

VERSI 1.0  
AGUSTUS, 2024



# [PEMROGRAMAN DASAR]

## MODUL 4 – LOOP

DISUSUN OLEH:  
- WEMPY ADITYA WIRYAWAN  
- MUHAMMAD ZAKY DARAJAT

DIAUDIT OLEH:  
- HARDIANTO WIBOWO, S.KOM, M.T

LAB. INFORMATIKA  
UNIVERSITAS MUHAMMADIYAH MALANG

## [PEMROGRAMAN DASAR]

---

### PERSIAPAN MATERI

Mahasiswa diharapkan mempelajari materi sebelum mengerjakan tugas, materi yang tercakup antara lain:

- Pengenalan Loop
- While Loop
- For Loop
- Nested Loop
- Break/Continue

---

### TUJUAN

- Mahasiswa dapat memahami konsep perulangan (loop) dalam pemrograman dan kegunaannya untuk menjalankan blok code berulang kali.
- Mahasiswa dapat memahami sintaks dan cara kerja while loop untuk menjalankan code berulang kali selama kondisi tertentu terpenuhi.
- Mahasiswa dapat memahami struktur dan penggunaan for loop sebagai salah satu bentuk perulangan dalam bahasa C.
- Mahasiswa dapat memahami konsep nested loop (perulangan bersarang) dan cara menggunakannya untuk mengatasi masalah yang kompleks.
- Mahasiswa dapat memahami pernyataan break dan continue untuk mengendalikan aliran perulangan dan kondisi tertentu dalam loop.

---

### TARGET MODUL

- Mahasiswa dapat menjelaskan apa itu loop dan mengidentifikasi situasi yang memerlukan penggunaan perulangan dalam program.
- Mahasiswa dapat membuat loop menggunakan while dan menghindari infinite loop dengan tepat.
- Mahasiswa dapat menggunakan for loop untuk mengulang code dengan jumlah iterasi yang diketahui sebelumnya.

- Mahasiswa dapat membuat dan mengelola nested loop dengan benar untuk memproses data secara berlapis-lapis.
- Mahasiswa dapat menggunakan break untuk menghentikan perulangan secara paksa dan continue untuk melompati iterasi tertentu dalam loop dengan tepat.

---

## PERSIAPAN SOFTWARE/APLIKASI

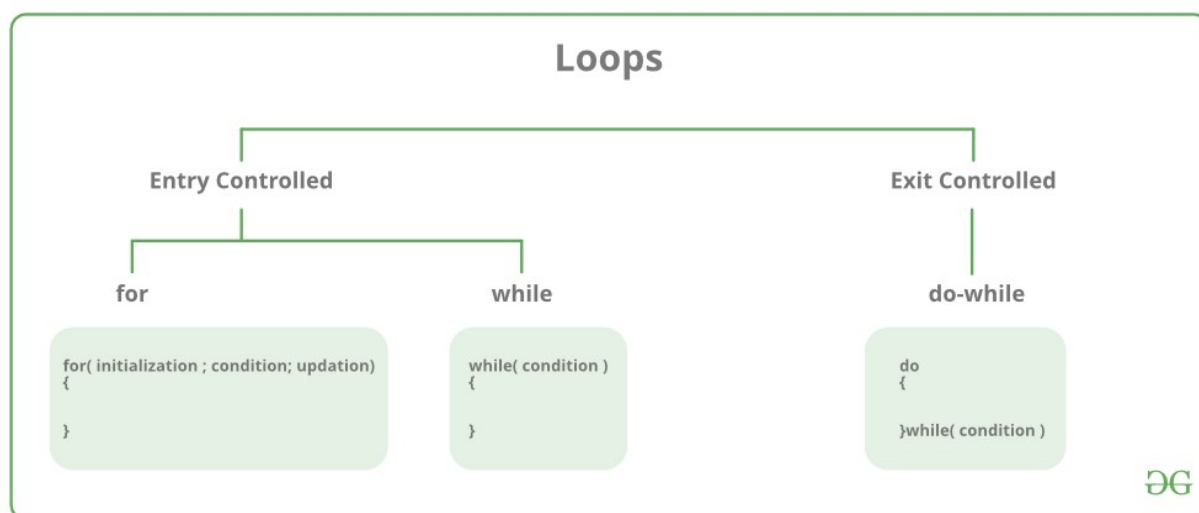
- Komputer/Laptop
- Software Aplikasi (Falcon/Dev C++)

---

## MATERI PRAKTIKUM

### LOOP - INTRODUCTION

Loop atau perulangan adalah struktur kontrol dalam pemrograman yang memungkinkan untuk melakukan serangkaian tindakan berulang kali berdasarkan kondisi tertentu. Loop memungkinkan kita untuk mengeksekusi blok kode secara berulang tanpa harus menulis kode yang sama berulang kali. Hal ini sangat berguna ketika kita ingin mengulang suatu tindakan beberapa kali atau hingga suatu kondisi terpenuhi. Terdapat dua jenis loop yang umum digunakan dalam bahasa pemrograman, yaitu Loop dengan kondisi (Conditional Loop) dan Loop dengan perulangan tetap (Counted Loop)



Loop dengan Kondisi (Conditional Loop), juga dikenal sebagai loop while atau loop do-while, akan mengulang blok kode selama suatu kondisi tertentu terpenuhi. Pada awal setiap iterasi, kondisi akan dievaluasi, dan jika kondisi tersebut benar (true), blok kode di dalam loop akan dijalankan. Loop akan

terus berlanjut selama kondisi tersebut masih benar. Jika kondisi salah (false), maka loop akan berhenti dan program akan melanjutkan eksekusi pada bagian setelah loop.

Loop dengan Perulangan Tetap (Counted Loop), juga dikenal sebagai loop for, memungkinkan kita untuk mengulang blok kode sejumlah tertentu kali. Pada loop for, kita mendeklarasikan variabel kontrol, mendefinisikan kondisi iterasi, dan menentukan langkah iterasi.

Loop sangat berguna dalam situasi-situasi di mana kita ingin melakukan tugas yang sama berulang kali atau ketika ingin melakukan perulangan dengan jumlah iterasi yang telah ditentukan. Dengan loop, kita dapat menulis kode yang lebih efisien dan mudah dibaca daripada menulis instruksi yang sama berulang kali secara manual.

## LOOP – WHILE LOOP

"LOOP" (perulangan) adalah konsep penting dalam pemrograman yang memungkinkan kita untuk menjalankan sekelompok pernyataan berulang kali selama kondisi tertentu terpenuhi. "WHILE LOOP" merupakan salah satu jenis perulangan dalam bahasa C yang memungkinkan kita untuk melakukan tugas-tugas berulang berdasarkan kondisi yang diberikan.

Pada dasarnya, "WHILE LOOP" memungkinkan program untuk menjalankan blok kode tertentu selama kondisi yang diberikan tetap benar (true). Setiap kali blok kode dijalankan, kondisi akan dievaluasi kembali. Jika kondisi masih benar, maka blok kode akan dijalankan lagi, dan begitu seterusnya. Namun, jika kondisi menjadi salah (false), "WHILE LOOP" akan berhenti dan eksekusi program akan melanjutkan ke pernyataan setelah while loop.

Penggunaan "WHILE LOOP" bergantung pada kondisi yang akan dievaluasi. Jika kondisi awalnya salah, maka blok kode dalam while loop mungkin tidak akan dieksekusi sama sekali. Oleh karena itu, penting untuk memastikan bahwa kondisi dapat menjadi benar pada suatu titik dalam iterasi sehingga loop tidak menjadi tak berujung.

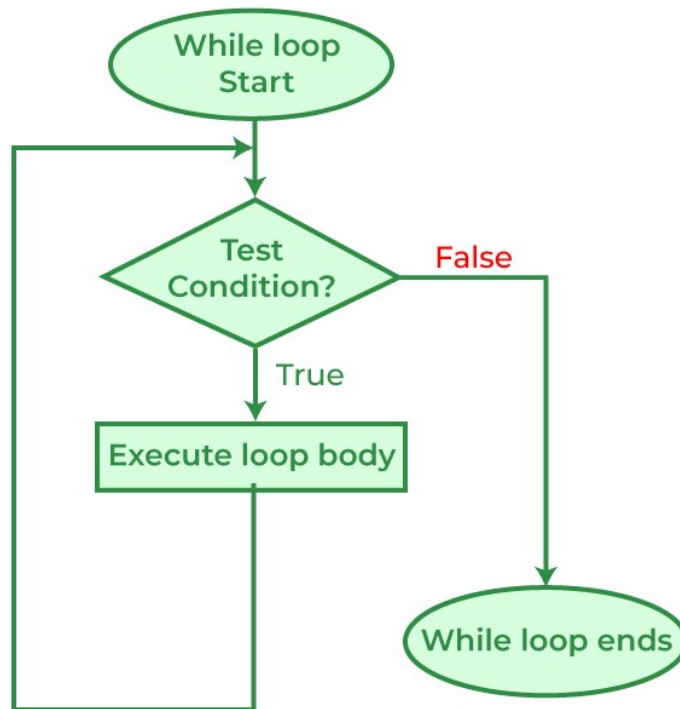
While loop adalah salah satu jenis loop (perulangan) dalam bahasa C yang akan terus mengulang blok kode selama kondisi tertentu terpenuhi. Loop ini akan mengevaluasi kondisi di awal setiap iterasi dan hanya akan menjalankan blok kode selama kondisi tersebut benar (true). Jika kondisi menjadi salah (false), maka while loop akan berhenti dan program akan melanjutkan eksekusi ke pernyataan setelah while loop. Struktur dari while loop adalah sebagai berikut:

```
while (kondisi) {  
    // Blok kode yang akan diulang selama kondisi benar  
    // Perintah atau tindakan yang ingin diulang  
}
```

**while (condition) {**, ini adalah awal dari konstruksi while loop di banyak bahasa pemrograman. Di sini, condition adalah ekspresi yang harus dievaluasi menjadi nilai Boolean, yaitu true

atau false. Jika kondisi ini bernilai true, maka blok code di dalamnya akan dieksekusi. Jika kondisi bernilai false, maka program akan keluar dari while loop dan melanjutkan ke pernyataan berikutnya setelahnya.

Flowchart cara kerja while loop:



Jadi, while loop tersebut akan terus mengeksekusi blok code di dalamnya selama kondisi yang diberikan bernilai true. Jika kondisi tersebut akhirnya bernilai false, maka eksekusi akan keluar dari loop dan melanjutkan ke pernyataan berikutnya setelahnya. Penting untuk memastikan bahwa kondisi akan berubah dari true ke false pada suatu saat, agar program tidak terjebak dalam perulangan yang tak terbatas (infinite loop).

Pada contoh di bawah ini, code dalam perulangan akan berjalan berulang kali, selama variabel (**i**) kurang dari 5:

```
int i = 0;

while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

Pada program di atas, `int i = 0;` adalah deklarasi variabel `i` dengan tipe data `int` (integer) dan nilai awal `0`. Variabel `i` digunakan sebagai penghitung atau indeks dalam loop.

`While (i < 5) {`, adalah awal dari while loop. Selama nilai dari variabel `i` kurang dari 5, blok code di dalamnya akan terus dieksekusi. Jika nilai `i` sudah mencapai 5 atau lebih, maka eksekusi akan keluar dari loop.

`Printf("%d\n", i);`, adalah pernyataan cetak yang menggunakan fungsi `printf`. `%d` merupakan format specifier untuk mencetak nilai integer. Pernyataan ini akan mencetak nilai variabel `i` ke layar, diikuti dengan karakter newline (`\n`) agar setiap nilai `i` dicetak dalam baris yang berbeda.

`i++;`, adalah pernyataan untuk menambahkan nilai 1 ke variabel `i`. Ekspresi ini bersifat singkat dan setara dengan `i = i + 1` atau `i += 1`. Hal ini diperlukan agar nilai `i` berubah pada setiap iterasi loop, sehingga loop dapat berjalan hingga kondisi pada baris kedua tidak terpenuhi.

Jadi, while loop di atas akan mencetak nilai dari `i` mulai dari 0 hingga 4, karena ketika `i` mencapai nilai 5, kondisi pada baris kedua tidak lagi terpenuhi dan loop akan berakhir. Hasilnya akan mencetak angka 0, 1, 2, 3, dan 4 masing-masing dalam baris yang berbeda.

Output program:

```
0
1
2
3
4

=== Code Execution Successful ===
```

```
#include <stdio.h>

int main() {
    int number = 0;
    int sum = 0;

    printf("Enter numbers to add to the sum (enter 0 to stop):\n");

    // Meminta pengguna untuk memasukkan angka
    scanf("%d", &number);

    // Loop berlanjut sampai angka 0 dimasukkan
    while (number != 0) {
        sum += number; // Menambahkan angka ke jumlah total
        printf("Current sum: %d\n", sum);

        // Meminta pengguna untuk memasukkan angka berikutnya
        scanf("%d", &number);
    }

    printf("Final sum: %d\n", sum);
    return 0;
}
```

Output:

```
Enter numbers to add to the sum (enter 0 to stop):
5
Current sum: 5
10
Current sum: 15
-3
Current sum: 12
0
Final sum: 12
```

Program ini menggunakan while loop untuk menambahkan angka-angka yang dimasukkan oleh pengguna sampai pengguna memasukkan angka nol (0). Program dimulai dengan menginisialisasi variabel number dan sum masing-masing dengan nilai 0. Pengguna kemudian diminta untuk memasukkan angka pertama. Di dalam while loop, program mengecek apakah number tidak sama dengan 0. Jika kondisi ini benar, nilai number ditambahkan ke sum, dan jumlah saat ini dicetak. Setelah itu, program meminta pengguna untuk memasukkan angka berikutnya. Proses ini diulangi hingga pengguna memasukkan 0, yang menyebabkan loop berhenti dan mencetak jumlah total akhir.



## LOOP – DO/WHILE

"LOOP" (perulangan) merupakan mekanisme yang penting dalam pemrograman yang memungkinkan program untuk menjalankan serangkaian pernyataan secara berulang sesuai dengan kondisi yang ditentukan. Salah satu jenis perulangan yang sering digunakan dalam bahasa pemrograman C adalah "DO/WHILE LOOP".

"DO/WHILE LOOP" adalah varian dari "WHILE LOOP" yang memiliki karakteristik unik. Pada dasarnya, loop ini akan menjalankan blok kode setidaknya satu kali sebelum memeriksa kondisi yang ditentukan. Ini berarti bahwa bahkan jika kondisi awalnya salah (false), setidaknya satu iterasi akan dilakukan.

Langkah-langkah dalam "DO/WHILE LOOP" adalah sebagai berikut:

- Blok kode di dalam "DO" akan dieksekusi pertama kali tanpa memeriksa kondisi.
- Setelah blok kode dieksekusi, kondisi di dalam "WHILE" akan dievaluasi.
- Jika kondisi masih benar (true), blok kode dalam "DO" akan dijalankan kembali, dan langkah 2 akan diulangi.
- Jika kondisi menjadi salah (false), loop akan berhenti dan program akan melanjutkan ke pernyataan setelah "DO/WHILE LOOP".

Penggunaan "DO/WHILE LOOP" dapat berguna dalam situasi di mana kita ingin memastikan bahwa blok kode dijalankan setidaknya sekali sebelum melakukan pengecekan kondisi. Ini sangat berguna saat kita ingin memproses input dari pengguna atau menjalankan tugas-tugas tertentu sebelum mengevaluasi kondisi.

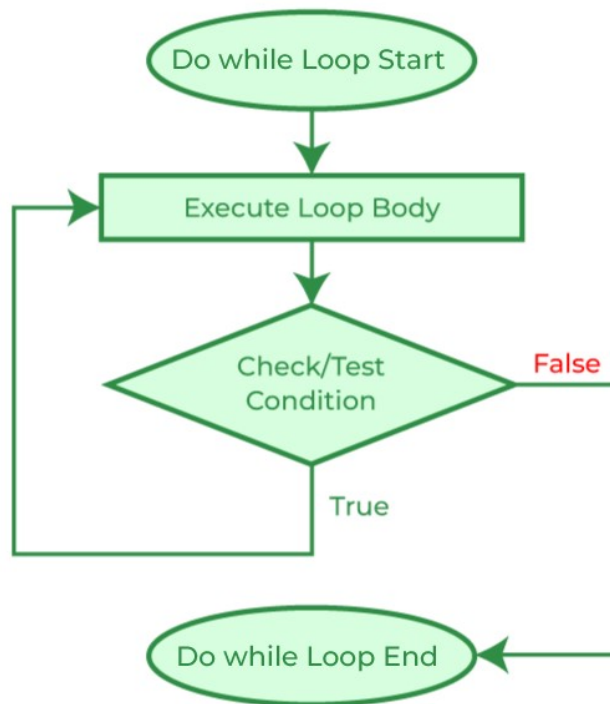
Loop do/while adalah varian dari while loop. Loop ini akan mengeksekusi blok code satu kali, sebelum memeriksa apakah kondisinya benar, maka loop akan berulang selama kondisinya benar. Struktur dari do/while loop adalah sebagai berikut:

```
do {  
    // blok kode yang akan dieksekusi  
}  
while (condition);
```

**do {**, adalah awal dari do-while loop. Blok code di dalamnya akan selalu dieksekusi terlebih dahulu sebelum kondisi dievaluasi.

**} while (condition);**, adalah akhir dari do-while loop. Setelah blok kode di dalamnya dieksekusi, kondisi pada bagian **while** akan dievaluasi. Jika kondisi tersebut bernilai true, maka eksekusi akan kembali ke awal do-while loop dan blok code di dalamnya akan dieksekusi kembali. Jika kondisi bernilai false, maka eksekusi keluar dari loop dan melanjutkan ke pernyataan berikutnya setelahnya.

Flowchart cara kerja do while loop:



Hal penting yang membedakan do-while loop dari while loop adalah bahwa do-while loop akan mengeksekusi blok code di dalamnya minimal satu kali, bahkan jika kondisi awalnya bernilai false. Ini karena evaluasi kondisi dilakukan setelah eksekusi blok code pertama, sehingga minimal satu iterasi pasti terjadi.

Jadi, do-while loop akan terus mengeksekusi blok code di dalamnya selama kondisi yang diberikan masih bernilai true. Setelah kondisi menjadi false, eksekusi keluar dari loop dan melanjutkan ke pernyataan berikutnya setelahnya.

Contoh di bawah ini menggunakan do/while loop. Loop akan selalu dieksekusi setidaknya sekali, bahkan jika kondisi salah, karena blok code dieksekusi sebelum kondisi diuji:

```

int i = 0;

do {
    printf("%d\n", i);
    i++;
}
while (i < 5);

```

**int i = 0;**, adalah deklarasi variabel **i** dengan tipe data **int** (integer) dan nilai awal 0. Variabel **i** digunakan sebagai penghitung atau indeks dalam do-while loop.

**do {**, adalah awal dari do-while loop. Blok code di dalamnya akan dieksekusi setidaknya satu kali, karena evaluasi kondisi dilakukan setelah eksekusi blok code pertama.

**printf("%d\n", i);**, adalah pernyataan cetak yang menggunakan fungsi **printf**. **%d** merupakan format specifier untuk mencetak nilai integer. Pernyataan ini akan mencetak nilai variabel **i** ke layar, diikuti dengan karakter newline (**\n**) agar setiap nilai **i** dicetak dalam baris yang berbeda.

**i++;**, adalah pernyataan untuk menambahkan nilai 1 ke variabel **i**. Ekspresi ini akan dieksekusi setiap kali do-while loop berjalan untuk meningkatkan nilai **i** pada setiap iterasi.

**}**, adalah akhir dari do-while loop. Setelah blok code di dalamnya dieksekusi, program akan kembali ke kata kunci **while** untuk mengevaluasi kondisi pada baris berikutnya.

**while (i < 5);**, adalah kondisi do-while loop. Selama nilai dari variabel **i** kurang dari 5, eksekusi akan kembali ke awal do-while loop dan blok code di dalamnya akan dieksekusi lagi. Jika nilai **i** sudah mencapai 5 atau lebih, maka eksekusi keluar dari loop dan melanjutkan ke pernyataan berikutnya setelahnya.

Jadi, do-while loop di atas akan mencetak nilai dari **i** mulai dari 0 hingga 4. Setiap nilai **i** akan dicetak dalam baris yang berbeda karena pernyataan **printf** pada baris ke-3 mencetak nilai **i** diikuti dengan karakter newline (**\n**).

Output program:

```

0
1
2
3
4

=== Code Execution Successful ===

```

```
#include <stdio.h>
#include <string.h>
int main() {
    char password[] = "secret";
    char input[20];
    do {
        printf("Enter the password: ");
        scanf("%s", input);
    } while (strcmp(input, password) != 0);
    printf("Access granted!\n");
    return 0;
}
```

Output:

```
Enter the password: 123
Enter the password: abc
Enter the password: secret
Access Granted!
```

Program di atas meminta pengguna untuk memasukkan kata sandi yang benar sebelum memberikan akses. Pertama, variabel `password` diinisialisasi dengan string "secret", dan sebuah array `input` disiapkan untuk menyimpan masukan pengguna. Program kemudian masuk ke dalam `do-while` loop, yang memastikan bahwa setidaknya satu iterasi dilakukan. Di dalam loop, program meminta pengguna untuk memasukkan kata sandi menggunakan `scanf`, yang membaca string yang dimasukkan dan menyimpannya ke dalam array `input`. Fungsi `strcmp` kemudian membandingkan string `input` dengan `password`. Jika `input` tidak sama dengan "secret", loop akan berlanjut, meminta pengguna untuk memasukkan kembali kata sandi. Loop ini terus berulang sampai pengguna memasukkan "secret" sebagai kata sandi yang benar. Setelah pengguna memasukkan kata sandi yang benar, perbandingan `strcmp(input, password) != 0` menjadi salah, menghentikan loop dan mencetak "Access granted!", menandakan bahwa akses telah diberikan. Program ini berguna dalam situasi di mana hanya pengguna yang mengetahui kata sandi yang benar yang dapat melanjutkan.

**Perbedaan do-while dan while**

Aspek	while loop	do-while loop
Sintaks	<code>while (kondisi) { ... }</code>	<code>do { ... } while (kondisi);</code>
Eksekusi Badan Loop	Kondisi diperiksa sebelum eksekusi.	Badan dieksekusi sebelum kondisi.
Eksekusi Pertama	Kondisi harus benar sejak awal.	Badan dieksekusi setidaknya sekali.
Eksekusi Loop	Mungkin dieksekusi nol kali atau lebih.	Akan dieksekusi setidaknya sekali.
Contoh	<code>while (i &lt; 5) { printf("%d\n", i); i++; }</code>	<code>do { printf("%d\n", i); i++; } while (i &lt; 5);</code>
Kasus Penggunaan Umum	Ketika loop mungkin tidak dijalankan sama sekali.	Ketika Anda ingin loop dijalankan setidaknya sekali.

## LOOP – FOR LOOP

"LOOP" (perulangan) adalah fitur penting dalam pemrograman yang memungkinkan kita untuk menjalankan serangkaian pernyataan berulang kali sesuai dengan kondisi tertentu. "FOR LOOP" adalah salah satu bentuk perulangan yang paling umum digunakan dalam bahasa pemrograman C. Loop ini memberikan fleksibilitas dalam mengulangi blok kode berdasarkan kondisi yang ditentukan.

"FOR LOOP" memiliki struktur yang terdiri dari tiga bagian:

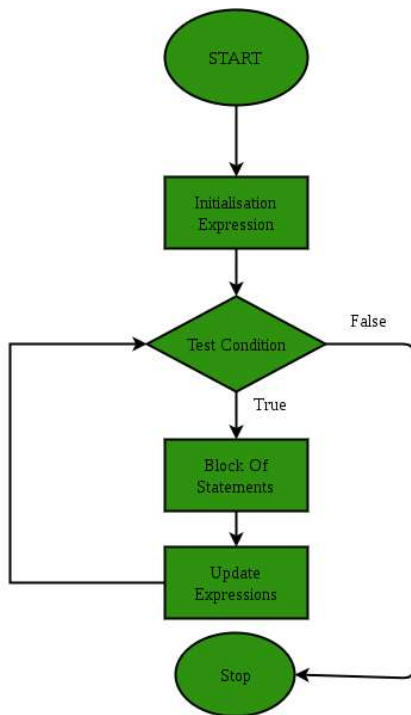
1. Inisialisasi: Bagian pertama biasanya digunakan untuk menginisialisasi variabel penghitung yang akan digunakan selama iterasi.
2. Kondisi: Bagian kedua adalah kondisi yang akan dievaluasi sebelum setiap iterasi. Jika kondisi masih benar (true), blok kode dalam loop akan dijalankan. Jika kondisi menjadi salah (false), loop akan berhenti.
3. Pembaruan: Bagian ketiga digunakan untuk memperbarui variabel penghitung setelah setiap iterasi.

Syntax dasar for loop:

```
for (initialize expression; test expression; update expression)
{
    //
    // body of for loop
    //
}
```

Dalam setiap iterasi, blok kode dalam loop akan dieksekusi. Setelah blok kode selesai dieksekusi, variabel penghitung akan diperbarui sesuai dengan langkah pembaruan, dan kondisi akan dievaluasi kembali. Loop akan terus berlanjut selama kondisi masih benar.

Flowchart cara kerja for loop:



"FOR LOOP" sangat berguna ketika kita tahu berapa kali perulangan harus dijalankan atau ketika kita ingin melakukan perulangan dengan menggunakan variabel penghitung. Ini membantu menghindari perulangan tak terbatas dan memberikan cara yang terstruktur untuk mengulang sejumlah kali tertentu.

For loop adalah salah satu bentuk perulangan dalam bahasa pemrograman C yang memungkinkan kita untuk mengulangi blok code secara berulang berdasarkan suatu kondisi yang ditentukan. For loop sangat berguna ketika kita tahu berapa kali perulangan harus berlangsung, atau ketika kita ingin melakukan perulangan dengan menggunakan sebuah variabel penghitung. Contoh sederhana for loop:

```

#include <stdio.h>

int main() {
    // Contoh for loop dengan variabel penghitung
    for (int i = 0; i < 5; i++) {
        printf("Iterasi ke-%d\n", i);
    }

    return 0;
}
  
```

**for (int i = 0; i < 5; i++) {**, adalah for loop yang akan berjalan selama **i** kurang dari

1. Di sini, **int i = 0** adalah inisialisasi variabel **i** dengan nilai awal 0 sebelum memulai perulangan. **i < 5** adalah kondisi perulangan, dan **i++** adalah ekspresi increment untuk menambahkan nilai **i** setiap kali perulangan selesai.

**printf("Iterasi ke-%d\n", i);**, baris code ini akan mencetak pesan "Iterasi ke-X" di mana X adalah nilai variabel **i**. Pesan ini akan dicetak dalam setiap iterasi for loop.

**}**, ini menandakan akhir dari for loop. Setelah blok kode di dalam for loop dieksekusi, program akan kembali ke awal for loop untuk mengevaluasi kondisi pada baris ke-3. Jika kondisi masih terpenuhi, for loop akan dijalankan kembali.

**return 0;**, adalah pernyataan return yang mengindikasikan akhir dari fungsi main dan mengembalikan nilai 0 ke sistem operasi. Nilai 0 menandakan bahwa program berakhir dengan sukses.

Hasil dari code di atas adalah mencetak pesan "Iterasi ke-0", "Iterasi ke-1", "Iterasi ke-2", "Iterasi ke-3", dan "Iterasi ke-4" masing-masing dalam baris yang berbeda. Ini karena for loop akan berjalan dari **i = 0** hingga **i < 5** dan mencetak nilai **i** dalam setiap iterasi.

Output program:

```
Iterasi ke-0
Iterasi ke-1
Iterasi ke-2
Iterasi ke-3
Iterasi ke-4

=== Code Execution Successful ===
```



```

#include<stdio.h>
int main(){
int i=1,number=0;
printf("Enter a number: ");
scanf("%d",&number);
for(i=1;i<=10;i++){
printf("%d \n",(number*i));
}
return 0;
}

```

Output program:

```

Enter a number: 2
2
4
6
8
10
12
14
16
18
20

```

Program di atas adalah sebuah aplikasi sederhana yang menampilkan tabel perkalian dari suatu bilangan yang dimasukkan oleh pengguna. Ketika program dijalankan, ia pertama-tama menginisialisasi variabel `i` dengan nilai 1 dan `number` dengan nilai 0. Program kemudian meminta pengguna untuk memasukkan sebuah bilangan melalui `printf` dan `scanf`. Misalkan pengguna memasukkan angka 2, program kemudian memasuki `for loop`, yang akan berjalan dari `i = 1` hingga `i = 10`. Pada setiap iterasi, program mencetak hasil perkalian antara `number` (2) dan `i`, dimulai dari 1 hingga 10. Jadi, outputnya adalah deretan hasil perkalian: 2, 4, 6, 8, 10, 12, 14, 16, 18, dan 20, masing-masing pada baris baru. Program ini efektif untuk menampilkan tabel perkalian dari angka yang diberikan, dalam hal ini angka 2.

### Keuntungan Menggunakan for Loop

Ada beberapa keuntungan menggunakan for loop dalam bahasa pemrograman C:

1. Penggunaan Ulang Kode: Dengan menggunakan for loop, kita dapat menulis kode yang dapat digunakan kembali untuk berbagai kasus yang mirip. Misalnya, jika kita perlu melakukan sesuatu untuk setiap elemen dalam array, kita dapat menulis satu for loop yang akan bekerja untuk array apa pun.

2. Ukuran Kode Berkurang: Menggunakan for loop dapat mengurangi jumlah baris kode yang perlu kita tulis. Sebagai contoh, daripada menulis perintah yang sama berulang kali, kita bisa menulis satu for loop yang menjalankan perintah tersebut berkali-kali.
3. Mudah Traversing Struktur Data: Dengan for loop, kita dapat dengan mudah menjelajahi (traversing) elemen-elemen dalam struktur data seperti array dan string. Ini sangat berguna saat kita perlu mengakses atau memproses setiap elemen dalam struktur data tersebut.

### **Kekurangan Menggunakan for Loop**

Meskipun memiliki banyak keuntungan, for loop juga memiliki beberapa kekurangan:

1. Tidak Bisa Melewati Elemen: Saat menggunakan for loop, kita tidak bisa dengan mudah melewati elemen tertentu dalam sebuah struktur data. For loop biasanya berjalan melalui setiap elemen satu per satu tanpa pengecualian.
2. Hanya Mengikuti Satu Kondisi: For loop biasanya hanya mengevaluasi satu kondisi untuk menentukan apakah loop harus terus berjalan atau berhenti. Ini bisa menjadi keterbatasan jika kita perlu menggunakan lebih dari satu kondisi untuk mengendalikan loop.

## LOOP – NESTED LOOP

"NESTED LOOP" (perulangan bersarang) adalah konsep yang kuat dalam pemrograman di mana sebuah loop ditempatkan di dalam loop lainnya. Ini memungkinkan kita untuk melakukan perulangan berulang kali untuk setiap iterasi dari loop luar. Dengan nested loop, kita dapat mengakses setiap elemen dalam struktur data yang lebih kompleks, seperti matriks dua dimensi atau array bersarang.

Syntax dasar nested loop:

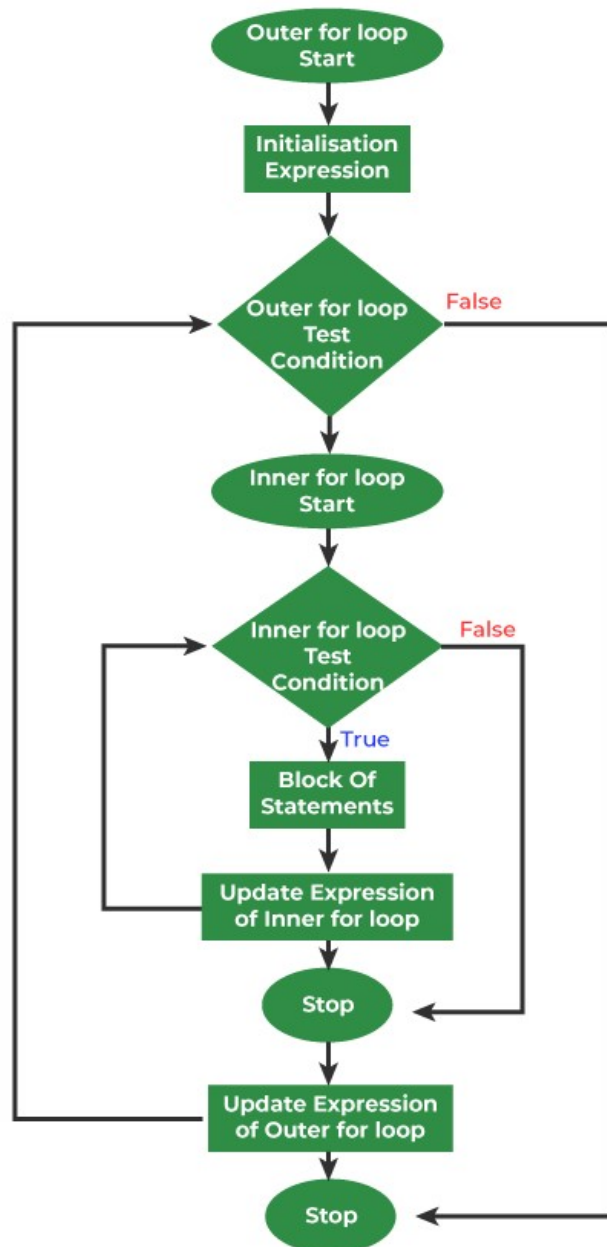
```
for ( initialization; condition; increment ) {  
    for ( initialization; condition; increment ) {  
        // statement of inside loop  
    }  
    // statement of outer loop  
}
```

Konsep nested loop memungkinkan kita untuk mengulang kode di dalam kode, menciptakan hirarki perulangan. Setiap iterasi dari loop luar akan memicu seluruh rangkaian iterasi dari loop dalam. Ini memungkinkan kita untuk mengakses dan memanipulasi data secara lebih terperinci, baik dalam struktur data yang kompleks maupun dalam situasi di mana kita perlu mengulang tugas-tugas berulang.

Salah satu contoh penerapan nested loop adalah saat kita ingin mengakses setiap elemen dalam matriks dua dimensi. Matriks ini adalah struktur data yang memiliki baris dan kolom, sehingga diperlukan dua loop yang bersarang untuk mengakses setiap elemen.

"NESTED LOOP" adalah alat yang sangat berguna dalam situasi di mana kita perlu mengakses elemen-elemen dalam struktur data kompleks atau melakukan tugas-tugas berulang dalam pola yang lebih kompleks. Namun, perlu diingat bahwa penggunaan nested loop dapat mempengaruhi kinerja program, terutama jika jumlah perulangan sangat besar.

Flowchart nested loop



Nested loop adalah konsep dalam pemrograman di mana sebuah loop ditempatkan di dalam loop lainnya. Dengan menggunakan nested loop, kita dapat melakukan perulangan bersarang untuk mengakses setiap elemen dalam suatu struktur data yang memiliki bentuk yang lebih kompleks, seperti matriks dua dimensi atau array bersarang. Setiap iterasi dari loop luar akan memicu seluruh iterasi dari loop dalam. Contoh sederhana nested loop:

```

#include <stdio.h>

int main() {
    // Contoh nested loop untuk mencetak pola segitiga
    int tinggi = 5;

    for (int baris = 1; baris <= tinggi; baris++) {
        for (int kolom = 1; kolom <= baris; kolom++) {
            printf("* ");
        }
        printf("\n");
    }

    return 0;
}

```

**int tinggi = 5;**, adalah deklarasi variabel tinggi dengan tipe data **int** dan nilai **5**. Variabel **tinggi** akan menentukan tinggi dari segitiga yang akan dicetak. **for (int baris = 1; baris <= tinggi; baris++) {**, adalah for loop pertama untuk mengontrol baris dari segitiga. Loop ini akan berjalan dari **baris = 1** hingga **baris <= tinggi**. **for (int kolom = 1; kolom <= baris; kolom++) {**, adalah nested for loop kedua untuk mengontrol kolom dari segitiga. Loop ini akan berjalan dari **kolom = 1** hingga **kolom <= baris**.

**printf("\* ");**, pada baris ini mencetak karakter '\*' dengan diikuti oleh spasi pada setiap iterasi dari nested loop. Dengan demikian, kita akan mendapatkan pola segitiga berbentuk bintang. **printf("\n");**, kemudian ini mencetak karakter newline (**\n**) untuk berpindah ke baris berikutnya setelah selesai mencetak setiap baris dari segitiga. **}**, ni menandakan akhir dari nested for loop kedua, kemudian **}** setelahnya menandakan akhir dari for loop pertama.

Dalam nested loop, setiap iterasi dari loop luar akan memicu seluruh iterasi dari loop dalam, sehingga perlu diperhatikan agar tidak terjadi nested loop yang terlalu dalam atau terlalu rumit untuk mencegah kinerja yang buruk atau kesulitan dalam pemahaman code.

Output program:

```

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

=== Code Execution Successful ===

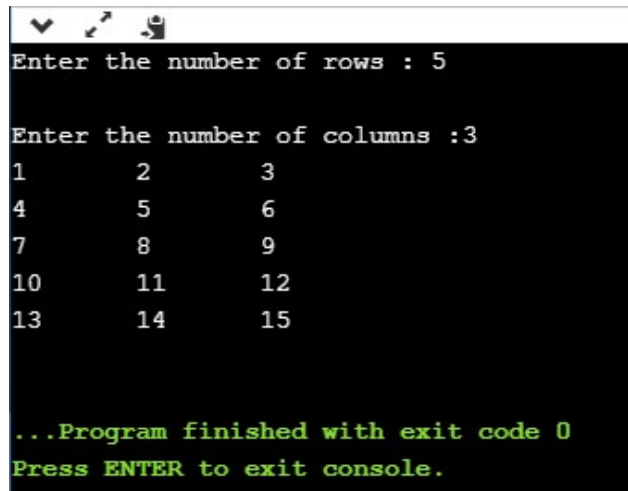
```

```

#include <stdio.h>
int main()
{
    int rows; // deklarasi variabel
    int columns; // deklarasi variabel
    int k=1; // inisialisasi variabel
    printf("Enter the number of rows :"); // input jumlah baris
    scanf("%d",&rows);
    printf("\nEnter the number of columns :"); // input jumlah kolom
    scanf("%d",&columns);
    int a[rows][columns]; //deklarasi 2d array
    int i=1;
    while(i<=rows) // loop luar
    {
        int j=1;
        while(j<=columns) // loop dalam
        {
            printf("%d\t",k); // printing nilai k.
            k++; // increment
            j++;
        }
        i++;
        printf("\n");
    }
}

```

Output:



```

Enter the number of rows : 5

Enter the number of columns : 3
1      2      3
4      5      6
7      8      9
10     11     12
13     14     15

...Program finished with exit code 0
Press ENTER to exit console.

```

Program di atas menggunakan `while loop` bersarang untuk mencetak angka-angka berurutan dalam bentuk matriks sesuai dengan jumlah baris dan kolom yang ditentukan oleh pengguna. Pertama, pengguna diminta untuk memasukkan jumlah baris dan kolom. Setelah menerima input, program mendeklarasikan array dua dimensi `a[rows][columns]`, meskipun array ini tidak digunakan lebih lanjut dalam program. Variabel `k` diinisialisasi dengan nilai 1 dan digunakan untuk mencetak angka berurutan. Loop luar `while(i<=rows)` mengendalikan iterasi

untuk setiap baris, dimulai dari `i = 1` hingga `i` lebih besar dari `rows`. Di dalam loop luar, terdapat loop dalam `while(j<=columns)` yang mengendalikan iterasi untuk setiap kolom dalam baris saat ini, dimulai dari `j = 1` hingga `j` lebih besar dari `columns`. Di setiap iterasi loop dalam, program mencetak nilai `k` dan menambah `k` dengan 1, serta mencetak tab (`\t`) setelah setiap angka untuk memformat keluaran sebagai tabel. Setelah selesai dengan satu baris, loop dalam berhenti, `i` ditambah dengan 1, dan program mencetak baris baru (`\n`). Proses ini diulang hingga semua baris dan kolom terisi dengan angka berurutan yang dimulai dari 1.

## LOOP – BREAK/CONTINUE

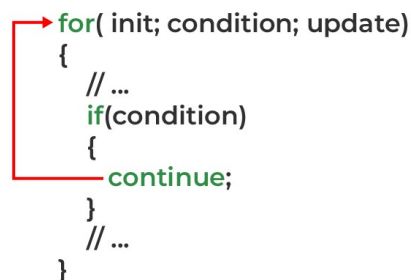
"LOOP - BREAK/CONTINUE" merupakan konsep yang penting dalam pemrograman untuk mengendalikan alur eksekusi dalam perulangan. Dua pernyataan khusus yang sering digunakan untuk mengontrol loop adalah "break" dan "continue".

Break adalah pernyataan yang memiliki kekuatan untuk menghentikan eksekusi loop secara paksa. Ketika pernyataan break dieksekusi dalam loop, loop tersebut akan langsung dihentikan, dan program akan melanjutkan eksekusi dari pernyataan setelah loop. Break umumnya digunakan untuk menghentikan loop ketika kondisi tertentu terpenuhi, sehingga kita tidak perlu menjalankan sisa iterasi yang tidak diperlukan.

**Break** dan **continue** adalah pernyataan yang digunakan untuk mengontrol alur eksekusi dalam loop. **Break** digunakan untuk menghentikan eksekusi loop secara paksa dan keluar dari loop. Ketika **break** dieksekusi, program akan melanjutkan eksekusi dari pernyataan setelah loop.

**Continue** digunakan untuk menghentikan iterasi saat ini dalam loop dan melanjutkan ke iterasi berikutnya. Pada saat **continue** dieksekusi, pernyataan-pernyataan di bawahnya dalam blok loop akan diabaikan, dan program akan melanjutkan dengan menguji kondisi loop untuk iterasi berikutnya.

### Cara Kerja Continue Statement:



```

for( init; condition; update)
{
    // ...
    if(condition)
    {
        continue;
    }
    // ...
}

```

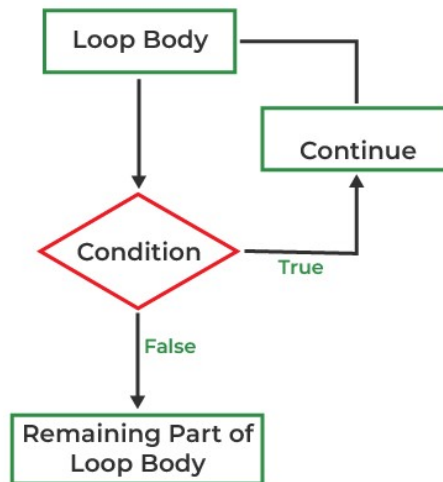
Cara kerja pernyataan continue adalah sebagai berikut:

- LANGKAH 1: Eksekusi loop dimulai setelah kondisi loop dievaluasi menjadi benar.
- LANGKAH 2: Kondisi dari pernyataan continue akan dievaluasi.
- LANGKAH 3A: Jika kondisinya salah, eksekusi normal akan berlanjut.
- LANGKAH 3B: Jika kondisinya benar, kontrol program akan melompat ke awal loop dan semua pernyataan di bawah continue akan dilewati.



- LANGKAH 4: Langkah 1 hingga 4 akan diulangi sampai akhir loop.

Flowchart pernyataan continue:



### Cara Kerja Break Statement

```

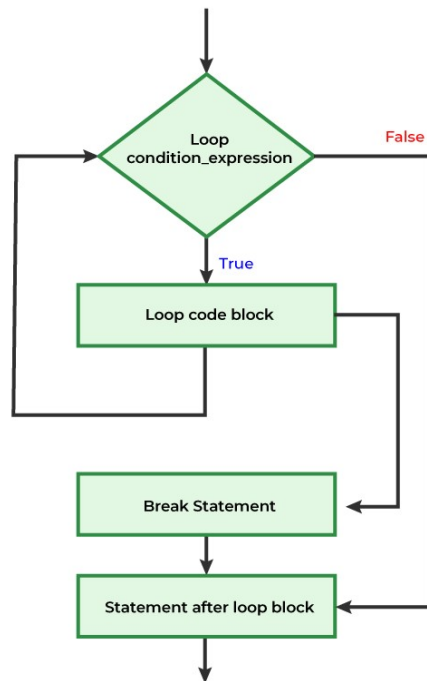
for( init; condition; operation)
{
    // code
    if(condition to break)
    {
        break;
    }
    // code
}
  
```

Cara kerja pernyataan break dalam C dijelaskan di bawah ini:

- LANGKAH 1: Eksekusi loop dimulai setelah kondisi uji dievaluasi.
- LANGKAH 2: Jika ada kondisi break, kondisi tersebut akan dievaluasi.
- LANGKAH 3A: Jika kondisinya benar, kontrol program mencapai pernyataan break dan melewati eksekusi selanjutnya dari loop dengan melompat ke pernyataan langsung di bawah loop.
- LANGKAH 3B: Jika kondisinya salah, alur normal dari kontrol program berlanjut.

Flowchart pernyataan break:

**Break Statement Flow Diagram**



Perbedaan break dan continue

break	continue
The break statement terminates the loop and brings the program control out of the loop.	The continue statement terminates only the current iteration and continues with the next iterations.
The syntax is: break;	The syntax is: continue;
The break can also be used in switch case.	Continue can only be used in loops.

Penggunaan "BREAK" dan "CONTINUE" adalah cara efektif untuk mengontrol alur eksekusi dalam loop sesuai dengan kondisi yang diinginkan. Namun, harus diperhatikan bahwa penggunaan berlebihan atau tidak tepat dari pernyataan ini dapat membuat kode sulit dibaca dan dipahami, jadi penting untuk menggunakan mereka dengan bijak. Contoh sederhana **break** dan **continue**:

```

#include <stdio.h>

int main() {
    // Contoh penggunaan break dalam loop
    for (int i = 1; i <= 5; i++) {
        printf("%d ", i);
        if (i == 3) {
            break;
        }
    }
    printf("\n");

    // Contoh penggunaan continue dalam loop
    for (int j = 1; j <= 5; j++) {
        if (j == 3) {
            continue;
        }
        printf("%d ", j);
    }
    printf("\n");

    return 0;
}

```

Contoh penggunaan **break** dalam loop:

- **for (int i = 1; i <= 5; i++) {**, adalah for loop yang berjalan dari **i = 1** hingga **i <= 5**.
- **printf("%d ", i);**, baris ini mencetak nilai **i** di setiap iterasi loop.
- **if (i == 3) { break; }**, adalah pernyataan **if** yang menguji apakah nilai **i** sama dengan 3. Jika kondisi terpenuhi, maka **break** akan dieksekusi, menghentikan loop secara paksa.
- **printf("\n");**, pada baris ini mencetak karakter newline (**\n**) setelah loop berakhir.

Contoh penggunaan **continue** dalam loop:

- **for (int j = 1; j <= 5; j++) {**, adalah for loop yang berjalan dari **j = 1** hingga **j <= 5**.
- **if (j == 3) {continue;}**, adalah pernyataan **if** yang menguji apakah nilai **j** sama dengan 3. Jika kondisi terpenuhi, maka **continue** akan dieksekusi, melewati iterasi saat ini dan melanjutkan ke iterasi berikutnya.
- **printf("%d ", j);**, ini mencetak nilai **j** di setiap iterasi loop (kecuali ketika nilai **j** adalah 3 karena akan dilewati oleh **continue**).
- **printf("\n");**, ini mencetak karakter newline (**\n**) setelah loop berakhir.

Pada contoh pertama dengan **break**, loop berhenti saat **i** mencapai nilai 3. Sedangkan pada contoh kedua dengan **continue**, nilai 3 tidak dicetak karena dilewati oleh pernyataan **continue**, dan loop melanjutkan hingga selesai mencetak nilai 1, 2, 4, dan 5.

Output program:

```
123
1245

=== Code Execution Successful ===
```

Contoh penggunaan break pada switch case

```
#include <stdio.h>

int main()
{
    // switch variable
    int var = 1;

    // switch statement
    switch (var) {
        case 1:
            printf("Case 1 is Matched.");
            break;

        case 2:
            printf("Case 2 is Matched.");
            break;

        case 3:
            printf("Case 3 is Matched.");
            break;

        default:
            printf("Default case is Matched.");
            break;
    }

    return 0;
}
```

Output program:

```
Case 1 is Matched.

=== Code Execution Successful ===
```

Penjelasan:

- Langkah 1: Variabel switch dievaluasi.
- Langkah 2: Nilai yang dievaluasi dicocokkan dengan semua kasus yang ada.
- Langkah 3A: Jika nilai kasus yang cocok ditemukan, kode yang terkait dijalankan.
- Langkah 3B: Jika nilai kasus yang cocok tidak ditemukan, maka kasus default dijalankan jika ada.
- Langkah 4A: Jika kata kunci break ada dalam kasus, maka kontrol program keluar dari pernyataan switch.
- Langkah 4B: Jika kata kunci break tidak ada, maka semua kasus setelah kasus yang cocok akan dijalankan.
- Langkah 5: Pernyataan setelah pernyataan switch dijalankan.

---

**CODELAB 1****Menghitung Total dan Rata-Rata Bilangan**

**Deskripsi Tugas:** Buatlah sebuah program dalam bahasa C yang meminta pengguna untuk memasukkan beberapa bilangan bulat positif. Program ini akan menggunakan loop untuk menghitung total dan rata-rata dari bilangan-bilangan tersebut hingga pengguna memasukkan angka negatif.

**Langkah-langkah:**

1. Program meminta pengguna memasukkan bilangan bulat positif satu per satu.
2. Program akan terus meminta masukan hingga pengguna memasukkan angka negatif.
3. Gunakan while loop untuk menjumlahkan bilangan yang dimasukkan.
4. Setelah loop selesai, hitung rata-rata dari bilangan-bilangan tersebut.
5. Cetak total dan rata-rata bilangan yang telah dimasukkan.

Contoh output program yang diharapkan:

```
Masukkan bilangan bulat (masukkan angka negatif untuk berhenti): 5
Masukkan bilangan bulat (masukkan angka negatif untuk berhenti): 8
Masukkan bilangan bulat (masukkan angka negatif untuk berhenti): 2
Masukkan bilangan bulat (masukkan angka negatif untuk berhenti): 9
Masukkan bilangan bulat (masukkan angka negatif untuk berhenti): 2
Masukkan bilangan bulat (masukkan angka negatif untuk berhenti): -3

Total bilangan: 26
Rata-rata bilangan: 5.20

=== Code Execution Successful ===
```

**CODELAB 2****Mencari Karakter Unik dalam String**

**Deskripsi Tugas:** Andi adalah seorang penggemar seni dan ingin membuat program dalam bahasa C yang dapat mencetak karakter unik dari sebuah string. Andi ingin program tersebut menggunakan for loop, break, dan continue.

**Langkah-langkah:**

1. Program meminta pengguna memasukkan sebuah string.
2. Gunakan for loop untuk memeriksa setiap karakter dalam string.
3. Gunakan break untuk menghentikan pencarian saat menemukan karakter yang telah diulang.
4. Gunakan continue untuk melompati karakter yang sudah dicetak sebelumnya.

**Contoh output program yang diharapkan:**

```
Masukkan sebuah string: pemrograman
pemrogan

=== Code Execution Successful ===
```

**KEGIATAN 1**

Budi adalah seorang akuntan yang sedang menyelesaikan laporan keuangannya. Dia ingin membuat program untuk menghitung total dari sejumlah transaksi yang dia masukkan. Bantu Budi membuat program tersebut! Manfaatkan semua sub materi pada modul 4 ini. Semakin lengkap maka nilai yang kalian dapatkan semakin tinggi.

**Contoh output program yang diharapkan:**

```
=== Pencatatan Transaksi ===

Masukkan transaksi selanjutnya (gunakan 0 untuk menyelesaikan):
1000
2000
3000
4000
0

Total dari transaksi tersebut adalah: 10000

Apakah Anda ingin mencatat transaksi lagi? (1=Ya, 0=Tidak): 1

=== Pencatatan Transaksi ===

Masukkan transaksi selanjutnya (gunakan 0 untuk menyelesaikan):
30000
2200
12000
32000
0

Total dari transaksi tersebut adalah: 76200

Apakah Anda ingin mencatat transaksi lagi? (1=Ya, 0=Tidak): 0

Terima kasih!

=== Code Execution Successful ===
```



---

**KRITERIA & DETAIL PENILAIAN**

Kriteria	Poin
Codelab 1	10
Codelab 2	10
Kegiatan 1	35
Pemahaman	25
Ketepatan Menjawab	20