

VERSI 1.0
AGUSTUS, 2024



[PEMROGRAMAN DASAR]

MODUL 2 – USER INPUTS, CONSTANTS, OPERATORS, BOOLEANS, STRING,
POINTERS

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:

- User Inputs
- Constants
- Operators
- Booleans
- String
- Pointer

TUJUAN

- Mahasiswa dapat mengembangkan program yang interaktif dengan menerima input dari pengguna untuk memodifikasi perilaku program sesuai kebutuhan.
- Mahasiswa dapat menggunakan konstanta untuk menyimpan nilai tetap yang tidak berubah selama program berjalan.
- Mahasiswa dapat menggunakan berbagai operator matematika dan logika untuk melakukan perhitungan dan pengambilan keputusan dalam program.
- Mahasiswa dapat menggunakan tipe data boolean untuk mengimplementasikan logika pengambilan keputusan dalam program.
- Mahasiswa dapat memanipulasi data string untuk mengolah dan memanipulasi teks dalam program.
- Mahasiswa dapat memahami konsep pointer dan penerapannya untuk mengakses dan memanipulasi data dengan efisien.

TARGET MODUL

- Mahasiswa dapat membuat program sederhana yang meminta dan menggunakan input dari pengguna.
- Mahasiswa dapat mendeklarasikan dan memanfaatkan konstanta dalam program.
- Mahasiswa dapat mengaplikasikan operator dalam ekspresi dan kondisi program dengan tepat.

- Mahasiswa dapat membuat struktur keputusan sederhana dengan menggunakan nilai boolean.
- Mahasiswa dapat melakukan operasi sederhana pada string, seperti penggabungan dan pemotongan.
- Mahasiswa dapat menggunakan pointer untuk memanipulasi variabel dan data dalam program.

PERSIAPAN SOFTWARE/APLIKASI

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

MATERI PRAKTIKUM

USER INPUTS – INT INPUT

Dalam konteks bahasa pemrograman C, "USER INPUTS" merujuk pada interaksi antara program yang sedang berjalan dengan pengguna melalui keyboard atau perangkat masukan lainnya. Ini memberikan kesempatan bagi pengguna untuk memberikan informasi atau data kepada program yang sedang dijalankan. Data yang dimasukkan oleh pengguna dapat berupa angka, teks, atau jenis data lainnya yang relevan dengan tujuan program.

Ketika kita menulis program, kita sering ingin program tersebut tidak hanya menjalankan serangkaian instruksi yang telah ditentukan sebelumnya, tetapi juga bisa menyesuaikan perilakunya berdasarkan input yang diberikan oleh pengguna. Inilah peran penting dari "USER INPUTS". Dengan menggunakan input dari pengguna, kita dapat menciptakan program yang lebih interaktif dan lebih responsif terhadap kebutuhan pengguna.

Dalam bahasa pemrograman C, salah satu cara umum untuk mengambil "USER INPUTS" adalah melalui fungsi **scanf()**. Fungsi ini memungkinkan program untuk meminta pengguna memasukkan nilai dan kemudian mengambil nilai tersebut dari keyboard. Data yang dimasukkan oleh pengguna melalui **scanf()** bisa disimpan dalam variabel yang telah ditentukan sebelumnya dalam program.

Misalnya, jika Kalian sedang membuat program yang menghitung luas segitiga, Kalian dapat meminta pengguna untuk memasukkan panjang alas dan tinggi segitiga menggunakan "USER INPUTS"

melalui fungsi **scanf()**. Program kemudian dapat menghitung luas segitiga berdasarkan nilai-nilai yang dimasukkan oleh pengguna dan menghasilkan output yang sesuai.

Jadi, "USER INPUTS" adalah komponen kunci dalam membuat program yang interaktif dan dapat menanggapi kebutuhan serta preferensi pengguna secara dinamis. Dengan memahami cara mengambil dan memproses input dari pengguna, kita dapat menciptakan pengalaman yang lebih baik bagi pengguna program yang kita buat.

Pada modul 1 telah mempelajari bahwa **printf()** digunakan untuk menampilkan nilai dalam C. Untuk mendapatkan input pengguna, dapat menggunakan fungsi **scanf()**, contohnya:

```
// Buat variabel integer yang akan menyimpan angka yang kita dapatkan dari user
int number;

// Minta user untuk mengetik nomor
printf("Masukkan Nomor: \n");

// Dapat dan simpan nomor yang diketik user
scanf("%d", &number);

// Keluarkan nomor yang diketik user
printf("Angka yang Diinputkan: %d", number);
```

Fungsi **scanf()** mengambil dua argumen, yaitu pertama penentu format variabel (**%d** pada contoh di atas) dan yang kedua operator referensi (**&number**) yang menyimpan alamat memori variabel. Pada baris pertama, sebuah variabel bernama **number** dengan tipe data **int** (integer) dideklarasikan. Variabel ini akan digunakan untuk menyimpan angka yang dimasukkan oleh pengguna nantinya.

Pada baris kedua, menggunakan fungsi **printf()** untuk mencetak teks "Masukkan Nomor: " ke layar. Ini adalah instruksi bagi pengguna untuk memasukkan nomor. Pada baris ketiga, fungsi **scanf()** digunakan untuk mendapatkan input dari pengguna. **%d** merupakan spesifier format yang digunakan untuk membaca input berupa bilangan bulat (integer). Fungsi **scanf()** akan menyimpan nilai yang dimasukkan oleh pengguna ke dalam variabel **number** menggunakan operator **&** (alamat variabel). Dengan ini, nilai yang dimasukkan oleh pengguna akan disimpan dalam variabel **number**.

Pada baris terakhir, hasil dari input yang dimasukkan oleh pengguna akan dicetak ke layar menggunakan fungsi **printf(). %d** merupakan spesifikier format yang digunakan untuk mencetak nilai variabel **number**, sehingga akan mencetak angka yang dimasukkan oleh pengguna setelah teks "Angka yang Diinputkan: ".

USER INPUTS – STRING INPUT

Dalam dunia pemrograman, "USER INPUTS" adalah inti dari interaksi antara program dan pengguna. Ini menciptakan jembatan antara kode yang telah ditulis dengan dunia nyata, memungkinkan program untuk beradaptasi dengan data yang diberikan oleh pengguna. Dalam bahasa pemrograman C, ini berarti menerima dan memproses informasi yang diinputkan melalui keyboard atau perangkat masukan lainnya. Data ini bisa berupa angka, karakter, atau, yang paling sering, String.

"STRING INPUT" dalam bahasa pemrograman C mengacu pada cara kita menerima dan memanipulasi input teks dari pengguna. Ini sangat penting ketika kita ingin mengambil input dalam bentuk kalimat, nama, atau data berbasis teks lainnya. Meskipun sebagian besar data pada akhirnya dianalisis dalam bentuk numerik, string memberikan konteks dan makna pada data tersebut.

Dalam C, kita menggunakan array karakter (dikenal sebagai string) untuk menangani input teks. "STRING INPUT" melibatkan proses menerima input dari pengguna, menyimpannya dalam array karakter, dan kemudian memproses atau mengolah string sesuai kebutuhan program. Penggunaan String Input memungkinkan program untuk mengambil informasi seperti nama, alamat, kata sandi, dan banyak lagi.

Contoh yang umum adalah ketika kita meminta pengguna untuk memasukkan nama mereka. Dengan "STRING INPUT", kita bisa membuat program yang dapat menangani berbagai nama, baik yang pendek maupun yang panjang. Ini memungkinkan pengguna untuk berinteraksi dengan program dengan cara yang lebih alami, seperti saat berkomunikasi dengan manusia lainnya.

Kalian juga bisa mendapatkan string yang dimasukkan oleh pengguna, contohnya:

```
// Buat String
Char namaDepan[30];

// Minta User untuk Memasukkan Teks
printf("Ketikkan Nama Depan Kamu: \n");

//Dapatkan dan Simpan Teks
scanf("%s", &namaDepan);

// Keluarkan Teks
printf("Hai %s", namaDepan);
```

Pada baris pertama, mendeklarasikan sebuah array karakter bernama **namaDepan**. Array ini digunakan untuk menyimpan string (teks) yang dimasukkan oleh pengguna. Ukuran array **namaDepan** adalah 30, yang berarti dapat menampung sebuah string dengan panjang hingga 29 karakter (sisa 1 karakter untuk null-terminator '\0' yang menandakan akhir dari string). Pada baris kedua, menggunakan fungsi **printf()** untuk mencetak teks "Ketikkan Nama Depan Kamu: " ke layar. Ini adalah instruksi bagi pengguna untuk memasukkan nama depan mereka.

Pada baris ketiga, fungsi **scanf()** digunakan untuk mendapatkan input teks dari pengguna. **%s** merupakan spesifier format yang digunakan untuk membaca string (kata atau teks) dari pengguna. Fungsi **scanf()** akan menyimpan string yang dimasukkan oleh pengguna ke dalam array **namaDepan**. Perlu diingat, penggunaan **scanf("%s", &namaDepan);** memiliki beberapa keterbatasan. Jika input pengguna berisi spasi atau karakter selain huruf dan angka, maka fungsi ini akan berhenti membaca setelah menemukan karakter pertama yang tidak valid. Jadi, sebaiknya gunakan fungsi **fgets()** jika kamu ingin membaca seluruh baris input termasuk spasi.

Pada baris terakhir, hasil dari input yang dimasukkan oleh pengguna (nama depan) akan dicetak ke layar menggunakan fungsi **printf()**. **%s** merupakan spesifier format yang digunakan untuk mencetak string (array karakter). Program akan menampilkan pesan "Hai" diikuti dengan nama depan yang dimasukkan oleh pengguna.

CONSTANTS

Dalam pemrograman, "CONSTANTS" (konstanta) adalah nilai yang tetap dan tidak dapat diubah selama eksekusi program. Konstanta ini dapat digunakan untuk menyimpan nilai-nilai yang relevan dan krusial dalam program, seperti nilai konversi, konstanta fisika, atau nilai-nilai tetap lainnya. Konsep

konstanta sangat penting dalam pemrograman karena memungkinkan penggunaan nilai yang dapat diandalkan dan dijamin tidak akan berubah selama eksekusi program.

Dalam bahasa pemrograman C, Anda dapat mendefinisikan konstanta menggunakan kata kunci `const`. Ini memberi tahu kompiler bahwa nilai variabel tersebut tidak akan berubah selama eksekusi program. Konstanta ini juga meningkatkan pembacaan kode karena menyiratkan bahwa nilai tersebut tidak boleh dimodifikasi.

Jika kamu tidak ingin orang lain (atau diri kamu sendiri) mengubah nilai variabel yang ada, maka kamu dapat menggunakan keyword **`const`** yang akan mendeklarasikan variabel sebagai "constant", yang berarti tidak dapat diubah dan hanya bisa dibaca. Kamu harus selalu mendeklarasikan variabel sebagai konstanta ketika kamu memiliki nilai yang tidak mungkin berubah, contohnya:



```
const int number = 15; // number akan selalu 15
number = 10; // error: penugasan variabel read-only 'number'
```

Pada code di atas, kita mendeklarasikan sebuah variabel **`number`** sebagai konstanta dengan menggunakan kata kunci **`const`**. Konstanta ini diberi nilai awal 15. Dengan mendeklarasikan **`number`** sebagai konstanta dengan nilai 15, kita menyatakan bahwa nilai variabel **`number`** tidak dapat diubah setelah diberi nilai awal. Ini berarti **`number`** akan selalu memiliki nilai 15 selama program berjalan, dan tidak dapat diubah menjadi nilai lain. Namun, kemudian pada baris berikutnya, kita mencoba untuk mengubah nilai variabel **`number`** menjadi 10. Code ini akan menyebabkan kesalahan kompilasi karena mencoba untuk melakukan penugasan pada variabel yang dinyatakan sebagai konstanta. Variabel konstan memiliki sifat "read-only", yang berarti nilai yang sudah diberikan pada saat deklarasi tidak bisa diubah selama program berjalan.

Kesalahan yang dihasilkan dari code tersebut adalah "penugasan variabel read-only 'number'", yang mengindikasikan bahwa kamu mencoba untuk mengubah nilai dari variabel yang tidak dapat diubah (konstanta). Dengan menggunakan **`const`** pada deklarasi variabel, kamu menetapkan bahwa variabel tersebut akan selalu memiliki nilai yang sama sepanjang program berjalan. Penggunaan variabel konstan (**`const`**) sangat berguna ketika kamu ingin menyimpan nilai tetap atau nilai konstan yang tidak akan berubah selama eksekusi program. Ini membantu untuk mencegah perubahan nilai yang tidak disengaja

dan meningkatkan kejelasan code, karena pembacaan code akan dengan jelas menunjukkan bahwa variabel tersebut merupakan konstanta yang nilainya tidak berubah.

Contoh Penggunaan

```
#include <stdio.h>

int main() {
    const int MAX_USERS = 100;
    printf("Jumlah maksimum pengguna: %d\n", MAX_USERS);
    // MAX_USERS = 200; // Ini akan menyebabkan kesalahan kompilasi
    return 0;
}
```

Penjelasan:

- **const int MAX_USERS = 100;** Mendeklarasikan variabel **MAX_USERS** sebagai konstanta dengan nilai 100.
- Setiap usaha untuk mengubah nilai **MAX_USERS** akan menghasilkan kesalahan kompilasi.

```
Jumlah maksimum pengguna: 100
```

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

```
#include <stdio.h>

int main() {
    const int MAX_USERS = 100;
    printf("Jumlah maksimum pengguna: %d\n", MAX_USERS);
    MAX_USERS = 200; // Ini akan menyebabkan kesalahan kompilasi
    return 0;
}
```

```
6 |         MAX_USERS = 200; // Ini akan menyebabkan kesalahan kompilasi
  |         ^
```

=== Code Exited With Errors ===

CONSTANTS – DEFINE

Dalam Constants terdapat **#define**, dimana ini adalah direktif preprocessor dalam banyak bahasa pemrograman, termasuk C. Ini digunakan untuk mendefinisikan konstanta simbolik atau makro

dalam kode program sebelum proses kompilasi dimulai. Penggunaan umum **#define** biasanya dimulai dengan simbol **#** diikuti oleh kata kunci **define**, seperti berikut:

```
#define NAMA_KONSTANTA nilai_konstan
```

(#) adalah tanda pengenalan untuk direktif preprocessor dalam banyak bahasa pemrograman. **Define** adalah kata kunci yang menunjukkan bahwa kita akan mendefinisikan sesuatu. **NAMA_KONSTANTA** adalah nama yang diberikan untuk konstanta simbolik atau makro yang akan didefinisikan. **Nilai_konstan** adalah nilai yang akan terkait dengan konstanta ini dan akan menggantikan setiap kemunculan **NAMA_KONSTANTA** dalam kode.

Dengan cara ini, **#define** memungkinkan kalian memberi nama dan memberikan nilai kepada simbolik yang akan digunakan dalam program kalian. Hal ini membantu dalam membaca dan memahami kode serta memudahkan pemeliharaan kode, karena kalian hanya perlu mengubah nilai di satu tempat jika perlu, dan itu akan mempengaruhi semua kemunculan simbolik tersebut dalam program kalian.

Contoh penggunaan :

```
#include <stdio.h>

#define PI 3.14159
#define AREA_CIRCLE(r) (PI * (r) * (r))

int main() {
    double radius = 5.0;
    double area = AREA_CIRCLE(radius);

    printf("Jari-jari: %f\n", radius);
    printf("Luas lingkaran: %f\n", area);
    return 0;
}
```

Penjelasan:

- **#define PI 3.14159** mendefinisikan konstanta **PI** dengan nilai 3.14159.
- **#define AREA_CIRCLE(r) (PI * (r) * (r))** mendefinisikan makro untuk menghitung luas lingkaran berdasarkan jari-jari **r**.
- Program utama menghitung dan mencetak area lingkaran menggunakan konstanta dan makro yang telah didefinisikan.

Output:

```
Jari-jari: 5.000000  
Luas lingkaran: 78.539750  
  
=== Code Execution Successful ===|
```

OPERATORS - INTRODUCTION

Dalam bahasa pemrograman C, "OPERATORS" (operator) adalah unsur penting yang menggunakan simbol atau tanda untuk melaksanakan berbagai macam operasi pada variabel dan nilai dalam sebuah program. Dalam esensinya, operator bertindak sebagai alat yang memungkinkan kita untuk melakukan tugas-tugas matematis, membandingkan nilai, memanipulasi data, serta mengambil keputusan dalam kode program.

Pada dasarnya, operator adalah "perintah" yang memberi petunjuk kepada komputer tentang bagaimana melaksanakan operasi tertentu. Dengan menggunakan operator, kita dapat melakukan perhitungan matematis seperti penambahan, pengurangan, perkalian, dan pembagian. Selain itu, kita juga bisa membandingkan nilai untuk menentukan kondisi, misalnya apakah suatu nilai lebih besar dari atau sama dengan yang lain. Operator juga memberikan kemampuan untuk menggabungkan data, mengubah nilai variabel, serta melakukan berbagai manipulasi data.

Dalam bahasa pemrograman C, operator dapat dikelompokkan ke dalam beberapa jenis berdasarkan fungsi dan sifat operasi yang dilakukan. Ada operator aritmatika untuk operasi matematis, operator perbandingan untuk membandingkan nilai, operator logika untuk mengambil keputusan berdasarkan logika boolean, dan banyak lagi. Setiap jenis operator memiliki peran khusus dalam membantu kita membangun program yang fungsional, efisien, dan mampu mengolah data dengan baik.

Jadi, dengan menggunakan operator, kita dapat memberi "instruksi" kepada komputer tentang bagaimana mengelola data dan menjalankan operasi tertentu, yang pada akhirnya memungkinkan kita mengembangkan program yang sesuai dengan kebutuhan dan tujuan yang diinginkan.

Operator digunakan untuk melakukan operasi pada variabel dan nilai. Pada bahasa C, membagi operator ke dalam grup berikut:

- Operator Arithmetic (Aritmatika)
- Operator Assignment (Penugasan)
- Operator Comparison (Perbandingan)
- Operator Logical (Logika)
- Operator Bitwise

OPERATORS – ARITHMETIC

Dalam bahasa pemrograman C, "OPERATORS ARITHMETIC" (operator aritmatika) adalah kelompok operator yang digunakan untuk melaksanakan berbagai operasi matematika pada operand-operand numerik (bilangan). Operator ini memungkinkan kita untuk melakukan perhitungan matematis dasar dan kompleks, yang seringkali diperlukan dalam pengembangan program.

Dalam bahasa pemrograman C, operator aritmatika memiliki prioritas tertentu, yang mengatur urutan evaluasi dalam operasi matematis yang lebih kompleks. Jika ada beberapa operator dalam satu ekspresi, operator dengan prioritas lebih tinggi akan dievaluasi terlebih dahulu. Namun, kita bisa mengubah urutan evaluasi dengan menggunakan tanda kurung.

Operator aritmatika digunakan untuk melakukan operasi matematika pada operand-operand numerik (bilangan). Berikut ini adalah contoh operator aritmetika:

Operator	Name	Description	Example
+	Addition	Adds together two values	$x + y$
-	Subtraction	Subtracts one value from another	$x - y$
*	Multiplication	Multiplies two values	$x * y$
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	$x \% y$
++	Increment	Increases the value of a variable by 1	$++x$
--	Decrement	Decreases the value of a variable by 1	$--x$

Berikut adalah contoh penggunaan beberapa operator aritmetika:

```
#include <stdio.h>

int main() {
    int num1 = 10;
    int num2 = 5;

    // Penjumlahan
    int hasilPenjumlahan = num1 + num2;
    printf("Hasil penjumlahan: %d\n", hasilPenjumlahan);

    // Pengurangan
    int hasilPengurangan = num1 - num2;
    printf("Hasil pengurangan: %d\n", hasilPengurangan);

    // Perkalian
    int hasilPerkalian = num1 * num2;
    printf("Hasil perkalian: %d\n", hasilPerkalian);

    // Pembagian
    int hasilPembagian = num1 / num2;
    printf("Hasil pembagian: %d\n", hasilPembagian);

    // Sisa bagi (modulo)
    int hasilModulo = num1 % num2;
    printf("Hasil modulo (sisa bagi): %d\n", hasilModulo);

    return 0;
}
```

Output:

```
Hasil penjumlahan: 15
Hasil pengurangan: 5
Hasil perkalian: 50
Hasil pembagian: 2
Hasil modulo (sisa bagi): 0
```

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

Penjelasan:

- Operator: +

Deskripsi: Operator penjumlahan digunakan untuk menambah dua operand. dalam contoh di atas menjumlahkan antara variable num1 dan num2.

- Operator: -

Deskripsi: Operator pengurangan digunakan untuk mengurangi satu operand dari operand yang lain. dalam contoh di atas mengurangi antara variable num1 dan num2.

- Operator: *
Deskripsi: Operator perkalian digunakan untuk mengalikan dua operand. dalam contoh di atas mengalikan antara variable num1 dan num2.
- Operator: /
Deskripsi: Operator pembagian digunakan untuk membagi satu operand dengan operand lain. dalam contoh di atas membagi antara variable num1 dan num2.
- Operator: %
Deskripsi: Operator modulus digunakan untuk mendapatkan sisa pembagian dari dua operand. dalam contoh di atas mendapatkan sisa bagi antara variable num1 dan num2.

OPERATORS – ASSIGNMENT

Dalam bahasa pemrograman C, "OPERATORS ASSIGNMENT" (operator penugasan) adalah kelompok operator yang digunakan untuk memberikan nilai kepada variabel. Operator ini memungkinkan kita untuk menginisialisasi variabel, mengubah nilai variabel yang ada, serta melakukan operasi penugasan dengan nilai lainnya.

Operator penugasan utama adalah operator tanda sama dengan (=), yang digunakan untuk memberikan nilai ke variabel. Operator ini menempatkan nilai dari ekspresi di sebelah kanan tanda sama dengan ke dalam variabel di sebelah kiri. Ini memberi fleksibilitas untuk mengatur nilai variabel sesuai dengan kebutuhan program.

Selain operator penugasan dasar, ada juga operator penugasan dengan operasi matematis, seperti += (penugasan dengan penambahan), -=, *=, /=, dan %= . Operator-operator ini memungkinkan kita untuk melakukan operasi matematis dan kemudian menugaskan hasilnya ke variabel yang sama.

Operator penugasan juga terlibat dalam ekspresi yang lebih kompleks. Sebagai contoh, `a = b = 5;` adalah ekspresi yang menugaskan nilai 5 ke b, lalu menugaskannya ke a. Operator penugasan penting untuk mengontrol aliran data dan mengelola nilai-nilai dalam program. Dengan menggunakan operator ini, kita bisa memastikan bahwa variabel memiliki nilai yang sesuai sebelum digunakan dalam operasi lainnya.

Dalam pemrograman, operator penugasan adalah salah satu elemen dasar dalam memanipulasi variabel dan data, yang menghasilkan kode yang lebih mudah dibaca dan dimengerti oleh pengembang lain atau bahkan oleh diri sendiri di masa depan.

Operator penugasan digunakan untuk memberikan nilai ke variabel atau mengubah nilai variabel dengan nilai baru. Berikut ini adalah contoh operator penugasan:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Berikut adalah contoh penggunaan beberapa operator penugasan:

```
#include <stdio.h>

int main() {
    int x = 5;

    // Operator penugasan dengan penambahan
    x += 3;
    printf("Nilai x setelah penambahan: %d\n", x);

    // Operator penugasan dengan pengurangan
    x -= 2;
    printf("Nilai x setelah pengurangan: %d\n", x);

    // Operator penugasan dengan perkalian
    x *= 4;
    printf("Nilai x setelah perkalian: %d\n", x);

    // Operator penugasan dengan pembagian
    x /= 2;
    printf("Nilai x setelah pembagian: %d\n", x);

    // Operator penugasan dengan modulo
    x %= 3;
    printf("Nilai x setelah sisa bagi: %d\n", x);

    return 0;
}
```

output:

```
Nilai x setelah penambahan: 8
Nilai x setelah pengurangan: 6
Nilai x setelah perkalian: 24
Nilai x setelah pembagian: 12
Nilai x setelah sisa bagi: 0
```

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

Penjelasan:

- Operator (+=):
 - Menambah nilai operand di sebelah kanan (3) dengan nilai variabel di sebelah kiri (x), kemudian menyimpan hasilnya ke dalam x.
 - Output: Nilai x setelah penambahan: 8
- Operator (-=):
 - Mengurangi nilai operand di sebelah kanan (2) dari nilai variabel di sebelah kiri (x), kemudian menyimpan hasilnya ke dalam x.
 - Output: Nilai x setelah pengurangan: 6
- Operator (*=):
 - Mengalikan nilai operand di sebelah kanan (4) dengan nilai variabel di sebelah kiri (x), kemudian menyimpan hasilnya ke dalam x.
 - Output: Nilai x setelah perkalian: 24
- Operator (/=):
 - Membagi nilai variabel di sebelah kiri (x) dengan nilai operand di sebelah kanan (2), kemudian menyimpan hasilnya ke dalam x.
 - Output: Nilai x setelah pembagian: 12
- Operator (%=):
 - Menghitung sisa pembagian nilai variabel di sebelah kiri (x) dengan nilai operand di sebelah kanan (3), kemudian menyimpan hasilnya ke dalam x.
 - Output: Nilai x setelah sisa bagi: 0

OPERATORS – COMPARISON

Dalam dunia pemrograman, "OPERATORS" adalah simbol atau tanda yang digunakan untuk melakukan operasi matematika, perbandingan, atau manipulasi data lainnya. Salah satu jenis operator yang sangat umum adalah "COMPARISON OPERATORS" atau operator perbandingan. "COMPARISON OPERATORS" memungkinkan kita untuk membandingkan dua nilai atau ekspresi dan menghasilkan hasil yang bernilai benar (true) atau salah (false) berdasarkan hasil perbandingan tersebut.

Dalam bahasa pemrograman C, "COMPARISON OPERATORS" memainkan peran penting dalam pengambilan keputusan dan pengendalian alur program. Dengan menggunakan operator perbandingan, kita bisa membuat program untuk memutuskan tindakan apa yang harus diambil berdasarkan hubungan antara nilai-nilai yang diperbandingkan.

Operator perbandingan digunakan untuk membandingkan dua nilai (atau variabel). Ini penting dalam pemrograman, karena membantu kita menemukan jawaban dan mengambil keputusan. Nilai kembalian dari perbandingan adalah salah satu 1 atau 0, yang berarti benar (1) atau salah (0). Nilai ini dikenal sebagai nilai Boolean. Berikut ini adalah contoh operator perbandingan:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Berikut adalah contoh penggunaan operator perbandingan:

```
#include <stdio.h>

int main() {
    int a = 10, b = 5;

    if (a == b) {
        printf("%d sama dengan %d\n", a, b);
    } else {
```

```

    printf("%d tidak sama dengan %d\n", a, b);
}

if (a != b) {
    printf("%d tidak sama dengan %d\n", a, b);
} else {
    printf("%d sama dengan %d\n", a, b);
}

if (a > b) {
    printf("%d lebih besar dari %d\n", a, b);
} else {
    printf("%d tidak lebih besar dari %d\n", a, b);
}

if (a < b) {
    printf("%d lebih kecil dari %d\n", a, b);
} else {
    printf("%d tidak lebih kecil dari %d\n", a, b);
}

if (a >= b) {
    printf("%d lebih besar atau sama dengan %d\n", a, b);
} else {
    printf("%d tidak lebih besar atau sama dengan %d\n", a, b);
}

if (a <= b) {
    printf("%d lebih kecil atau sama dengan %d\n", a, b);
} else {
    printf("%d tidak lebih kecil atau sama dengan %d\n", a, b);
}

return 0;

```

Output:

```

10 tidak sama dengan 5
10 tidak sama dengan 5
10 lebih besar dari 5
10 tidak lebih kecil dari 5
10 lebih besar atau sama dengan 5
10 tidak lebih kecil atau sama dengan 5

```

=== Code Execution Successful ===

Penjelasan:

1. Operator (==):

- Memeriksa apakah dua nilai operand sama.
- Output: 10 tidak sama dengan 5

2. Operator (!=):

- Memeriksa apakah dua nilai operand tidak sama.
- Output: 10 tidak sama dengan 5

3. Operator (>):

- Memeriksa apakah nilai operand di sebelah kiri lebih besar dari operand di sebelah kanan.
- Output: 10 lebih besar dari 5

4. Operator (<):

- Memeriksa apakah nilai operand di sebelah kiri lebih kecil dari operand di sebelah kanan.
- Output: 10 tidak lebih kecil dari 5

5. Operator (>=):

- Memeriksa apakah nilai operand di sebelah kiri lebih besar atau sama dengan operand di sebelah kanan.
- Output: 10 lebih besar atau sama dengan 5

6. Operator (<=):

- Memeriksa apakah nilai operand di sebelah kiri lebih kecil atau sama dengan operand di sebelah kanan.
- Output: 10 tidak lebih kecil atau sama dengan 5

OPERATORS – LOGICAL

Operator logika digunakan untuk menggabungkan atau memanipulasi nilai-nilai kebenaran (true atau false). Berikut adalah contoh operator logika:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	<code>x < 5 && x < 10</code>
	Logical or	Returns true if one of the statements is true	<code>x < 5 x < 4</code>
!	Logical not	Reverse the result, returns false if the result is true	<code>!(x < 5 && x < 10)</code>

Berikut adalah contoh penggunaan beberapa operator logika:

```
#include <stdio.h>

int main() {
    int x = 5;
    int y = 3;

    // Operator logika AND (&&)
    if (x > 0 && y > 0) {
        printf("Kedua nilai x dan y lebih besar dari 0\n");
    } else {
        printf("Salah satu atau kedua nilai x dan y tidak lebih besar dari 0\n");
    }

    // Operator logika OR (||)
    if (x > 10 || y > 10) {
        printf("Salah satu dari nilai x atau y lebih besar dari 10\n");
    } else {
        printf("Kedua nilai x dan y tidak lebih besar dari 10\n");
    }

    // Operator logika NOT (!)
    if (!(x > y)) {
        printf("x tidak lebih besar dari y\n");
    } else {
        printf("x lebih besar dari y\n");
    }

    return 0;
}
```

Pada contoh di atas, kita memiliki dua variabel **x** dan **y** yang diberi nilai 5 dan 3. Operator logika AND (**&&**) digunakan untuk menggabungkan dua kondisi. Jika kedua kondisi benar (true), maka hasilnya adalah true. Jika salah satu atau kedua kondisi salah (false), maka hasilnya adalah false. Pada contoh pertama, kita menggunakan operator logika AND untuk memeriksa apakah kedua nilai **x** dan **y** lebih besar dari 0.

Operator logika OR (**| |**) juga menggabungkan dua kondisi. Jika salah satu atau kedua kondisi benar (true), maka hasilnya adalah true. Hanya jika kedua kondisi salah (false), maka hasilnya adalah false. Pada contoh kedua, kita menggunakan operator logika OR untuk memeriksa apakah salah satu dari nilai **x** atau **y** lebih besar dari 10.

Operator logika NOT (**!**) digunakan untuk membalikkan nilai kondisi. Jika kondisi awalnya benar (true), maka NOT akan menghasilkan false, dan sebaliknya. Pada contoh ketiga, kita menggunakan operator logika NOT untuk memeriksa apakah nilai **x** tidak lebih besar dari **y**.

Semua operator logika ini sangat berguna dalam mengendalikan alur program berdasarkan kondisi-kondisi tertentu. Mereka memungkinkan kita untuk membuat keputusan berdasarkan hasil perbandingan atau nilai kebenaran dari beberapa kondisi.

OPERATORS – BITWISE

Dalam dunia pemrograman, terdapat "OPERATORS" khusus yang digunakan untuk melakukan operasi pada level bit dari nilai-nilai dalam bentuk bilangan biner. Ini dikenal sebagai "BITWISE OPERATORS." Operasi-operasi ini memungkinkan kita untuk melakukan manipulasi bit yang lebih mendalam, yang seringkali sangat berguna dalam pengembangan program yang berhubungan dengan pengolahan data tingkat rendah atau operasi perangkat keras.

Operasi "BITWISE" biasanya digunakan dalam pemrograman terkait pengolahan data atau perangkat keras. Misalnya, dalam pengkodean warna dalam format gambar atau manipulasi nilai-nilai yang disimpan dalam representasi biner.

Operator	Meaning of operator
&	Bitwise AND operator
	Bitwise OR operator
^	Bitwise exclusive OR operator
~	One's complement operator (unary operator)
<<	Left shift operator
>>	Right shift operator

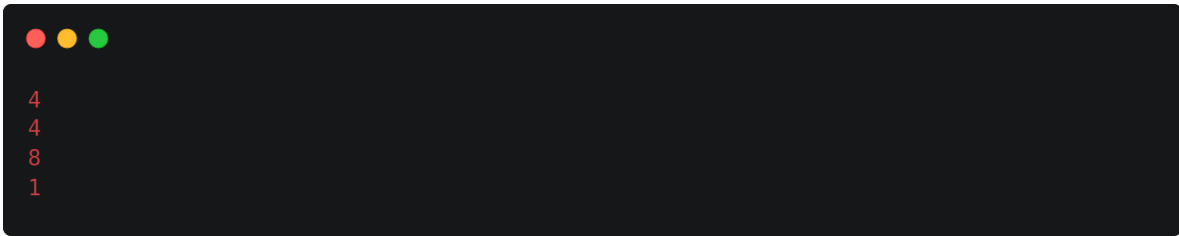
Operator bitwise digunakan untuk melakukan operasi pada level bit dari operand. Berikut contoh dari penggunaan operator bitwise:

```

int myInt;
float myFloat;
double myDouble;
char myChar;

printf("%lu\n", sizeof(myInt));
printf("%lu\n", sizeof(myFloat));
printf("%lu\n", sizeof(myDouble));
printf("%lu\n", sizeof(myChar));

```


A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the output of a program, showing the sizes of variables in bytes: 4, 4, 8, and 1, each on a new line.

Pada baris pertama program, kita memiliki deklarasi variabel `myInt` dengan tipe data `int`. Pada baris kedua, kita memiliki deklarasi variabel `myFloat` dengan tipe data `float`. Pada baris ketiga, kita memiliki deklarasi variabel `myDouble` dengan tipe data `double`. Pada baris keempat, kita memiliki deklarasi variabel `myChar` dengan tipe data `char`. Kemudian, program menampilkan ukuran (size) dari masing-masing variabel menggunakan operator `sizeof()` dan specifier `%lu` dalam fungsi `printf()`.

Ukuran dari tipe data pada setiap komputer dapat bervariasi tergantung pada arsitektur sistem dan kompilernya. Namun, dalam hasil yang ditampilkan di atas, dapat dijelaskan bahwa `int` memiliki ukuran 4 byte (32 bit) dalam sistem yang digunakan, `float` memiliki ukuran 4 byte (32 bit) dalam sistem yang digunakan, `double` memiliki ukuran 8 byte (64 bit) dalam sistem yang digunakan, `char` memiliki ukuran 1 byte (8 bit) dalam sistem yang digunakan. Itulah mengapa hasil yang ditampilkan menunjukkan ukuran seperti di atas. Penting untuk diingat bahwa ukuran tipe data dapat bervariasi di berbagai sistem, dan dapat dipengaruhi oleh sistem operasi, arsitektur CPU, dan compiler yang digunakan.

Contoh Program:

```

#include <stdio.h>

int main() {
    // Deklarasi variabel
    unsigned int a = 12; // 12 dalam biner: 0000 1100
    unsigned int b = 25; // 25 dalam biner: 0001 1001

    // Operator AND
    unsigned int hasil_and = a & b; // 0000 1100 & 0001 1001 = 0000 1000 (8)
    printf("Hasil dari %u & %u = %u\n", a, b, hasil_and);

    // Operator OR
    unsigned int hasil_or = a | b; // 0000 1100 | 0001 1001 = 0001 1101 (29)
    printf("Hasil dari %u | %u = %u\n", a, b, hasil_or);

    // Operator XOR
    unsigned int hasil_xor = a ^ b; // 0000 1100 ^ 0001 1001 = 0001 0101 (21)
    printf("Hasil dari %u ^ %u = %u\n", a, b, hasil_xor);

    // Operator NOT
    unsigned int hasil_not = ~a; // ~0000 1100 = 1111 0011 (bitwise NOT hasilkan bilangan negatif di signed int)
    printf("Hasil dari ~%u = %u\n", a, hasil_not);

    // Left shift
    unsigned int hasil_left_shift = a << 2; // 0000 1100 << 2 = 0011 0000 (48)
    printf("Hasil dari %u << 2 = %u\n", a, hasil_left_shift);

    // Right shift
    unsigned int hasil_right_shift = a >> 2; // 0000 1100 >> 2 = 0000 0011 (3)
    printf("Hasil dari %u >> 2 = %u\n", a, hasil_right_shift);

    return 0;
}

```

Output:

```

Hasil dari 12 & 25 = 8
Hasil dari 12 | 25 = 29
Hasil dari 12 ^ 25 = 21
Hasil dari ~12 = 4294967283
Hasil dari 12 << 2 = 48
Hasil dari 12 >> 2 = 3

```

Program ini mendemonstrasikan penggunaan operator bitwise dalam bahasa C dengan dua bilangan bulat, `a` dan `b`, yang diinisialisasi masing-masing dengan nilai 12 dan 25. Operator bitwise yang digunakan meliputi AND (`&`), OR (`|`), XOR (`^`), dan NOT (`~`), serta operasi geser bit kiri (`<<`) dan kanan (`>>`). Operator AND mengembalikan bit 1 jika kedua bit yang bersesuaian adalah 1,

Agustus, 2024 [PEMROGRAMAN DASAR] 12

menghasilkan nilai 8. Operator OR mengembalikan bit 1 jika salah satu atau kedua bit adalah 1, menghasilkan 29. Operator XOR mengembalikan bit 1 jika salah satu bit adalah 1 dan yang lain 0, menghasilkan 21. Operator NOT membalik semua bit dari `a`, memberikan nilai komplementer yang besar dalam konteks bilangan tak bertanda. Operasi geser bit kiri memindahkan bit `a` dua posisi ke kiri, mengisi sisi kanan dengan nol, sehingga menghasilkan 48, sementara geser bit kanan menggeser bit `a` dua posisi ke kanan, menghasilkan 3. Operator bitwise ini sangat berguna dalam pemrograman yang membutuhkan manipulasi bit langsung untuk tugas seperti pengaturan bit flags, kompresi data, dan algoritma kriptografi.

BOOLEANS

Sangat sering, dalam pemrograman, kalian memerlukan tipe data yang hanya dapat memiliki satu dari dua nilai, seperti:

- YES / NO
- ON / OFF
- TRUE / FALSE

Untuk ini, C memiliki bool tipe data yang dikenal sebagai **boolean**. Boolean mewakili nilai yang baik true atau false. Contoh penggunaan Booleans dapat dilihat pada contoh penggunaan operator perbandingan pada modul ini.

```
#include <stdio.h>
#include <stdbool.h> // Menyertakan pustaka stdbool.h untuk menggunakan tipe boolean

int main() {
    // Deklarasi variabel
    int angka;
    bool isGenap;

    // Meminta pengguna memasukkan sebuah angka
    printf("Masukkan sebuah angka: ");
    scanf("%d", &angka);

    // Menentukan apakah angka tersebut genap atau ganjil
    if (angka % 2 == 0) {
        isGenap = true;
    } else {
        isGenap = false;
    }

    // Menampilkan hasil
    if (isGenap) {
        printf("Angka %d adalah bilangan genap.\n", angka);
    } else {
        printf("Angka %d adalah bilangan ganjil.\n", angka);
    }

    return 0;
}
```

Program ini menggunakan tipe data boolean untuk menentukan apakah sebuah angka yang dimasukkan oleh pengguna adalah bilangan genap atau ganjil. Dalam program ini, pustaka `stdbool.h` digunakan untuk mendefinisikan tipe data `bool`, serta nilai `true` dan `false`. Setelah pengguna memasukkan sebuah angka, program memeriksa apakah angka tersebut habis dibagi dua menggunakan operator modulus (%). Jika hasil modulus adalah 0, maka variabel boolean `isGenap` diatur ke `true`, menandakan bahwa angka tersebut genap; jika tidak, `isGenap` diatur ke `false`, menandakan bahwa angka tersebut ganjil. Akhirnya, program menggunakan pernyataan `if` untuk memeriksa nilai `isGenap` dan mencetak pesan yang sesuai, apakah angka tersebut genap atau ganjil. Penggunaan boolean dalam program ini meningkatkan keterbacaan dan menyederhanakan logika pemeriksaan genap/ganjil.

STRINGS

Dalam pemrograman, "STRINGS" adalah kumpulan karakter yang digunakan untuk merepresentasikan teks atau data berbasis karakter lainnya. String sering digunakan untuk menyimpan informasi seperti kata-kata, frasa, nama, alamat, dan banyak lagi. Namun, dalam bahasa pemrograman C, "STRINGS" diimplementasikan dengan sedikit kompleksitas dibandingkan dengan beberapa bahasa pemrograman lain yang memiliki tipe data string khusus.

Sebagai catatan penting, dalam C, "STRINGS" sebenarnya adalah array karakter (array of characters). Setiap karakter dalam string disimpan dalam elemen-elemen array secara berurutan, dengan karakter null ('\0') menandai akhir dari string.

String digunakan untuk menyimpan teks/karakter. Misalnya, "Hello World" adalah serangkaian karakter. Tidak seperti banyak bahasa pemrograman lainnya, C tidak memiliki tipe String untuk membuat variabel string dengan mudah. Sebagai gantinya, kalian harus menggunakan tipe **char** dan membuat larik karakter untuk membuat string di C. Berikut adalah contohnya:

```
#include <stdio.h>

int main() {
    // Mendeklarasikan dan menginisialisasi string
    char greeting[] = "Hello, World!";

    // Menampilkan string menggunakan printf
    printf("Pesan: %s\n", greeting);

    // Menampilkan karakter-karakter individual dalam string
    printf("Karakter pertama: %c\n", greeting[0]);
    printf("Karakter kedua: %c\n", greeting[1]);
    printf("Karakter ketiga: %c\n", greeting[2]);

    return 0;
}
```

Hasil:

```
Pesan: Hello, World!
Karakter pertama: H
Karakter kedua: e
Karakter ketiga: l
```

Pada contoh di atas, kita mendeklarasikan sebuah array karakter **greeting** dan menginisialisasinya dengan teks "Hello, World!". Compiler akan menambahkan karakter null (`'\0'`) pada akhir array secara otomatis, sehingga array ini menjadi string yang valid dalam bahasa C. Kita kemudian menggunakan fungsi **printf()** untuk menampilkan keseluruhan string dengan specifier **%s**. Selain itu, kita juga dapat mengakses karakter-karakter individual dalam string menggunakan indeks array seperti `greeting[0]`, `greeting[1]`, dan seterusnya. Ingat, dalam bahasa C, indeks dimulai dari 0, sehingga `greeting[0]` mengakses karakter pertama dalam string C mendukung berbagai fungsi yang memanipulasi string yang diakhiri null.

Sr.No.	Fungsi & Tujuan
1	strcpy(s1, s2); Salin string s2 menjadi string s1.
2	strcat(s1, s2); Menggabungkan string s2 ke ujung string s1.
3	strlen(s1); Mengembalikan panjang string s1.
4	strcmp(s1, s2); Mengembalikan 0 jika s1 dan s2 sama; kurang dari 0 jika s1 < s2; lebih besar dari 0 jika s1 > s2.
5	strchr(s1, ch); Mengembalikan pointer ke kemunculan pertama karakter ch dalam string s1.
6	strstr(s1, s2); Mengembalikan pointer ke kemunculan pertama string s2 dalam string s1.

Contoh Implementasi:

1. Strcpy

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

int main() {
    char sumber[] = "Halo, Dunia!";
    char tujuan[50];

    // Menyalin string dari 'sumber' ke 'tujuan'
    strcpy(tujuan, sumber);

    printf("String sumber: %s\n", sumber);
    printf("String tujuan setelah strcpy: %s\n", tujuan);

    return 0;
}
```

Output:

```
String sumber: Halo, Dunia!
String tujuan setelah strcpy: Halo, Dunia!
```

Program ini menyalin string dari variabel sumber ke variabel tujuan menggunakan fungsi strcpy dari pustaka string.h. Fungsi strcpy mengambil dua argumen: variabel tujuan dan variabel sumber. Setelah penyalinan, string yang ada di sumber dicetak, diikuti dengan string yang telah disalin ke tujuan. Fungsi ini sangat berguna untuk membuat salinan string tanpa mengubah string asli.

2. Strcat

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

int main() {
    char pertama[50] = "Halo";
    char kedua[] = ", Dunia!";

    // Menggabungkan string 'kedua' ke 'pertama'
    strcat(pertama, kedua);

    printf("Hasil setelah strcat: %s\n", pertama);

    return 0;
}
```

Output:

```
Hasil setelah strcat: Halo, Dunia!
```

Program ini menggabungkan dua string menggunakan fungsi `strcat`. Variabel `pertama` diinisialisasi dengan string "Halo", dan `kedua` dengan string ", Dunia!". Fungsi `strcat` kemudian menambahkan `kedua` ke akhir `pertama`, menghasilkan string gabungan yang dicetak sebagai hasilnya. `strcat` mengubah string tujuan dengan menambahkan konten string sumber ke ujungnya, jadi pastikan buffer tujuan cukup besar untuk menampung hasil penggabungan.

3. Strlen

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

int main() {
    char teks[] = "Halo, Dunia!";

    // Menghitung panjang string
    int panjang = strlen(teks);

    printf("Panjang string '%s': %d\n", teks, panjang);

    return 0;
}
```

Output:

```
Panjang string 'Halo, Dunia!': 12
```

Program ini menghitung panjang string menggunakan fungsi `strlen`. Variabel `teks` berisi string "Halo, Dunia!". Fungsi `strlen` mengembalikan panjang string (jumlah karakter sebelum karakter null `\0`), yang kemudian dicetak. Fungsi ini berguna untuk mendapatkan ukuran string, terutama ketika bekerja dengan alokasi memori dinamis atau saat memanipulasi string.

4. Strcmp

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

int main() {
    char string1[] = "Halo";
    char string2[] = "halo";

    // Membandingkan dua string
    int hasil = strcmp(string1, string2);

    if (hasil == 0) {
        printf("String '%s' dan '%s' adalah sama.\n", string1, string2);
    } else if (hasil < 0) {
        printf("String '%s' lebih kecil dari '%s'.\n", string1, string2);
    } else {
        printf("String '%s' lebih besar dari '%s'.\n", string1, string2);
    }

    return 0;
}
```

Output:

```
String 'Halo' lebih kecil dari 'halo'.
```

Program ini membandingkan dua string, `string1` dan `string2`, menggunakan fungsi `strcmp`. Fungsi ini mengembalikan 0 jika kedua string sama, nilai negatif jika `string1` lebih kecil secara leksikografis, dan nilai positif jika `string1` lebih besar. Program mencetak hasil perbandingan ini dalam bentuk kalimat yang menjelaskan hubungan antara kedua string, membantu memahami perbandingan leksikografis berbasis ASCII.

5. Strchr

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

int main() {
    char teks[] = "Halo, Dunia!";
    char karakter = 'D';

    // Mencari karakter 'D' dalam string
    char *hasil = strchr(teks, karakter);

    if (hasil != NULL) {
        printf("Karakter '%c' ditemukan di posisi: %ld\n", karakter, hasil - teks);
    } else {
        printf("Karakter '%c' tidak ditemukan.\n", karakter);
    }

    return 0;
}
```

Output:

```
Karakter 'D' ditemukan di posisi: 6
```

Program ini mencari karakter D dalam string teks menggunakan fungsi strchr. Jika karakter ditemukan, fungsi mengembalikan pointer ke posisi karakter tersebut dalam string; jika tidak, mengembalikan NULL. Program kemudian menghitung posisi karakter dengan mengurangi pointer awal dari pointer yang dikembalikan oleh strchr dan mencetak hasilnya. Ini berguna untuk menemukan karakter dalam string dan menentukan posisi atau memulai operasi lanjutan dari posisi tersebut.

6. Strstr

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

int main() {
    char teks[] = "Halo, Dunia!";
    char substring[] = "Dunia";

    // Mencari substring 'Dunia' dalam string
    char *hasil = strstr(teks, substring);

    if (hasil != NULL) {
        printf("Substring '%s' ditemukan di posisi: %ld\n", substring, hasil - teks);
    } else {
        printf("Substring '%s' tidak ditemukan.\n", substring);
    }

    return 0;
}
```

Output:

```
Substring 'Dunia' ditemukan di posisi: 6
```

Program ini mencari substring "Dunia" dalam string teks menggunakan fungsi `strstr`. Jika substring ditemukan, fungsi mengembalikan pointer ke awal substring dalam string utama; jika tidak, mengembalikan `NULL`. Program kemudian mencetak posisi awal substring dengan menghitung selisih pointer, membantu menentukan lokasi substring pertama kali muncul, berguna dalam pencarian teks dan parsing string yang

lebih kompleks.

POINTERS

Dalam bahasa pemrograman C, "POINTERS" adalah salah satu konsep paling fundamental dan kuat. Mereka memungkinkan kita untuk mengakses dan memanipulasi alamat memori tempat nilai disimpan. Secara sederhana, "POINTERS" adalah variabel yang berisi alamat memori suatu nilai, bukan nilai itu sendiri.

Konsep "POINTERS" dapat membingungkan pada awalnya, tetapi mereka memainkan peran penting dalam fleksibilitas dan efisiensi dalam pemrograman C. Dengan "POINTERS," kita bisa mengirimkan data antara fungsi, mengelola alokasi memori dinamis, dan melakukan manipulasi langsung pada nilai dalam memori.

Kalian telah belajar bahwa kita bisa mendapatkan alamat memori dari sebuah variabel dengan operator referensi **&**. Pointer adalah salah satu fitur penting dalam bahasa pemrograman C. Sebuah pointer adalah variabel yang berisi alamat memori dari suatu variabel lain. Dengan menggunakan pointer, Kalian dapat secara langsung mengakses atau memanipulasi nilai dari variabel yang sebenarnya melalui alamatnya. Hal ini memberikan fleksibilitas dan efisiensi dalam penggunaan memori, serta memungkinkan kita untuk mengoperasikan data secara langsung.

Dalam bahasa C, untuk mendeklarasikan pointer, kita menggunakan tanda asterisk (*) sebelum nama variabel pointer. Sebagai contoh, **int *ptr;** akan mendeklarasikan **ptr** sebagai pointer ke variabel bertipe integer. Misalnya, jika kita memiliki variabel **num** bertipe integer dan pointer **ptr**, maka kita dapat menggunakan pointer untuk menyimpan alamat memori dari variabel **num** dan kemudian mengakses nilai atau mengubah nilainya melalui pointer. Berikut contohnya:

Pada contoh di atas, kita mendeklarasikan sebuah variabel **num** bertipe **integer** dengan nilai 10.

```
#include <stdio.h>

int main() {
    int num = 10;
    int *ptr; // Deklarasi pointer bertipe integer

    // Mengisi pointer dengan alamat memori variabel num
    ptr = &num;

    printf("Nilai dari num: %d\n", num);
    printf("Alamat memori dari num: %p\n", &num);
    printf("Nilai yang diakses melalui pointer ptr: %d\n", *ptr);
    printf("Alamat memori yang disimpan dalam pointer ptr: %p\n", ptr);

    // Mengubah nilai variabel num melalui pointer ptr
    *ptr = 20;
    printf("Nilai baru dari num setelah diubah melalui pointer: %d\n", num);

    return 0;
}
```

Kemudian, kita mendeklarasikan sebuah pointer **ptr** yang menunjuk ke variabel bertipe **integer**. Setelah itu, kita mengisi pointer **ptr** dengan alamat memori dari variabel **num** menggunakan operator **&** (address-of). Dalam **printf()**, kita menggunakan specifier **%p** untuk mencetak alamat memori dari variabel dan pointer. Ketika kita mencetak nilai yang diakses melalui pointer **ptr** menggunakan ***ptr**, itu menghasilkan nilai dari variabel **num**. Selanjutnya, kita mengubah nilai variabel **num** melalui pointer **ptr** dengan menugaskan nilai baru ke ***ptr**. Hasilnya menunjukkan bahwa perubahan nilai melalui pointer juga mempengaruhi nilai variabel aslinya.

CODELAB 1

Penghitung Kalori

Buatlah sebuah program untuk menghitung total kalori harian yang dikonsumsi berdasarkan input dari pengguna. Program ini harus mencakup:

1. Pengenalan Program:
 - Menampilkan judul program dan deskripsi singkat kepada pengguna menggunakan fungsi `printf`.
2. Mengambil input dari pengguna untuk: Kalori per makanan yang dikonsumsi.
3. Menghitung total kalori:
 - Jumlahkan kalori yang dihitung dari masing-masing makanan.
4. Memeriksa batas kalori harian:
 - Gunakan konstanta untuk menyimpan nilai batas kalori harian.
 - Memeriksa apakah total kalori melebihi batas kalori harian.
5. Menampilkan hasil:
 - Menampilkan total kalori yang dihitung.
 - Menampilkan pesan apakah total kalori melebihi batas kalori harian atau tidak.

Output yang diharapkan:

```
===== Program Penghitung Kalori Harian =====  
Program ini akan menghitung total kalori harian yang dikonsumsi.  
  
Masukkan kalori makanan pertama: 1000  
Masukkan kalori makanan kedua: 500  
Masukkan kalori makanan ketiga: 1000  
  
Total kalori: 2500  
Total kalori melebihi batas harian.  
  
=== Code Execution Successful ===
```


CODELAB 2

Pada codelab ini, Anda diberikan sebuah program yang telah ditulis untuk melakukan operasi dasar pada boolean, string, dan pointer. Tugas Anda adalah menulis ulang program ini, memperbaiki kesalahan sintaks, dan menjalankan program untuk memastikan semuanya bekerja dengan benar.

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

int main() {
    // Boolean values
    boolean flagTrue = true;
    boolean flagFalse = false;

    printf("Boolean values demonstration:\n");
    printf("flagTrue: %d\n", );
    printf("flagFalse: %d\n", );

    // String operations
    char greeting[] = "Hello, World!";
    char copyGreeting[20];
    strcpy(copyGreeting, greeting);

    printf("\nString operations:\n");
    printf("Original greeting: %s\n", string1);
    printf("Copied greeting: %s\n", string2);

    // Pointer demonstrations
    int number = 100;
    int *pointer = &number;

    printf("\nPointer demonstrations:\n");
    printf("Value of number: %d\n", );
    printf("Address of number: %p\n", &number);
    printf("Value of pointer: %p\n", pointer);
    printf("Value pointed by pointer: %d\n", );

    return 0;
}
```

KEGIATAN 1

Menghitung Luas dan Keliling Lingkaran

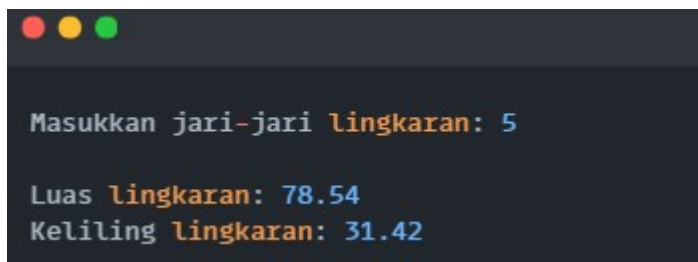
Seorang siswa ingin membuat program yang dapat menghitung luas dan keliling dari sebuah lingkaran. Bantulah siswa tersebut dengan membuat program yang:

1. Meminta pengguna untuk memasukkan jari-jari lingkaran.
2. Menghitung luas lingkaran menggunakan rumus: $\text{Luas} = \pi \times \text{jari-jari}^2$
3. Menghitung keliling lingkaran menggunakan rumus: $\text{Keliling} = 2 \times \pi \times \text{jari-jari}$
4. Menampilkan hasil perhitungan luas dan keliling lingkaran kepada pengguna.

Petunjuk:

- Gunakan nilai π (pi) dengan presisi 3.14159.
- Gunakan fungsi `printf` untuk menampilkan pesan dan `scanf` untuk membaca input dari pengguna.
- Pastikan program memvalidasi input dan memberikan output yang jelas.

Contoh Output:

A screenshot of a terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The terminal displays the following text: 'Masukkan jari-jari lingkaran: 5' on the first line, 'Luas lingkaran: 78.54' on the second line, and 'Keliling lingkaran: 31.42' on the third line. The word 'lingkaran' is highlighted in orange in all three lines.

```
Masukkan jari-jari lingkaran: 5
Luas lingkaran: 78.54
Keliling lingkaran: 31.42
```

KRITERIA & DETAIL PENILAIAN

	Poin
Codelab 1	15
Codelab 2	5
Kegiatan 1	25
Pemahaman	35
Ketepatan Menjawab	20