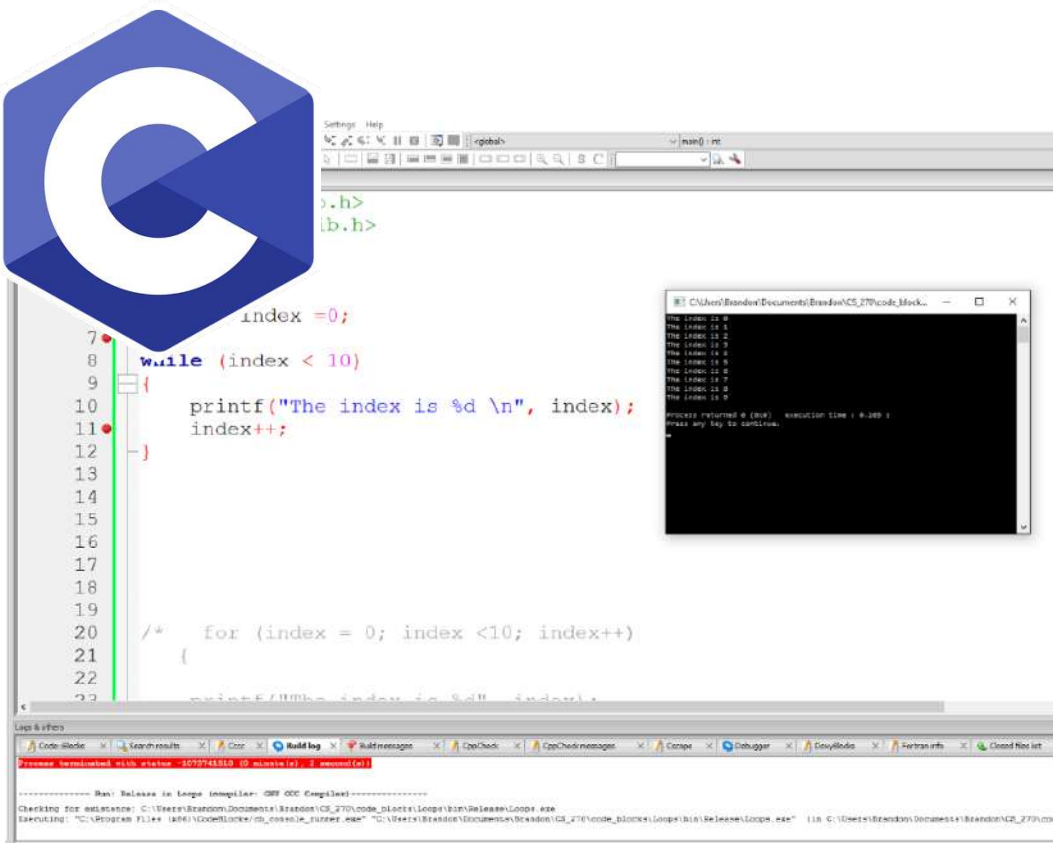
MODUL PRAKTIKUM DASAR PEMROGRAMAN V1.0

PROGRAM STUDI INFORMATIKA
UNIVERSITAS PEMBANGUNAN JAYA

PEMROGRAMAN BAHASA C

Prio Handoko, S,Kom., M.T.I.
Penyusun

MODUL PRAKTIKUM
DASAR-DASAR PEMROGRAMAN



Disusun Oleh:
Prio Handoko, S.Kom., M.T.I.

FAKULTAS DESAIN & TEKNOLOGI
PROGAM STUDI INFORMATIKA
UNIVERSITAS PEMBANGUNAN JAYA
TANGERANG SELATAN 2019

MODUL PRAKTIKUM I
STRUKTUR BAHASA C DAN I/O

Modul ini membahas mengenai struktur bahasa C dan sekaligus membahas I/O (input output) dalam pemrograman bahasa C.

Baris Komentar
Baris komentar adalah baris-baris yang menjelaskan maksud dari perubah yang digunakan atau maksud dari program itu sendiri. Hal ini dimaksudkan untuk memudahkan pelacakan atas perubah yang digunakan apabila program yang digunakan cukup besar atau memudahkan orang lain memahami program yang kita buat. Dalam program, baris komentar diletakkan diantara tanda /* dan */ dan baris ini tidak dikerjakan oleh komputer, hanya dianggap sebagai baris kosong.

Struktur Bahasa C
Bentuk program C mirip dengan kebanyakan program bahasa tingkat tinggi lainnya. Bentuk programnya adalah :

```
Judul Program
Header File
Deklarasi
Deskripsi
```

Judul Program
Judul program sifatnya sebagai dokumentasi saja, tidak signifikan terhadap proses program. Ditulis dalam bentuk baris komentar.
Syntax:
```
/*<judul program>*/
```

Contoh:
```
/* Program Menghitung Rata-Rata */
```

Header File
C menyediakan sejumlah file judul (*header file*) yaitu file yang umumnya berisi prototipe fungsi, definisi makro, variabel dan definisi tipe. File ini mempunyai ciri yaitu namanya diakhiri dengan extension .h (dot h).
Syntax:
```
#define <<nama header>.h>
```

Tabel 1. Daftar header file

Header File	Fungsi
<assert.h>	Melakukan diagnosa terhadap kesalahan program yang kita buat, yaitu dengan mengirimkannya ke dalam suatu file yang sering disebut dengan *standard error file*.
<ctype.h>	Melakukan pengecekan dan konversi terhadap sebuah karakter, *bukan* string. Fungsi-fungsi yang terdapat di dalam file header ini diawali dengan awalan **is...** dan **to...**.
<errno.h>	Mengatasi kesalahan-kesalahan (*error*) yang muncul di dalam program.
<float.h>	Menyimpan nilai-nilai konstan yang didefinisikan untuk melakukan perhitungan-perhitungan pada bilangan riil.
<limits.h>	Menyimpan makro yang merupakan nilai-nilai konstan yang siap pakai.
<math.h>	Mendefinisikan fungsi-fungsi yang digunakan untuk melakukan perhitungan- perhitungan matematika.
<signal.h>	Mengetahui signal atau tanda-tanda yang perlu dilaporakan dalam eksekusi sebuah program.
<stdarg.h>	membentuk fungsi yang terdiri dari parameter (argumen) dinamis, artinya banyaknya parameter dapat berubah-ubah dalam setiap fungsi tersebut dipanggil.
<stddef.h>	menyimpan beberapa variabel dan makro standar. Beberapa diantaranya juga dapat Anda temui di dalam file header yang lain.
<stdio.h>	Header ini adalah header yang hampir digunakan dalam semua program C, yaitu untuk melakukan proses input-output.
<stdlib.h>	menyimpan fungsi-fungsi yang akan digunakan untuk melakukan pengkonversian bilangan, alokasi memori dan pekerjaan-pekerjaan pemrograman lainnya yang sejenis.
<string.h>	menyimpan fungsi-fungsi yang digunakan untuk melakukan pemanipulasian string.
<time.h>	menyimpan fungsi-fungsi yang digunakan untuk melakukan pembacaan maupun pengkonversian terhadap waktu dan tanggal.

Contoh:

```
#include <stdio.h>
```

Keterangan: menyatakan bahwa agar membaca file bernama **stdio.h** saat pelaksanaan kompilasi.

Deklarasi
Deklarasi adalah bagian untuk mendefinisikan semua nama yang dipakai dalam program. Nama tersebut dapat berupa nama tetapan (konstanta), nama variabel, nama tipe, nama prosedur, nama fungsi.

Deskripsi
Bagian inti dari suatu program yang berisi uraian langkah-langkah penyelesaian masalah. Program C pada hakekatnya tersusun atas sejumlah blok fungsi. Sebuah program minimal mengandung sebuah fungsi. Setiap fungsi terdiri dari satu atau beberapa pernyataan, yang secara keseluruhan dimaksudkan untuk melaksanakan tugas khusus.

Bagian pernyataan fungsi (disebut tubuh fungsi) diawali dengan tanda "**{**" (kurung kurawal buka) dan diakhiri dengan tanda "**}**" (kurung kurawal tutup)

Variabel
Variabel dalam program digunakan untuk menyimpan suatu nilai tertentu dimana nilai tersebut dapat berubah-ubah. Setiap variabel mempunyai tipe dan hanya data yang bertipe sama dengan tipe variabel yang dapat disimpan di dalam variabel tersebut. Setiap variabel mempunyai nama. Pemisahan antar variabel dilakukan dengan memberikan tanda koma.
Syntax:
```
<nama_tipe_data> <nama_variabel>;
```

Contoh :
```
int jumlah;
float harga_per_unit, total_biaya;
```

Dari contoh diatas, variabel jumlah hanya boleh menerima data yang bertipe integer (bulat), tidak boleh menerima data bertipe lainnya. Variabel **`harga_per_unit`** dan **`total_biaya`** hanya bisa diisi dengan bilangan *float* (pecahan).

Konstanta
Berbeda dengan variabel yang isinya bisa berubah selama eksekusi program berlangsung, nilai suatu konstanta tidak bisa berubah.

Syntax:
Cara 1.
```
<nama_tipe_data> <nama_variabel> = <nilai_variabel>;
```

Cara 2.

```
#define <nama_variabel> <nilai_variabel>
```

Contoh :

```
const  int  m  =  8;
#define pajak 0.05
```

Fungsi main() atau int main() atau int main(void)

Fungsi main() harus ada pada program, karena fungsi inilah yang menjadi titik awal dan titik akhir eksekusi program. Tanda "**{**" di awal fungsi menyatakan awal tubuh fungsi sekaligus awal eksekusi program, sedangkan tanda "**}**" di akhir fungsi merupakan akhir tubuh fungsi dan sekaligus akhir eksekusi program.

Operasi Input dan Output

Berkenaan dengan mempelajari bahasa pemrograman C, tentunya tidak terlepas dari sebuah proses penting yang sering digunakan hampir di setiap program C yang dibuat, yaitu sebuah proses yang dinamakan masukan (input) dan keluaran (output). Proses ini penting karena dalam sebuah program terkadang pemrogram membutuhkan proses memasukkan data untuk mendapatkan sebuah keluaran tertentu. Input adalah sebuah proses menerima inputan dari pengguna dan output merupakan proses menampilkan data/nilai dari sebuah proses. Nilai yang dimasukkan dan ditampilkan oleh program nantinya akan disimpan dalam sebuah wadah penyimpanan nilai yang disebut dengan *variabel*. Tabel 2. di bawah ini memperlihatkan beberapa tipe data yang akan sering digunakan dalam bahasa C.

Tabel 2. Tabel tipe data

Jenis	Ukuran	Range	Format	Keterangan
Char	1 byte	-128 s/d 127	`%c`	Karakter/Huruf
Int	2 byte	-32768 s/d 32767	`%d`	Integer/Bilangan Bulat
Float	4 byte	-3.4E-38 s/d 3.4E+38	`%f`	Float/Bilangan Pecahan
Double	8 byte	-1.7E-308 s/d 1.7E+308	`%lf`	Pecahan Presisi Ganda

Fungsi Perintah Input

`scanf()`

Fungsi ini digunakan untuk menampilkan data yang dimasukkan dari keyboard. Guna membedakan antara setiap tipe data yang digunakan untuk melakukan proses input, maka dalam penulisannya perlu dituliskan simbol string kontrol yang tepat untuk mewakili tipe data yang digunakan.

Syntax:

```
scanf("<simbol_string_kontrol>", &<nama_variabel>);
```

Tabel 3. berikut menunjkkan beragam simbol string kontrol yang dapat dituliskan untuk mewakili tipe data yang digunakan.

Tabel 3. Simbol string kontrol

Simbol	Tipe Data
`%d`	Integer
`%c`	Char
`%s`	String
`%f`	Float
`%lf`	Double/Long Float
`%s`	String
`%[^\n]`	Until breakline. Menerima inputan hingga enter. (seperti gets)

Contoh:

```
/*Contoh penggunaan fungsi scanf()*/
#include <stdio.h>
void main()
{
	int angka;
	scanf("%d", &angka);
	fflush(stdin); //fflush digunakan untuk membersihkan buffer.
}
```

Keterangan: Jika menggunakan **`scanf()`**, pada penulisan perintahnya harus disertai dengan simbol "**`&`**" (dan) di depan variabel penampung. Hal ini akan dibahas nanti saat Alasannya akan dijelaskan saat mempelajari tentang *pointer*.

`gets()`

Fungsi yang digunakan untuk menerima inputan berupa kata/kalimat yang akan dibaca sampai negasi enter (**`\n`**).

Syntax:

```
gets(<nama_variabel>);
```

Contoh:

```
/*Contoh penggunaan fungsi gets()*/
#include <stdio.h>
void main()
{
	char nama[20];
	gets(nama);
	fflush(stdin); //fflush digunakan untuk membersihkan buffer.
}
```

getchar()

Fungsi yang digunakan untuk menerima inputan berupa karakter.

Syntax:

```
getchar();
```

Contoh:

```
/*Contoh penggunaan fungsi getchar()*/
#include <stdio.h>
void main()
{
      char grade;
      grade = getchar();
      fflush(stdin); //fflush digunakan untuk membersihkan buffer.
}
```

Fungsi Perintah Ouput

printf()

Merupakan fungsi yang digunakan untuk menampilkan output ke layar. Dengan menggunakan fungsi ini, tampilan dapat diatur (diformat) dengan mudah.

Syntax :

Opsi 1.

```
printf(”string”);
```

Contoh:

```
/*Program Opsi 1*/
#include<stdio.h>
int main()
{
     printf(”Belajar Pemrograman Komputer”);
     getchar();
}
```

Opsi 2.

```
printf(”string string_kontrol”, argumen);
```

Contoh:

```
/*Program Opsi 2*/
#include<stdio.h>
int main()
{
     int nilai;
     printf("Nilai saya : %d", nilai); getchar();
}
```

String kontrol dapat berupa keterangan beserta penentu format (seperti %d, %f). Argumen adalah data yang akan ditampilkan, dapat berupa variabel, konstanta, maupun ungkapan.

puts()

Merupakan fungsi untuk menampilkan string (kata dan kalimat) disertai dengan breakline (**\n**).

Syntax:

```
puts("string/kalimat");
```

Contoh:

```
/* Program penggunaan fungsi puts()*/
#include <stdio.h>
int main()
{
     puts("Halo, selamat datang di Prodi Informatika");
     puts("Selamat belajar giat ☺ ");
     getchar();
}
```

putchar()

Merupakan fungsi untuk menampilkan karakter.

Syntax:

```
putchar("nama_variabel");
```

Contoh:

```
/*Program penggunaan fungsi putchar()*/
#include <stdio.h>
void main()
{
     char grade;

     //proses input grade
     printf("Grade = ");
     scanf("%c", &grade);
     fflush(stdin);

     //proses output
     printf("Grade anda  : ");
     putchar(grade);
     getchar();
}
```

LATIHAN.

Buatlah program-program berikut.

1. Menampilkan:
   A. tulisan **"Halo, siapa namamu?"**, lalu meminta pengguna memasukkan namanya;
   B. tulisan **"Berapa usiamu?"**, lalu meminta pengguna memasukkan usianya, ;
   C. tulisan **"Dimanakah tempat tinggalmu?"**, lalu meminta pengguna memasukkan alamat tempat tinggalnya;
   D. tulisan **"Dimanakah tempat kuliahmu?"**, lalu meminta pengguna memasukkan nama kampus tempat kuliah;
   E. tulisan **"Program studimu apa?"**, lalu meminta pengguna memasukkan nama program studinya;

   dan akhirnya menuliskan pesan "**Hallo <nama>, senang berteman denganmu. Usiamu sekarang sudah <usia> tahun ya? Makin keren aja kamu, apalagi sekarang kamu kuliah di <nama kampus> di program studi <nama prodi>. Rumah kamu di <alamat rumah> kan? Kapan-kapan kita pergia sama-sama ya ke kampus? Aku juga mahasiswa <nama kampus>**".

   Keterangan:
   <nama> , **<usia>**, **<nama kampus>**, **<nama prodi>**, dan **<alamat rumah>** adalah data yang dibaca dari hasil input sebelumnya.

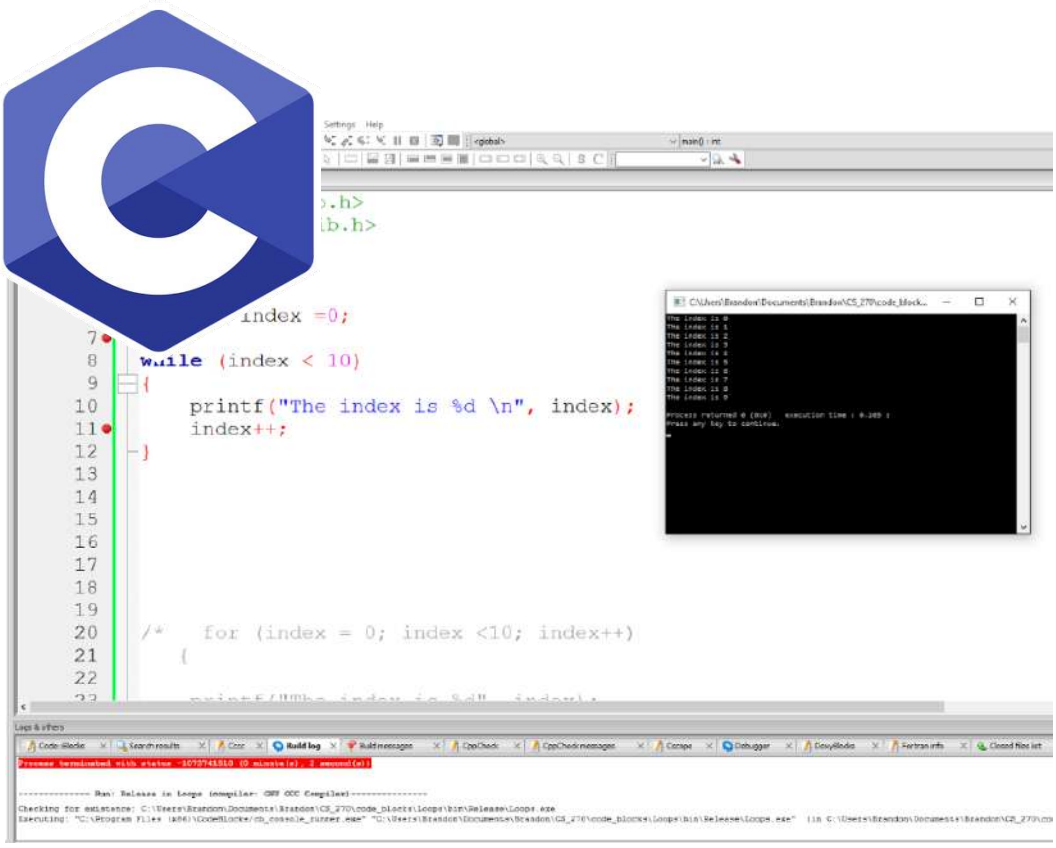
2. Menukarkan dua buah nilai dari dua buah variabel. Misalnya, sebelum pertukaran nilai a=5, nilai b=3, maka setelah pertukaran nilai a=3, nilai b=5.

3. Menghitung luas dan keliling persegi panjang. Data masukan dibaca dari piranti masukan dan luas dan keliling bangun persegi panjang ditampilkan sebagai keluaran.

4. Mengkonversikan total detik menjadi jam menit detik. Petunjuk: 1 menit = 60 detik dan 1 jam = 3600 detik.

5. Menampilkan hasil konversi jarak x dalam km ke dalam meter dan sentimeter dimana setiap kali hasil konversi akan dimunculkan didahului dengan penekanan tombol Enter. Ingat bahwa 1 m = 100 cm dan 1 km = 1000 m = 100.000 cm.

MODUL PRAKTIKUM
DASAR-DASAR PEMROGRAMAN



Disusun Oleh:
Prio Handoko, S.Kom., M.T.I.

FAKULTAS DESAIN & TEKNOLOGI
PROGAM STUDI INFORMATIKA
UNIVERSITAS PEMBANGUNAN JAYA
TANGERANG SELATAN 2019

MODUL PRAKTIKUM 2
OPERATOR

Dalam kegiatan pemrograman kita selalu dilibatkan dalam kesibukan melakukan operasi-operasi tertentu, misalnya seperti perhitungan-perhitungan matematik, pemanipulasian *string*, pemanipulasian bit ataupun operasi-operasi lainnya. Untuk melakukan hal tersebut tentu kita harus menguasai dengan benar akan penggunaan operator-operator yang digunakan dalam suatu bahasa pemrograman tertentu. Operator itu sendiri adalah tanda yang digunakan untuk meyelesaikan suatu operasi tertentu.

Operator Assignment

Operator *assignment* adalah suatu operator penugasan yang digunakan untuk memasukkan nilai ke dalam suatu variabel. Dalam bahasa C, operator *assignment* ini dilambangkan dengan tanda sama dengan (**=**). Untuk lebih memahaminya, perhatikan contoh sintak berikut.

```
C = 15;
```

Sintak tersebut berarti memasukkan nilai 15 ke dalam variabel **C**. Selain itu bahasa C juga menawarkan kita untuk memasukkan nilai ke dalam beberapa variabel secara sekaligus, berikut ini contoh yang akan menunjukkan hal tersebut.

```
a = b = c = 20;
```

Contoh di atas memperlihatkan bahwa nilai 20 dimuatkan ke dalam variabel **a**, **b** dan **c**. Berikut ini program yang akan menunjukkan penggunaan operator *assignment*.

```
#include   <stdio.h>
int main(void)
{
     int a, b, c, d;
     a = 10;
     b = c = 35;
     d = a;
     printf("Nilai a \t= %d\n", a);
     printf("Nilai b \t= %d\n", b);
     printf("Nilai c \t= %d\n", c);
     printf("Nilai d \t= %d\n", d);
     return 0;
}
```

Berbeda dengan kebanyakan bahasa pemrograman lainnya, bahasa C juga menawarkan cara penulisan sintak untuk mempersingkat proses *assignment*. Sebagai contoh apabila terdapat statemen `j = j + 4`, maka *statement* tersebut dapat kita tulis dengan `j += 4`. Berikut ini bentuk penyingkatan yang terdapat dalam bahasa C.

Tabel 1. Bentuk penyingkatan *statement* operasi aritmatika

Contoh Statemen	Bentuk Penyingkatan
`J = J + 4`	`J += 4`
`J = J - 4`	`J -= 4`
`J = J * 4`	`J *= 4`
`J = J / 4`	`J /= 4`
`J = J % 4`	`J %= 4`
`J = J << 4`	`J <<= 4`
`J = J >> 4`	`J >>= 4`
`J = J & 4`	`J &= 4`
`J = J ^ 4`	`J ^= 4`
`J = J | 4`	`J |= 4`

Operator Unary

Operator *unary* adalah operator yang digunakan untuk melakukan operasi-operasi matematik yang hanya melibatkan satu buah *operand*. Dalam bahasa C, yang termasuk ke dalam operator *unary* adalah seperti yang tampak pada tabel di bawah ini.

Tabel 2. Operator *unary*

Operator	Jenis Operasi	Contoh Penggunaan
`+`	Membuat nilai positif	`+10`
`-`	Membuat nilai negatif	`-10`
`++`	Increment *(menambahkan nilai 1)*	`x++`
`--`	Decrement *(mengurangi nilai 1)*	`x--`

Berikut ini program yang akan menunjukkan penggunaan operator unary **+** dan **-**.

```
#include <stdio.h> int main(void)
{
int x, y;
x = +5;
y = -10;
printf("%d x (%d) = %d", x, y, x*y);
return 0;
}
```

Increment

Increment adalah suatu proses menaikkan (menambahkan) nilai dengan nilai 1. Adapun operator dalam bahasa C yang digunakan untuk melakukan proses tersebut adalah operator **++**. Maka dari itu operator **++** ini disebut dengan *operator increment*. Sebagai contoh apabila kita memiliki variabel **x** yang bertipe int dengan nilai 10, maka setelah operasi **++x** atau **x++**, maka nilai **x** akan bertambah satu, yaitu menjadi 11.

Dalam bahasa C, *increment* terbagi lagi ke dalam dua bagian, yaitu *pre-increment* dan *post-increment*. Berikut ini penjelasan dari masing-masing topik tersebut.

Pre-Increment

Pre-increment berarti menaikkan nilai yang terdapat pada sebuah variabel sebelum nilai dari variabel tersebut diproses di dalam program. Operator **++** akan dianggap sebagai *pre-increment* apabila dituliskan di depan nama variabel atau nilai yang akan dinaikkan. Sebagai contoh, misalnya kita memiliki variabel **x** bertipe int dengan nilai 10 dan di sini kita akan melakukan pemrosesan terhadap variabel tersebut dengan cara menampilkan nilainya ke layar monitor. Apabila kita melakukan operasi *pre-increment*, yaitu dengan menuliskan **++x**, maka nilai **x** akan dinaikkan terlebih dahulu sebelum nilai tersebut ditampilkan ke layar. Hal ini menyebabkan nilai yang akan ditampilkan adalah 11. Untuk membuktikannya, coba perhatikan program di bawah ini.

```
#include   <stdio.h>
int main(void)
{
  int x=10;
  printf("Nilai  x  awal  \t=  %d\n",  x);
  printf("Nilai   ++x   \t=   %d\n",   ++x);
  printf("Nilai x akhir \t= %d\n",x);
  return 0;
}
```

Post-Increment

Post-increment berarti menaikkan nilai yang terdapat pada sebuah variabel setelah nilai dari variabel tersebut diproses di dalam program. Pada *post-increment* operator **++** ditulis setelah variabel atau nilai yang akan dinaikkan. Sebagai contoh, misalkan kita memiliki variabel **x** yang bernilai 10, maka nilai **x++** yang akan ditampilkan di layar adalah 10 (bukan 11). Kenapa demikian? Hal ini disebabkan karena nilai dari variabel **x** tersebut akan ditampilkan terlebih dahulu, selanjutnya baru dinaikkan nilainya. Untuk lebih jelasnya, perhatikan kembali program di bawah ini.

```
#include <stdio.h>
int main(void)
{
  int x=10;
  printf("Nilai  x  awal  \t=  %d\n",  x);
  printf("Nilai   x++   \t=   %d\n",   x++);
  printf("Nilai x akhir \t= %d\n",x);
  return 0;
}
```

Decrement

Decrement merupakan kebalikan dari *increment*, yang merupakan proses penurunan nilai dengan nilai 1. *Decrement* juga dibagi menjadi dua macam, yaitu *pre-decrement* dan *post-decrement*. Namun di sini kita tidak akan membahas tentang kedua jenis *decrement* tersebut karena konsepnya sama persis dengan *pre-increment* dan *post- increment*. Untuk lebih memahaminya, berikut ini program yang akan menunjukkan penggunaan operator *decrement*.

```
#include <stdio.h>
int main(void)
{
  int x=10, y=10;

  printf("Nilai x awal \t= %d\n", x);
  printf("Nilai --x \t= %d\n", --x);
  printf("Nilai x akhir \t= %d\n\n", x);
  printf("\nNilai y awal \t= %d\n", y);
  printf("Nilai y-- \t= %d\n", y--);
  printf("Nilai y akhir \t= %d\n", y);
  return 0;
}
```

Operator Binary

Dalam ilmu matematika, operator *binary* adalah operator yang digunakan untuk melakukan operasi yang melibatkan dua buah *operand*. Pada pembahasan ini, kita

akan mengelompokkan operator *binary* ini ke dalam empat jenis, yaitu operator *aritmetika*, *logika*, *relasional* dan *bitwise*.

Operator Aritmatika

Operator aritmetika adalah operator yang berfungsi untuk melakukan operasi-operasi aritmetika seperti penjumlahan, pengurangan, perkalian dan pembagian. Berikut ini operator aritmetika yang terdapat dalam bahasa C.

Tabel 3. Operator Aritmatika

Operator	Jenis Operasi	Contoh Penggunaan
+	Penjumlahan	12 + 13 = 25
-	Pengurangan	15 – 12 = 13
*	Perkalian	12 * 3 = 36
/	Pembagian	10.0 / 3.0 = 3.3333
%	Sisa bagi (modulus)	10 % 4 = 2

Agar Anda dapat lebih memahaminya, di sini dituliskan beberapa program yang akan menunjukkan penggunaan dari masing-masing operator tersebut.

Menggunakan Operator Penjumlahan ("+")

```
#include <stdio.h>
int main(void)
{
   int a=12, b=13;
   int c;
   c = a + b;
   printf("%d + %d = %d", a, b, c);
   return 0;
}
```

Menggunkaan Operator Pengurangan ("-")

```
#include <stdio.h>
int main(void)
{
   int a=25, b=12;
   int c;
   c = a - b;
   printf("%d - %d = %d", a, b, c);
   return 0;
}
```

Menggunakan Operator Perkalian ("*")

```
#include <stdio.h>
int main(void)
{
   int a=12, b=3;
   int c;
   c = a * b;
   printf("%d x %d = %d", a, b, c);
   return 0;
}
```

Menggunakan Operator Pembagian ("/")

Dalam menggunakan operator `/`, kita harus memperhatikan tipe data dari *operand* yang kita definisikan. Pasalnya, perbedaan tipe data akan menyebabkan hasil yang berbeda pula. Apabila kita mendefinisikan *operand* dengan tipe data riil (**`float`** atau **`double`**), maka hasilnya juga berupa data riil. Misalnya **`10.0 / 3.0`**, maka hasil yang akan didapatkan adalah **`3.3333`**. Sedangkan apabila kita mendefinisikan *operand* dengan tipe data bilangan bulat (**`int`** atau **`long`**), maka nilai yang dihasilkan juga berupa bilangan bulat (tanpa memperdulikan sisa baginya). Misalnya **`10 / 3`**, maka hasil yang didapatkan adalah **`3`**. Adapun nilai 1 yang merupakan sisa bagi dari operasi pembagian tersebut akan diabaikan oleh program. Berikut ini contoh program yang akan membuktikan hal tersebut.

```
#include <stdio.h>
int main(void)
{
   int a=10, b=3, c;
   double x = 10.0, y = 3.0, z;
   c = a / b;
   z = x / y;
   printf("%d  /  %d  \t\t  =  %d\n",  a,  b,  c);
   printf("%.1f / %.1f \t = %.4f\n", x, y, z);
   return 0;
}
```

Menggunakan Operator Modulo/Modulus ("%")

Operator `%` ini digunakan untuk mendapatkan sisa bagi dari operasi pembagian pada bilangan bulat. Misalnya **`10 % 3`**, maka hasilnya adalah **`1`**. Nilai tersebut merupakan sisa bagi yang diperoleh dari proses pembagian 10 dibagi 3. Berikut ini contoh program yang menunjukkan penggunaan operator `%`.

```
#include <stdio.h>
int main(void)
{
   printf("%2d %s %d \t = %d\n", 10, "%", 3, (10 % 3));
   printf("%2d %s %d \t = %d\n", 8, "%", 3, (8 % 3));
   printf("%2d %s %d \t = %d\n", 15, "%", 4, (15 % 4));
   return 0;
}
```

Operator Logika

Operator logika adalah operator digunakan di dalam operasi yang hanya dapat menghasilkan nilai benar (true) dan salah (false). Nilai seperti ini disebut dengan nilai boolean. Berbeda dengan bahasa pemrograman lain (misalnya Pascal), bahasa C tidak menyediakan tipe data khusus untuk merepresentasikan nilai boolean tersebut. Dalam bahasa C, nilai benar akan direpresentasikan dengan menggunakan nilai selain nol. Namun, pada kenyataannya para programmer C umumnya menggunakan nilai 1 untuk merepresentasikan nilai benar. Adapun nilai salah akan direpresentasikan dengan nilai 0. Untuk memudahkan dalam proses penulisan sintak, maka biasanya nilai-nilai tersebut dijadikan sebagai makro dengan sintak berikut.

```
#define TRUE   1
#define FALSE  0
```

Adapun yang termasuk ke dalam operator logika dalam bahasa C adalah seperti yang tampak dalam tabel berikut.

Tabel 4. Operator logika

Operator	Jenis Operasi	Contoh Penggunaan
`&&`	AND (dan)	`1 && 0 = 0`
`||`	OR (atau)	`1 || 0 = 1`
`!`	NOT (negasi / ingkaran)	`!1 = 0`

`Operator AND ("&&")`

Operasi AND akan memberikan nilai benar apabila semua operand-nya bernilai benar, selain itu operasi ini akan menghasilkan nilai salah. Berikut ini tabel yang menyatakan nilai yang dihasilkan dari operasi AND.

Tabel 5. Tabel kebenaran operator AND ("**&&**")

X	Y	X && Y
1	1	1
1	0	0
0	1	0
0	0	0

Berikut ini contoh program yang akan menunjukkan penggunaan operator **&&**.

```
#include <stdio.h>
int main(void)
{
   printf("1 && 1 = %d\n", (1 && 1));
   printf("1 && 0 = %d\n", (1 && 0));
   printf("0 && 1 = %d\n", (0 && 1));
   printf("0 && 0 = %d\n", (0 && 0));
   return 0;
}
```

Operator OR ("||")

Operasi OR akan menghasilkan nilai salah apabila semua operand-nya bernilai salah, selain itu operasi tersebut akan menghasilkan nilai benar. Berikut ini tabel yang menyatakan hasil dari operasi OR.

Tabel 6. Tabel kebenaran operator OR ("**||**")

X	Y	X || Y
1	1	1
1	0	1
0	1	1
0	0	0

Berikut ini contoh program yang akan membuktikan hal tesebut.

```
#include <stdio.h>
int main(void)
{
   printf("1 || 1 = %d\n", (1 || 1));
   printf("1 || 0 = %d\n", (1 || 0));
   printf("0 || 1 = %d\n", (0 || 1));
   printf("0 || 0 = %d\n", (0 || 0));
   return 0;
}
```

Operator NOT ("!")

Operasi NOT merupakan operasi yang menghasilkan nilai ingkaran (negasi) dari nilai operand yang diberikan. Berikut ini tabel yang menyatakan hasil dari opearsi NOT.

Tabel 7. Tabel kebenaran operator NOT ("!")

X	!X
1	0
0	1

Adapun contoh program yang akan menunjukkan hal tersebut adalah sebagai berikut.

```
#include <stdio.h>
int main(void)
{
   printf("!1 = %d\n", (!1));
   printf("!0 = %d\n", (!0));
   return 0;
}
```

Operator Operasional

Operator relasional adalah operator yang digunakan untuk menentukan relasi atau hubungan dari dua buah nilai atau operand. Operator ini terdapat dalam sebuah ekspresi yang selanjutnya akan menentukan benar atau tidaknya ekspresi tersebut. Berikut ini operator yang termasuk ke dalam operator relasional dalam bahasa C.

Operator ini pada umumnya digunakan untuk melakukan pengecekan terhadap ekspresi di dalam blok pemilihan, yang baru akan kita bahas pada bab selanjutnya, yaitu bab 4 – Kontrol Program. Namun sebagai gambaran bagi Anda, berikut ini contoh program yang akan menunjukkan penggunaan salah satu dari operator di atas.

Tabel 8. Tabel operator operasional

Operator	Jenis Operasi	Contoh Penggunaan
==	Sama dengan	(8 == 8) = 1
>	Lebih besar	(8 > 9) = 0
<	Lebih kecil	(8 < 9) = 1
>=	Lebih besar atau sama dengan	(8 >= 7) = 0
<=	Lebih kecil atau sama dengan	(7 <= 8) = 1
!=	Tidak sama dengan	(8 != 9) = 1

Operator ini pada umumnya digunakan untuk melakukan pengecekan terhadap ekspresi di dalam blok pemilihan, yang baru akan kita bahas pada bab selanjutnya, yaitu bab 4 – Kontrol Program. Namun sebagai gambaran bagi Anda, berikut ini contoh program yang akan menunjukkan penggunaan salah satu dari operator di atas.

```
#include <stdio.h>
int main(void)
{
  int x;
  printf("Masukkan sebuah bilangan bulat : ");
  scanf("%d", &x);
  if (x % 2 == 0 )
  {
    printf("%d adalah bilangan genap", x);
   } else
      {
        printf("%d adalah bilangan ganjil", x);
      }
  return 0;
}
```

Operator Bitwise

Operator bitwise digunakan untuk menyelesaikan operasi-operasi bilangan dalam bentuk biner yang dilakukan bit demi bit. Dengan kata lain, operator *bitwise* ini berfungsi untuk melakukan pemanipulasian bit. Operasi ini merupakan hal vital apabila program yang kita buat kan melakukan interaksi dengan perangkat keras (hardware). Meskipun bahasa pemrograman lebih bersifat data-oriented, namun perangkat keras masihlah bersifat bit-oriented. Ini artinya, perangkat keras menginginkan input dan output data yang dilakukan terhadapnya tetap dalam bentuk bit tersendiri.

Perlu ditekankan di sini bahwa operasi pemanipulasian bit ini hanya dapat dilakukan pada bilangan-bilangan yang bertipe char dan int saja karena keduanya dapat berkoresponden dengan tipe byte dan word di dalam bit. Adapun yang termasuk ke dalam operator bitwise di dalam bahasa C adalah seperti yang tertera dalam tabel di bawah ini.

Tabel 9. Tabel operator bitwise

Operator	Jenis Operasi	Contoh Penggunaan
`&`	Bitwise AND	`1 & 1 = 1`
`|`	Bitwise OR	`1 | 0 = 1`
`^`	*Bitwise XOR (Exclusive OR)*	`1 ^ 1 = 0`
`~`	Bitwise Complement (NOT)	`~0 = 1`
`>>`	*Shift right (geser kanan)*	`4 << 1 = 8`
`<<`	Shift left (geser kiri)	`4 >> 1 = 2`

Fungsi dari operator **&**, | dan **~** di atas sebenarnya sama dengan fungsi operator logika **&&**, **||** dan **!**. Perbedaannya hanya operator bitwise ini melakukan operasinya bit demi bit, sedangkan operator logika melakukan operasi pada nilai totalnya. Sebagai contoh apabila kita melakukan operasi logika 7 **||** 8, maka hasil yang akan didapatkan adalah 1, pasalnya nilai 7 dan 8 akan dianggap sebagai nilai benar (true) sehingga operasi OR tersebut juga akan menghasilkan nilai true yang direpresentasikan dengan nilai 1. Namun, jika kita melakukan operasi bitwise 7 **|** 8, maka nilai 7 dan 8 tersebut akan dikonversi ke dalam bilangan biner, setelah itu baru dilakukan operasi OR untuk setiap bit-nya. Proses ini dapat kita representasikan dengan cara berikut.

0 0 0 0 1 0 0 0 nilai 8 dalam bentuk biner
0 0 0 0 0 1 1 1 nilai 7 dalam bentuk biner
---------- |
0 0 0 0 1 1 1 1 hasil = 15

Cara kerja dari operator **&** dan **~** juga sama seperti di atas. Untuk itu di sini kita tidak akan membahas lebih detil tentang kedua operator tersebut. Adapun operator lain yang perlu Anda ketahui di sini adalah operator **^** (bitwise XOR), >> (shift right) dan << (shift left). Penjelasan dari masing-masing operator tersebut dapat Anda lihat dalam subbab di bawah ini.

Operator Exclusive-OR ("^")

Operasi XOR (exlusive OR) akan memberikan nilai benar apabila hanya terdapat satu buah operand yang bernilai benar, selain itu akan menghasilkan nilai salah. Dengan demikian, apabila kedua operand-nya bernilai benar, operasi ini tetap akan menghasilkan nilai salah. Berikut ini tabel yang menunjukkan hasil dari operasi XOR.

Tabel 10. Tabel kebenaran XOR (**^**)

X	Y	X ^ Y
1	1	0
1	0	1
0	1	1
0	0	0

Sebagai contoh apabila kita ingin melakukan operasi 45 ^ 23, maka hasilnya adalah 58. Berikut ini proses yang menunjukkan operasi tersebut.

0 0 1 0 1 1 0 1 nilai 45 dalam bentuk biner
0 0 0 1 0 1 1 1 nilai 23 dalam bentuk biner
---------- ^
0 0 1 1 1 0 1 0 hasil = 58

Anda dapat membuktikannya ke dalam sebuah program sederhana seperti di bawah ini.

```
#include <stdio.h>
#define X  0x2D /* nilai 45 dalam bentuk heksadesimal */
#define Y  0x17 /* nilai 23 dalam bentuk heksadesimal */

int main(void)
{
  printf("%d ^ %d = %d", (X ^ Y));
  return 0;
}
```

Operator Shift-Right (">>")

Operator *shift-right* (geser kanan) ini digunakan untuk melakukan penggeseran bit ke arah kanan sebanyak nilai yang didefinisikan. Apabila terdapat operasi X **>>** 3 berarti melakukan penggeseran 3 bit ke kanan dari nilai X yang telah dikonversi ke dalam bilangan biner. Adapun bentuk umum dari penggunaan operator **>>** adalah sebagai berikut.

```
nilai >> banyaknya_pergeseran_bit_ke_arah_kanan
```

Untuk memudahkan Anda dalam menentukan hasil yang diberikan dari operasi ini, ingatlah bahwa setiap proses pergeseran bit yang terjadi, operator >> akan membagi suatu nilai dengan 2. Sebagai contoh 128 **>>** 1, maka hasil yang akan didapatkan

adalah 64. Sedangkan 128 **>>** 2 akan menghasilkan nilai 32, begitu seterusnya.

Berikut ini contoh program yang akan membuktikan hal tersebut.

```
#include <stdio.h>
#define X 0x80   /* nilai 128 dalam bentuk heksadesimal */
int main(void)
{
  printf("%d >> 1 = %d\n", X, (X>>1));
  printf("%d >> 2 = %d\n", X, (X>>2));
  printf("%d >> 3 = %d\n", X, (X>>3));
  printf("%d >> 4 = %d\n", X, (X>>4));
  printf("%d >> 5 = %d\n", X, (X>>5));
  printf("%d >> 6 = %d\n", X, (X>>6));
  printf("%d >> 7 = %d\n", X, (X>>7));
  return 0;
}
```

Berikut ini tabel yang akan mengilustrasikan proses yang terjadi dalam program di atas.

Tabel 11. Tabel proses *shift-right* ("**>>**")

Nilai X	X dalam bilangan biner	Hasil
X = 128	1 0 0 0 0 0 0 0	128
X = 128 >> 1	0 1 0 0 0 0 0 0	64
X = 128 >> 2	0 0 1 0 0 0 0 0	32
X = 128 >> 3	0 0 0 1 0 0 0 0	16
X = 128 >> 4	0 0 0 0 1 0 0 0	8
X = 128 >> 5	0 0 0 0 0 1 0 0	4
X = 128 >> 6	0 0 0 0 0 0 1 0	2
X = 128 >> 7	0 0 0 0 0 0 0 1	1

Operasi Shift-Left ("<<")

Operator *shift-left* (geser kiri) merupakan kebalikan dari operator **>>**, artinya di sini kita melakukan pergeseran bit ke arah kiri sebanyak nilai yang didefinisikan. Berikut ini bentuk umum penggunaan operator **<<**.

```
nilai << banyaknya_pergeseran_bit_ke_arah_kiri
```

Dalam setiap pergeseran bit-nya, operator ini akan mengalikan suatu nilai dengan 2. Misalnya 1 **<<** 1, maka hasil yang akan didapatkan adalah 2 (berasal dari 1 x 2).

Sedangkan 1 **<<** 2 akan memberikan hasil 4 (berasal dari 1 x 2 x 2), 1 **<<** 3 memberikan hasil 8 (berasal dari 1 x 2 x 2 x 2), begitu seterusnya. Perhatikan program di bawah ini yang akan membuktikan hal tersebut.

```
#include <stdio.h>
#define X 0x01   /* nilai 1 dalam bentuk heksadesimal */

int main(void)
{
  printf("%d << 1 = %d\n", X, (X<<1));
  printf("%d << 2 = %d\n", X, (X<<2));
  printf("%d << 3 = %d\n", X, (X<<3));
  printf("%d << 4 = %d\n", X, (X<<4));
  printf("%d << 5 = %d\n", X, (X<<5));
  printf("%d << 6 = %d\n", X, (X<<6));
  printf("%d << 7 = %d\n", X, (X<<7));
  return 0;
}
```

Untuk mengilustrasikan proses yang terjadi dalam program atas, perhatikanlah tabel di bawah ini.

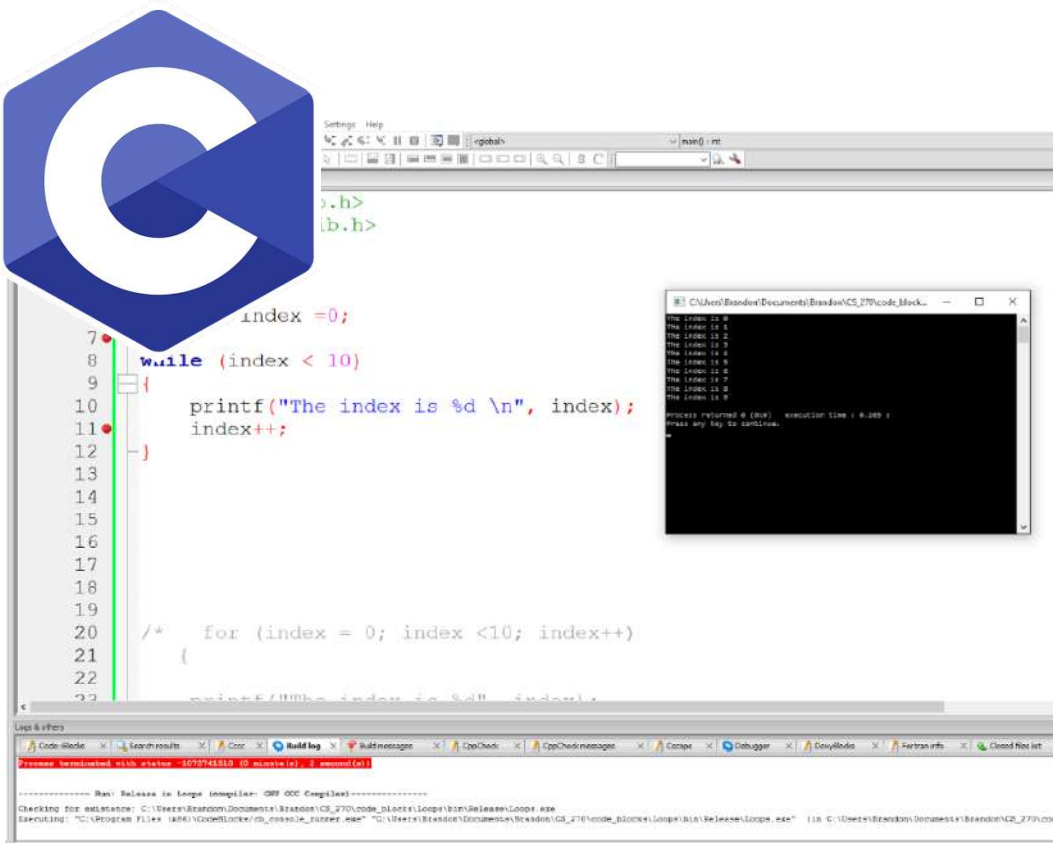
Tabel 12. Tabel operasi *shift-left* ("**<<**")

Nilai X	X dalam bentuk bilangan biner	Hasil
X = 1	0 0 0 0 0 0 0 1	1
X = 1 << 1	0 0 0 0 0 0 1 0	2
X = 1 << 2	0 0 0 0 0 1 0 0	4
X = 1 << 3	0 0 0 0 1 0 0 0	8
X = 1 << 4	0 0 0 1 0 0 0 0	16
X = 1 << 5	0 0 1 0 0 0 0 0	32
X = 1 << 6	0 1 0 0 0 0 0 0	64
X = 1 << 7	1 0 0 0 0 0 0 0	128

LATIHAN

1. Perhatikan contoh-contoh program pada pembahasan mengenai semua jenis operator aritmatika. Ketiklah contoh-contoh progran tersebut menggunakan IDE ***code::blocks*** amati kemudian tentukan output dari setiap contoh program tersebut.
2. Buatlah program untuk menyelesaikan operasi aritmatika berikut.
   a. `A + B – C`
   b. `A * ((B – C) / D)`
3. Berdasarkan contoh program dan tabel kebenaran untuk operator logika AND, OR dan NOT, maka buatlah program untuk menentukan apakah operasi logika di bawah ini bernilai 1 (TRUE) atau 0 (FALSE).
   a. `A && B || C`
   b. `A' ! ((B && C') || D)`

MODUL PRAKTIKUM
DASAR-DASAR PEMROGRAMAN



Disusun Oleh:
Prio Handoko, S.Kom., M.T.I.

FAKULTAS DESAIN & TEKNOLOGI
PROGAM STUDI INFORMATIKA
UNIVERSITAS PEMBANGUNAN JAYA
TANGERANG SELATAN 2019

MODUL PRAKTIKUM 3
SELEKSI KONDISI (IF)

Setelah mempelajari mengenai variabel dan jenis-jenisnya, tipe data, operator serta ekspresi yang telah dibahas pada bab sebelumnya, maka bahasan berikutnya yang akan dipelajari adalah bagaimana cara melakukan seleksi kondisi terhadap program di dalam bahasa C. Membuat sebuah program dalam bahasa C tentunya kita harus mengetahui terlebih dahulu aturan-aturan pemrograman yang berlaku, seperti melakukan pemilihan statemen yang didasarkan atas kondisi tertentu maupun melakukan pengulangan statemen di dalam program. Setelah mempelajari materi ini, diharapkan pemahaman mengenai konsep seleksi kondisi dalam bahasa C dengan baik, sehingga mampu mengimplementasikannya di dalam kasus-kasus program yang dihadapi.

Pemilihan/Seleksi Kondisi

Dalam kehidupan sehari-hari, kadang kala kita disudutkan pada beberapa pilihan atau harus memilih salah satu dari opsi yang ada dimana pilihan-pilihan tersebut didasarkan atas kondisi tertentu. Dengan kata lain, pilihan tersebut hanya dapat dilakukan apabila kondisi telah terpenuhi. Sebagai contoh, perhatikan statemen di bawah ini.

Jika Gunawan memiliki banyak uang, maka ia akan membeli mobil mewah

Pada statemen di atas, Gunawan akan dapat membeli mobil mewah hanya apabila ia memiliki banyak uang. Hal ini berarti apabila ternyata Gunawan tidak memiliki banyak uang (kondisi tidak terpenuhi), maka Gunawan pun tidak akan pernah membeli mobil mewah. Begitupun di dalam bahasa pemrograman, kita juga dapat melakukan pemilihan statemen yang akan dieksekusi, yaitu dengan melakukan pengecekan terhadap kondisi tertentu yang didefinisikan. Adapun kondisi yang dimaksud di dalam program tidak lain adalah suatu ekspresi. Dalam bahasa C, pemilihan statemen dapat dilakukan melalui dua buah cara, yaitu dengan menggunakan statemen **`if`** dan statemen **`switch`**.

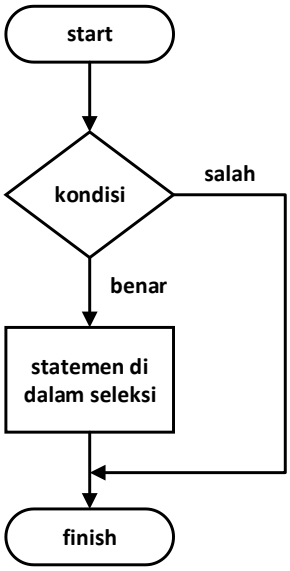
Statement **`if`**

Untuk memudahkan pembahasan, di sini kita akan mengklas**`if`**ikasikan pemilihan dengan menggunakan statemen **`if`** tersebut ke dalam tiga bagian, yaitu pemilihan yang didasarkan atas satu kasus, dua kasus dan lebih dari dua kasus.

`Satu Kasus`

Pemilihan jenis ini adalah pemilihan yang paling sederhana karena hanya mengandung satu kondisi yang akan diperiksa. Berikut ini gambar yang akan menunjukkan konsep

dari pemilihan yang didasarkan atas satu kasus.



Gambar 1. Flowchart seleksi kondisi **`if`** untuk satu kasus

Pada Gambar 1. di atas terlihat bahwa mula-mula program akan mengecek kondisi yang didefinisikan. Apabila kondisi bernilai benar (kondisi terpenuhi), maka program akan melakukan statemen-statemen yang terdapat di dalam blok pengecekan. Namun apabila ternyata kondisi bernilai salah (kondisi tidak terpenuhi), maka program akan langsung keluar dari blok pengecekan dengan melanjutkan eksekusi terhadap statemen-statemen berikutnya di luar blok pengecakan (jika ada).

Adapun bentuk umum atau kerangka dari blok pemilihan menggunakan statemen **`if`** untuk satu kasus di dalam bahasa C adalah seperti yang tampak di bawah ini.

```
if (kondisi)
     Statemen_yang_akan_dieksekusi;
```

Bentuk umum di atas berlaku apabila Anda hanya memiliki sebuah statemen di dalam blok pengecekan. Namun, apabila Anda memiliki dua statemen atau lebih, maka bentuk umumnya menjadi seperti di bawah ini.

```
if (kondisi) {
     Statemen_yang_akan_dieksekusi1;
     Statemen_yang_akan_dieksekusi2;
     ...
}
```

Perlu sekali untuk diperhatikan bahwa dalam bahasa C, kondisi harus diapit oleh tanda kurung. Selain itu bahasa C juga tidak memiliki kata kunci then seperti yang terdapat pada kebanyakan bahasa pemrograman lainnya, misalnya bahasa Pascal.

Untuk lebih memahami konsep yang terdapat di dalamnya, perhatikan program berikut.

```
#include   <stdio.h>

int main(void)

{
  int x;
  printf("Masukkan sebuah bilangan bulat : "); scanf("%d", &x);
  if (x > 0)
    printf("\n%d adalah bilangan positif\n", x);

  printf("\nStatemen di luar blok kontrol pengecekan");
  return 0;
}
```

Tugas 1.

Tuliskan program di atas kemudian lakukan kompilasi dan jalankan program tersebut, kemudian masukkan sembarang nilai **x**. *Amatilah, lalu tuliskan keluaran program tersebut pada logbook Anda!*

```
Masukkan sebuah bilangan bulat : 10

10 adalah bilangan positif

Statemen di luar blok kontrol pengecekan
```

Dari hasil di atas dapat kita lihat bahwa nilai x sama dengan 10 dan ekspresi **(10 > 0)** bernilai benar. Hal ini tentu menyebabkan statemen di dalam blok pengecekan akan dieksekusi oleh program. Namun apabila Anda memasukkan nilai **x** dengan nilai 0 atau negatif (misalnya –5), maka hasil yang akan diberikan adalah sebagai berikut.

```
Masukkan sebuah bilangan bulat : 10
Statemen di luar blok kontrol pengecekan
```

Sekarang, statemen di dalam blok pengecekan tidak ikut dieksekusi. Hal ini

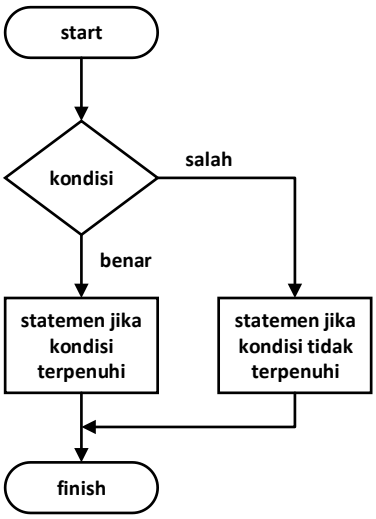
disebabkan karena ekspresi **(-5 > 0)** bernilai salah sehingga program akan langsung keluar dari blok pengecekan.

Sebagai contoh penggunaan statemen if, di sini kita akan membuat program untuk menentukan apakah suatu tahun merupakan tahun kabisat atau bukan. Adapun sintaknya adalah sebagai berikut.

```
#include <stdio.h>
int main(void)
{
  long tahun;
  printf("Masukkan tahun yang akan diperiksa : ");
  scanf("%ld", &tahun);

  //Mengecek tahun kabisat
  if ((tahun % 4 == 0) &&((tahun % 100 != 0) ||
     (tahun % 400 = 0)))

  printf("\n%ld merupakan tahun kabisat", tahun);

  return 0;
}
```

Dua Kasus

Bentuk pemilihan ini merupakan perluasan dari bentuk pertama, hanya saja di sini didefinisikan pula statemen yang akan dilakukan apabila kondisi yang diperiksa bernilai salah (tidak terpenuhi). Adapun cara yang digunakan untuk melakukan hal tersebut adalah dengan menambahkan kata kunci else di dalam blok pengecekan. Hal ini menyebabkan statemen untuk pemilihan untuk dua kasus sering dikenal dengan statemen **if-else**. Berikut ini gambar yang akan menunjukkan konsep dari pemilihan yang didasarkan atas dua kasus.



Gambar 2. Flowchart seleksi kondisi **if** untuk dua kasus

Pada Gambar 2. di atas terlihat jelas bahwa pada pemilihan untuk dua kasus, apabila kondisi yang diperiksa tidak terpenuhi maka terlebih dahulu program akan mengeksekusi sebuah (atau lebih) statemen sebelum program melanjutkan eksekusi ke statemen-statemen berikutnya di luar atau setelah blok pengecekan.

Bentuk umum atau kerangka yang digunakan dalam bahasa C untuk melakukan pemilihan dua kasus adalah sebagai berikut.

```
if (kondisi)
  Statemen_jika_kondisi_benar; /* Ingat, harus menggunakan
                                  tanda titik koma */
else
  Statemen_jika_kondisi_salah;
```

Bentuk umum di atas dilakukan apabila statemen yang kita definisikan untuk sebuah nilai kondisi tertentu (benar atau salah) hanya terdiri dari satu statemen. Namun apabila kita akan mendefinisikan lebih dari satu statemen, maka bentuk umumnya adalah sebagai berikut.

```
if (kondisi)
{ Statemen_jika_kondisi_benar1; Statemen_jika_kondisi_benar2;
  …
} else
  {  Statemen_jika_kondisi_salah1; Statemen_jika_kondisi_salah2;
     …
  }
```

Sebagai contoh termudah untuk menunjukkan penggunaan statemen **if-else** di dalam program adalah untuk menentukan suatu bilangan bulat apakah merupakan

```
#include   <stdio.h>
int main(void)
{
  int x;
  printf("Masukkan bilangan bulat yang akan diperiksa : ");
  scanf("%d", &x);

  if (x % 2 == 0)
    printf("%d  merupakan  bilangan  GENAP", x);
  else
    printf("%d merupakan bilangan GANJIL", x);

  return 0;
}
```

bilangan genap atau ganjil. Adapun sintak programnya adalah sebagai berikut.

Contoh hasil yang akan diberikan dari program tersebut adalah sebagai berikut.

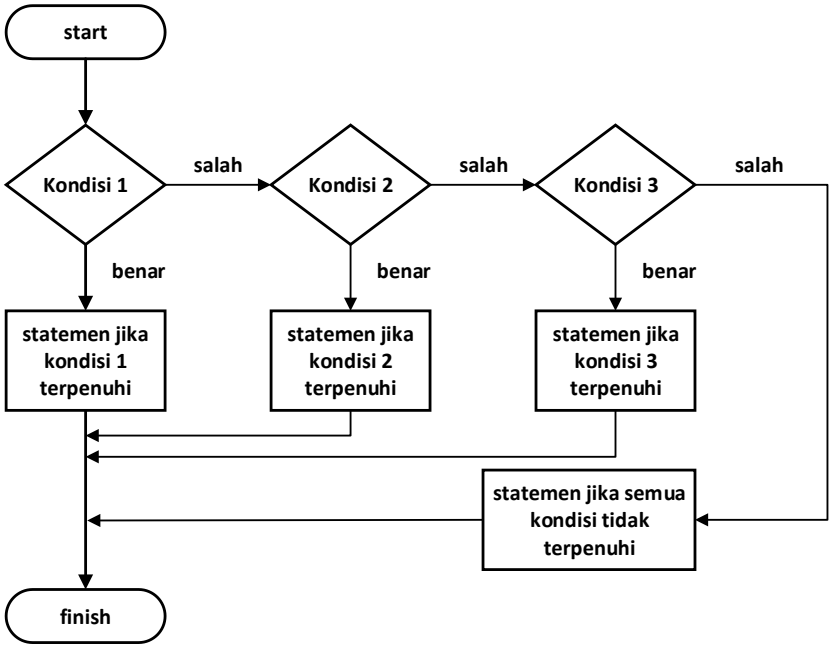
```
Masukkan bilangan bulat yang akan diperiksa : 5
5 merupakan bilangan GANJIL
```

Perhatikan kembali sintak program di atas, di sana tertulis ekspresi **`(x % 2 == 0)`** yang berarti melakukan pengecekan terhadap suatu bilangan yang dibagi dengan 2, apakah sisa baginya sama dengan 0 atau tidak. Apabila bilangan tersebut habis dibagi 2 (artinya sisa = 0) maka bilangan tersebut merupakan bilangan genap, sebaliknya apabila sisa bagi = 1 maka bilangan tersebut termasuk ke dalam bilangan ganjil. Pada kasus ini, kita melakukan pengecekan terhadap bilangan 5. Kita tahu bahwa 5 dibagi 2 menghasilkan nilai 2 dan sisa baginya 1, ini berarti ekspresi di atas bernilai salah (tidak terpenuhi) dan menyebabkan program akan mengeksekusi statemen yang berada pada bagian else, yaitu statemen di bawah ini.

```
printf("%d merupakan bilangan GANJIL", x);
```

Lebih Dari Dua Kasus

Pada pemilihan jenis ini kita diizinkan untuk menempatkan beberapa (lebih dari satu) kondisi sesuai dengan kebutuhan program yang akan kita buat. Berikut ini gambar yang akan menunjukkan konsep dari pemilihan statemen yang didasarkan atas tiga kasus atau lebih.



Gambar 3. Flowchart seleksi kondisi **`if`** untuk tiga kasus atau lebih

Gambar 3. di atas, mula-mula program akan melakukan pengecekan terhadap kondisi1. Apabila kondisi1 benar, maka program akan langsung mengeksekusi statemen yang didefinisikan di dalamnya. Namun, apabila kondisi1 bernilai salah maka program akan melakukan pengecekan terhadap kondisi2. Apabila kondisi2 juga bernilai salah maka program akan melanjutkan ke pengecekan kondisi3. Apabila ternyata kondisi3 juga bernilai salah maka program akan mengeksekusi statemen alternatif yang didefinisikan, yaitu statemen yang terdapat pada bagian akhir blok pengecekan (pada bagian **`else`**). Adapun bentuk umum dari pemilihan yang melibatkan tiga buah kasus atau lebih adalah sebagai berikut.

```
if (kondisi 1) { Statemen_jika_kondisi_1_benar;
  …
} if else (kondisi 2){Statemen_jika_kondisi_2_benar;
   …
  }if else (kondisi 3){Statemen_jika_kondisi_3_benar;
      …
     }else {Statemen_jika_semua_kondisi_salah;}
```

Sebagai contoh sederhana untuk mengimplementasikan pemilihan yang didasarkan atas tiga kasus adalah pembuatan program untuk menentukan wujud air yang berada pada suhu tertentu. Adapun ketentuan-ketentuannya adalah sebagai berikut.

suhu ≤ 0	air akan berwujud padat (es)
0 < suhu < 100	air akan berwujud cair
suhu ≥ 100	air akan berwujud gas

Apabila dituliskan dalam bentuk program, maka sintaknya adalah seperti yang tertera di bawah ini.

```
#include <stdio.h>
int main(void)
{
  int suhu;
  printf("Masukkan besarnya suhu : "); scanf("%d", suhu);

  if (suhu <= 0) { printf("Pada suhu %d derajat Celcius, air akan
  berwujud " \ "padat (es)", suhu);
  } else if ((suhu > 0) && (suhu < 100)) { printf("Pada suhu %d
  derajat Celcius, air akan berwujud " \ "cair", suhu);
  } else {printf("Pada suhu %d derajat Celcius, air akan berwujud
  " \ "gas", suhu);
  }

  return 0;
}
```

Berikut ini contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan besarnya suhu : 28
Pada suhu 28 derajat Celcius, air akan berwujud cair
```

Statement **switch**

Statemen **switch** digunakan untuk melakukan pemilihan terhadap ekspresi atau kondisi yang memiliki nilai-nilai konstan. Oleh karena itu, ekspresi yang didefinisikan harus menghasilkan nilai yang bertipe bilangan bulat atau karakter. Untuk mendefinisikan nilai-nilai konstan tersebut adalah dengan menggunakan kata kunci **case**. Hal yang perlu Anda perhatikan juga dalam melakukan pemilihan dengan menggunakan statamen **switch** ini adalah kita harus menambahkan statemen **break** pada setiap nilai yang kita definisikan.

Untuk lebih memahaminya, coba Anda perhatikan bentuk umum dari statemen **switch** di bawah ini.

```
switch (ekspresi) {
  case nilai_konstan1:
  {
    Statemen_yang_akan_dieksekusi;
    … break;
  }
  case nilai_konstan2:
  {
    Statemen_yang_akan_dieksekusi;
    … break;
  }
  … default:
  {
    Statemen_alternatif;    /* apabila semua nilai di
                               atas tidak terpenuhi */
  }
}
```

Kata kunci default di atas berguna untuk menyimpan statemen alternatif, yang akan dieksekusi apabila semua nilai yang didefinisikan tidak ada yang sesuai dengan ekspresi ada.

Berikut ini contoh program yang akan menunjukkan penggunaan statemen switch untuk melakukan suatu pemilihan nilai.

```
#include <stdio.h>
int main(void)
{
  int nohari;
  printf("Masukkan nomor hari (1-7): "); scanf("%d", &nohari);

  switch (nohari)
  {
    case 1: printf("Hari ke-%d adalah hari Minggu", nohari);
            break;
    case 2: printf("Hari ke-%d adalah hari Senin", nohari);
            break;
    case 3: printf("Hari ke-%d adalah hari Selasa", nohari);
            break;
    case 4: printf("Hari ke-%d adalah hari Rabu", nohari);
            break;
    case 5: printf("Hari ke-%d adalah hari Kamis", nohari);
            break;
    case 6: printf("Hari ke-%d adalah hari Jumat", nohari);
            break;
    case 7: printf("Hari ke-%d adalah hari Sabtu", nohari);
            break;
    default: printf("Nomor hari yang dimasukkan salah");
  }

  return 0;
}
```

Program di atas berguna untuk menentukan nama hari dari nomor hari yang dimasukkan. Adapun contoh hasil yang akan diberikan dari program di atas adalah sebagai berikut.

```
Masukkan nomor hari : 6
Hari ke-6 adalah hari Jumat
```

Apabila kita ingin mendefinisikan satu blok statemen yang dapat digunakan untuk beberapa nilai konstan, maka kita dapat menuliskan sintaknya seperti di bawah ini.

```
switch (ekspresi) {
  case nilai1: case nilai2: case nilai3:
  {
    /* Statemen yang berlaku untuk nilai1, nilai2 dan nilai3 */
    break;
  }
  case nilai4: case nilai5:
  {
    /* Statemen yang berlaku untuk nilai4 dan nilai5 */
    break;
  }
  ...
}
```

Untuk lebih jelasnya, di sini ini kita akan membuat program untuk menentukan bulan tertentu masuk ke dalam caturwulan ke berapa. Sebelumnya kita asumsikan bahwa bulan 1-4 (Januari sampai April) termasuk ke dalam caturwulan 1, bulan 5-8 (Mei sampai Agustus) termasuk ke dalam caturwulan 2 dan bulan 9-12 (September sampai Desember) masuk ke dalam caturwulan 3.

Adapun contoh sintak untuk mengimplementasikan kasus ini adalah seperti di bawah ini.

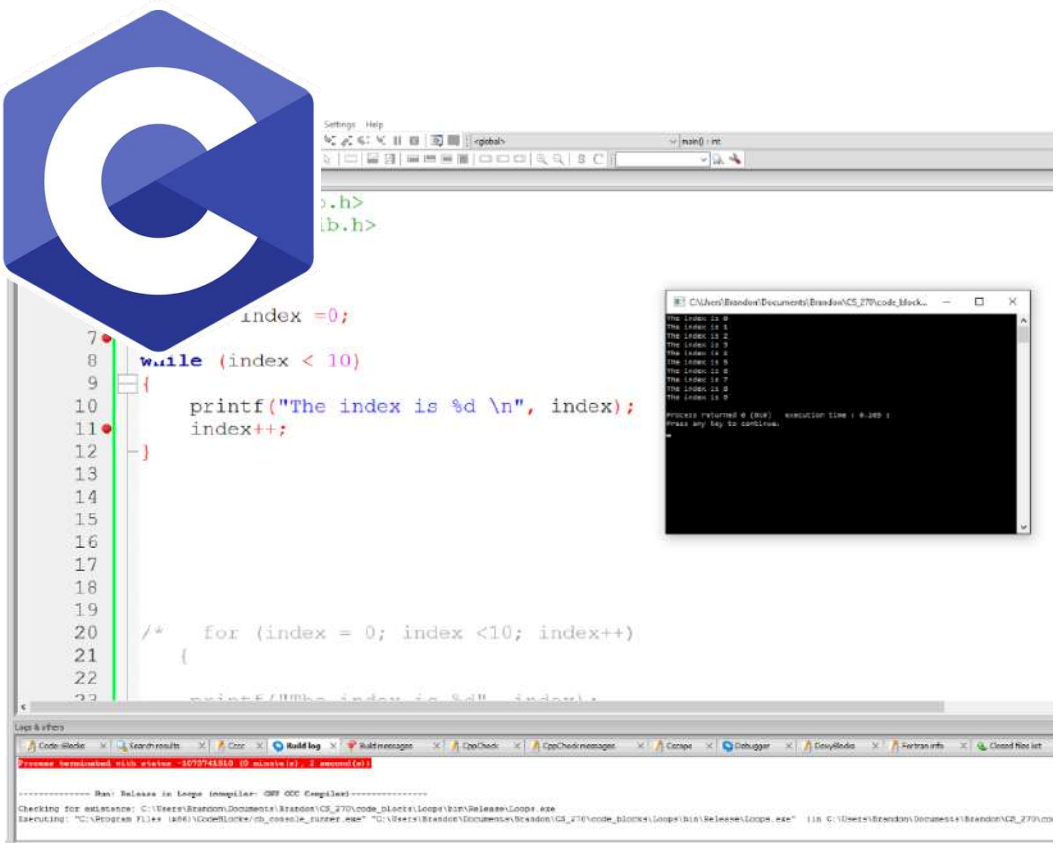
```
#include <stdio.h>

/* Mendeklarasikan array konstan untuk nama bulan */
const char namabulan[][12] =
{"Januari","Februari","Maret","April","Mei","Juni","Juli",
"Agustus", "September","Oktober","November","Desember"};

int main(void)
{
   int nobulan;
   printf("Masukkan nomor bulan (1-12) : "); scanf("%d",
   &nobulan);

switch (nobulan)
   {
     case 1: case 2: case 3: case 4:
             printf("Bulan %s termasuk ke dalam caturwulan 1",
             namabulan[nobulan-1]);
             break;
      case 5: case 6: case 7: case 8:
             printf("Bulan %s termasuk ke dalam caturwulan 2",
             namabulan[nobulan-1]);
             break;
      case 9: case 10: case 11: case 12:
             printf("Bulan %s termasuk ke dalam caturwulan 3",
             namabulan[nobulan-1]);
             break;
     default: printf("Nomor bulan yang dimasukkan salah");
   }

   return 0;
}
```

Apabila program di atas dijalankan dan kita memasukkan nomor bulan dengan nilai 1, 2, 3 atau 4 maka statemen yang akan dieksekusi adalah statamen yang didefinisikan untuk blok nilai-nilai tersebut. Begitu juga dengan nilai 5, 6, 7 dan 8 serta nilai 9, 10, 11 dan 12. Berikut ini contoh hasil yang akan diberikan dari program di atas.

```
Masukkan nomor bulan : 10
Bulan Oktober termasuk ke dalam caturwulan 3
```

MODUL PRAKTIKUM

DASAR-DASAR PEMROGRAMAN



Disusun Oleh:
Prio Handoko, S.Kom., M.T.I.

FAKULTAS DESAIN & TEKNOLOGI
PROGAM STUDI INFORMATIKA
UNIVERSITAS PEMBANGUNAN JAYA
TANGERANG SELATAN 2019

MODUL PRAKTIKUM 4
SELEKSI KONDISI (LOOPING)

Dalam pembuatan program, terkadang kita harus melakukan pengulangan suatu aksi, misalnya untuk melakukan perhitungan berulang dengan menggunakan **for**mula yang sama. Sebagai contoh, misalnya kita ingin membuat program yang dapat menampilkan teks "**Saya sedang belajar bahasa C**" sebanyak 10 kali, maka kita tidak perlu untuk menuliskan 10 buah statemen melainkan kita hanya tinggal menempatkan satu buah statemen ke dalam suatu struktur pengulangan. Dengan demikian program kita akan lebih efesien.

Sebagai gambaran bagi Anda untuk dapat lebih menyerap konsep pengulangan, coba Anda perhatikan terlebih dahulu contoh program di bawah ini.

```
#include   <stdio.h>
int main(void)
{
  /* Mencetak teks ke layar sebanyak 10 kali */
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya sedang belajar bahasa C");

  return 0;
}
```

Dalam bahasa C, terdapat tiga buah struktur pengulangan yang akan digunakan sebagai kontrol dalam melakukan pengulangan proses, yaitu struktur **for**, **while** dan **do-while**.

Setiap struktur yang ada masing-masing memiliki aturan tersendiri dan digunakan dalam konteks yang berbeda. Kita harus bisa menentukan kapan sebaiknya kita harus menggunakan struktur **for**, struktur while ataupun do-while. Untuk itu, pada bagian ini kita akan membahas secara detil apa perbedaan dan penggunaan dari masing-masing struktur tersebut. Hal ini juga bertujuan agar Anda dapat mengimplementasikannya dengan benar ke dalam kasus-kasus program yang Anda hadapi.

Struktur `for`

Struktur **`for`** ini digunakan untuk menuliskan jenis pengulangan yang banyaknya sudah pasti atau telah diketahui sebelumnya. Oleh karena itu, di sini kita harus melakukan inisialisai nilai untuk kondisi awal pengulangan dan juga harus menuliskan kondisi untuk menghentikan proses pengulangan. Adapun bentuk umum dari pendefinisian struktur **`for`** (untuk sebuah statemen) dalam bahasa C adalah sebagai berikut.

```
for (ekpresi1; ekspresi2; ekspresi3)
  statemen_yang_akan_diulang;
```

Sedangkan untuk dua statemen atau lebih.

```
for (ekpresi1; ekspresi2; ekspresi3)
{
  Statemen_yang_akan_diulang1;
  Statemen_yang_akan_diulang2;
  …
}
```

Ekspresi1 di atas digunakan sebagai proses inisialisasi variabel yang akan dijadikan sebagai pencacah (counter) dari proses pengulangan, dengan kata lain ekspresi ini akan dijadikan sebagai kondisi awal.

Ekspresi2 digunakan sebagai kondisi akhir, yaitu kondisi dimana proses pengulangan harus dihentikan. Perlu untuk diketahui bahwa pengulangan masih akan dilakukan selagi kondisi akhir bernilai benar.

Ekspresi3 digunakan untuk menaikkan (*increment*) atau menurunkan (*decrement*) nilai variabel yang digunakan sebagai pencacah. Apabila pengulangan yang kita lakukan bersifat menaik, maka kita akan menggunakan statemen increment, sedangkan apabila pengulangan yang akan kita lakukan bersifat menurun maka kita harus menggunakan statemen decrement.

Berikut ini contoh untuk mengilustrasikan struktur pengulangan **`for`** yang telah diterangkan di atas.

```
for (int j=0; j<10; j++)
{
  /* Statemen yang akan diulang */
  ...
}
```

Pada sintak di atas, mula-mula kita menginisialisasi variabel **j** dengan nilai 0, kemudian karena ekspresi (0 < 10) bernilai benar maka program akan melakukan statemen untuk pertama kalinya. Setelah itu variabel j akan dinaikkan nilainya sebesar 1 melalui statemen increment **j**++sehingga nilai j sekarang menjadi 1. Sampai di sini program akan mengecek ekspresi (**j** < 10). Oleh karena ekspresi (2 < 10) bernilai benar, maka program akan melakukan statemen yang kedua kalinya. Begitu seterusnya sampai nilai **j** bernilai 9. Namun pada saat variabel j telah bernilai 10 maka program akan keluar dari proses pengulangan. Hal ini disebabkan oleh karena ekspresi (10 < 10) bernilai salah.

Untuk membuktikan hal tersebut, perhatikan contoh program di bawah ini dimana kita akan menampilkan teks "**Saya sedang belajar bahasa C**" sebanyak 10 kali.

```
#include   <stdio.h>
int main(void)
{
  for (int j=0; j<10; j++)
  {
    printf("Saya sedang belajar bahasa C\n");
  }

  return 0;

}
```

Pengulangan yang dilakukan pada program di atas bersifat menaik sehingga kita menggunakan increment. Kita juga dapat melakukan pengulangan tersebut secara menurun, yaitu dengan sintak di bawah ini.

```
#include   <stdio.h>
int main(void) {
  for (int j=10; j>0; j--) {
    printf("Saya sedang belajar bahasa C\n");
  }

  return 0;
}
```

Pada sintak di atas, mula-mula variabel **j** bernilai 10 dan setiap pengulangan dilakukan menyebabkan variabel tersebut dikurangi satu. Hal ini disebabkan karena statemen decrement **j--** di atas. Dalam program tersebut, pengulangan baru akan dihentikan ketika variabel j bernilai 0. Apabila dijalankan program di atas akan memberikan hasil yang sama dengan program sebelumnya.

Indeks dari variabel yang digunakan sebagai nilai awal dalam struktur **for** tidak selalu harus bernilai 0, artinya kita dapat memanipulasinya sesuai dengan keinginaan kita (misalnya dengan nilai 1, 2, 3 ataupun lainnya).

```
for (int j=1; j<=5; j++)
{
  /* Statemen yang akan diulang */
  ...
}
```

atau dapat juga dituliskan sebagai berikut.

```
for (int j=10; j>=5; j--)
{
  /* Statemen yang akan diulang */
  ...
}
```

Selain tipe **int**, kita juga dapat menggunakan variabel yang bertipe **char** sebagai pencacah dalam proses pengulangan. Sebagai contoh apabila kita akan melakukan pengulangan sebanyak 3 kali, maka kita dapat menuliskannya sebagai berikut.

```
for (char j='a'; j<='c'; j++)
{
  /* Statemen yang akan diulang */
  ...
}
```

Berikut ini contoh program yang akan menunjukkan hal tersebut.

```
#include   <stdio.h>
int main(void)
{
  for (char j='A'; j<='E'; j++)
  {
    printf("%c = %d\n", j, j);
  }
  return 0;
}
```

Melakukan Beberapa Inisialisasi dalam Struktur `for`

Dalam bahasa C, kita diizinkan untuk melakukan banyak inisialisasi di dalam struktur `for`. Hal ini tentu akan dapat mengurangi banyaknya baris kode program. Adapun cara untuk melakukan hal ini, yaitu dengan menambahkan tanda koma (,) di dalam bagian inisialisasi, seperti yang tertulis di bawah ini.

```
int a, b;
for (a=0, b=0; a<5; a++, b +=5)
{
  ...
}
```

Sintak di atas sama seperti penulisan sintak berikut.

```
int a, b;
b = 0;
for (a=0; a<5; a++)
{
  ...
  b += 5;  /* dapat ditulis dengan b = b + 5 */
}
```

Untuk membuktikannya, perhatikan dua buah contoh program di bawah ini.

a. Menggunakan dua inisialisasi

```
#include   <stdio.h>

int main(void) {

  for (int a=0, int b=0; a < 5; a++, b += 5)
  {
    printf("Baris ke-%d : a = %d, b = %2d\n", a+1, a, b);
  }

  return 0;
}
```

b. Menggunakan satu inisialisasi

```
#include   <stdio.h>
int main(void)
{
  int b=0;
  for (int a=0, a < 5; a++)
  {
    printf("Baris ke-%d : a = %d,   b = %2d\n", a+1, a, b);
    b += 5;
  }

  return 0;
}
```

Struktur `for` Bersarang (*Nested for*)

Dalam pemrograman sering kali kita dituntut untuk melakukan proses pengulangan di dalam strukur pengulangan yang sedang dilakukan. Begitupun dalam bahasa C, kita juga dapat melakukkannya dengan menggunakan struktur **`for`**. Sebagai contoh di sini kita akan mendemonstrasikannya dengan sebuah program yang dapat menampilkan tabel perkalian dari 1 sampai 10.
Adapun sintaknya adalah sebagai berikut:

```
#include   <stdio.h>

int main(void) {
  int baris, kolom;

  /* Proses pengulangan pertama */
  for (baris=1; baris<=10; baris++)
  {
    /* Proses pengulangan kedua yanag terdapat dalam
    pengulangan pertama */
    for (kolom=1; kolom<=10; kolom++)
    {
      printf("%3d ", baris*kolom);
    }
    printf("\n");
  }
  return 0;
}
```

LATIHAN

1. Buatlah deret bilangan urut dari:
   a. 1 – 10
   b. 1 – 100
   c. 10 – 20
   d. 50 – 75
   e. 90 – 115
2. Buatlah sebuah program menggunakan perintah peruangan **`for`** untuk mencetak deret bilanga ganjil dan genap sekaligus dalam 2 baris berbeda jika nilai awal dan nilai akhir tidak ditentukan dari program tetapi diinputkan secara manual dari *console* (layar **cmd**).
3. Buatlah sebuah program untuk menampilkan keluaran berikut.

   a.
   ```
   *
   * *
   * * *
   * * * *
   * * * * *
   ```

   b.
   ```
                 *
               * *
             * * *
           * * * *
         * * * * *
   ```

4. Buatlah sebuah program menggunakan printah **`for`** untuk menentukan IPS semester berjalan dari 3 mata kuliah yang diinputkan berulang secara manual dengan aturan sebagai berikut.
   Tampilan input informasi mata kuliah dan nilai.

```
Mata kuliah ke-1
-----------------------------------------------------
Kode Mata Kuliah : <input>
Nama Mata Kuliah : <input>
SKS              : <input>
Nilai            : <input>
Nilai x Mutu     : <output>
-----------------------------------------------------

Mata kuliah ke-2
-----------------------------------------------------
Kode Mata Kuliah : <input>
Nama Mata Kuliah : <input>
SKS              : <input>
Nilai            : <input>
Nilai x Mutu     : <output>
-----------------------------------------------------
```

```
Mata kuliah ke-3
---------------------------------------------------
Kode Mata Kuliah : <input>
Nama Mata Kuliah : <input>
SKS              : <input>
Nilai            : <input>
Nilai x Mutu     : <output>
---------------------------------------------------
```

Tampilan nilai IPS.

```
---------------------------------------------------
Indeks Prestasi Semester (IPS): <output>
---------------------------------------------------
```

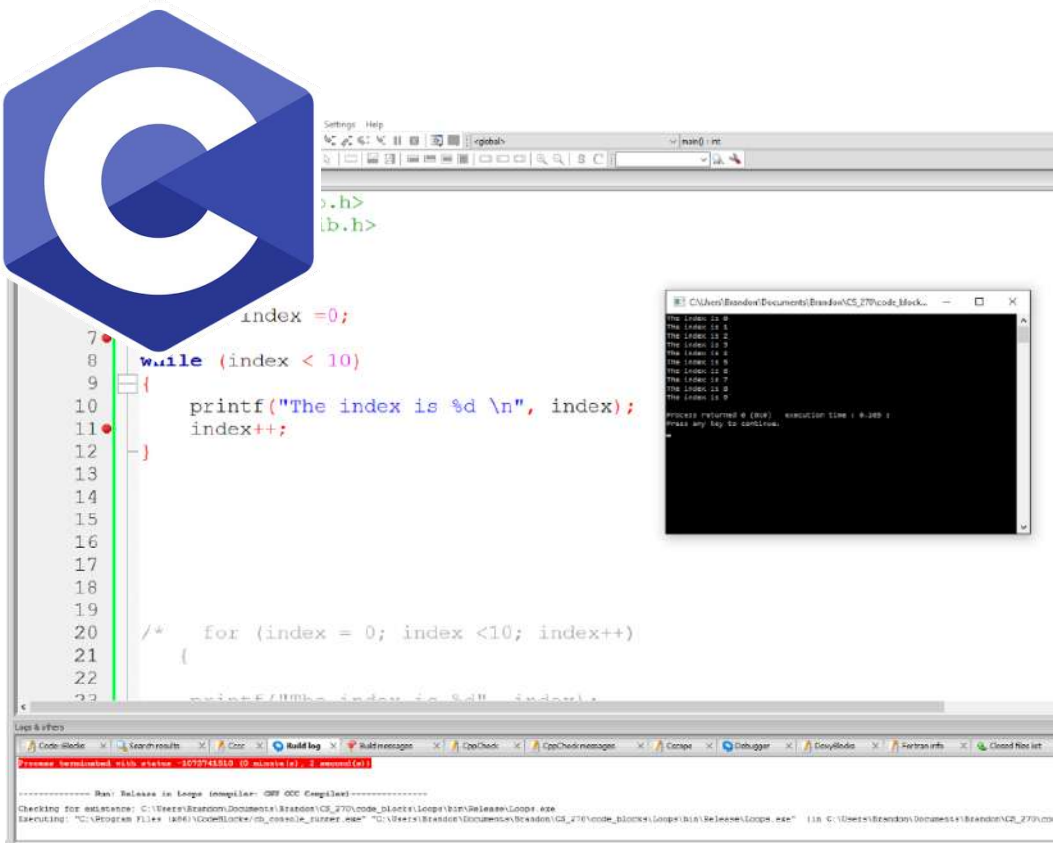
Tabel angka mutu.

KRITERIA	HURUF MUTU	KLASIFIKASI	BOBOT NILAI*	ANGKA MUTU
Sangat Baik	A	A	90.00 – 100.00	4
		A-	80.00 – 89.99	3.7
Baik	B	B+	75.00 – 79.99	3.3
		B	70.00 – 74.99	3.0
		B-	65.00 – 69.99	2.7
Cukup	C	C+	60.00 – 64.99	2.3
		C	55.00 – 59.99	2.0
Kurang	D	C-	50.00 – 54.99	1.7
		D	40.00 – 49.99	1
Sangat Kurang (Tidak Lulus)	E	E	< 40.00	0

**untuk Penilaian Acuan Patokan (PAP)*

MODUL PRAKTIKUM

DASAR-DASAR PEMROGRAMAN



Disusun Oleh:
Prio Handoko, S.Kom., M.T.I.

FAKULTAS DESAIN & TEKNOLOGI
PROGAM STUDI INFORMATIKA
UNIVERSITAS PEMBANGUNAN JAYA
TANGERANG SELATAN 2019

MODUL PRAKTIKUM 4
SELEKSI KONDISI (LOOPING)

Dalam pembuatan program, terkadang kita harus melakukan pengulangan suatu aksi, misalnya untuk melakukan perhitungan berulang dengan menggunakan **for**mula yang sama. Sebagai contoh, misalnya kita ingin membuat program yang dapat menampilkan teks "**Saya sedang belajar bahasa C**" sebanyak 10 kali, maka kita tidak perlu untuk menuliskan 10 buah statemen melainkan kita hanya tinggal menempatkan satu buah statemen ke dalam suatu struktur pengulangan. Dengan demikian program kita akan lebih efesien.

Sebagai gambaran bagi Anda untuk dapat lebih menyerap konsep pengulangan, coba Anda perhatikan terlebih dahulu contoh program di bawah ini.

```
#include   <stdio.h>
int main(void)
{
  /* Mencetak teks ke layar sebanyak 10 kali */
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya   sedang   belajar   bahasa   C");
  printf("Saya sedang belajar bahasa C");

  return 0;
}
```

Dalam bahasa C, terdapat tiga buah struktur pengulangan yang akan digunakan sebagai kontrol dalam melakukan pengulangan proses, yaitu struktur **for**, **while** dan **do-while**.

Setiap struktur yang ada masing-masing memiliki aturan tersendiri dan digunakan dalam konteks yang berbeda. Kita harus bisa menentukan kapan sebaiknya kita harus menggunakan struktur **for**, struktur while ataupun do-while. Untuk itu, pada bagian ini kita akan membahas secara detil apa perbedaan dan penggunaan dari masing-masing struktur tersebut. Hal ini juga bertujuan agar Anda dapat mengimplementasikannya dengan benar ke dalam kasus-kasus program yang Anda hadapi.

Struktur `for`

Struktur **`for`** ini digunakan untuk menuliskan jenis pengulangan yang banyaknya sudah pasti atau telah diketahui sebelumnya. Oleh karena itu, di sini kita harus melakukan inisialisai nilai untuk kondisi awal pengulangan dan juga harus menuliskan kondisi untuk menghentikan proses pengulangan. Adapun bentuk umum dari pendefinisian struktur **`for`** (untuk sebuah statemen) dalam bahasa C adalah sebagai berikut.

```
for (ekpresi1; ekspresi2; ekspresi3)
  statemen_yang_akan_diulang;
```

Sedangkan untuk dua statemen atau lebih.

```
for (ekpresi1; ekspresi2; ekspresi3)
{
  Statemen_yang_akan_diulang1;
  Statemen_yang_akan_diulang2;
  …
}
```

Ekspresi1 di atas digunakan sebagai proses inisialisasi variabel yang akan dijadikan sebagai pencacah (counter) dari proses pengulangan, dengan kata lain ekspresi ini akan dijadikan sebagai kondisi awal.

Ekspresi2 digunakan sebagai kondisi akhir, yaitu kondisi dimana proses pengulangan harus dihentikan. Perlu untuk diketahui bahwa pengulangan masih akan dilakukan selagi kondisi akhir bernilai benar.

Ekspresi3 digunakan untuk menaikkan (*increment*) atau menurunkan (*decrement*) nilai variabel yang digunakan sebagai pencacah. Apabila pengulangan yang kita lakukan bersifat menaik, maka kita akan menggunakan statemen increment, sedangkan apabila pengulangan yang akan kita lakukan bersifat menurun maka kita harus menggunakan statemen decrement.

Berikut ini contoh untuk mengilustrasikan struktur pengulangan **`for`** yang telah diterangkan di atas.

```
for (int j=0; j<10; j++)
{
  /* Statemen yang akan diulang */
  ...
}
```

Pada sintak di atas, mula-mula kita menginisialisasi variabel **j** dengan nilai 0, kemudian karena ekspresi (0 < 10) bernilai benar maka program akan melakukan statemen untuk pertama kalinya. Setelah itu variabel j akan dinaikkan nilainya sebesar 1 melalui statemen increment **j**++sehingga nilai j sekarang menjadi 1. Sampai di sini program akan mengecek ekspresi (**j** < 10). Oleh karena ekspresi (2 < 10) bernilai benar, maka program akan melakukan statemen yang kedua kalinya. Begitu seterusnya sampai nilai **j** bernilai 9. Namun pada saat variabel j telah bernilai 10 maka program akan keluar dari proses pengulangan. Hal ini disebabkan oleh karena ekspresi (10 < 10) bernilai salah.

Untuk membuktikan hal tersebut, perhatikan contoh program di bawah ini dimana kita akan menampilkan teks "**Saya sedang belajar bahasa C**" sebanyak 10 kali.

```
#include   <stdio.h>
int main(void)
{
  for (int j=0; j<10; j++)
  {
    printf("Saya sedang belajar bahasa C\n");
  }

  return 0;

}
```

Pengulangan yang dilakukan pada program di atas bersifat menaik sehingga kita menggunakan increment. Kita juga dapat melakukan pengulangan tersebut secara menurun, yaitu dengan sintak di bawah ini.

```
#include   <stdio.h>
int main(void) {
  for (int j=10; j>0; j--) {
    printf("Saya sedang belajar bahasa C\n");
  }

  return 0;
}
```

Pada sintak di atas, mula-mula variabel **j** bernilai 10 dan setiap pengulangan dilakukan menyebabkan variabel tersebut dikurangi satu. Hal ini disebabkan karena statemen decrement **j--** di atas. Dalam program tersebut, pengulangan baru akan dihentikan ketika variabel j bernilai 0. Apabila dijalankan program di atas akan memberikan hasil yang sama dengan program sebelumnya.

Indeks dari variabel yang digunakan sebagai nilai awal dalam struktur **for** tidak selalu harus bernilai 0, artinya kita dapat memanipulasinya sesuai dengan keinginaan kita (misalnya dengan nilai 1, 2, 3 ataupun lainnya).

```
for (int j=1; j<=5; j++)
{
  /* Statemen yang akan diulang */
  ...
}
```

atau dapat juga dituliskan sebagai berikut.

```
for (int j=10; j>=5; j--)
{
  /* Statemen yang akan diulang */
  ...
}
```

Selain tipe **int**, kita juga dapat menggunakan variabel yang bertipe **char** sebagai pencacah dalam proses pengulangan. Sebagai contoh apabila kita akan melakukan pengulangan sebanyak 3 kali, maka kita dapat menuliskannya sebagai berikut.

```
for (char j='a'; j<='c'; j++)
{
  /* Statemen yang akan diulang */
  ...
}
```

Berikut ini contoh program yang akan menunjukkan hal tersebut.

```
#include   <stdio.h>
int main(void)
{
  for (char j='A'; j<='E'; j++)
  {
    printf("%c = %d\n", j, j);
  }
  return 0;
}
```

Melakukan Beberapa Inisialisasi dalam Struktur `for`

Dalam bahasa C, kita diizinkan untuk melakukan banyak inisialisasi di dalam struktur `for`. Hal ini tentu akan dapat mengurangi banyaknya baris kode program. Adapun cara untuk melakukan hal ini, yaitu dengan menambahkan tanda koma (,) di dalam bagian inisialisasi, seperti yang tertulis di bawah ini.

```
int a, b;
for (a=0, b=0; a<5; a++, b +=5)
{
  ...
}
```

Sintak di atas sama seperti penulisan sintak berikut.

```
int a, b;
b = 0;
for (a=0; a<5; a++)
{
  ...
  b += 5;  /* dapat ditulis dengan b = b + 5 */
}
```

Untuk membuktikannya, perhatikan dua buah contoh program di bawah ini.

a. Menggunakan dua inisialisasi

```
#include   <stdio.h>

int main(void) {

  for (int a=0, int b=0; a < 5; a++, b += 5)
  {
    printf("Baris ke-%d : a = %d, b = %2d\n", a+1, a, b);
  }

  return 0;
}
```

b. Menggunakan satu inisialisasi

```
#include   <stdio.h>
int main(void)
{
  int b=0;
  for (int a=0, a < 5; a++)
  {
    printf("Baris ke-%d : a = %d,   b = %2d\n", a+1, a, b);
    b += 5;
  }

  return 0;
}
```

Struktur `for` Bersarang (*Nested for*)

Dalam pemrograman sering kali kita dituntut untuk melakukan proses pengulangan di dalam strukur pengulangan yang sedang dilakukan. Begitupun dalam bahasa C, kita juga dapat melakukkannya dengan menggunakan struktur **`for`**. Sebagai contoh di sini kita akan mendemonstrasikannya dengan sebuah program yang dapat menampilkan tabel perkalian dari 1 sampai 10.
Adapun sintaknya adalah sebagai berikut:

```
#include   <stdio.h>

int main(void) {
  int baris, kolom;

  /* Proses pengulangan pertama */
  for (baris=1; baris<=10; baris++)
  {
    /* Proses pengulangan kedua yanag terdapat dalam
    pengulangan pertama */
    for (kolom=1; kolom<=10; kolom++)
    {
      printf("%3d ", baris*kolom);
    }
    printf("\n");
  }
  return 0;
}
```

Struktur `while`

Struktur pengulangan **`while`** ini kondisi akan diperiksa di bagian awal. Hal ini tentu menyebabkan kemungkinan bahwa apabila ternyata kondisi yang kita definisikan tidak terpenuhi (bernilai salah), maka proses pengulangan pun tidak akan pernah dilakukan. Adapun bentuk umum dari struktur while adalah seperti yang tampak di bawah ini.

```
while (ekspresi)
{
   Statemen_yang_akan_diulang1;
   Statemen_yang_akan_diulang2;
   …
}
```

Sama seperti pada struktur **`for`**, struktur pengulangan jenis ini juga memerlukan suatu inisialisasi nilai pada variabel yang akan digunakan sebagai pencacah, yaitu dengan menuliskannya di atas blok pengulangan. Selain itu kita juga harus melakukan penambahan ataupun pengurangan terhadap nilai dari variabel pencacah di dalam blok pengulangan tersebut. Hal ini bertujuan untuk menghentikan pengulangan sesuai dengan kondisi yang didefinisikan. Sebagai contoh apabila kita ingin melakukan pengulangan proses sebanyak 5 kali, maka kita akan menuliskannya sebagai berikut.

```
int j = 0; /* Melakukan inisialisasi terhadap variabel j dengan
              nilai 0 */
while (j<5)
{
  /* Statemen yang akan diulang */
  ...
  j++;     /* Melakukan increment terhadap variabel j */
}
```

Untuk menunjukkan bagaimana struktur pengulangan **`while`** ini bekerja, perhatikan contoh program untuk menghitung jumlah 5 buah bilangan positif pertama ini.

Struktur `do-while`

Berbeda dengan struktur **`while`** dimana kondisinya terletak di awal blok pengulangan, pada struktur **`do-while`** kondisi diletakkan di akhir blok pengulangan. Hal ini menyebabkan bahwa statemen yang terdapat di dalam blok pengulangan ini pasti akan dieksekusi minimal satu kali, walaupun kondisinya bernilai salah sekalipun. Maka dari itu struktur do-while ini banyak digunakan untuk kasus-kasus pengulangan yang tidak mempedulikan benar atau salahnya kondisi pada saat memulai proses pengulangan.

Adapun bentuk umum dari struktur pengulangan do-while adalah seperti yang tertulis di bawah ini.

```
do
{
   Statemen_yang_akan_diulang;
   …
} while (ekspresi); /* Ingat tanda semicolon (;) */
```

Mungkin bagi Anda yang merupakan programmer pemula akan merasa bingung untuk membedakan struktur pengulangan **`while`** dan **`do-while`**. Maka dari itu, perhatikan dulu dua buah contoh program berikut ini.

a. Menggunakan struktur **`while`**

```
#include   <stdio.h>

int main(void) {
  int j;
  printf("Statemen sebelum blok pengulangan\n");
  j = 10;

  while (j < 5) {
    printf("Statemen di dalam blok pengulangan\n");
    j++;
  }
  printf("Statemen setelah blok pengulangan\n");
  return 0;
}
```

b. Menggunakan struktur **`do-while`**

```
#include   <stdio.h>

int main(void) {
  int j;
  printf("Statemen sebelum blok pengulangan\n");
  j = 10;
  do {
    printf("Statemen di dalam blok pengulangan\n");
    j++;
  } while (j < 5);
  printf("Statemen setelah blok pengulangan\n");

  return 0;
}
```

Statemen Peloncatan

Statemen peloncatan pada umumnya digunakan dalam sebuah proses pengulangan, yang menentukan apakah pengulangan akan diteruskan, dihentikan atau dipindahkan ke statemen lain di dalam program. Dalam bahasa C, terdapat tiga buah kata kunci yang digunakan untuk melakukan proses peloncatan tersebut, yaitu **`break`**, **`continue`** dan **`goto`**.

Menggunakkan Kata Kunci ***`break`***

Statemen **`break`** digunakan untuk menghentikan sebuah pengulangan dan program akan langsung meloncat ke statemen yang berada di bawah blok pengulangan. Ini biasanya dilakukan karena alasan efesiensi program, yaitu untuk menghindari proses pengulangan yang sebenarnya sudah tidak diperlukan lagi. Sebagai contoh, di sini kita akan menuliskan program untuk menentukan suatu bilangan apakah termasuk ke dalam bilangan prima atau tidak. Adapun sintak programnya adalah sebagai berikut.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk mengecek bilangan prima atau
   bukan */
int CekPrima(int x) {
  int prima = 1;/* mula-mula variabel prima bernilai 1 (true) */
  int i;        /* variabel untuk indeks pengulangan */

  if (x <= 1) {
    prima = 0;  /* apabila x ≤ 1 maka prima bernilai 0
                   (false) */
  } else {
    for (i=2; i<=(x/2); i++) {
      if (x % i == 0) {
        prima = 0;    /* prima bernilai false */
        break;        /* menghentikan proses pengulangan */
      }
    }
  }
```

Pada program di atas, apabila dalam proses pengulangan kita telah mengeset nilai variabel **`prima`** menjadi 0 (**`false`**), maka kita tidak perlu lagi untuk melanjutkan proses pengulangan, karena bilangan yang diperiksa sudah pasti bukan merupakan bilangan prima. Dengan demikian apabila kita masih melanjutkan proses pengulangan maka hal tersebut dapat dikatakan sebagai hal yang sia-sia. Oleh karena itu, pada kasus ini kita harus menambahkan statemen **`break`** untuk menghentikan pengulangan tersebut.

Menggunakan Kata Kunci ***continue***

Berbeda dengan statemen **break** di atas yang berguna untuk menghentikan suatu proses pengulangan, statemen **continue** justru digunakan untuk melanjutkan proses pengulangan. Sebagai contoh apabila kita akan membuat program untuk melakukan pembagian dua buah bilangan, maka kita harus menjaga agar bilangan pembagi (penyebut) harus tidak sama dengan nol. Untuk kasus ini, kita akan membuat sebuah pengulangan untuk melakukan input sampai bilangan pembagi yang dimasukkan tidak sama dengan nol. Berikut ini contoh sintak program yang dimaksud.

```
#include <stdio.h>

#define  TRUE    1
#define FALSE    0

int main(void) {
  double a = 1; /* Menginisialisasi bilangan yang akan di bagi
                   (pembilang) */
  double b;     /* Variabel penampung nilai pembagi (penyebut) */

  /* Memaksa proses pengulangan */
  while (TRUE) {
    printf("Masukkan bilangan pembagi : "); scanf("%lf", &b);
    if (b == 0) {
      continue; /* Apabila pembagi 0,
                   maka lanjutkan pengulangan */
    }
```

Menggunakan Kata Kunci ***goto***

Selain cara-cara yang telah dijelaskan di atas, bahasa C juga menyediakan kata kunci **goto** yang digunakan agar program dapat meloncat ke baris tertentu yang kita pilih. Adapun untuk menentukan baris tersebut kita harus membuat suatu label, yaitu dengan menempatkan tanda *colon* atau titik dua (**:**) di belakangnya, misalnya **LBL:, LABEL:, mylabel:** atau nama-nama lain yang Anda kehendaki. Berbeda dengan statemen **break** dan **continue** yang umumnya digunakan untuk proses pengulangan, statemen **goto** dapat ditempatkan di mana saja sesuai dengan kebutuhan program. Berikut ini contoh program yang akan menunjukkan penggunaan statemen **goto** di dalam proses pengulangan.

```
#include <stdio.h>

#define       TRUE     1
#define FALSE    0

int  main(void)  {
  int counter = 0;    /* Variabel untuk indeks pengulangan */
  while (TRUE) {
    counter++;
    if (counter > 10) {
      goto LBL;
    }
    printf("Baris ke-%d\n", counter);
  }
```

```
  LBL       /* Membuat label dengan nama LBL */
  printf("Statemen yang terdapat di luar blok pengulangan");

  return 0;
}
```

Pada program di atas terlihat jelas bahwa penggunaan statemen **goto** akan menyebabkan eksekusi program akan langsung berpindah atau meloncat ke label yang telah didefinisikan. Dalam hal ini, karena label didefinisikan di luar blok pengulangan maka proses peloncatan tersebut secara otomatis akan menyebabkan terhentinya proses pengulangan yang sedang berlangsung.

LATIHAN

1. Buatlah deret bilangan urut untuk rentang di bawah ini menggunakan perulangan **`for`**, **`while`**, dan **`do-while`**.
   a. 1 – 10
   b. 1 – 100
   c. 10 – 20
   d. 50 – 75
   e. 90 – 115
2. Buatlah sebuah program menggunakan perintah peruangan **`for`**, **`while`**, dan **`do-while`** untuk mencetak deret bilanga ganjil dan genap sekaligus dalam 2 baris berbeda jika nilai awal dan nilai akhir tidak ditentukan dari program tetapi diinputkan secara manual dari *console* (layar **cmd**).
3. Buatlah sebuah program untuk menampilkan keluaran berikut menggunakkan perintah perulangan **`for`**, **`while`**, dan **`do-while`**.

   a.
   ```
   *
   * *
   * * *
   * * * *
   * * * * *
   ```

   b.
   ```
                 *
               * *
             * * *
           * * * *
         * * * * *
   ```

4. Buatlah sebuah program menggunakan printah **`for`**, **`while`**, dan **`do-while`** untuk menentukan IPS semester berjalan dari 3 mata kuliah yang diinputkan berulang secara manual dengan aturan sebagai berikut.
   Tampilan input informasi mata kuliah dan nilai.

```
Mata kuliah ke-1
-----------------------------------------------------
Kode Mata Kuliah : <input>
Nama Mata Kuliah : <input>
SKS              : <input>
Nilai            : <input>
Nilai × Mutu     : <output>
-----------------------------------------------------

Mata kuliah ke-2
-----------------------------------------------------
Kode Mata Kuliah : <input>
Nama Mata Kuliah : <input>
SKS              : <input>
Nilai            : <input>
```

```
Nilai x Mutu      : <output>
-------------------------------------------------------
Mata kuliah ke-3
-------------------------------------------------------
Kode Mata Kuliah : <input>
Nama Mata Kuliah : <input>
SKS              : <input>
Nilai            : <input>
Nilai x Mutu     : <output>
-------------------------------------------------------
```

Tampilan nilai IPS.

```
-------------------------------------------------------
Indeks Prestasi Semester (IPS): <output>
-------------------------------------------------------
```

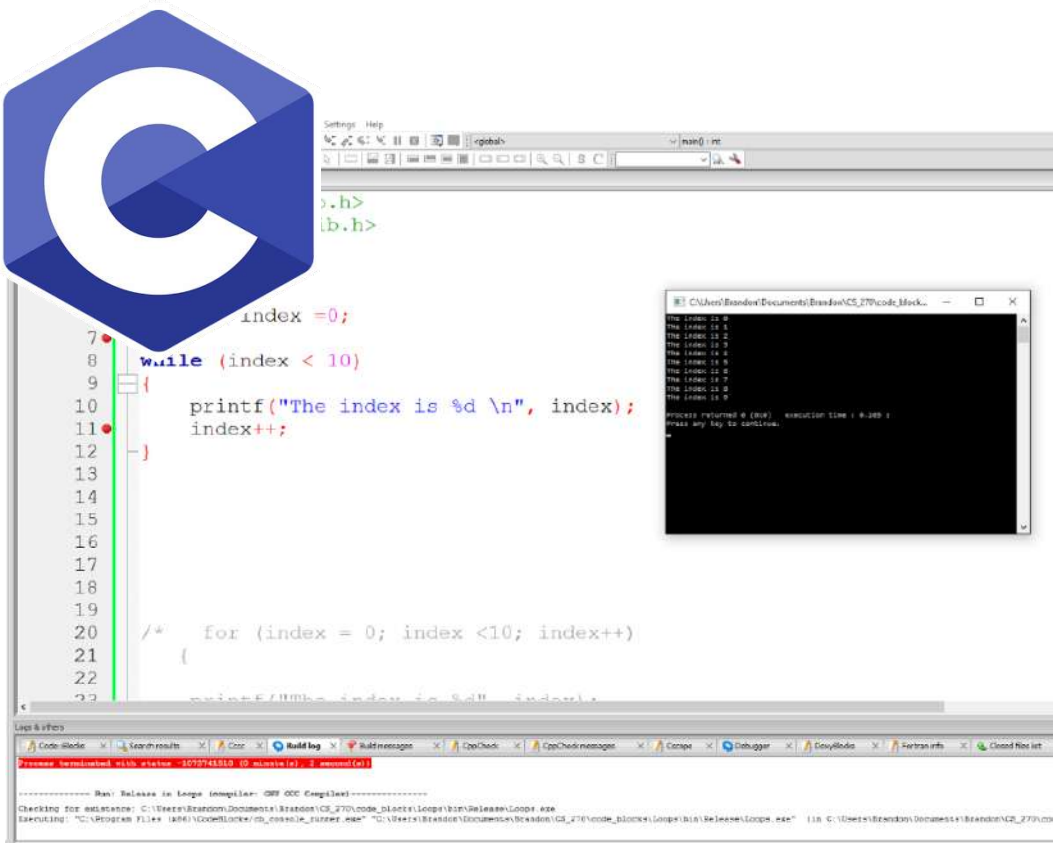
Tabel angka mutu.

KRITERIA	HURUF MUTU	KLASIFIKASI	BOBOT NILAI*	ANGKA MUTU
Sangat Baik	A	A	90.00 – 100.00	4
		A-	80.00 – 89.99	3.7
Baik	B	B+	75.00 – 79.99	3.3
		B	70.00 – 74.99	3.0
		B-	65.00 – 69.99	2.7
Cukup	C	C+	60.00 – 64.99	2.3
		C	55.00 – 59.99	2.0
Kurang	D	C-	50.00 – 54.99	1.7
		D	40.00 – 49.99	1
Sangat Kurang (Tidak Lulus)	E	E	< 40.00	0

**untuk Penilaian Acuan Patokan (PAP)*

MODUL PRAKTIKUM

DASAR-DASAR PEMROGRAMAN



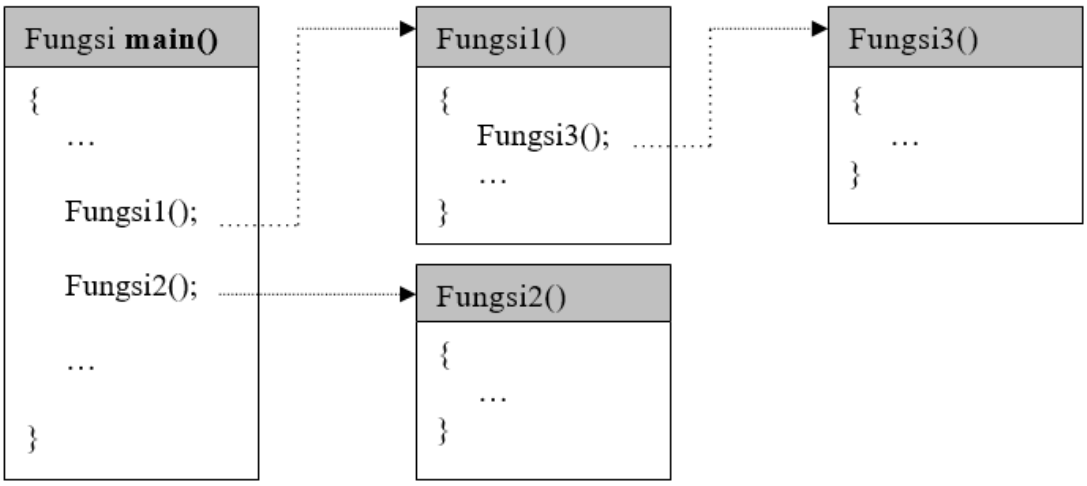
Disusun Oleh:
Prio Handoko, S.Kom., M.T.I.

FAKULTAS DESAIN & TEKNOLOGI
PROGAM STUDI INFORMATIKA
UNIVERSITAS PEMBANGUNAN JAYA
TANGERANG SELATAN 2019

MODUL PRAKTIKUM 5
FUNGSI

Menurut definisinya, fungsi adalah suatu blok program yang digunakan untuk melakukan proses-proses tertentu. Sebuah fungsi dibutuhkan untuk menjadikan program yang akan kita buat menjadi lebih modular dan mudah untuk dipahami alurnya. Dengan adanya fungsi, maka kita dapat mengurangi duplikasi kode program sehingga performa dari program yang kita buat pun akan meningkat. Dalam bahasa C, sebuah program terdiri atas fungsi-fungsi, baik yang didefinisikan secara langsung di dalam program maupun yang disimpan di dalam file lain (misalnya file *header*). Satu fungsi yang pasti terdapat dalam program yang ditulis menggunakan bahasa C adalah fungsi **`main()`**. Fungsi tersebut merupakan fungsi utama dan merupakan fungsi yang akan dieksekusi pertama kali.

Dalam bahasa C, fungsi terbagi menjadi dua macam, yaitu fungsi yang mengembalikan nilai (*return value*) dan fungsi yang tidak mengembalikan nilai. Fungsi yang tidak mengembalikan nilai tersebut dinamakan dengan *void function*. Bagi Anda yang sebelumnya pernah belajar bahasa Pascal, *void function* ini serupa dengan *procedure* yang terdapat di dalam bahasa Pascal. Gambar di bawah ini menerangkan bagaimana kompilator C membaca fungsi-fungsi yang didefiniskan di dalam program secara berurutan sesuai dengan waktu pemanggilannya.



Gambar 1. Pemanggilan fungsi dalam Bahasa C

Mula-mula fungsi **`main()`** akan dieksekusi oleh kompilator. Oleh karena fungsi **`main()`** tersebut memanggil **`Fungsi1()`**, maka kompilator akan mengeksekusi **`Fungsi1()`**. Dalam **`Fungsi1()`** juga terdapat pemanggilan **`Fungsi3()`**, maka kompilator akan mengeksekusi **`Fungsi3()`** dan kembali lagi mengeksekusi baris

selanjutnya yang terdapat pada **`Fungsi1()`** (jika ada). Setelah **`Fungsi1()`** selesai dieksekusi, kompilator akan kembali mengeksekusi baris berikutnya pada fungsi **`main()`**, yaitu dengan mengeksekusi kode-kode yang terdapat pada **`Fungsi2()`**. Setelah selesai, maka kompilator akan melanjutkan pengeksekusian kode pada baris- baris selanjutnya dalam fungsi **`main()`**. Apabila ternyata dalam fungsi **`main()`** tersebut kembali terdapat pemanggilan fungsi lain, maka kompilator akan meloncat ke fungsi lain tersebut, begitu seterusnya sampai semua baris kode dalam fungsi **`main()`** selesai dieksekusi.

Apa Nilai yang dikembalikan Oleh Fungsi `main()`?

Pada bab-bab sebelumnya kita telah banyak menggunakan fungsi **`main()`** di dalam program yang kita buat. Mungkin sekarang Anda akan bertanya apa sebenarnya nilai yang dikembalikan oleh fungsi **`main()`** tersebut? Jawabnya adalah nilai 0 dan 1. Apabila fungsi **`main()`** mengembalikan nilai 0 ke sistem operasi, maka sistem operasi akan mengetahui bahwa program yang kita buat tesebut telah dieksekusi dengan benar tanpa adanya kesalahan. Sedangkan apabila nilai yang dikembalikan ke sistem operasi adalah nilai 1, maka sistem operasi akan mengetahui bahwa program telah dihentikan secara tidak normal (terdapat kesalahan). Dengan demikian, seharusnya fungsi **`main()`** tidaklah mengembalikan tipe **`void`**, melainkan tipe data **`int`**, seperti yang terlihat di bawah ini.

```
int main(void) {
   ...
   return 0;            /* Mengembalikan nilai 0 */
}
```

Namun, apabila Anda ingin mendefinisikan nilai kembalian tersebut dengan tipe **`void`**, maka seharusnya Anda menggunakan fungsi **`exit()`** yang dapat berguna untuk mengembalikan nilai ke sistem operasi (sama seperti halnya **`return`**). Adapun parameter yang dilewatkan ke fungsi **`exit()`** ini ada dua, yaitu:

1. Nilai 0 (`EXIT_SUCCESS`), yaitu menghentikan program secara normal
2. Nilai 1 (`EXIT_FAILURE`), yaitu menghentikan program secara tidak normal

Berikut ini contoh penggunaannya di dalam fungsi **main()**.

```
void  main(void) {
  ...
  exit(0);       /* dapat ditulis exit(EXIT_SUCCESS) */
}
```

Fungsi Tanpa Nilai Balik (Void Function)

Pada umumnya fungsi tanpa nilai balik (*return value*) ini digunakan untuk melakukan proses-proses yang tidak menghasilkan nilai, seperti melakukan pengulangan, proses pengesetan nilai ataupun yang lainnya. Dalam bahasa C, fungsi semacam ini tipe kembaliannya akan diisi dengan nilai **void**. Adapun bentuk umum dari pendefinisian fungsi tanpa nilai balik adalah sebagai berikut:

```
void nama_fungsi(parameter1, parameter2,…) {
  Statemen_yang_akan_dieksekusi;
  ...
}
```

Berikut ini contoh dari pembuatan fungsi tanpa nilai balik.

```
void Tulis10Kali(void) {
  int j;
  for (j=0; j<10; j++) {
    printf("Saya sedang belajar bahasa C");
  }
}
```

Adapun contoh program lengkap yang akan menggunakan fungsi tersebut adalah seperti yang tertulis di bawah ini.

```
#include <stdio.h>

/* Mendefinisikan sebuah fungsi dengan nama Tulis10Kali */
void Tulis10Kali(void) {
  int j;

  for (j=0; j<10; j++) {
    printf("Saya sedang belajar bahasa C");
  }
}
```

```
  int main(void) {
  Tulis10Kali();        /* Memanggil fungsi Tulis10Kali() */
  return 0;
}
```

Fungsi dengan Nilai Balik

Berbeda dengan fungsi di atas yang hanya mengandung proses tanpa adanya nilai kembalian, di sini kita akan membahas mengenai fungsi yang digunakan untuk melakukan proses-proses yang berhubungan dengan nilai. Adapun cara pendefinisiannya adalah dengan menuliskan tipe data dari nilai yang akan dikembalikan di depan nama fungsi, berikut ini bentuk umum dari pendefinisian fungsi dengan nilai balik di dalam bahasa C.

```
tipe_data nama_fungsi(parameter1, parameter2,...) {
  Statemen_yang_akan_dieksekusi;
  ...
  return nilai_balik;
}
```

Sebagai contoh, di sini kita akan membuat fungsi sederhana yang berguna untuk menghitung nilai luas bujursangkar. Adapun sintak untuk pendefinisiannya adalah sebagai berikut.

```
int HitungLuasBujurSangkar(int sisi)
{
  int L;  /* mendeklarasikan variabel L untuk menampung nilai
             luas */
  L = sisi * sisi;     /* memasukkan nilai sesuai dengan rumus
                          yang berlaku */
 return L;             /* mengembalikan nilai yang didapat dari
                          hasil proses */
```

Sedangkan untuk menggunakan fungsi tersebut, Anda harus menuliskan program lengkap seperti di bawah ini.

```
int main(void)
{ int S, Luas;

  /* Mengeset nilai variabel S dengan nilai 10*/
  S = 10;

  /* Memanggil fungsi HitungLuasBujurSangkar
     dan menampung nilainya ke variabel Luas
  */
  Luas = HitungLuasBujurSangkar(S);

  /* Mencetak hasil perhitungan ke layar monitor */
  printf("Luas bujur sangkar dengan sisi %d adalah %d", S,
          Luas);

  return 0;
}
```

Fungsi dengan Parameter

Parameter adalah suatu variabel yang berfungsi untuk menampung nilai yang akan dikirimkan ke dalam fungsi. Dengan adanya parameter, sebuah fungsi dapat bersift dinamis. Parameter itu sendiri terbagi menjadi dua macam, yaitu parameter formal dan parameter aktual. Parameter formal adalah parameter yang terdapat pada pendefinisian fungsi, sedangkan parameter aktual adalah parameter yang terdapat pada saat pemanggilan fungsi. Untuk lebih memahaminya, perhatikan contoh pendefinisian fungsi di bawah ini.

```
int  TambahSatu(int  x)  {
     return ++x;
}
```

Pada sintak di atas, variabel **x** dinamakan sebagai *parameter formal*. Sekarang perhatikan sintak berikut.

```
int main(void) { int
  a = 10, hasil;
  hasil  =  TambahSatu(a);
  return 0;
}
```

Pada saat pemanggilan fungsi **TambahSatu()** di atas, variabel **a** dinamakan dengan *parameter aktual*.

Jenis Parameter

Dalam dunia pemrograman dikenal tiga jenis parameter, yaitu parameter masukan, keluaran dan masukan/keluaran. Untuk memahami perbedaan dari setiap jenis parameter, di sini kita akan membahasnya satu per satu.

Parameter Masukan

Parameter masukan adalah parameter yang digunakan untuk menampung nilai data yang akan dijadikan sebagai masukan (*input*) ke dalam fungsi. Artinya, sebuah fungsi dapat menghasilkan nilai yang berbeda tergantung dari nilai parameter yang dimasukkan pada saat pemanggilan fungsi tersebut. Berikut ini contoh program yang akan menunjukkan kegunaan dari parameter masukan.

```
#include    <stdio.h>

#define pi 3.14159

/* Mendefinisikan suatu fungsi dengan parameter berjenis
   masukan */
double HitungKelilingLingkaran(int radius) {
  double k;
  k  =  2  *  pi  *  radius;
  return k;
}

/* Fungsi Utama */
int  main(void)  {
  int r;
  printf("Masukkan nilai jari-jari lingkaran : ");
  scanf("%d", &r);
  double    keliling    =    HitungKelilingLingkaran(r);
  printf("Keliling lingkaran dengan jari-jari %d : %f", r,
         keliling);
  return 0;
}
```

Pada sintak program di atas, variabel **r** merupakan parameter aktual yang berfungsi sebagai parameter masukan karena variabel tersebut digunakan untuk menampung nilai yang akan menjadi masukan (*input*) untuk proses perhitungan di dalam fungsi **HitungKelilingLingkaran()**.

Parameter Keluaran

Kebalikan dari parameter masukan, parameter keluaran adalah parameter yang digunakan untuk menampung nilai kembalian / nilai keluaran (*output*) dari suatu proses. Umumnya parameter jenis ini digunakan di dalam fungsi yang tidak mempunyai nilai balik. Untuk lebih memahaminya, perhatikan contoh program di bawah ini yang merupakan modifikasi dari program sebelumnya.

```
#include    <stdio.h>
#define PI 3.14159

/* Mendefinisikan fungsi yang mengandung parameter keluaran */
void HitungKelilingLingkaran(int radius, double *K) {
  *K = 2 * PI * radius;
}

/* Fungsi Utama */
int  main(void)  {
  int R;
  double Keliling;

  printf("Masukkan  nilai  jari-jari  lingkaran  :  ");
  scanf("%d", &R);
  HitungKelilingLingkaran(R, Keliling);
  printf("Keliling  lingkaran  dengan  jari-jari  %d  :  %f",  R,
         Keliling);

  return 0;
}
```

Pada sintak program di atas, variabel `Keliling` berfungsi sebagai parameter keluaran karena variabel tersebut digunakan untuk menampung nilai hasil dari proses yang terdapat di dalam fungsi. Sedangkan variabel `R` adalah variabel yang bersifat sebagai parameter masukan dimana nilainya digunakan untuk menampung nilai yang akan dilewatkan ke dalam fungsi. Adapun contoh hasil yang akan diberikan dari program di atas adalah seperti yang tertera di bawah ini.

Parameter Masukan/Keluaran

Selain parameter masukan dan keluaran, terdapat parameter jenis lain, yaitu parameter masukan/keluaran dimana parameter tersebut mempunyai dua buah kegunaan, yaitu sebagai berikut:

- Pertama parameter ini akan bertindak sebagai parameter yang menampung nilai masukan.
- Setelah itu, parameter ini akan bertindak sebagai parameter yang menampung nilai keluaran.

Berikut ini diberikan contoh program dimana di dalamnya terdapat sebuah parameter yang berperan sebagai parameter masukan/keluaran.

```
#include    <stdio.h>
#define PI 3.14159

/* Mendefinisikan fungsi dengan parameter masukan/keluaran */
void HitungKelilingLingkaran(double *X) {
  *X = 2 * PI * (*X);
}

/* Fungsi Utama */
int  main(void)  {
  int R;
  double param;
  printf("Masukkan nilai jari-jari lingkaran : ");
  scanf("%d", &int);

  /* Melakukan typecast dari tipe int ke tipe double */
  param = (double) R;

  HitungKelilingLingkaran(&param);
  printf("Keliling lingkaran dengan jari-jari %d : %f", R,
          param);
  return 0;
```

Melewatkan Parameter Berdasarkan Nilai (*Pass By Value*)

Terdapat dua buah cara untuk melewatkan parameter ke dalam sebuah fungsi, yaitu dengan cara melewatkan berdasarkan nilainya (*pass by value*) dan berdasarkan alamatnya (*pass by reference*). Namun pada bagian ini hanya akan dibahas mengenai pelewatan parameter berdasarkan nilai saja, sedangkan untuk pelewatan parameter berdasarkan alamat akan kita bahas pada sub bab berikutnya. Pada pelewatan parameter berdasarkan nilai, terjadi proses penyalinan (*copy*) nilai dari parameter formal ke parameter aktual. Hal ini akan meyebabkan nilai variabel yang dihasilkan oleh proses di dalam fungsi tidak akan berpengaruh terhadap nilai variabel yang terdapat di luar fungsi. Untuk lebih memahaminya, perhatikan contoh program berikut dimana di dalamnya terdapat sebuah parameter yang dilewatkan dengan cara *pass by value*.

```
#include <stdio.h>
/* Mendefinisikan fungsi dengan melewatkan parameter berdasarkan
nilai */
void  TambahSatu(int  X)  {
  X++;
  /* Menampilkan nilai yang terdapat di dalam fungsi */
  printf("Nilai di dalam fungsi : %d\n", X);
}
```

```
/* Fungsi Utama */
int  main(void)  {
  int Bilangan;
  printf("Masukkan sebuah bilangan bulat : ");
  scanf("%d", &Bilangan);

  /* Menampilkan nilai awal */
  prinf("\nNilai awal : %d\n", Bilangan);

  /* Memanggil fungsi TambahSatu */
  TambahSatu(Bilangan);

  /* Menampilkan nilai akhir */
  prinf("Nilai akhir : %d\n", Bilangan);

  return 0;
}
```

Seperti yang kita lihat pada hasil program di atas bahwa nilai dari variabel Bilangan tetap bernilai 10 walaupun kita telah memanggil fungsi **`TambahSatu()`**. Hal ini disebabkan karena variabel **`Bilangan`** dan variabel **`X`** merupakan dua variabel yang tidak saling berhubungan dan menempati alamat memori yang berbeda sehingga yang terjadi hanyalah proses penyalinan (peng-*copy*-an) nilai dari variabel **`Bilangan`** ke variabel **`X`**. Dengan demikian, perubahan nilai variabel **`X`** tentu tidak akan mempengaruhi nilai variabel **`Bilangan`**.

Melewatkan Parameter Berdasarkan Alamat (*Pass By Reference*)

Di sini, parameter yang dilewatkan ke dalam fungsi bukanlah berupa nilai, melainkan suatu alamat memori. Pada saat kita melewatkan parameter berdasarkan alamat, terjadi proses referensial antara variabel yang terdapat pada parameter formal dengan variabel yang terdapat parameter aktual. Hal tersebut menyebabkan kedua variabel tersebut akan berada pada satu alamat di memori yang sama sehingga apabila terdapat perubahan nilai terhadap salah satu dari variabel tersebut, maka nilai variabel satunya juga akan ikut berubah. Untuk melakukan hal itu, parameter fungsi tersebut harus kita jadikan sebagai *pointer* (akan dibahas pada bab terpisah).

Agar dapat lebih memahami materi ini, di sini kita akan menuliskan kembali kasus di atas ke dalam sebuah program. Namun, sekarang kita akan melakukannya dengan menggunakan cara *pass by reference*.

```
#include <stdio.h>

/* Mendefinisikan fungsi dengan melewatkan parameter berdasarkan
alamat */
void  TambahSatu(int  *X) {
  (*X)++;
  /* Menampilkan nilai yang terdapat di dalam fungsi */
  printf("Nilai di dalam fungsi : %d\n", *X);
}

/* Fungsi Utama */
int  main(void)  {
  int Bilangan;
  printf("Masukkan sebuah bilangan bulat : ");
  scanf("%d", &Bilangan);

  /* Menampilkan nilai awal */
  prinf("\nNilai awal : %d\n", Bilangan);

  /* Memanggil fungsi TambahSatu dengan mengirimkan
     alamat variabel Bilangan */
  TambahSatu(&Bilangan);

  /* Menampilkan nilai akhir */
  prinf("Nilai akhir : %d\n", Bilangan);

  return 0;
}
```

Rekursi

Rekursi adalah proses pemanggilan fungsi oleh dirinya sendiri secara berulang. Istilah 'rekursi' sebenarnya berasal dari bahasa Latin 'recursus', yang berarti 'menjalankan ke belakang'. Rekursi digunakan untuk penyederhanaan algoritma dari suatu proses sehingga program yang dihasilkan menjadi lebih efesien. Pada bagian ini kita akan mempelajarinya langsung melalui contoh-contoh program.

Menentukan Nilai Faktorial

Pada bagian ini kita akan membuat sebuah fungsi rekursif untuk menentukan nilai faktorial dengan memasukkan nilai yang akan dihitung sebagai parameter fungsi ini. Sebagai contoh apabila parameter yang kita masukkan adalah 5, maka hasilnya adalah

5! = 5 x 4 x 3 x 2 x 1 = 120

Proses tersebut dapat kita sederhanakan melalui fungsi matematis sebagai berikut. F! **(N) = N * F! (N-1)**

Namun yang harus kita perhatikan di sini adalah F! (0) = 1, ini adalah suatu tetapan

numerik yang tidak dapat diubah. Berikut ini contoh implementasi kasus tersebut ke dalam sebuah program.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk menghitung nilai faktorial */
int Faktorial(int N) {
  if (N == 0) {
    return 1;
  }else {
    return N * Faktorial(N-1);
  }
}

int  main(void)  {
  int bilangan;
  printf("Masukkan bilangan yang akan dihitung : ");
  scanf("%d", &bilangan);
  printf("%d! = %d", bilangan, Faktorial(bilangan));
  return 0;
}
```

Konsep dari proses di atas sebenarnya sederhana, yaitu dengan melakukan pemanggilan fungsi **`Faktorial()`** secara berulang. Untuk kasus ini, proses yang dilakukan adalah sebagai berikut.

Faktorial(5) = 5 * Faktorial(4)
Faktorial(4) = 4 * Faktorial(3)
Faktorial(3) = 3 * Faktorial(2)
Faktorial(2) = 2 * Faktorial(1)
Faktorial(1) = 1 * Faktorial(0)
Faktorial(0) = 1
Faktorial(1) = 1 * 1
Faktorial(2) = 2 * 1
Faktorial(3) = 3 * 2
Faktorial(4) = 4 * 6
Faktorial(5) = 5 * 24
= 120

Menentukan Nilai Perpangkatan

Sekarang kita akan melakukan rekursi untuk menghitung nilai BN, dimana B adalah bilangan basis dan N adalah nilai eksponen. Kita dapat merumuskan fungsi tersebut seperti di bawah ini.

BN = B * BN-1

Dengan demikian apabila kita implementasikan ke dalam program, maka sintaknya kurang lebih sebagai berikut.

```
#include <stdio.h>

/* Mendefinisikan fungsi untuk menghitung nilai eksponensial */
int Pangkat(int basis, int e) {
  if (e == 0) {
    return 1;
  }else {
    return basis * Pangkat(basis, e-1);
  }
}

int  main(void)  {
  int B, N;
  printf("Masukkan bilangan basis : "); scanf("%d", &B);
  printf("Masukkan bilangan eksponen : "); scanf("%d", &N);
  printf("%d^%d = %d", B, N, Pangkat(B, N));
  return 0;
}
```

Proses yang terdapat di atas adalah sebagai berikut.

Pangkat(2, 5) = 2 * Pangkat(2, 4)
Pangkat(2, 4) = 2 * Pangkat(2, 3)
Pangkat(2, 3) = 2 * Pangkat(2, 2)
Pangkat(2, 2) = 2 * Pangkat(2, 1)
Pangkat(2, 1) = 2 * Pangkat(2, 0)
Pangkat(2, 0) = 1
Pangkat(2, 1) = 2 * 1
Pangkat(2, 2) = 2 * 2
Pangkat(2, 3) = 2 * 4
Pangkat(2, 4) = 2 * 8
Pangkat(2, 5) = 2 * 16
= 32

Konversi Bilangan Desimal ke Bilangan Biner

Kali ini kita akan membuat sebuah fungsi rekursif tanpa nilai balik yang digunakan untuk melakukan konversi bilangan desimal ke bilangan biner. Untuk informasi lebih detil mengenai bilangan biner dan heksadesimal, Anda dapat melihat lampiran B – Bit dan Byte di bagian akhir buku ini. Artinya, di sini kita tidak akan membahas bagaimana proses pengkonversian tersebut, melainkan kita lebih berkonsentrasi ke pembahasan mengenai rekursi.

Adapun sintak program untuk melakukan hal tersebut adalah sebagai berikut.

```
#include <stdio.h>

void DesimalKeBiner(int n) {
  if (n>1) {
    DesimalKeBiner(n/2);
  }
  printf("%d", n%2);
}

int main(void)
{
  int a;
  printf("Masukkan bilangan desimal yang akan dikonversi : ");
  scanf("%d",&a);
  printf("%d dalam biner : ", a)
  DesimalKeBiner(a);

  return 0;
}
```

LATIHAN

1. Buatlah program untuk menyelesaikan perhitungan aritmatika di bawah ini dimana setiap proses yang dikerjakan diwakilkan oleh sebuah fungsi dengan nilai balik yang memiliki 2 variabel.

$$X = (A + B) * ((C – D) / E)$$

2. Buatlah sebuah program rekursif untuk melakukan konversi dari bilangan desimal ke *oktadecimal*!