

PENGULANGAN (LOOPING)

8



Overview

Pengulangan (*Loop*) merupakan sebuah konsep yang penting dalam pemrograman. Dengan struktur pengulangan, program dapat berjalan beberapa kali sesuai inisialisasi, jumlah iterasi dan kondisi berhenti yang ditentukan. Hal ini dapat menyederhanakan program yang kompleks menjadi lebih sederhana. Dalam C disediakan berbagai perintah *Loop*, dimana setiap perintah *loop* memiliki keunikan tersendiri. Di dalam bab ini akan dijelaskan tentang struktur pengulangan dalam algoritma serta implementasi struktur perulangan dengan perintah *loop* yang ada di dalam C.



Tujuan

- Memahami konsep dasar dan struktur pengulangan
- Memahami perintah pengulangan dalam C
- Menerapkan sintaks-sintaks pengulangan dalam menyelesaikan persoalan

8.1 Konsep Pengulangan

Program yang efisien adalah program yang memungkinkan pengguna bekerja sesedikit mungkin dan komputer bekerja sebanyak mungkin. Salah satu cara melakukan hal tersebut adalah dengan menggunakan kembali sekumpulan baris program yang terdapat pada bagian lain dari program tersebut atau baris program yg terdapat di dalam program lain.

Pengulangan merupakan sebuah konsep pemrograman yang penting karena konsep ini memungkinkan pengguna menggunakan sekumpulan baris program berulang kali dengan tiga komponen yang mengendalikannya, yaitu:

- ↳ **Inisialisasi;** menentukan kondisi awal dilakukannya pengulangan.
- ↳ **Jumlah iterasi;** menunjukkan berapa kali pengulangan akan dilakukan.
- ↳ **Kondisi berhenti;** menentukan kondisi yang dapat mengakhiri pengulangan.

Contoh kasus dunia nyata yang dapat digunakan untuk menggambarkan ketiga komponen ini adalah cerita ibu mengupas sepuluh (10) butir kentang. Ibu akan mengumpulkan dulu 10 butir kentang yang akan dikupas, kemudian Ibu akan mengambil sebuah kentang kemudian mengupasnya, setelah selesai mengupas Ibu akan mengambil kentang berikutnya dan melakukan aksi mengupas lagi. Ibu akan melakukan pengupasan kentang satu persatu hingga mencapai kentang ke-10, dan seluruh kentang tersebut telah terkupas. Ibu akan melakukan sederetan aksi yang tepat sama terhadap kesepuluh butir kentang tersebut. Maka,

- ↳ **Inisialisasi:** 10 butir kentang yang masih utuh dengan kulitnya
- ↳ **Jumlah iterasi:** 10 (sesuai jumlah kentang)
- ↳ **Kondisi berhenti:** 10 butir kentang sudah terkupas.

Ketika mengimplementasikan dalam program, ketiga komponen ini tidak selalu dapat didefinisikan dalam struktur pengulangan. Mungkin saja salah satu komponen tersebut tidak didefinisikan. Pengulangan tetap dapat berjalan, asal komponen yang tidak didefinisikan tersebut dapat diketahui secara tersirat

berdasarkan komponen lain yang didefinisikan. Hal lain yang perlu diperhatikan adalah bahwa **pengulangan harus berhenti**. Jika pengulangan tidak pernah berhenti, maka logika program salah. Pengulangan akan berhenti jika jumlah iterasi yang diminta sudah tercapai atau kondisi berhenti bernilai benar. Maka, dalam setiap pengulangan, pemrogram perlu menentukan jumlah iterasi atau kondisi berhenti dan langkah pencapaian menuju kondisi berhenti tersebut.

Pada bab ini akan dijelaskan 3 struktur perulangan dan implementasinya di dalam C, yaitu struktur perulangan While, For, dan Do While

8.1.1 Sintaks WHILE

Pengulangan dengan menggunakan WHILE merupakan sebuah pengulangan yang dikendalikan oleh suatu kondisi tertentu, dimana kondisi tersebut yang akan menentukan apakah perulangan itu akan terus dilaksanakan atau dihentikan. Kondisi tersebut akan dicek disetiap awal iterasi, apakah sebuah kondisi terpenuhi atau tidak. Jika kondisi terpenuhi (bernilai benar), maka iterasi akan dilanjutkan. Jika kondisi tidak terpenuhi, maka iterasi dihentikan.

Perulangan dengan WHILE dapat digunakan pada struktur perulangan yang diketahui jumlah iterasinya dan juga pada struktur perulangan yang tidak diketahui jumlah iterasinya, tetapi harus selalu terdapat kondisi berhenti. Struktur pengulangan WHILE adalah:

```
while (ungkapan);  
{  
    pernyataan1;  
    pernyataan2;  
    .....  
    pernyataanN;  
}
```

Pencacah adalah variabel pengendali iterasi yang harus diinisialisasi, dicek dalam kondisi, dan terus berubah nilainya

setiap iterasi dilakukan. Pencacahan inilah yang akan membuat sebuah kondisi berhenti tercapai. Pada struktur pengulangan dengan sintaks WHILE, nilai pencacahan akan diubah di akhir aksi pengulangan.

Contoh: Ibu mengupas 10 butir kentang dapat direpresentasikan dengan pengulangan WHILE sebagai berikut :

ALGORITMA Kupas Kentang	
IS :	Terdapat 10 kentang belum dikupas
FS :	10 kentang telah terkupas
KAMUS DATA	
	kentang : integer
1	{inisialisasi jumlah kentang yang sudah dikupas}
2	kentang \leftarrow 0
3	WHILE (kentang < 10) {kondisi iterasi dilakukan}
4	Ambil sebuah kentang
5	Kupas kulit kentang
	kentang \leftarrow kentang + 1 {pencapaian kondisi berhenti}

Telah diketahui bahwa Ibu akan melakukan pengupasan sebanyak 10 kentang, maka sebelum masuk struktur pengulangan, variabel kentang berisi 0 {menunjukkan bahwa di awal iterasi belum ada kentang yang dikupas atau jumlah kentang yg telah dikupas = 0}.

Pada iterasi pertama, terjadi pengecekan kondisi (kentang < 10) dimana artinya “apakah jumlah kentang yang dikupas belum mencapai 10”, karena nilai **kentang = 0** maka kondisi bernilai benar, ibu akan mengambil sebuah kentang kemudian mengupas kulitnya. Di akhir iterasi pertama, jumlah kentang yang sudah dikupas menjadi $(0+1) = 1$. Pada titik ini, variabel kentang berisi 1 (artinya, 1 kentang sudah dikupas).

Pada iterasi kedua, terjadi pengecekan kondisi (kentang < 10), sedangkan kentang = 1, maka kondisi bernilai benar, ibu akan mengambil sebuah kentang kemudian mengupas kulitnya. Di akhir iterasi kedua, jumlah kentang yang sudah dikupas menjadi $(1+1)=2$.

Iterasi ini terus berulang sampai iterasi ke 10, variabel kentang berisi 9. Saat terjadi pengecekan kondisi (kentang < 10) bernilai benar, maka ibu mengambil sebuah kentang kemudian

mengupas kulitnya. Di akhir iterasi ke sepuluh, jumlah kentang yang sudah dikupas menjadi $(9+1) = 10$.

Kemudian, pada iterasi ke 11, terjadi pengecekan kondisi ($kentang < 10$), karena $kentang = 10$ maka kondisi bernilai salah, sehingga iterasi diakhiri.

Hasil akhirnya, variabel kentang berisi 10 (artinya, 10 kentang sudah dikupas).

Jalannya iterasi ini dapat ditulis dalam bentuk tabel berikut:

Iterasi ke-	Kentang	$kentang <$ 10	ambil & kupas	kentang
1	0	Ya	Ya	1
2	1	Ya	Ya	2
3	2	Ya	Ya	3
4	3	Ya	Ya	4
5	4	Ya	Ya	5
6	5	Ya	Ya	6
7	6	Ya	Ya	7
8	7	Ya	Ya	8
9	8	Ya	Ya	9
10	9	Ya	Ya	10
11	10	Tidak	Tidak	-

Dari contoh di atas, variabel kentang merupakan pengendali iterasi. Iterasi dapat terus dilakukan atau tidak, bergantung pada nilai variabel kentang ini. Selanjutnya, variabel penentu iterasi ini disebut dengan pencacah. Pencacah harus berupa nilai yang memiliki urutan, yaitu dapat bertipe integer atau karakter. Di setiap struktur pengulangan, pencacah selalu ada dan jangan lupa untuk menginisialisasi pencacah. Nilai pencacah akan berubah pada setiap iterasi.

Hal lain yang perlu diperhatikan adalah bahwa di akhir iterasi, variabel kentang bernilai 10. Nilai ini tidak berubah lagi karena iterasi tidak dilakukan lagi, dan disebut sebagai *loop invariant*.

Contoh lain struktur pengulangan ini:

↳ Algoritma untuk menampilkan karakter '*' sebanyak 5 kali.

ALGORITMA Tampil_Bintang

IS :-

FS : Jumlah bintang yg tampil = 5

KAMUS DATA

i : integer

```
1     i ← 0 {inisialisasi pencacah i}
2     WHILE (i < 5) {jumlah iterasi}
3         Output ('*') {aksi}
4     i ← i + 1 {pencapaian kondisi berhenti}
```

Output dari algoritma ini:

```
*
```

↳ Algoritma untuk menampilkan iterasi ke 1 sampai 5 dengan pengulangan.

ALGORITMA Iterasi_Angka

IS :-

FS : Tampil angka 1 hingga 5

KAMUS DATA

i : integer

```
1     i ← 1 {inisialisasi pencacah i}
2     WHILE (i ≤ 5)
3         Output ('Ini adalah iterasi ke-',i)
4         i ← i + 1
5
```

Output dari algoritma ini:

```
Ini adalah iterasi ke-1
Ini adalah iterasi ke-2
Ini adalah iterasi ke-3
Ini adalah iterasi ke-4
Ini adalah iterasi ke-5
```

- ↳ Algoritma untuk memasukkan nilai tiga (3) orang mahasiswa

ALGORITMA Input_Nilai

IS :-

FS : Menerima inputan nilai dari user

KAMUS DATA

i : integer

nilai : integer

```

1      i ← 1
2      WHILE (i <= 3)
3          Output ('Nilai mahasiswa ke-‘,i,’ adalah:')
4          Input (nilai)
5          i ← i + 1
    
```

Output dari algoritma ini:

Nilai mahasiswa ke-1 adalah: _____

Nilai mahasiswa ke-2 adalah: _____

Nilai mahasiswa ke-3 adalah: _____

- ↳ Algoritma untuk menghitung nilai rata-rata dari tiga bilangan yang diinputkan.

ALGORITMA Nilai_Rata_Rata

IS :-

FS : Tampil rata-rata dari nilai yang diinputkan

KAMUS DATA

jumlah : float

bilangan : integer

x : float

rerata : float

```

1      jumlah ← 0      {variabel jumlah bilangan}
2      bilangan ← 3    {inisialisasi variabel pencacah}
3      WHILE (bilangan > 0)
4          Output('Masukkan angka : ')
5          Input (x)     {masukkan sebuah bilangan}
6          jumlah ← jumlah + x {tambahkan bilangan}
7          bilangan ← bilangan - 1 {pencacah mundur}
    
```

8	rerata \leftarrow jumlah/ 3 {menghitung rerata}
9	Output ('Rerata : ',rerata)

Pada contoh algoritma yang ke-4, ditunjukkan bahwa iterasi dapat dilakukan dengan pencacah mundur. Selama kondisi bilangan > 0 terpenuhi, maka pengguna diminta memasukkan sebuah bilangan yang kemudian dijumlahkan dengan bilangan-bilangan sebelumnya (pada iterasi pertama, bilangan dijumlahkan dengan nol). Karena menggunakan iterasi mundur, maka pencacah akan dikurangkan. Algoritma akan memberikan output berupa hasil perhitungan rerata dari ketiga bilangan yang diinputkan.

Jika pada contoh algoritma yang ke-4 diatas diinputkan angka 60, 70, dan 90, maka jalannya iterasi ini dapat ditulis dalam bentuk tabel berikut:

Iterasi ke-	bilangan	x	jumlah	rerata	bilangan
1	3	60	60	-	2
2	2	70	130	-	1
3	1	90	220	-	0
4	0	-	220	73.33	-

Penggunaan sintaks WHILE dalam bahasa pemrograman C :

Algoritma	C
WHILE (<i>kondisi</i>) <i>aksi</i> <i>ubah pencacah</i>	while (<i>kondisi</i>) { // <i>aksi</i> // <i>ubah pencacah</i> }
... <i>i</i> \leftarrow 0 WHILE (<i>i</i> < 5) Output('*') <i>i</i> \leftarrow <i>i</i> + 1 <i>i</i> = 0; while (<i>i</i> < 5) { printf("*"); <i>i</i> = <i>i</i> + 1; } ...

Implementasi algoritma contoh ke-4 ke dalam C adalah :

```
1 //Nilai_Rata_Rata.c
2 //Menerima 3 angka dan menampilkan rata-ratanya
3 #include <stdio.h>
4 main ()
5 {
6     //deklarasi variabel yg digunakan
7     float jumlah;
8     int bilangan;
9     float x;
10    float rerata;
11    //inisialisasi variabel jumlah
12    jumlah = 0;
13    //inisialisasi variabel pencacah bilangan
14    bilangan = 3;
15    while (bilangan > 0 )
16    {
17        //meminta inputan dari user
18        printf("Masukkan angka : ");
19        scanf("%f",&x);
20        //menjumlahkan angka yg diinputkan
21        jumlah = jumlah + x;
22        //mengurangkan pencacah untuk mencapai
23        //kondisi berhenti
24        bilangan = bilangan - 1;
25    }
26    //menghitung rata-rata dari 3 inputan angka
27    rerata = jumlah / 3;
28    //menampilkan hasil rata-rata
29    printf("Rerata : %.2f",rerata);
30 }
```

Telah dikatakan di awal bahwa perulangan dengan WHILE dapat digunakan untuk struktur pengulangan yang belum diketahui jumlah iterasinya, tetapi tetap mempunyai kondisi berhenti. Contoh struktur pengulangan ini adalah sebagai berikut :

ALGORITMA Tebak_Huruf

IS :-

FS : Menampilkan pesan “Maaf Anda salah” jika tebakan salah, dan menampilkan “Anda Benar” jika tebakan benar

KAMUS DATA

huruf : character

```
1 {inisialisasi huruf tebakan pertama }
2 Output('Masukkan tebakan : ')
3 Input(huruf)
4 WHILE (huruf != 'q') {pengecekan kondisi}
5   Output('Maaf Anda salah ')
6   Output('Masukkan tebakan : ')
7   Input(huruf) {input huruf berikutnya}
8 Output ('Anda Benar')
```

Pada algoritma diatas tidak terlihat adanya deklarasi variabel pencacah dan inisialisasinya, juga tidak terlihat adanya perubahan terhadap variabel pencacah untuk mencapai kondisi berhenti. Pengulangan diatas tetap dapat berjalan, karena komponen yang tidak didefinisikan tersebut dapat diketahui secara tersirat berdasarkan komponen lain yang didefinisikan. Dan yang paling penting adalah perulangan ini mempunyai kondisi berhenti.

Pada baris ke-4 user akan memasukkan sebuah karakter. Karakter inilah yang akan menjadi pemicu dilaksanakan pengulangan atau tidak, jika karakter yang dimasukkan adalah ‘q’, maka pengulangan tidak dilanjutkan, tetapi jika karakter yang dimasukkan bukan ‘q’, maka pengulangan dilanjutkan. Dapat dilihat pada baris ke-5 bahwa kondisi berhenti pengulangan adalah ketika user memasukkan (input) huruf ‘q’.

Jumlah iterasi tidak ditentukan oleh variabel pencacah, melainkan ditentukan sendiri oleh user yang menginputkan huruf. Berikut adalah implementasi dari algoritma Tebak_Huruf dalam bahasa pemrograman C.

```
1 //Tebak_Huruf.c
2 //User memasukkan sebuah karakter untuk menebak
3 #include <stdio.h>
```

```

4 main()
5 {
6     //deklarasi variabel untuk menerima input
7     char huruf;
8     //menginisialisasi huruf awal untuk dicek
9     printf("Tebakan : ");
10    scanf("%c",&huruf);
11    //pengecekan kondisi terhadap huruf inputan
12    while (huruf!='q')
13    {
14        //jika huruf bukan 'q' maka input huruf lain
15        printf("Maaf anda salah");
16        printf("\nTebakan : ");
17        scanf("%c",&huruf);
18    }
19    //jika huruf = 'q' maka tidak masuk ke while
20    printf("Anda Benar");
21 }

```

Bagian pernyataan yang mengikuti **while** akan dieksekusi selama *ungkapan* pada **while** bernilai benar (tidak sama dengan nol). Pengujian terhadap ungkapan **while** dilakukan sebelum bagian pernyataan. Perhatikan contoh-contoh berikut :

Contoh 24	Hasil
<pre> /* ** Contoh 24 : pemakaian while untuk * ** menampilkan tulisan C++ * ** sebanyak 10 kali * */ #include <iostream.h> #include <conio.h> void main() { int i; // Sebagai variabel pencacah yang menyatakan // jumlah tulisan C++ yang harus ditampilkan clrscr(); // Hapus layar i = 0; // Mula-mula diisi sama dengan nol while (i < 10) { cout << " C++ " << endl; i ++ ; // Menaikkan pencacah sebesar 1 } } </pre>	C++ C++ C++ C++ C++ C++ C++ C++ C++ C++

Pada program diatas, variabel i bertindak sebagai pencacah yang gunanya untuk mengingat jumlah tulisa C++ yang telah ditampilkan. Itulah sebabnya mula-mula didisi dengan nol. Kemudian untuk setiap putaran, isi variabel ini dinaikkan. Oleh karena variabel i dijadikan sebagai kondisi pada **while**, suatu ketika ketika kondisi $i < 10$ akan bernilai salah, maka **while** berakhir.

8.1.2 Sintaks DO... WHILE

Sintaks DO... WHILE... melakukan pengulangan serupa dengan sintaks WHILE. Penggunaan sintaks ini juga tidak harus menyebutkan jumlah pengulangan yang harus dilakukan, karena dapat digunakan untuk perulangan dengan jumlah iterasinya yang belum diketahui, tetapi harus mempunyai kondisi berhenti.

Bedanya, jika pada sintaks WHILE kondisi dievaluasi/ diuji sebelum aksi pengulangan dilakukan, sedangkan pada sintaks DO... WHILE pengujian kondisi dilakukan setelah aksi pengulangan dilakukan.

Struktur pengulangan DO... WHILE yaitu:

```
do
{
    pernyataan1;
    pernyataan2;
    ....
    pernyataanN;
} while (ungkapan)
```

Pada struktur pengulangan dengan sintaks DO... WHILE..., aksi akan terus dilakukan hingga kondisi yang dicek di akhir pengulangan, bernilai benar. Dengan sintaks ini, pengulangan pasti dilakukan minimal satu kali, yakni pada iterasi pertama sebelum pengecekan kondisi. WHILE dengan DO WHILE seringkali memberikan hasil yang sama, tetapi ada kalanya hasilnya akan berbeda, sehingga harus berhati-hati dalam penggunaan kondisi antara WHILE dengan DO WHILE. Dengan kata lain Bagian *pernyataan1* hingga *pernyataanN* dijalankan secara berulang sampai *ungkapan* bernilai salah (sama dengan nol). Namun berbeda

dengan **while**, pengujian *ungkapan* dilakukan dibelakang (setelah bagian *pernyataan*).

Beberapa contoh penerapan struktur pengulangan DO... WHILE... :

↳ Algoritma ibu mengupas Kentang

ALGORITMA Kupas_Kentang	
IS : Terdapat 10 kentang belum dikupas	
FS : 10 kentang telah terkupas	
KAMUS DATA	
kentang : integer	
1	{inisialisasi jumlah kentang yang sudah dikupas}
2	kentang \leftarrow 0
3	DO
4	Ambil sebuah kentang
5	Kupas kulit kentang
6	{pencapaian kondisi berhenti}
7	kentang \leftarrow kentang + 1
8	WHILE (kentang < 10) {kondisi berhenti}

Pada potongan algoritma ini, aksi pasti dilakukan minimal satu kali, tanpa memperhatikan nilai pencacahan. Di akhir iterasi pertama, baru dilakukan pengecekan jumlah kentang yang sudah terkupas (pencacahan).

Jalannya iterasi ini dapat ditulis dalam bentuk tabel berikut:

Iterasi ke-	{aksi}	kentang	Kentang < 10
1	Ya	1	Ya
2	Ya	2	Ya
3	Ya	3	Ya
4	Ya	4	Ya
5	Ya	5	Ya
6	Ya	6	Ya
7	Ya	7	Ya
8	Ya	8	Ya
9	Ya	9	Ya
10	Ya	10	Tidak

Pengulangan dihentikan pada iterasi ke- 10 karena kondisi kentang < 10 bernilai salah.

- ↳ Algoritma untuk menampilkan karakter * sebanyak 5 kali.

ALGORITMA Tampil_Bintang	
IS : Jumlah bintang yg tampil = 0	
FS : Jumlah bintang yg tampil = 5	
KAMUS DATA	
i : integer	
1	i \leftarrow 0 {inisialisasi pencacah i}
2	DO
3	Output (*) {aksi}
4	i \leftarrow i + 1 {pencapaian kondisi berhenti}
5	WHILE (i < 5) {jumlah iterasi}
6	

Output dari algoritma ini:

*
*
*
*
*

Pengulangan dihentikan pada iterasi ke- 5 karena kondisi $i < 5$ salah.

- ↳ Algoritma untuk menampilkan iterasi ke 1 sampai 5 dengan pengulangan.

ALGORITMA Iterasi_Angka	
IS :-	
FS : Tampil angka 1 hingga 5	
KAMUS DATA	
i : integer	
1	i \leftarrow 1 {inisialisasi pencacah i}
2	DO
3	Output ('Ini adalah iterasi ke-', i) {aksi}
4	i \leftarrow i + 1
5	WHILE (i \leq 5) {kondisi berhenti}

Output dari algoritma ini:

```
Ini adalah iterasi ke-1  
Ini adalah iterasi ke-2  
Ini adalah iterasi ke-3  
Ini adalah iterasi ke-4  
Ini adalah iterasi ke-5
```

Pengulangan dihentikan pada iterasi ke- 5 dimana nilai i = 6 sehingga kondisi $i \leq 5$ salah.

↳ Algoritma untuk memasukkan nilai tiga (3) orang mahasiswa

ALGORITMA Input_Nilai

IS :-

FS : Menerima inputan nilai dari user

KAMUS DATA

i : integer

nilai : integer

```
1 BEGIN
2   i ← 1 {inisialisasi}
3   DO
4     Output ('Nilai mahasiswa ke- ', i, ' adalah: ')
5     Input (nilai)
6     i ← i + 1
7   WHILE(i <= 3) {kondisi berhenti}
```

Output dari algoritma ini:

```
Nilai mahasiswa ke-1 adalah: _
```

```
Nilai mahasiswa ke-2 adalah: _
```

```
Nilai mahasiswa ke-3 adalah: _
```

Pengulangan dihentikan pada iterasi ke- 3, ketika nilai i = 4 sehingga kondisi $i \leq 3$ salah.

↳ Algoritma untuk menghitung nilai rata-rata dari tiga bilangan yang diinputkan.

ALGORITMA Nilai_Rata_Rata

IS :-

FS : Tampil rata-rata dari nilai yang diinputkan

KAMUS DATA

jumlah : float

bilangan : integer

x : float

rerata : float

```

1 jumlah ← 0      {variabel jumlah bilangan}
2 bilangan ← 3    {inisialisasi variabel pencacah}
3 DO
4   Output('Masukkan angka : ')
5   Input (x)      {masukkan sebuah bilangan}
6   jumlah ← jumlah + x {tambahkan bilangan}
7   bilangan← bilangan – 1 {pencacah mundur}
8 WHILE (bilangan > 0)
9   rerata ← jumlah/ 3 {menghitung rerata}
10  Output ('Rerata : ',rerata)
```

Pada algoritma ini juga ditunjukkan bahwa iterasi pada pengulangan dengan sintaks DO... WHILE... dapat dilakukan dengan pencacah mundur. Pengguna terus diminta memasukkan sebuah bilangan yang kemudian dijumlahkan dengan bilangan-bilangan sebelumnya (pada iterasi pertama, bilangan dijumlahkan dengan nol) hingga pencacah bernilai 0. Program akan memberikan output berupa hasil perhitungan rerata dari ketiga bilangan yang diinputkan.

Penggunaan DO... WHILE dalam pemrograman C :

<i>Algoritma</i>	<i>C</i>
DO aksi ubah pencacah WHILE (kondisi)	do { //aksi //ubah pencacah } while (kondisi);

Implementasi algoritma contoh ke-8 ke dalam C adalah :

```

1 //InputNilai.c
2 //User diminta untuk input nilai sebanyak 3 kali
3 #include <stdio.h>
```

```

4 main()
5 {
6     int i; //deklarasi pencacah
7     int nilai;
8     i=1; //inisialisasi pencacah
9     do
10    {
11        printf("Nilai mahasiswa ke-%d adalah : ",i);
12        scanf("%d",&nilai); //input nilai dari user
13        i=i+1; //penambahan nilai pencacah
14    }
15    while (i<=3); //kondisi berhenti
16 }

```

Tidak semua implementasi kondisi pada DO...WHILE sama dengan implementasi pada WHILE, contohnya adalah algoritma berikut :

WHILE
ALGORITMA Tebak_Huruf
IS :-
FS : Menampilkan pesan “Maaf Anda salah” jika tebakan salah, dan menampilkan “Anda Benar” jika tebakan benar
KAMUS DATA
huruf : character
1 Output('Masukkan tebakan : ')
2 Input(huruf)
3 WHILE (huruf != 'q')
4 Output('Maaf Anda salah ')
5 Output('Masukkan tebakan : ')
6 Input(huruf)
7 Output ('Anda Benar')
DO...WHILE
ALGORITMA Tebak_Huruf
IS :-
FS : Menampilkan pesan “Maaf Anda salah” jika tebakan salah, dan menampilkan “Anda Benar” jika tebakan benar

KAMUS DATA

huruf : character

- 1 Output('Masukkan tebakan : ')
- 2 Input(huruf)
- 3 DO
- 4 Output('Maaf Anda salah ')
- 5 Output('Masukkan tebakan : ')
- 6 Input(huruf)
- 7 WHILE (huruf != 'q')
- 8 Output ('Anda Benar')

Perbandingan output dari algoritma diatas adalah sebagai berikut

WHILE

Masukkan tebakan : q
Anda Benar

DO... WHILE

Masukkan tebakan : q
Maaf Anda Salah
Masukkan tebakan :

Bila user memasukkan inputan 'q' pada pertanyaan yang kedua ini, maka pengulangan akan dihentikan, tetapi jika user memasukkan huruf yang lain maka pengulangan akan dilanjutkan. Dari contoh diatas dapat dilihat bahwa penggunaan sintaks WHILE dan DO... WHILE kadang akan memberikan output yang berbeda.

Sama seperti pada penggunaan sintaks WHILE, sintaks DO... WHILE dapat digunakan untuk struktur pengulangan yang belum diketahui jumlah iterasinya, tetapi tetap mempunyai kondisi berhenti.

Contoh struktur pengulangan tersebut adalah sebagai berikut :

ALGORITMA Menu

IS :-

FS : Menampilkan menu yang dipilih user

KAMUS DATA

pilihan : integer

- 1 DO
- 2 Output('MENU : ')

```

3     Output('1. Ulang')
4     Output('2. Keluar')
5     Output('Pilihan : ')
6     Input(pilihan)
7 WHILE (pilihan != 2) {pengecekan kondisi}
8 Output ('Anda Pilih Keluar')

```

Karena pada struktur DO...WHILE tidak terdapat pengecekan kondisi di awal, maka isi dari DO...WHILE akan langsung dijalankan, pengecekan akan dilakukan setelah user memasukkan pilihan. Yang paling penting adalah perulangan ini mempunyai kondisi berhenti, yaitu ketika user input angka 2. Jumlah iterasi tidak ditentukan oleh variabel pencacah, melainkan ditentukan sendiri oleh user yang menginputkan angka. Berikut adalah implementasi dari algoritma Menu dalam bahasa pemrograman C.

```

1 //Menu.c
2 //User memasukkan pilihan menu 1 atau 2
3 #include <stdio.h>
4 main()
5 {
6     int pilihan;
7     do
8     {
9         printf("MENU");
10        printf("\n1. Ulang");
11        printf("\n2. Keluar");
12        printf("\nPilihan : ");
13        scanf("%d",&pilihan);
14    }
15    while (pilihan != 2);
16    printf("\nAnda pilih keluar");
17 }

```

Output dari sintaks diatas adalah

```

MENU
1. Ulang
2. Keluar

```

Pilihan : 1
MENU
1. Ulang
2. Keluar
Pilihan : 1
MENU
1. Ulang
2. Keluar
Pilihan : 2
Anda pilih keluar

Bagian pernyataan yang mengikuti **while** akan dieksekusi selama *ungkapan* pada **while** bernilai salah (tidak sama dengan nol). Pengujian terhadap ungkapan **while** dilakukan sebelum bagian pernyataan. Perhatikan contoh-contoh berikut :

Contoh 25	Hasil
/*-----*	C++
/* Contoh 25 : pemakaian do-while untuk *	C++
/* menampilkan tulisan C++ *	C++
/* sebanyak 10 kali *	C++
/*-----*	C++
#include <iostream.h>	C++
#include <conio.h>	C++
void main()	C++
{	C++
int i; // Sebagai variabel pencacah yang menyatakan	C++
// jumlah tulisan C++ yang harus ditampilkan	C++
clrscr(); // Hapus layar	C++
i = 0; // Mula-mula diisi sama dengan nol	C++
do	C++
{	C++
cout << " C++ " << endl;	C++
i ++ ; // Menaikkan pencacah sebesar 1	C++
} while (i < 10);	C++
}	C++

Pada program diatas, variabel i bertindak sebagai pencacah yang gunanya untuk mengingat jumlah tulisa C++ yang telah ditampilkan. Itulah sebabnya mula-mula didisi dengan nol. Kemudian untuk setiap putaran, isi variabel ini dinaikkan. Oleh karena variabel i dijadikan sebagai kondisi pada **while**, suatu ketika

ketika kondisi $i < 10$ akan bernilai benar, maka **while** berakhir. Dengan kata lain do..while kebalikan dari while. Dimana do..while akan mengulang saat kondisi bernilai salah, dan akan berhenti jika kondisi bernilai benar.

8.1.3 Sintaks FOR

Sintaks pengulangan FOR merupakan sintaks yang relatif paling mudah digunakan. Sintaks ini serupa dengan sintaks WHILE... DO... dalam hal pengecekan kondisi dilakukan di awal. Dalam menggunakan struktur pengulangan dengan sintaks FOR, pemrogram harus mendefinisikan nilai awal dan nilai akhir pencacah yang menunjukkan jumlah iterasi. Setiap kali iterasi berlangsung, nilai pencacah akan diubah. Jika pencacah sudah mencapai nilai akhir yang ditentukan, maka pengulangan akan berhenti.

Bila contoh ‘Ibu mengupas kentang’ ingin diubah ke dalam struktur pengulangan dengan sintaks FOR, pemrogram harus menentukan nilai awal dan akhir pencacah, yaitu variabel kentang. Karena ibu akan mengupas kentang pertama hingga kentang ke sepuluh, maka:

- ↳ Nilai awal pencacah: kentang = 1
- ↳ Nilai akhir pencacah: kentang = 10
- ↳ Selama kondisi $kentang \geq 1$ dan $kentang \leq 10$ terpenuhi, aksi pengulangan akan dilakukan.

Pernyataan **for** berguna untuk mengulang pengeksekusian terhadap satu atau sejumlah pernyataan. Bentuk format :

```
for (ungkapan1; ungkapan2; ungkapan3)  
pernyataan;
```

Dimana :

- ↳ Ungkapan1 : untuk memberikan nilai awal untuk variabel pencacah.
- ↳ Ungkapan2 : kondisi pengulangan akan berhenti atau tidak.

- ↳ Ungkapan3 : pengubahan nilai variabel pencacah untuk mencapai kondisi berhenti, dapat berupa kenaikan ataupun penurunan. Pengubah variabel pencacah tidak harus selalu naik atau turun satu, tetapi dapat dilakukan pengubahan variabel pencacah lebih dari satu.
- ↳ Pernyataan perintah : aksi yang akan diulang

Maka, pada contoh-contoh sebelumnya dapat diubah dalam struktur FOR menjadi :

- ↳ Algoritma ibu mengupas kentang

ALGORITMA Kupas_Kentang	
IS : Terdapat 10 kentang belum dikupas	
FS : 10 kentang telah terkupas	
KAMUS DATA	
1	kentang : integer
2	//inisialisasi,kondisi, dan pengubah pencacah
3	//dideklarasikan dalam sintaks for
4	FOR(kentang ← 0; kentang < 10; kentang++)
5	{aksi pengulangan}
6	Ambil sebuah kentang
	Kupas kulit kentang

Perhatikan bahwa potongan algoritma di atas tidak mencantumkan aksi pengubah pencacah \leftarrow kentang + 1. Inisialisasi, pengecekan kondisi, dan pengubah variabel pencacah sudah terdapat dalam argumen FOR. Pada posisi pengubah variabel, pernyataan kentang++ sama dengan kentang \leftarrow kentang + 1, penambahan akan dilakukan setelah aksi pengulangan dilaksanakan.

Jalannya iterasi ini dapat ditulis dalam bentuk tabel berikut:

Iterasi ke-	kentang	kentang < 10?	{aksi}
1	0	Ya	Ya
2	1	Ya	Ya
3	2	Ya	Ya
4	3	Ya	Ya

5	4	Ya	Ya
6	5	Ya	Ya
7	6	Ya	Ya
8	7	Ya	Ya
9	8	Ya	Ya
10	9	Ya	Ya
11	10	Tidak	-

Pengulangan dihentikan pada iterasi ke- 11 karena kondisi $k < 10$ bernilai salah.

↳ Algoritma untuk menampilkan karakter ‘ * ’ sebanyak 5 kali.

ALGORITMA Tampil_Bintang
IS : Jumlah bintang yg tampil = 0
FS : Jumlah bintang yg tampil = 5
KAMUS DATA
i : integer
1 FOR($i \leftarrow 0$; $i < 5$; $i++$)
2 Output (“*”) {aksi}

Output dari algoritma ini:

*
*
*
*
*

Pengulangan dihentikan pada iterasi ke- 6 karena kondisi $i < 5$ bernilai salah. Perhatikan bahwa pencacah dapat dimulai dari angka berapapun hingga berapapun sesuai kebutuhan program.

↳ Algoritma untuk menampilkan iterasi ke 1 sampai 5 dengan pengulangan.

ALGORITMA Iterasi_Angka
IS : -
FS : Jumlah bintang yg tampil = 5
KAMUS DATA
i : integer

```
1 FOR(i ← 1; i <= 5;i++)
2   Output ('Ini adalah iterasi ke-',i){aksi}
```

Output dari algoritma ini:

```
Ini adalah iterasi ke-1
Ini adalah iterasi ke-2
Ini adalah iterasi ke-3
Ini adalah iterasi ke-4
Ini adalah iterasi ke-5
```

Pengulangan dihentikan pada iterasi ke- 6 karena kondisi $i \leq 5$ bernilai salah.

↳ Algoritma untuk memasukkan nilai tiga (3) orang mahasiswa

ALGORITMA Input_Nilai

IS : -

FS : Menerima inputan nilai dari user

KAMUS DATA

i : integer

nilai : integer

```
1 FOR(i ← 1; i <= 3;i++)
2   Output ('Nilai mahasiswa ke-','i,'adalah:')
3   Input (nilai)
```

Output dari algoritma ini:

Nilai mahasiswa ke-1 adalah: _

Nilai mahasiswa ke-2 adalah: _

Nilai mahasiswa ke-3 adalah: _

Pengulangan dihentikan pada iterasi ke- 4 karena kondisi $i \leq 3$ bernilai salah.

Serupa dengan struktur pengulangan dengan sintaks lain, struktur pengulangan dengan sintaks FOR juga dapat dilakukan dengan pencacah mundur. Berikut ini adalah beberapa contoh penggunaan struktur pengulangan FOR dengan pencacah mundur :

- ↳ Program untuk menghitung rerata dari tiga bilangan yang diinputkan.

ALGORITMA Nilai Rata Rata

IS :-

FS : Tampil rata-rata dari nilai yang diinputkan

KAMUS DATA

jumlah : float

bilangan : integer

x : float

rerata : float

```

1 jumlah ← 0      {variabel jumlah bilangan}
2           {inisialisasi variabel pencacah}
3 FOR(bilangan ← 3; bilangan > 0; bilangan--)
4   Output('Masukkan angka : ')
5   Input (x)
6   jumlah ← jumlah + x
7 rerata ← jumlah/ 3 {menghitung rerata}
8 Output ('Rerata : ',rerata)

```

Pada algoritma, pengguna terus diminta memasukkan sebuah bilangan yang kemudian dijumlahkan dengan bilangan-bilangan sebelumnya (pada iterasi pertama, bilangan dijumlahkan dengan nol) hingga pencacah bernilai 0. Pengulangan dihentikan pada iterasi ke-4 karena kondisi $\text{bilangan} > 0$ bernilai salah. Program akan memberikan output berupa hasil perhitungan rerata dari ketiga bilangan yang diinputkan.

- ↳ Algoritma petasan: program akan menghitung mundur dengan menampilkan sejumlah bilangan tertentu, kemudian mengeluarkan pesan 'DOR!!!' di akhir perhitungan mundur.

ALGORITMA Hitung Mundur

IS:

FS : Menampilkan angka dari 5 hingga 1

KAMUS DATA

hitung : integer

```
1 FOR(hitung ← 5; hitung > 0; bilangan--)
```

```
2     Output(hitung)
3     Output ('DOR!!!')
```

Output dari algoritma ini:

```
5
4
3
2
1
DOR!!!
```

- ↳ Algoritma ibu mengupas kentang dengan perhitungan mundur

ALGORITMA Kupas Kentang

IS : Terdapat 10 kentang belum dikupas

FS : 10 kentang telah terkupas

KAMUS DATA

kentang : integer

```
1 //inisialisasi,kondisi, dan pengubah pencacah
2 //dideklarasikan dalam sintaks for
3 FOR(kentang < 10; kentang > 0; kentang--)
4     Ambil sebuah kentang
5     Kupas kulit kentang
```

Pada algoritma di atas, iterasi dilakukan 10 kali dan akan dihentikan pada iterasi ke 11 karena kondisi $kentang > 0$ bernilai salah.

Penulisan sintaks FOR dalam bahasa pemrograman C:

For dengan satu aksi

```
int i;
for(i=0;i<5;i++)
    printf("%d",i);
```

For dengan banyak aksi

```
int i;
for(i=0;i<5;i++)
{
```

```

    printf("Masukkan angka ke-%d",i);
    scanf("%d",&nilai);
}

```

Dalam perulangan mekanismenya saja yang berbeda. Pelaksanaan atau output yang dihasilkan tetap sama. Dalam arti apabila kita membuat suatu program perulangan dengan menggunakan while, maka dapat dikonversi ke bentuk do while dan for. Perhatikan contoh for berikut :

Contoh 26	Hasil
<pre> /* ... */ /* Contoh 26 : Menampilkan bilangan genap * /* yang nilainya kurang atau sama * /* dengan n dan ditampilkan dari * /* terbesar sampai nol * /* ... #include <iostream.h> #include <conio.h> void main() { int n; clrscr(); cout << "Menampilkan bilangan genap yang nilainya " << endl; cout << "kurang atau sama dengan n " << endl; cout << "Masukkan nilai n = "; cin >> n; // Jika n ganjil, maka dikurangi 1 if (n % 2) n--; // tampilkan deret bilangan genap dari besar ke kecil for (; n >= 0; n -= 2) cout << n << ' '; } </pre>	Menampilkan bilangan genap yang nilainya kurang atau sama dengan n Masukkan nilai n = 11 ↴ 10 8 6 4 2 0

Pada program diatas terdapat :

n --; ungkapan kosong
for (; n >= 0; n -= 2)
sama artinya dengan :
for (n -- ; n >= 0 ; n -= 2)

Contoh 27	Hasil
<pre>/* * Contoh 27 : Memebentuk segitiga yang berisi * * karakter '*' dengan menggunakan * * for didalam for * */ #include <iostream.h> #include <conio.h> void main() { int tinggi, // Menyatakan tinggi segi tiga baris, // Pencacah untuk baris kolom; // Pencacah untuk kolom clrscr(); cout << " Tinggi segitiga = " ; cin >> tinggi; cout << endl; //Membuat baris kosong for (baris = 1; kolom <= baris; kolom ++) { for (klom = 1; kolom <= baris ; klom ++) cout << '*'; cout << endl ; // Pindah baris } }</pre>	<p>Tinggi segitiga = 5 ↴</p> <p>*</p> <p>**</p> <p>***</p> <p>****</p> <p>*****</p>



Rangkuman

- Struktur pengulangan memungkinkan program melakukan satu atau lebih aksi beberapa kali.
- Tiga komponen pengendali aksi adalah inisialisasi, jumlah iterasi, dan kondisi berhenti.
- Tiga struktur pengulangan yang dapat digunakan adalah struktur pengulangan dengan sintaks WHILE, DO...WHILE, dan FOR.
- Struktur pengulangan dapat dibuat bersarang dengan sintaks pengulangan yang sama atau berbeda, bahkan dapat digabungkan dengan struktur pemilihan.
- Untuk keluar dari struktur pengulangan sebelum kondisi berhenti, kita dapat menggunakan sintaks BREAK
- Hal yang terpenting dari struktur pengulangan adalah kondisi berhenti yang akan memberikan kondisi apakah pengulangan dilakukan atau tidak.