INFORMATIKA Fungsi

Ikhtisar

Fungsi adalah bagian kode yang dapat digunakan kembali yang melayani tujuan tertentu. Fungsi dapat mengambil *input* dan *output*, dan dapat digunakan kembali di seluruh program. Mengorganisir program ke dalam fungsi membantu mengatur dan menyederhanakan kode. Ini adalah contoh **abstraksi**; setelah Anda menulis suatu fungsi, Anda dapat menggunakan fungsi tersebut tanpa harus khawatir tentang detail bagaimana fungsi tersebut diimplementasikan. Karena abstraksi, orang lain dapat menggunakan (atau "memanggil") fungsi tersebut tanpa mengetahui detail level bawahnya juga.

Istilah Kunci

- fungsi
- abstraksi
- *return type*
- efek samping
- *return value*
- cakupan

```
1   #include <stdio.h>
2
3   int say_hi(void)
4   {
5       printf("Hi!\n");
6   }
7
8   int main(void)
9   {
10      say_hi();
11      say_hi();
12  }
```

Sintak Fungsi

Semua program yang Anda tulis dalam C sudah memiliki satu fungsi: `main`. Namun, program di C dapat memiliki lebih banyak fungsi juga. Program di sebelah kiri mendefinisikan fungsi baru bernama `say_hi()`.

Baris pertama suatu fungsi membutuhkan tiga bagian. Pertama, ***return type*** fungsi, yang merupakan tipe data dari *output* fungsi yang "dikembalikan" ke tempat fungsi dipanggil. Jika fungsi tidak mengembalikan nilai, *return type* yang digunakan adalah `void`. Kedua, nama fungsi; ini tidak dapat menggunakan spasi dan tidak dapat menggunakan salah satu kata kunci C yang sudah ada. Ketiga, dalam tanda kurung, parameter fungsi, juga dikenal sebagai argumen. Ini adalah *input* fungsi (jika tidak ada, gunakan `void`). Setelah baris pertama ini (dikenal sebagai baris deklarasi), kode yang mendefinisikan fungsi itu sendiri diapit oleh kurung kurawal.

Pada contoh di atas, fungsi `say_hi()` menyebabkan `"Hi\n"` dicetak ke layar. Ini disebut **efek samping**, yang merupakan fungsi yang dilakukan di luar cakupannya yang tidak mengembalikan nilai. Fungsi `say_hi()` kemudian dipanggil dua kali pada fungsi `main`. Fungsi dipanggil dengan menuliskan nama fungsi, diikuti oleh argumen apa pun dalam tanda kurung (jika ada), diikuti dengan tanda titik koma. Saat program dijalankan, `"Hi\n"` dicetak ke layar dua kali.

Input dan Output

Contoh di sebelah kanan menunjukkan fungsi menghitung kuadrat, `square`, yang mengambil *input* dan *output*. `square` mengambil satu *input*: bilangan bulat yang disebut `x`. Ini juga mengembalikan `int` ke tempat fungsi dipanggil. Baris 5 dari fungsi menentukan ***return value*** fungsi, diawali dengan kata `return`. Dalam kasus ini, fungsi `square` mengembalikan nilai *input* `x` dikalikan dengan angka yang sama. Ketika sampai pada baris `return`, fungsi berhenti dan *return value* dikembalikan ke tempat fungsi awalnya dipanggil.

```
1   #include <stdio.h>
2
3   int square(int x)
4   {
5       return x * x;
6   }
7
8   int main(void)
9   {
10      printf("%i\n", square(2));
11      printf("%i\n", square(4));
12      printf("%i\n", square(8));
13  }
```

Setelah kita telah menulis fungsi ini, kita dapat menggunakan `square` di tempat lain di program kita kapan saja kita ingin membuat angka kuadrat. Dalam fungsi `main` di sebelah kanan, fungsi `square` disebut tiga kali: setiap kali, fungsi dievaluasi dan mengembalikan *return value* yang sesuai di tempat fungsi dipanggil. Jadi `printf("%i\n, square(2))` memiliki efek yang setara dengan menulis `printf("%i\n", 2 * 2)` atau `printf("%i\n", 4)`.

Cakupan

Variabel yang didefinisikan di dalam fungsi atau dalam daftar parameter fungsi memiliki **cakupan** lokal, artinya variabel-variabel tersebut hanya ada di dalam fungsi itu sendiri dan tidak memiliki arti di tempat lain. Dalam contoh di atas, jika Anda mencoba untuk memanggil variabel `x` di dalam fungsi `main`, *compiler* akan memberi Anda *error*; fungsi `main` tidak tahu apa artinya `x`, hanya `square` yang tahu. Demikian juga, variabel apa pun yang didefinisikan di dalam `main` tidak dapat diakses dari dalam `square`.

Jika variabel didefinisikan di luar fungsi mana pun, mereka memiliki cakupan global dan bukan cakupan lokal. Ini berarti mereka dapat diakses dari fungsi mana pun dalam *file*. Namun, variabel global lebih sulit untuk dilacak dan dapat diubah dari lokasi mana pun dalam program. Karena variabel global memiliki cakupan global, nama variabel yang sama tidak dapat digunakan kembali di bagian lain dari program Anda.