

PERULANGAN

OBJEKTIF :

1. Mahasiswa mampu memahami tentang operator *increment*
 2. Mahasiswa mampu memahami tentang operator *decrement*.
 3. Mahasiswa mampu memahami tentang statement `while`.
 4. Mahasiswa mampu memahami tentang statement `do...while`.
 5. Mahasiswa mampu memahami tentang statement nested `for`.
-

3.1 OPERATOR INCREMENT

Operator *increment* merupakan jenis operator yang digunakan untuk menaikkan nilai variabel sebesar satu. Biasanya operator ini digunakan pada operand bertipe bilangan bulat. Operator *increment* identik dengan adanya simbol `++` yang berada sebelum atau setelah suatu nama variabel. Ada dua jenis operator *increment*, yaitu:

- Operator *pre increment*
- Operator *post increment*

3.1.1 OPERATOR PRE INCREMENT

Operator *pre increment* merupakan operator yang di mana peletakkan simbol `++` berada sebelum nama variabel. Variabel dengan operator *pre increment* digambarkan seperti `++A`. Operator *pre increment* akan menambahkan nilai sebanyak satu terlebih dahulu kepada suatu variabel terkait, yang mana kenaikan nilai tersebut mempengaruhi operasi lain yang melibatkan variabel dengan operator *pre increment* tersebut. Di bawah ini merupakan contoh program `preIncrement.c` yang terdapat operator *pre increment* di dalamnya:

```
#include <stdio.h>

int main()
{
    int angka1 = 5;
    int angka2;

    // Inisialisasi variabel angka2 dengan menugaskan ekspresi 2 dijumlahkan dengan
    ++angka1
    angka2 = 2 + ++angka1;

    // Mencetak nilai
    printf("angka1 = %d\n", angka1);
    printf("angka2 = %d\n", angka2);

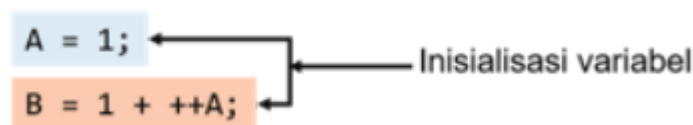
    return 0;
}
```

Output program `preIncrement.c`:

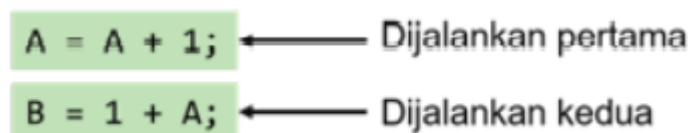
```
angka1 = 6
angka2 = 8
```

Pada output di atas, menampilkan bahwa variabel `angka1` telah mengalami perubahan nilai dengan menambahkan nilai `+1` ke variabel `angka1`, sehingga nilai variabel `angka1` setelah melalui proses *increment* berubah menjadi `6`. Sedangkan keberadaan operator pre increment pada statement `angka2 = 2 + ++angka1;` memberi arti bahwa nilai dari variabel `angka2` mengandung ekspresi `2 + 6`, di mana nilai `6` merupakan hasil dari *increment* terhadap variabel `angka1`.

Berdasarkan program di atas, di bawah ini merupakan penggambaran mengenai cara kerja operator *pre increment*:



maka urutan pengerjaannya seperti di bawah ini:



Baik operator *pre increment* ataupun *post increment* sama-sama menambah nilai `+1`. Namun, pada *pre increment* apabila suatu variabel melibatkan operasi dengan variabel *pre increment* seperti pada program `preIncrement.c` di atas, variabel yang mengandung ekspresi operasi perhitungan dengan variabel *pre increment* seperti variabel `angka2` tersebut, *compiler* secara otomatis akan menganggap nilai variabel `angka1` adalah `6`.

3.1.2 OPERATOR POST INCREMENT

Operator *post increment* merupakan operator yang di mana simbol `++` diletakkan setelah nama variabel. Variabel dengan operator post increment digambarkan seperti `A++`. Operator *post increment* akan menambahkan nilai sebanyak satu namun kenaikan nilai tersebut tidak mempengaruhi operasi lain yang melibatkan variabel dengan operator *post increment* tersebut. Di bawah ini merupakan contoh program `postIncrement.c` yang terdapat operator *post increment* di dalamnya:

```
#include <stdio.h>

int main()
{
    int angka1 = 5;
    int angka2;

    // Inisialisasi variabel angka2 dengan menugaskan ekspresi 2 dijumlahkan dengan
    angka1++
    angka2 = 2 + angka1++;
```

```
// Mencetak nilai
printf("angka1 = %d\n", angka1);
printf("angka2 = %d\n", angka2);

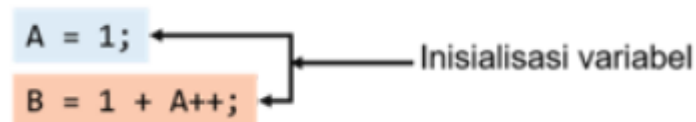
return 0;
}
```

Output program `postIncrement.c`:

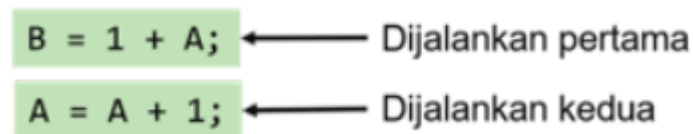
```
angka1 = 6
angka2 = 7
```

Berbeda dengan output program `preIncrement.c`, pada output program `postIncrement.c` variabel `angka2` bernilai `7`, hal ini dikarenakan *compiler* menganggap nilai variabel `angka1` masih bernilai `5`.

Berdasarkan program di atas, di bawah ini merupakan penggambaran mengenai cara kerja operator *post increment*:



maka urutan pengerjaannya seperti di bawah ini:



3.2 OPERATOR DECREMENT

Operator *decrement* merupakan operator yang mengubah nilai suatu variabel dengan cara mengurangi nilai pada variabel tersebut sebanyak satu (`-1`). Ada dua jenis operator *decrement*, diantaranya:

- Operator *pre decrement*
- Operator *post decrement*

3.2.1 OPERATOR PRE DECREMENT

Operator *pre decrement* mempunyai fungsi yang mirip dengan operator *pre increment*. Jika pada operator *pre increment* nilai suatu variabel akan ditambahkan satu atau `+1` terlebih dahulu, maka pada operator *pre decrement*, nilai suatu variabel akan dikurangi satu atau `-1` terlebih dahulu. Gambaran operator *pre decrement* adalah seperti `--A`. Di bawah ini merupakan program `preDecrement.c` di mana mengandung operator *pre decrement* di dalamnya:

```
#include <stdio.h>

int main()
{
    int angka1 = 5;
    int angka2;

    // Inisialisasi variabel angka2 dengan menugaskan ekspresi 2 dijumlahkan dengan
    --angka1
    angka2 = 2 + --angka1;

    // Mencetak nilai
    printf("angka1 = %d\n", angka1);
    printf("angka2 = %d\n", angka2);

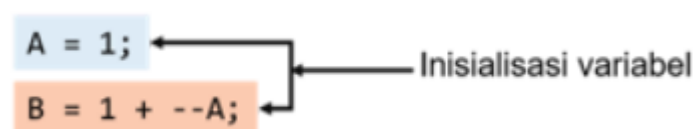
    return 0;
}
```

Output program `preDecrement.c`:

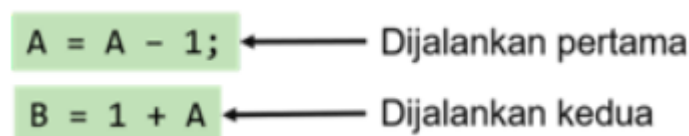
```
angka1 = 4
angka2 = 6
```

Penerapan perhitungan operator *pre decrement* sama seperti perhitungan pada operator *pre increment*, dimana variabel yang mengandung operasi perhitungan terhadap variabel yang memiliki simbol *pre decrement* di dalamnya, akan mempengaruhi nilai variabel tersebut. Maka dari itu nilai dari variabel `angka2` adalah `6`. Hal ini dikarenakan *compiler* akan secara otomatis menganggap nilai dari variabel `angka1` adalah `4`, dan kemudian angka `4` tersebut digunakan dalam ekspresi yang ditugaskan di dalam variabel `angka2` sehingga perhitungannya berubah menjadi `2 + 4`.

Di bawah ini merupakan gambaran mengenai cara kerja dari operator *pre decrement*:



maka urutan pengerjaannya seperti di bawah ini:



3.2.2 OPERATOR POST DECREMENT

Hal yang membedakan antara operator *pre decrement* dengan *post decrement* adalah pada operator *pre decrement*, simbol `--` diletakkan setelah variabel yang akan mengalami *decrement* atau penurunan nilai. Operator *post decrement* akan mengurangi nilai sebanyak satu namun kenaikan nilai tersebut tidak mempengaruhi operasi lain yang melibatkan variabel dengan operator *post decrement* tersebut. Di bawah ini merupakan contoh program `postDecrement.c` yang terdapat operator *postDecrement.c* di dalamnya:

```
#include <stdio.h>

int main()
{
    int angka1 = 5;
    int angka2;

    // Inisialisasi variabel angka2 dengan menugaskan ekspresi 2 dijumlahkan dengan
    --angka1
    angka2 = 2 + angka1--;

    // Mencetak nilai
    printf("angka1 = %d\n", angka1);
    printf("angka2 = %d\n", angka2);

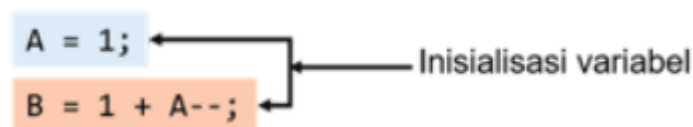
    return 0;
}
```

Output program `postDecrement.c`:

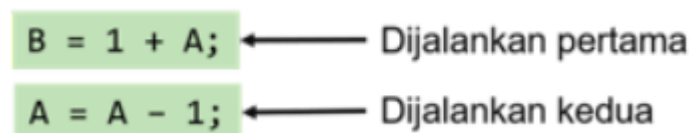
```
angka1 = 4
angka2 = 7
```

Pada output program yang menerapkan operator *post decrement*, terlihat bahwa variabel `angka2` ditugaskan untuk mempunyai nilai `7`. Hal ini dikarenakan pada statement `angka2 = 2 + angka1--`; *compiler* akan mengevaluasi nilai `5` dari variabel `angka1` bukan nilai `4`. Hal ini dikarenakan pengurangan nilai terjadi setelah ekspresi `2 + angka1` dijalankan.

Di bawah ini merupakan gambaran mengenai cara kerja dari operator *pre decrement*:



maka urutan pengerjaannya seperti di bawah ini:



3.3 STATEMENT `while`

Pernyataan `while` digunakan untuk memproses suatu *statement* atau beberapa *statement* beberapa kali. Di bawah ini merupakan bentuk sederhana dari *statement while*:

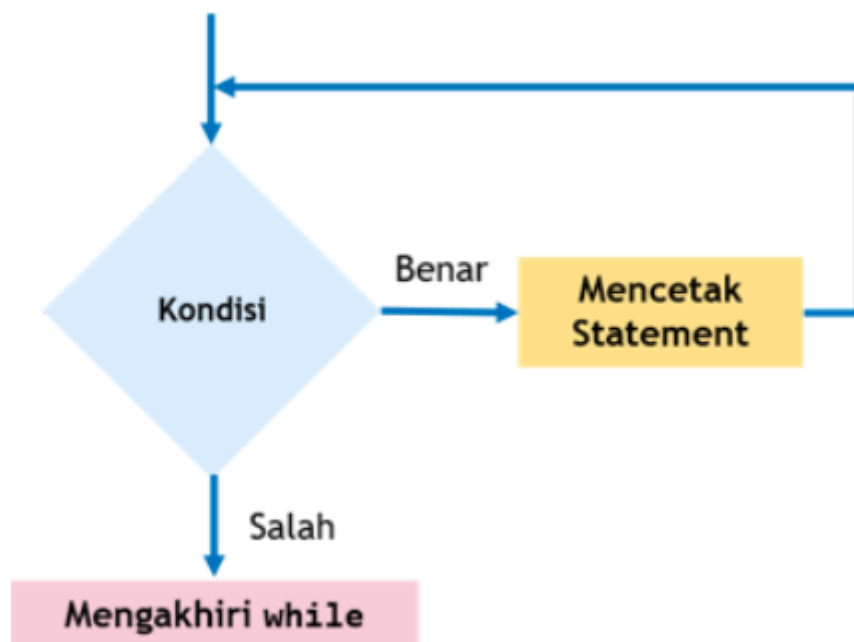
- Bentuk tunggal:

```
While (ekspresi)
{
    statement;
}
```

- Bentuk majemuk:

```
While (ekspresi)
{
    statement1;
    statement2;
    statementN;
}
```

Bagian *statement* yang mengikuti `while` akan dieksekusi selama ekspresi pada `while` bernilai benar (tidak sama dengan nol). Perlu diketahui, pengujian terhadap ekspresi pada `while` dilakukan sebelum bagian *statement*. Oleh karena itu, ada kemungkinan pernyataan `while` tidak dijalankan sama sekali jika kondisi yang pertama kali saat diuji bernilai salah. Di bawah ini merupakan flowchart yang menggambarkan alur perulangan `while`:



Di bawah ini merupakan file program `bilanganGenap.c` yang mengandung perulangan `while` di dalamnya:

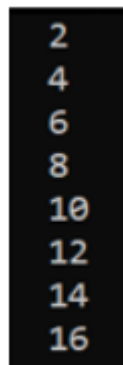
```
#include <stdio.h>

int main()
{
    // Inisialisasi variabel bil untuk ditugaskan dengan nilai 2
    int bil = 2;
    // Memulai kondisi perulangan while
    while (bil <= 16)
    {
        // Statement yang akan dicetak apabila kondisi bernilai benar
        printf ("%d\n", bil);

        // Statement increment untuk menambah nilai dari variabel bil
        bil = bil + 2;
    }

    return 0;
}
```

Output program `bilanganGenap.c`:



```
2
4
6
8
10
12
14
16
```

Pada program di atas, variabel `bil` digunakan sebagai nilai awal yang nantinya akan dijadikan parameter untuk dipanggil ke dalam kondisi. Kondisi dalam perulangan `while` digunakan sebagai penentu untuk mengakhiri pencetakan bilangan genap. Dalam hal ini, nilai dari variabel `bil` akan ditampilkan selama kondisi terpenuhi atau bernilai benar. Nilai dari variabel `bil` terus dicetak dikarenakan adanya *statement*:

```
printf ("%d\n", bil);
```

Penambahan nilai `bil` dikarenakan adanya *statement increment* seperti di bawah ini:

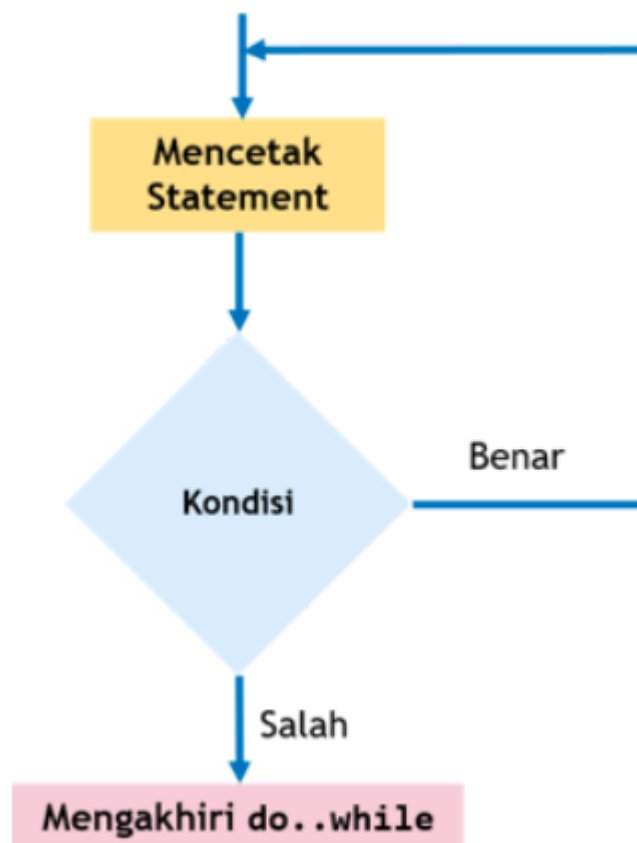
```
bil = bil + 2;
```

Di bawah ini merupakan tabel yang menggambarkan cara kerja perulangan:

Iterasi	bil	bil <= 16	Eksekusi
1	2	True	- Cetak "2" - num = num + 2 menghasilkan num = 4
2	4	True	- Cetak "2" - num = num + 2 menghasilkan num = 4
3	6	True	- Cetak "2" - num = num + 2 menghasilkan num = 4
4	8	True	- Cetak "2" - num = num + 2 menghasilkan num = 4
5	10	True	- Cetak "2" - num = num + 2 menghasilkan num = 4
6	12	True	- Cetak "2" - num = num + 2 menghasilkan num = 4
7	14	True	- Cetak "2" - num = num + 2 menghasilkan num = 4
8	16	True	- Cetak "2" - num = num + 2 menghasilkan num = 4
9	18	False	BERHENTI

3.4 STATEMENT `do...while`

Perulangan `do...while` merupakan perulangan yang akan mencetak *statement* secara berulang sampai ekspresi di dalam `while` tidak terpenuhi atau bernilai salah (sama dengan nol). Di bawah ini merupakan bentuk sederhana dari perulangan `do...while`:



Di bawah ini merupakan file program `bilanganGenap.c` yang mengandung perulangan `do...while` di dalamnya:

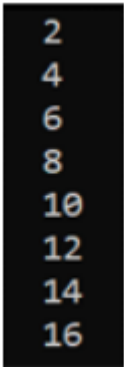
```
#include <stdio.h>

int main()
{
    // Inisialisasi variabel bil untuk ditugaskan dengan nilai 2
    int bil = 2;
    // Memulai kondisi perulangan do
    do
    {
        // Statement yang akan dicetak terlebih dahulu dan dicetak kembali apabila
        // kondisi bernilai benar
        printf ("%d\n", bil);

        // Statement increment untuk menambah nilai dari variabel bil
        bil = bil + 2;
    } while (bil <= 16); // kondisi di dalam while

    return 0;
}
```

Output program `bilanganGenap.c`:



```
2
4
6
8
10
12
14
16
```

Program `do...while` di atas memiliki output yang sama dengan perulangan `while`, yang membedakan adalah pada perulangan `do...while`, meskipun kondisi tidak terpenuhi pada proses seleksi, statement akan tetap dicetak. Di bawah ini merupakan perbedaan antara perulangan `while` dan `do...while` digambarkan dengan program dimana kondisi tidak terpenuhi:

- Statement `do..while`:

```
#include <stdio.h>

int main()
{
    // Inisialisasi variabel bil untuk ditugaskan dengan nilai 17
    int bil = 17;
    // Memulai kondisi perulangan do
    do
    {
```

```
// Statement yang akan dicetak terlebih dahulu dan dicetak kembali apabila
kondisi bernilai benar
printf ("%d\n", bil);

// Statement increment untuk menambah nilai dari variabel bil
bil = bil + 2;
} while (bil <= 16); // kondisi di dalam while

return 0;
}
```

Output:



- Statement `while`:

```
#include <stdio.h>

int main()
{
    // Inisialisasi variabel bil untuk ditugaskan dengan nilai 17
    int bil = 17;
    // Memulai kondisi perulangan while
    while (bil <= 16)
    {
        // Statement yang akan dicetak apabila kondisi bernilai benar
        printf ("%d\n", bil);

        // Statement increment untuk menambah nilai dari variabel bil
        bil = bil + 2;
    }

    return 0;
}
```

Output:



CATATAN:

Pada perulangan `while`, *compiler* akan menyeleksi kondisi terlebih dahulu, lalu ketika kondisi terpenuhi atau bernilai benar, maka *statement* akan dieksekusi. Nilai yang akan digunakan untuk seleksi kondisi adalah nilai dari variabel `bil`. Pada program di atas, nilai dari variabel `bil` adalah 17, sehingga program tidak akan menampilkan output apapun karena kondisi tidak terpenuhi. Sedangkan perulangan `do...while` adalah sebaliknya. Pada program ini *statement* akan dicetak terlebih dahulu, lalu, dilakukan seleksi kondisi. Apabila kondisi bernilai benar, maka *statement*

akan dicetak kembali, lalu ketika bernilai salah, program tidak akan mencetak *statement* tersebut kembali.

3.5 STATEMENT `for`

Pernyataan `for` berguna untuk mengulang eksekusi terhadap satu atau sejumlah *statement*. Di bawah ini merupakan bentuk umum dari *statement for*:

```
for(ekspresi1, ekspresi2, ekspresiN)
{
    statement1;
    statement2;
    statementN;
}
```

Contoh program `bilanganGenap.c`:

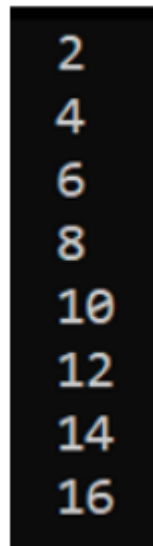
```
#include <stdio.h>

int main()
{
    // Inisialisasi variabel bil untuk ditugaskan dengan nilai 17
    int bil;

    // Memulai kondisi perulangan for
    for (bil = 2; bil <= 16; bil++)
    {
        if (bil%2==0){
            // Statement yang akan dicetak apabila kondisi bernilai benar
            printf ("%d\n", bil);
        }
    }

    return 0;
}
```

Output program `bilanganGenap.c`:



```
2
4
6
8
10
12
14
16
```

Pada *statement* `for`, kondisi dideklarasikan di dalam kurung setelah keyword `for`. Cara kerja perulangan `for` sama dengan perulangan `while`, di mana kondisi diseleksi terlebih dahulu lalu kemudian mencetak *statement* di dalam perulangan apabila kondisi bernilai benar. Perbedaannya adalah, dalam *statement* `for`, banyaknya perulangan telah diketahui, sedangkan pada perulangan `while`, lebih mengacu kepada syarat suatu kondisi dan tidak diketahui berapa jumlahnya.

3.5.1 FOR DENGAN EKSPRESI KOSONG

Statement `for` kadang dijumpai yang tidak mengandung bagian ekspresi yang lengkap (satu atau dua atau tiga ekspresi yang berada di dalam `()` setelah `for` dalam keadaan kosong). Hal itu memang diperbolehkan. Contoh:

```
for (abjad = 'A'; abjad <= `2` : abjad++)
    printf ("%c", abjad);
```

dapat ditulis menjadi:

```
abjad = 'A';
for (; abjad <= '2': abjad++)
    printf ("%c", abjad);
```

Tampak bahwa ekspresi tepat setelah tanda `(` pada `for` tidak ada. Pada contoh di atas, ekspresi

`abjad = 'A'`

diletakkan di atas `for`.

3.5.2 FOR BERSARANG (NESTED FOR)

Pada program tertentu, terkadang pernyataan `for` yang juga berada di dalam *statement* `for` digunakan. Di bawah ini merupakan contoh penerapan `for` bersarang pada suatu program:

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

```

int main()
{
    int tinggi, // Menyatakan tinggi segitiga
    pencacahBaris, // Pencacah untuk baris
    pencacahBintang; // Pencacah untuk menampilkan *

    printf("Tinggi segitiga = ");
    scanf("%d", &tinggi);

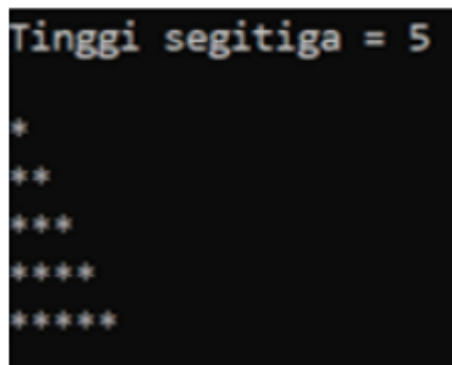
    printf("\n"); // Membuat baris kosong

    // Membuka statement for
    for(pencacahBaris = 1; pencacahBaris <= tinggi; pencacahBaris++)
    {
        for(pencacahBintang = 1; pencacahBintang <= pencacahBaris; pencacahBintang++)
        {
            printf("*");
        }
        printf("\n"); // Pindah baris
    }

    return 0;
}

```

Output:



```

Tinggi segitiga = 5

*
**
***
****
*****

```

Pada program di atas merupakan jenis program dengan perulangan bersarang atau **nested** `for`. Perulangan yang akan dijalankan terlebih dahulu adalah perulangan yang ada di luar, kemudian beralih ke perulangan yang ada di dalamnya. Jika kondisi di perulangan dalam terpenuhi, maka kondisi di perulangan dalam akan terus dieksekusi sampai kondisi tidak lagi terpenuhi, kemudian menyeleksi kondisi di perulangan luar kembali. Logika untuk menyelesaikan persoalan ini akan dijelaskan di bawah ini:

Baris 1: ada 1 *

Baris 2: ada 2 *

Baris 3: ada 3 *

Baris 4: ada 4 *

Baris 5: ada 5 *

Angka yang ada di sebelah kiri yang ada di samping kata baris adalah pencacah posisi baris, yang mana variabel `pencacahBaris` berfungsi untuk melakukan perulangan untuk mencetak `\n` atau pada output akan selalu tercetak baris baru selama kondisi ekspresi `pencacahBaris <= tinggi`

masih terpenuhi atau bernilai benar. Perulangan yang mengandung ekspresi `pencacahBaris <= tinggi` merupakan perulangan yang berada di luar. Namun alurnya adalah, jika kondisi perulangan luar terpenuhi, maka akan menjalankan perintah yang ada di dalam perulangan ini, di mana perintah yang dijalankan adalah perulangan untuk mencetak `*`.

Sedangkan perulangan untuk mencetak bintang dimana banyaknya bintang sesuai dengan barisnya seperti pada baris pertama mencetak satu tanda bintang, pada baris kedua mencetak dua tanda bintang, dan seterusnya memerlukan variabel `pencacahBintang`. Apabila ekspresi `pencacahBintang <= pencacahBaris;` terpenuhi, maka bintang akan terus dicetak. Perulangan ini berada di dalam perulangan kondisi dengan ekspresi `pencacahBaris <= tinggi`.

REFERENSI

[1] Abdul Kadir. 2015. *From Zero to a Pro*. Yogyakarta. Andi