksi pada prosedur `sapa()`.

```
int main()  
  
    sapa("Tiara");  
  
    sapa("Nabila");  
  
}
```



```
void sapa(string nama){  
  
    cout<<"Selamat Datang "<<nama<<endl;  
  
    cout<<"Terima Kasih Telah Mendaftar"<<endl;  
  
}
```

3. Ketika prosedur dipanggil, kendali program kini berpindah ke prosedur `sapa()`
4. Ketika dipanggil semua instruksi dalam badan prosedur dieksekusi.
5. Setelah semua instruksi selesai dilaksanakan, kendali program berpindah secara otomatis kembali ke instruksi sesudah pemanggilan prosedur.

Penggunaan parameter pada prosedur akan menjadikan prosedur lebih dinamis.