Subprogram: Fungsi dan Prosedur

Tim Olimpiade Komputer Indonesia

Pendahuluan

Melalui dokumen ini, kalian akan:

- Memahami mengapa perlu ada subprogram.
- Membedakan fungsi dan prosedur.
- Mempelajari cara menulis fungsi dan prosedur.

Motivasi-1

- Ketika menulis program, kadang-kadang kita memerlukan suatu rutinitas yang sama di beberapa tempat.
- Sebagai gambaran, perhatikan contoh soal berikut:
  - Pak Dengklek menancapkan tiga buah tiang pancang di halaman rumahnya untuk membangun sebuah kandang bebek.
  - Setiap tiang pancang bisa dianggap terletak di suatu sistem koordinat Kartesius, yaitu di (A_x, A_y), (B_x, B_y), dan (C_x, C_y).
  - Pagar akan dibentangkan menurut garis lurus antar setiap tiang pancang.
  - Sekarang Pak Dengklek ingin tahu berapa luas kandang bebeknya.
- Persoalan yang kita hadapi adalah menghitung luas dari segitiga, jika hanya diberikan titik-titik sudutnya.

Motivasi-1 (lanj.)

Salah satu penyelesaian untuk soal tersebut adalah menggunakan rumus Heron:

Jika sebuah segitiga memiliki panjang sisi sebesar a, b, dan c, maka luas segitiga tersebut adalah:
$L = \sqrt{S(S-a)(S-b)(S-c)}$, dengan $S = \frac{a+b+c}{2}$

Bagaimana implementasinya pada program?

Motivasi-1 (lanj.)

- Kita perlu menghitung jarak antar titik terlebih dahulu, baru bisa menghitung luasnya.
- Pascal tidak menyediakan fungsi dasar untuk menghitung jarak antar dua titik, sehingga kita harus menuliskannya:

```
(* tA, tB, tC merupakan ketiga titik yang diberikan *)
a := sqrt(sqr(tA.x - tB.x) + sqr(tA.y - tB.y));
b := sqrt(sqr(tA.x - tC.x) + sqr(tA.y - tC.y));
c := sqrt(sqr(tB.x - tC.x) + sqr(tB.y - tC.y));
```

- Perhatikan bahwa hal yang sama, yaitu menghitung jarak titik, dituliskan secara berulang-ulang.
- Bagaimana jika pada salah satunya terdapat kesalahan pengetikan? Atau suatu ketika rumusnya perlu diubah? Sungguh merepotkan!

Motivasi-2

- Seringkali ketika kita menulis program yang panjang, program menjadi lebih sulit dipahami, meskipun telah ditulis komentar sekalipun.
- Alangkah baiknya jika kita bisa membuat subprogram untuk suatu rutinitas tertentu dan menyatukannya di akhir, seperti:

```
bacaMasukan(N);
bangkitkanPrimaSampai(N);
writeln('faktorisasi:');
temp := N;
while (not cekPrima(temp)) do begin
  d := cariPembagiTerkecil(temp);
  temp := temp div d;
  writeln(d);
end;
if (temp > 1) then begin
  writeln(temp);
end;
```

Bagian 1

Konsep Subprogram

Konsep Subprogram

- Sesuai dengan namanya, subprogram adalah bagian dari program.
- Jika program merupakan serangkaian instruksi untuk mencapai suatu tujuan tertentu, maka subprogram bisa dianggap sebagai serangkaian instruksi untuk mencapai suatu tujuan tertentu, sebagai bagian dari tujuan program.
- Subprogram bisa dipanggil di bagian manapun pada program.

Konsep Prosedur

- Kita bisa memindahkan serangkaian kode menjadi sebuah subprogram, lalu memanggilnya pada program utama.
- Perhatikan contoh pesan.pas berikut!

```
var
  pesan: string;

(* prosedur *)
procedure bacaPesan();
begin
  write('masukkan pesan: ');
  readln(pesan);
end;

(* program utama *)
begin
  bacaPesan();
  writeln('pesan = ', pesan);
end.
```

Penjelasan

- Pada pesan.pas, terdapat sebuah prosedur bernama **bacaPesan** yang melakukan perintah untuk membaca masukan.
- Ketika **bacaPesan** dipanggil pada program utama, bisa dianggap seluruh instruksi yang ada di dalam prosedur tersebut dipindahkan ke program utama yang memanggilnya.
- Sehingga, program utama seakan-akan menjadi:

```
(* program utama *)
begin
  write('masukkan pesan: ');
  readln(pesan);
  writeln('pesan = ', pesan);
end.
```

Penjelasan (lanj.)

- Tentu saja, sebuah prosedur bisa dipanggil berkali-kali, dan hal yang dilakukan tetap sama.
- Coba modifikasi blok progam utama pesan.pas sehingga menjadi:

```
(* program utama *)
begin
  bacaPesan();
  writeln('pesan = ', pesan);

  bacaPesan();
  writeln('sekarang pesan berisi = ', pesan);
end.
```

Bagian 2

Implementasi pada Pascal

Prosedur

Pada Pascal, prosedur bisa ditulis dengan format berikut:

```
procedure <nama>(<parameter...>);
var
  <deklarasi variabel...>
begin
  <instruksi...>
end;
```

- Blok **var** dan deklarasi variabel boleh dihilangkan jika tidak ada variabel perlu dideklarasikan.
- Perhatikan bahwa **end** diakhiri dengan titik koma (;), bukan titik (.) seperti pada blok program utama.

Konsep Parameter

Parameter merupakan tempat untuk "memberi masukan" bagi prosedur, sehingga prosedur bisa berperilaku berdasarkan masukan yang ia terima.

Perhatikan contoh berikut:

```
procedure gambar(x: longint);
var
  i: longint;
begin
  for i := 1 to x do begin
    write('*');
  end;
  writeln;
end;
```

Konsep Parameter (lanj.)

- Prosedur **gambar** berfungsi untuk menuliskan karakter '*' pada sebuah baris sebanyak x kali.
- Lalu apa x pada prosedur tersebut?
- Untuk menjawabnya, perhatikan blok program utama berikut:

```
(* program utama *)
begin
  gambar(3);
  gambar(5);
end.
```

- Yang akan tercetak adalah:

```
***
*****
```

Konsep Parameter (lanj.)

- Pada pemanggilan pertama, angka 3 pada **gambar(3)** mengakibatkan nilai x untuk prosedur **gambar** bernilai 3. Sehingga tercetak 3 karakter '*'.
- Pada pemanggilan kedua, nilai x yang diterima adalah 5. Sehingga tercetak 5 karakter '*'.
- Variabel x pada prosedur **gambar** disebut sebagai **parameter**.
- Melalui contoh ini, kalian dapat memahami bahwa nilai parameter dapat digunakan untuk mengatur perilaku prosedur.

Konsep Parameter (lanj.)

- Tentu saja, pemanggilan juga bisa dilakukan dengan variabel seperti contoh berikut:

```
var
  N: longint;

... (* prosedur gambar *)

begin
  readln(N);
  gambar(N);
end.
```

Parameter

- Parameter dituliskan seperti pada deklarasi variabel.
- Contoh:

```
(* tanpa parameter *)
procedure baca();

(* satu parameter *)
procedure tes(x: longint);

(* dua parameter *)
procedure sama(x, y: longint);

(* dua parameter, berbeda tipe data *)
procedure berbeda(x: string; y: longint);
```

Lingkup Variabel

Apa itu lingkup variabel?

Bisa dianggap sebagai "masa hidup" suatu variabel; kapan ia mulai terdefinisi dan kapan ia sudah tidak terdefinisi lagi.

Lingkup Variabel (lanj.)

- Perhatikan kembali prosedur **gambar** berikut:

```
procedure gambar(x: longint);
var
  i: longint;
begin
  for i := 1 to x do begin
    write('*');
  end;
  writeln;
end;
```

- Variabel **i** dan **x** pada prosedur tersebut hanya terdefinisi di antara blok **begin** dan **end** prosedur **gambar** saja.
- Artinya jika pada program utama terdapat pula variabel bernama **x** atau **i**, maka variabel tersebut bukan mengacu pada **x** dan **i** pada prosedur **gambar**.

Lingkup Variabel (lanj.)

- Variabel yang dideklarasikan pada blok **var subprogram** biasa disebut sebagai **variabel lokal**.
- Sementara variabel yang dideklarasikan pada blok **var program utama** disebut sebagai **variabel global**.
- Variabel global dapat diakses di mana saja, bahkan di dalam subprogram sekalipun.
- Namun, variabel lokal hanya bisa diakses pada subprogram yang mendeklarasikannya.

Fungsi

- Setelah kalian memahami tentang prosedur, mari kita membahas tentang fungsi.
- Secara konsep, fungsi mirip dengan prosedur.
- Bedanya adalah fungsi mengembalikan sebuah nilai.
- Perhatikan fungsi berikut:

```
function kubik(x: longint): longint;
begin
  kubik := x*x*x;
end;
```

Penjelasan

- Pada program utama, kita bisa melakukan:

```
(* program utama *)
begin
  volume := kubik(3);
  selisih := volume - kubik(2);
  writeln('4 kubik adalah ', kubik(4));
end.
```

- Teringat dengan sesuatu?
- Fungsi **kubik** kini terlihat seperti fungsi yang biasa kalian gunakan, seperti **sqrt**, **trunc**, atau **abs**!

Nilai Kembalian

- Ciri paling utama yang membedakan fungsi dengan prosedur adalah adanya **nilai kembalian**.
- Ketika fungsi dipanggil, fungsi akan mengembalikan nilai, dan nilai ini bisa dioperasikan ke dalam **ekspresi** atau **assignment**.
- Sebagai ilustrasi, perhatikan ekspresi berikut:

```
x := 2;
y := 3*sqr(x) - 1;
```

- Pada saat dijalankan, fungsi sqr(x) akan dieksekusi dan **mengembalikan nilai** 4.

```
y := 3*4 - 1;
```

- Setelah ekspresi itu dievaluasi, **y** bernilai 11.

Nilai Kembalian (lanj.)

- Hal semacam ini tidak berlaku pada prosedur, karena prosedur tidak mengembalikan nilai.
- Hal ini juga menjelaskan mengapa fungsi dan prosedur biasa dipanggil dengan cara:

```
(* prosedur *)
prosedurKerja();

(* fungsi *)
x := fungsiKerja();
```

Fungsi

- Pada Pascal, fungsi dituliskan dengan format:

```
function <nama>(<parameter...>): <tipe kembalian>
var
  <deklarasi variabel...>
begin
  <instruksi...>
  ...
  <nama> := <nilai kembalian>;
end;
```

- Dengan <tipe kembalian> adalah tipe data untuk nilai yang dikembalikan.
- Untuk memberi tahu Pascal nilai yang perlu dikembalikan, lakukan **assingment** pada nama fungsi dengan nilai yang akan dikembalikan, seperti yang dicontohkan baris akhir isi subprogram pada format di atas.

Fungsi (lanj.)

- Melakukan **assignment** pada nama fungsi tidak harus dilakukan pada bagian akhir.
- Yang pasti, Pascal akan mengembalikan nilai yang terakhir di-**assign** ke dalam nama fungsi tersebut.
- Seperti pada prosedur, bagian deklarasi variabel juga bisa dihilangkan jika tidak ada variabel lokal yang diperlukan.
- Cara penulisan parameter juga sama dengan prosedur.

Bagian 3

Passing Parameter

Passing Parameter

- *Passing parameter* merupakan aktivitas menyalurkan nilai pada parameter saat memanggil subprogram.
- Umumnya, dikenal dua macam *passing parameter*:
  - *By value*, yaitu mengirimkan nilai dari setiap parameter yang diberikan.
  - *By reference*, yaitu mengirimkan alamat dari setiap parameter yang diberikan.

Passing Parameter by Value

- Sebagai penjelasan, perhatikan program berikut:

```
var
  x, y: longint;

procedure tukar(a, b: longint);
var
  temp: longint;
begin
  temp := a;
  a := b;
  b := a;
end;

begin
  x := 1; y := 2;
  tukar(x, y);
  writeln('x=', x, ' y=', y);
end.
```

Passing Parameter by Value (lanj.)

- Jika dijalankan, apa keluaran dari program tersebut? Apakah "x=2 y=1"?
- Jawabannya tidak, yang tercetak adalah "x=1 y=2".
- Hal ini disebabkan karena ketika prosedur **tukar** dipanggil, nilai dari x dan y dikirim ke parameter a dan b pada prosedur **tukar**.
- Jadi a dan b hanya melakukan **assignment** nilai dari x dan y. Apapun yang terjadi pada a dan b selanjutnya tidak mempengaruhi x dan y karena mereka tidak berhubungan.

Passing Parameter by Reference

- Lain halnya ketika kita menambahkan kata kunci **var** pada penulisan parameter:

```
procedure tukar(var a, b: longint);
var
   temp: longint;
begin
   temp := a;
   a := b;
   b := a;
end;
```

Passing Parameter by Reference (lanj.)

- Dengan cara ini, ketika **tukar**(x, y) dipanggil, alamat memori variabel x dan y dikirimkan ke parameter a dan b.
- Kini, x dan a mengacu pada alamat memori yang sama.
- Apabila dilakukan perintah "$a := 3$", maka nilai x juga ikut menjadi 3. Hal ini disebabkan karena x dan a mengacu pada alamat memori yang sama.
- Demikian pula untuk y dan b.
- Sehingga keluaran program menjadi "x=2 y=1"!

Passing Parameter by Reference (lanj.)

- Dengan *passing parameter by reference*, kita tidak bisa melakukan:

```
tukar(2, 3);
```

- Mengirimkan alamat memori dari variabel memang bisa dilakukan, tetapi angka 2 atau 3 jelas bukan variabel dan jelas tidak punya alamat memori.
- Jika kita mengembalikan kedua parameter prosedur **tukar** untuk menggunakan *passing parameter by value*, barulah hal ini bisa dilakukan. Karena yang dikirimkan adalah nilainya.

Penulisan pada Pascal

- Untuk melakukan *passing parameter by reference*, cukup tambahkan kata kunci **var** di depan parameter yang akan di-*pass by reference*.
- Sebuah subprogram bisa juga menerima parameter dengan cara *passing parameter* yang campuran:

```
procedure bagi(a,b: longint; var hasil,sisa: longint);
begin
  hasil := a div b;
  sisa := a mod b;
end;
```

Bagian 4

Studi Kasus Subprogram

Fungsi Pangkat

Berikut ini adalah fungsi untuk menghitung a^b:

```
function pangkat(a, b: longint);
var
  i: longint;
  hasil: longint;
begin
  hasil := 1;
  for i := 1 to b do begin
    hasil := hasil * a;
  end;

  pangkat := hasil;
end;
```

Prosedur Pangkat

Berikut ini adalah prosedur untuk menghitung a^b, hasil ditampung pada variabel *hasil*:

```
procedure pangkat(a, b: longint; var hasil: longint);
var
  i: longint;
begin
  hasil := 1;
  for i := 1 to b do begin
    hasil := hasil * a;
  end;
end;
```

Fungsi dan Prosedur

- Baik dengan fungsi atau prosedur, kita bisa mencapai hal yang sama.
- Pertanyaannya adalah: mana yang lebih tepat?

Fungsi dan Prosedur (lanj.)

- Dengan fungsi, menghitung $y = 3x^5$ bisa dilakukan dengan:

```
y := 3 * pangkat(x, 5);
```

- Dengan prosedur, sedikit lebih rumit:

```
pangkat(x, 5, y);
y := 3 * y;
```

- Untuk kasus ini, penggunaan fungsi lebih tepat.

Kilas Balik: Prosedur tukar

- Sekarang coba ingat kembali prosedur **tukar** yang kita bahas sebelumnya.
- Kurang masuk akal apabila kita menggunakan fungsi untuk melakukan penukaran.
- Sehingga untuk kasus penukaran isi variabel, lebih tepat digunakan prosedur.

Penggunaan Fungsi dan Prosedur

- Dari sini kita mempelajari bahwa ada subprogram yang lebih cocok diimplementasikan menjadi fungsi, dan ada juga yang lebih cocok menjadi prosedur.
- Biasanya, fungsi bersifat:
  - Menghasilkan suatu nilai berdasarkan parameter.
  - Tidak mengakibatkan efek samping, misalnya adanya perubahan nilai pada parameter yang diberikan seperti pada prosedur **tukar**.
- Sementara prosedur bersifat:
  - Tidak menghasilkan suatu nilai.
  - Boleh jadi mengakibatkan perubahan pada variabel global atau parameter yang dikirimkan.

Manfaat Subprogram

- Meningkatkan daya daur ulang kode (*reusability*).
  Satu kali saja kita mendefinisikan prosedur untuk menukar isi variabel, berapa kali pun penukaran isi variabel bisa dilakukan tanpa perlu menuliskan algoritma penukaran (cukup memanggil prosedur).
- Memecah program menjadi beberapa subprogram yang lebih kecil.
  Keuntungannya adalah didapatkan kumpulan subprogram yang:
  - Fokus pada suatu tujuan tertentu.
  - Tersusun atas kode yang cenderung pendek.
  - Karena kedua hal di atas, lebih mudah dibaca dan ditelusuri.

Selanjutnya...

- Mengulas lebih dalam tentang tipe data **string** pada Pascal.
- Mengeksplorasi fungsi dan prosedur dasar pada pengolahan **string**.