Subprogram

Tim Olimpiade Komputer Indonesia

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.
- Kita dapat menuliskannya:

```
// tA, tB, tC merupakan ketiga titik yang diberikan
a = sqrt((tA.x - tB.x)*(tA.x - tB.x) + (tA.y -
    tB.y)*(tA.y - tB.y));
b = sqrt((tA.x - tC.x)*(tA.x - tC.x) + (tA.y -
    tC.y)*(tA.y - tC.y));
c = sqrt((tB.x - tC.x)*(tB.x - tC.x) + (tB.y -
    tC.y)*(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);
cariPrimaSampai(N);
printf("faktorisasi:\n");
temp = N;
while (!cekPrima(temp)) {
  d = cariPembagiTerkecil(temp);
  temp = temp / d;
  printf("%d\n", d);
}
if (temp > 1) {
  printf("%d\n", temp);
}
```

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.
- Pada C++, subprogram disebut dengan fungsi.

Contoh Subprogram

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

```
#include <cstdio>
#include <string>
using namespace std;

char buff[1001];
string pesan;

// Subprogram
void bacaPesan() {
  printf("masukkan pesan: \n");
  scanf("%s", buff);
  pesan = buff;
}

// Program utama
int main() {
  bacaPesan();
  printf("pesan = %s\n", pesan.c_str());
}
```

Penjelasan

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

```
int main() {
  printf("masukkan pesan: \n");
  scanf("%s", buff);
  pesan = buff;
  printf("pesan = %s\n", pesan.c_str());
}
```

Penjelasan (lanj.)

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

```
int main() {
  bacaPesan();
  printf("pesan = %s\n", pesan.c_str());

  bacaPesan();
  printf("sekarang pesan berisi = %s\n",
    pesan.c_str());
}
```

Bagian 2

Implementasi pada C++

Fungsi

Pada C++, fungsi bisa ditulis dengan format berikut:

```
<tipe> <nama>(<parameter...>) {
  <instruksi...>
}
```

- <tipe>: nilai kembalian yang dihasilkan fungsi. Untuk fungsi yang tidak menghasilkan nilai kembalian, kita gunakan tipe `void`. Fungsi yang menghasilkan nilai kembalian akan dipelajari di bagian selanjutnya.
- <nama>: nama dari fungsi.
- <parameter>: informasi yang hendak diberikan ke fungsi.

Konsep Parameter

Parameter merupakan tempat untuk "memberi masukan" bagi fungsi, sehingga fungsi bisa berperilaku berdasarkan masukan yang diterima.

Perhatikan contoh berikut:

```
void gambar(int x) {
  for (int i = 0; i < x; i++) {
    printf("*");
  }
  printf("\n");
}
```

Konsep Parameter (lanj.)

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

```
// Program utama
int main() {
  gambar(3);
  gambar(5);
}
```

- Yang akan tercetak adalah:

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

Konsep Parameter (lanj.)

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

Konsep Parameter (lanj.)

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

```
int main() {
  int n;
  scanf("%d", &n);
  gambar(n);
}
```

Parameter

- Parameter dituliskan dengan format tipe dan namanya.
- Suatu fungsi boleh memiliki beberapa parameter.
- Contoh:

```
// Tanpa parameter
void baca()

// Satu parameter
void tes(int x)

// Dua parameter
void sama(int x, int y)

// Dua parameter, berbeda tipe data
void berbeda(int x, string y)
```

Lingkup Variabel

- Perhatikan kembali fungsi `gambar` berikut:

```
void gambar(int x) {
  for (int i = 0; i < x; i++) {
    printf("*");
  }
  printf("\n");
}
```

- Variabel `i` dan `x` pada fungsi tersebut hanya terdefinisi di antara blok { dan } fungsi `gambar` saja.
- Artinya jika pada program utama terdapat pula variabel bernama `x` atau `i`, maka variabel tersebut bukan mengacu pada `x` dan `i` pada fungsi `gambar`.

Lingkup Variabel (lanj.)

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

Fungsi dengan Nilai Kembalian

- Setelah kalian memahami tentang fungsi, mari kita bahas tentang nilai kembalian.
- Perhatikan fungsi berikut:

```
int kubik(int x) {
  return x*x*x;
}
```

Penjelasan

- Pada program utama, kita bisa melakukan:

```
int main() {
  int volume = kubik(3);
  int selisih = volume - kubik(2);
  printf("4 kubik adalah %d\n", kubik(4));
}
```

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

Nilai Kembalian

- Ketika fungsi mengembalikan nilai, nilai ini bisa dioperasikan ke dalam ekspresi atau assignment.
- Sebagai ilustrasi, perhatikan ekspresi berikut:

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

- Pada saat dijalankan, fungsi kubik(x) akan dieksekusi dan mengembalikan nilai 8.

```
y := 3*8 - 1;
```

- Setelah ekspresi itu dievaluasi, y bernilai 23.

Nilai Kembalian (lanj.)

- Hal semacam ini tidak berlaku ketika fungsi tidak mengembalikan nilai.
- Hal ini juga membedakan cara pemanggilannya:

```
// Tidak mengembalikan nilai
kerja1()

// Mengembalikan nilai
x = kerja2();
```

Fungsi

- Untuk mengembalikan nilai, isikan tipe data nilai kembalian pada deklarasi fungsi.
- Selanjutnya, kembalikan nilai dengan cara menuliskan "`return`" diikuti dengan nilai kembaliannya.
- Perintah `return` akan menghentikan eksekusi, keluar dari fungsi, dan mengembalikan nilai ke pemanggilnya.
- Perintah `return` boleh dipanggil di baris fungsi manapun.

Contoh Fungsi Lain

- Perhatikan fungsi yang memeriksa keprimaan berikut:

```
bool prima(int x) {
  if (x < 2) {
    return false;
  }
  for (int i = 2; i*i <= x; i++) {
    if (x % i == 0) {
      return false;
    }
  }
  return true;
}
```

- Pertama, periksa apakah bilangan yang diberikan kurang dari dua. Bila ya, langsung kembalikan nilai `FALSE`.
- Kedua, periksa apakah ada angka di antara 2 dan $\sqrt{x}$ yang habis membali x. Bila ada, langsung kembalikan `FALSE`.
- Selain daripada itu, dijamin x prima.

Return pada Fungsi void

- Sebenarnya, `return` juga dapat dilakukan pada fungsi yang tidak mengembalikan nilai.
- Perintah `return` akan menghentikan eksekusi dan keluar dari program.
- Pada contoh berikut, gambar '*' tidak akan dicetak apabila x lebih dari 1000.

```
void gambar(int x) {
  if (x > 1000) {
    return;
  }
  for (int i = 0; i < x; i++) {
    printf("*");
  }
  printf("\n");
}
```

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:

```
#include <cstdio>

void tukar(int a, int b) {
  int temp = a;
  a = b;
  b = temp;
}

int main() {
  int x = 1;
  int y = 2;
  tukar(x, y);
  printf("x=%d y=%d\n", x, y);
}
```

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".
- Ketika fungsi `tukar` dipanggil, nilai dari x dan y dikirim ke parameter a dan b pada fungsi `tukar`.
- Jadi hanya dilakukan *assignment* nilai dari x dan y ke a dan b.
- 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 lambang **&** pada penulisan parameter:

```
void tukar(int &a, int &b) {
  int temp = a;
  a = b;
  b = temp;
}
```

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. Sebab 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.)

- Jika `tukar` ditulis 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 fungsi `tukar` untuk menggunakan *passing parameter by value*, barulah hal ini bisa dilakukan.

Penulisan pada C++

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

```
void bagi(int a, int b, int &hasil, int &sisa) {
  hasil = a / b;
  sisa = a % b;
}
```

Bagian 4

Studi Kasus Subprogram

Fungsi Pangkat (int)

Berikut ini adalah fungsi untuk menghitung a^b:

```
int pangkat(int a, int b) {
  int hasil = 1;
  for (int i = 0; i < b; i++) {
    hasil *= a;
  }
  return hasil;
}
```

Fungsi Pangkat (void)

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

```
void pangkat(int a, int b, int &hasil) {
  hasil = 1;
  for (int i = 0; i < b; i++) {
    hasil *= a;
  }
}
```

Mengembalikan Nilai atau Tidak

- Baik dengan kedua cara, kita bisa mencapai hal yang sama.
- Pertanyaannya adalah: mana yang lebih tepat?

Mengembalikan Nilai atau Tidak (lanj.)

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

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

- Tanpa mengembalikan nilai, sedikit lebih rumit:

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

- Untuk kasus ini, mengembalikan nilai lebih tepat.

Kilas Balik: Fungsi tukar

- Sekarang coba ingat kembali fungsi `tukar` yang kita bahas sebelumnya.
- Kurang masuk akal apabila kita menggunakan mengembalikan nilai saat melakukan penukaran.
- Sehingga untuk kasus penukaran isi variabel, tanpa mengembalikan nilai lebih tepat.

Penggunaan Subprogram

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

Manfaat Subprogram

- Meningkatkan daya daur ulang kode (*reusability*).
  Satu kali saja kita mendefinisikan subprogram untuk menukar isi variabel, berapa kali pun penukaran isi variabel bisa dilakukan tanpa perlu menuliskan algoritma penukaran lagi.
- 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.