

FUNGSI

OBJEKTIF

1. Mahasiswa mampu memahami konsep fungsi pada C .
2. Mahasiswa mampu memahami konsep lingkup variabel pada C.
3. Mahasiswa mampu memahami konsep fungsi *built-in* pada C.

4.1 Apa itu fungsi?

4.1.1 Pengantar Fungsi

Fungsi berisi sejumlah statement yang dikemas dalam sebuah nama. Selain dapat membuat fungsi sendiri yang disebut dengan *user-defined functions*, terdapat fungsi yang sering kita gunakan sebelumnya. Fungsi ini merupakan fungsi yang terdapat pada Bahasa C (*built-in*), salah satu contohnya yaitu `printf()`.

Contoh:

```
#include <stdio.h>

int main()
{
    // Fungsi printf() akan mencetak kalimat "Hello world"
    printf ("Hello world");
    return 0;
}
```

Fungsi *built-in* maupun yang kita buat sendiri, selanjutnya dipanggil beberapa kali di beberapa tempat di dalam program.

Fungsi dalam pemrograman memiliki beberapa tujuan. Biasanya digunakan untuk memecah sebuah program besar menjadi beberapa bagian kecil, sehingga program dapat lebih mudah dikelola.

Contoh:

Perhatikan program berikut ini.

[illegible]

Program di atas dapat memecah program menjadi bagian-bagian kecil, seperti berikut:

```
int main()
{
    statement;
    ...
}
int fungsi1()
{
    statement;
    ...
}
int fungsi2()
{
    statement;
    ...
}
```

Selain itu, fungsi dapat mengurangi pengulangan penulisan kode-kode program (*code reuse*). Sehingga dapat menghemat ukuran program, manfaat ini akan terasa jika ada beberapa deretan instruksi yang sama digunakan pada beberapa tempat di dalam program. Dapat dilihat pada fungsi `main`, terdapat `fungsi1` dan `fungsi2` yang dapat dipanggil berulang kali. Kedua fungsi didapatkan dari fungsi yang telah didefinisikan.

```
int main()
{
    statement;
    fungsi1();
    fungsi2();
    statement;
    fungsi2();
    fungsi1();
    statement;

    return 0;
}
int fungsi1()
{
    statement;
    ...
}
int fungsi2()
{
    statement;
    ...
}
```

4.1.2 Fungsi *Prototype*

Sama seperti variabel yang sebelumnya harus dideklarasikan terlebih dahulu. Pada fungsi, deklarasi fungsi disebut sebagai fungsi *prototype*.

Berikut ini merupakan bentuk umum dari fungsi *prototype*:

```
tipe_return namaFungsi(parameter)
```

Keterangan:

- **tipe_return** merupakan tipe data nilai kembali. Fungsi biasanya akan mengembalikan sebuah nilai dari hasil prosesnya, karena itu kita harus menentukan tipe data yang akan dikembalikan.
- **namaFungsi** merupakan nama fungsi yang kita buat dan dapat dipanggil berulang-ulang.
- **parameter** merupakan variabel yang menyimpan nilai untuk diproses di dalam fungsi. Parameter akan menyimpan nilai yang akan di-inputkan ke dalam fungsi, pada fungsi bisa juga terdapat lebih dari satu parameter.

Contoh:

```
// Contoh fungsi dengan satu parameter
int fungsiLinear(int x)

// Contoh fungsi dengan dua parameter
double hitungLuas(double panjang, double lebar)
```

4.1.3 Mendefinisikan Fungsi dengan Nilai Kembali

Setelah melakukan deklarasi fungsi, selanjutnya yaitu melakukan definisi fungsi. Setiap fungsi yang dipanggil di dalam program harus didefinisikan, letaknya dapat di mana saja.

Definisi fungsi terdiri dari blok kode yang mampu melakukan beberapa tugas tertentu.

Berikut ini merupakan contoh definisi fungsi dari `fungsiLinear`:

```
// Fungsi dengan tipe data nilai kembali int
int fungsiLinear(int x)
{
    // Blok kode yang mampu melakukan beberapa tugas tertentu
    int y;
    y = (x + 3) * 2;

    return y;
}
```

Fungsi akan mengembalikan sebuah nilai dari hasil akhir berupa sebuah nilai yang disebut *return value* dengan tipe `integer`. Fungsi di atas dapat dipanggil dengan cara memanggil nama fungsinya, yaitu `fungsiLinear(argument);`

Apabila fungsi tersebut tidak memiliki nilai kembalian (*return*), maka kita dapat menggunakan `void` untuk menyatakan kalau fungsi tersebut tidak akan mengembalikan nilai apapun.

Contoh:

```
// Fungsi tanpa nilai kembali
void garis()
{
    // Blok kode yang mencetak kalimat, tanpa adanya nilai
    // kembali (return)
    printf("-----");
    printf("\n")
}
```

Pada blok kode di atas, tidak ada *statement return*, mengingat fungsi tidak memiliki nilai kembali. Dapat juga ditulis dengan *return* saja tanpa nilai. Fungsi di atas dapat dipanggil dengan cara memanggil nama fungsinya, yaitu `garis();`.

Contoh Program:

```
#include <stdio.h>

// Fungsi prototype
int fungsiLinear(int x);

int main()
{
    // Deklarasi variabel nilaix dan nilaiy
    int nilaix, nilaiy;
    printf("f(x) = (x + 3) * 2\n");

    // Menugaskan nilaix dengan 2
    nilaix = 2;

    // Memanggil fungsiLinear dengan nilai argumennya nilaix
    // yang akan ditugaskan ke variabel nilaiy
    nilaiy = fungsiLinear(nilaix);

    // Mencetak kalimat dan memanggil variabel nilaiy untuk
    // ditampilkan
    printf("f(2) = %d\n", nilaiy);

    return 0;
}

int fungsiLinear(int x)
{
    // Deklarasi variabel y untuk menyimpan hasil ekspresi
    int y = (x + 3) * 2;
    // Nilai kembali fungsi
    return y;
}
```

Program di atas akan menampilkan output seperti berikut ini:

$$\begin{aligned}f(x) &= (x + 3) * 2 \\f(2) &= 10\end{aligned}$$

4.2 Lingkup Variabel

Lingkup variabel berfungsi agar tidak terdapat kesalahan dalam menggunakan variabel dalam penulisan fungsi. Lingkup variabel juga akan menentukan keberadaan suatu variabel di dalam fungsi.

Terdapat variabel yang hanya dikenal di satu fungsi, namun ada variabel yang dapat diakses oleh semua fungsi. Lingkup variabel terbagi menjadi tiga bagian, yaitu:

- Variabel otomatis
- Variabel eksternal
- Variabel statis

4.2.1 Variabel Otomatis

Variabel otomatis adalah variabel yang dideklarasikan dalam suatu fungsi berlaku sebagai variabel lokal bagi fungsi, artinya variabel tersebut hanya dikenal di dalam fungsi tempat variabel dideklarasikan.

Contoh:

```
#include <stdio.h>

int alpha();

int main()
{
    // variabel lokal di dalam fungsi main()
    int x = 20;

    printf("Di main(): x = %d\n", x);
    alpha();

    return 0;
}

int alpha()
{
    // variabel lokal di dalam fungsi alpha()
    int x = 10;

    printf("Di alpha(): x = %d\n", x);

    return x;
}
```

Apabila program di atas dijalankan, maka outputnya adalah sebagai berikut:

```
Di main(): x = 20
Di alpha(): x = 10
```

4.2.2 Variabel Eksternal

Variabel eksternal adalah variabel yang dideklarasikan di luar fungsi yang juga dikenal sebagai variabel global, karena dikenal di semua fungsi.

Contoh:

```
#include <stdio.h>

int alpha();
// variabel global berada di luar fungsi
int y = 5;

int main()
{
    // variabel lokal di dalam fungsi main()
    int x = 20;

    printf("Di main(): x = %d\n", x);
    alpha();
    printf("Variabel global y = %d\n", y);

    return 0;
}

int alpha()
{
    // variabel lokal di dalam fungsi alpha()
    int x = 10;

    printf("Di alpha(): x = %d\n", x);

    // Increment pada variabel global di dalam fungsi alpha()
    y++;
    return x;
}
```

Apabila program di atas dijalankan, maka outputnya adalah sebagai berikut:

```
Di main(): x = 20
Di alpha(): x = 10
Variabel global y = 6
```

Setelah dijalankan, output yang dihasilkan untuk nilai variabel global menjadi 6. Hal ini disebabkan karena telah terjadi proses *increment* pada variabel global `y` di dalam fungsi `alpha()`, sehingga variabel `y` yang awalnya ditugaskan dengan nilai 5 setelah mengalami proses *increment* akan bertambah nilainya menjadi 6.

4.2.3 Variabel Statis

Variabel statis adalah variabel yang dideklarasikan dengan menambahkan *keyword* `static`, sebelum mendeklarasikan variabel.

Contoh:

```
static int mamamia = 0;
```

Variabel statis juga memiliki sifat seperti berikut:

- Variabel tetap yang hanya dapat dideklarasikan pada fungsi yang mendeklarasikannya.
- Nilai pada variabel tetap.
- Jika tidak ada inisialisasi variabel, maka variabel bernilai nol.

Contoh:

```
#include <stdio.h>

void SayaIngat()
int main()
{
    // variabel lokal pada fungsi main()
    int mama = 55;
    SayaIngat();
    SayaIngat();
    printf("Variabel mama pada main = %d", mama);
    return 0;
}

void sayaIngat()
{
    // variabel statis di dalam fungsi SayaIngat()
    static int mama = 0;

    // Increment pada variabel mama
    mama++;
    printf("variabel mama pada SayaIngat = %d\n", mama);
}
```

Apabila program di atas dijalankan, maka outputnya adalah sebagai berikut:

```
Variabel mama pada SayaIngat = 1
Variabel mama pada SayaIngat = 2
Variabel mama pada main = 55
```

Setelah dijalankan, terdapat perbedaan pada output variabel `mama` yang pertama dan kedua. Hal ini disebabkan karena pada fungsi `SayaIngat()` menggunakan variabel statis, pada variabel statis akan menyimpan nilai terakhir yang diberikan kepada variabel tersebut. Sehingga, pada pemanggilan fungsi `SayaIngat()` yang pertama, variabel `mama` yang sebelumnya ditugaskan dengan nilai `0` akan mengalami proses *increment* dan nilainya menjadi `1`.

Lalu, pada pemanggilan fungsi `SayaIngat()` yang kedua, nilainya tidak lagi ditugaskan dengan `0` melainkan ditugaskan dengan nilai `1` karena pada pemanggilan fungsi `SayaIngat()` yang pertama, kita telah melakukan *increment* dan variabel statis telah menyimpan nilai tersebut.

4.3 Fungsi *Built-in*

4.3.1 Fungsi Matematika

C menyediakan banyak *standard library* yang di dalamnya terkandung sejumlah fungsi (fungsi *built-in*). Fungsi *built-in* yang digunakan untuk operasi matematika, mengharuskan penggunaan *header file* `#include` ditulis sebelum fungsi `main()`.

Di bawah ini adalah beberapa fungsi yang disediakan di dalam library `math.h`:

Fungsi	Kegunaan	Contoh
<code>int abs(int x)</code>	Memberikan nilai balik berupa nilai absolut argumennya ($ x $)	<code>abs(-4) = 4</code>
<code>double floor(double x)</code>	Memperoleh bilangan bulat yang tidak lebih besar daripada sebelumnya	<code>floor(4.3) = 4.0</code>
<code>double round(double x);</code>	Membulatkan bilangan yang terdekat dengan argumennya	<code>round(3.5) = 4.0</code>
<code>double pow(double x, double y);</code>	Menghitung pakat	<code>pow(2, 3) = 8</code>
<code>double sqrt(double x)</code>	Menghitung akar pangkat	<code>sqrt(25) = 5</code>

Contoh:

```
#include <stdio.h>
// Library math.h untuk menyertakan fungsi matematika
#include <math.h>

int main()
{
    double angka, akar;

    printf ("Masukkan angka: ");
    scanf ("%lf", &angka);

    // Fungsi sqrt digunakan untuk menghitung akar kuadrat nilai
    // dari variabel angka
    akar = sqrt(angka);

    printf("Akar kuadrat dari %.2f", angka, akar)
}
```

Apabila program di atas dijalankan, maka outputnya adalah sebagai berikut:

```
Masukkan angka: 25
Akar kuadrat dari 25.00 = 5.00
```


4.3.2 Fungsi Penggenerasi Bilangan Acak

Pengacakan nilai (*random*) pada bahasa C adalah suatu metode untuk mendapatkan bilangan bulat secara acak yang terletak diantara 0 sampai dengan nilai max yang telah ditentukan. Untuk menyertakan fungsi pengacakan nilai, maka diperlukan library `stdlib.h`.

Di bawah ini merupakan macam-macam perintah pengacakan nilai yang ada ada pada library `stdlib.h`:

- Fungsi `rand()` adalah penulisan fungsi *prototype* dari fungsi `rand()` adalah `int rand();`. Dimana di dalam kurung tidak terdapat parameter.
- Fungsi `srand()` adalah fungsi yang hasil pengecekan nilainya berubah setiap detik, namun nilai dapat bersifat konstan apabila di dalam kurung diberikan parameter. Parameter di dalam `srand()` biasa disebut dengan *seed*. Penulisan fungsi *prototype* dari fungsi `srand()` adalah `srand(unsigned int seed);`.

Berikut adalah perbedaan fungsi `rand()` dan fungsi `srand()` dalam melakukan pengacakan nilai:

Fungsi <code>rand()</code>	Fungsi <code>srand()</code>
Digunakan untuk mendapatkan bilangan bulat secara acak yang terletak antara 0 sampai dengan nilai <code>RAND_MAX</code>	Digunakan untuk memberikan <i>seed</i> atau parameter bilangan acak. Dengan memberikan nilai <i>seed</i> , akan diperoleh nilai acak yang bersifat menetap. Contoh : <code>srand(10)</code>
Konstanta <code>RAND_MAX</code> didefinisikan di file <code>stdlib.h</code>	Jika batasan nilainya bersifat acak, dideklarasikan seperti: <code>srand(time(NULL));</code> (memerlukan library tambahan <code><time.h></code>)

Contoh:

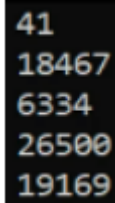
```
#include <stdio.h>
// Library stdlib.h digunakan untuk menyertakan fungsi rand() dan
// fungsi srand() ke dalam program
#include <stdlib.h>

int main()
{
    int i;

    for(i=0; i<5; i++)
        // Fungsi rand() digunakan untuk mengacak nilai
        printf("%d\n", rand());

    return 0;
}
```

Apabila program di atas dijalankan, maka outputnya adalah sebagai berikut:



```
41
18467
6334
26500
19169
```

Angka acak akan dicetak sebanyak 5 kali, dimana setiap baris atau setiap iterasinya akan menghasilkan angka acak yang berbeda.

Contoh:

```
#include <stdio.h>
// Library stdlib.h digunakan untuk menyertakan fungsi rand() dan
// fungsi srand() ke dalam program
#include <stdlib.h>

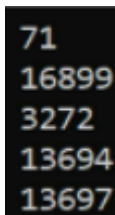
int main()
{
    // Fungsi srand(10) digunakan untuk menetapkan seed bilangan
    // acak bersifat tetap
    srand(10);

    int i;

    for(i=0; i<5; i++)
        // Fungsi rand() digunakan untuk mengacak nilai karena fungsi
        // srand() tidak dapat berjalan sendiri
        printf("%d\n", rand());

    return 0;
}
```

Apabila program di atas dijalankan, maka outputnya adalah sebagai berikut:



```
71
16899
3272
13694
13697
```

REFERENSI:

[1] Kadir, Abdul. 2015 . *From Zero to a Pro Pemrograman C*. ANDI