

VERSI 1.2

JULI, 2023



[PEMROGRAMAN DASAR]

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

DISUSUN OLEH:

- ANNISA ARTANTI WIDYADHANA

- KIARA AZZAHRA

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.

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:

A screenshot of a code editor window with a dark background. At the top left, there are three colored window control buttons: red, yellow, and green. The main area of the editor contains a single line of C preprocessor code: `#define NAMA_KONSTANTA nilai_konstan`. The text is in a light blue/cyan color.

```
#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.

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;
}
```

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;
}
```

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:

```
int x = 5;
int y = 3;
printf("%d", x > y); // mengembalikan 1 (benar) karena 5 lebih besar dari 3
```

Hasil:



```
1
```

Pada dua baris pertama, dua variabel **x** dan **y** dideklarasikan dan diberi nilai masing-masing. **x** diberi nilai 5 dan **y** diberi nilai 3. Mengapa hasilnya bisa 1? Karena nilai **x** (5) lebih besar dari nilai **y** (3), maka perbandingan **x > y** menghasilkan nilai true (benar). Dalam bahasa C, nilai true direpresentasikan dengan angka 1, sedangkan nilai false direpresentasikan dengan angka 0. Oleh karena itu, hasil perbandingan **x > y** adalah true (benar), yang dalam bentuk angka adalah 1, dan itulah yang dicetak ke layar menggunakan **printf()**.

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 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));
```

Hasil:

```
4
4
8
1
```

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.

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.

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 menginisiasinya 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.

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.

C mendukung berbagai fungsi yang memanipulasi string yang diakhiri null.

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:

```
#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;
}
```

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

Kamu adalah seorang programmer. Suatu hari seorang client ingin meminta jasmu untuk membuat program kalkulator yang **inputannya berasal dari user**. Client tersebut ingin hasil dari output dikategorikan sesuai jenis hewannya (sapi dan kambing) . Kebetulan client tersebut merupakan seorang peternak dimana dia memiliki sapi berbobot :

- Sapi Jantan I = 550 Kg
- Sapi Jantan II = 535,6 Kg
- Sapi Betina I = 498,23 Kg
- Sapi Betina II = 628 Kg

Selain sapi, dia juga memiliki kambing yang berbobot :

- Kambing Jantan I = 120 Kg
- Kambing Jantan II = 85.8 Kg
- Kambing Betina I = 23 Kg

Buatlah program tersebut dalam bahasa C dan **gunakan constant** untuk membuat nilai kambing menjadi tetap dan tidak bisa diubah.

Contoh hasil output :

```
Masukkan berat sapi 1: 550
Masukkan berat sapi 2: 535
Masukkan berat sapi 3: 498.23
Masukkan berat sapi 4: 628

Hasil Berat Sapi: 2211.23
Hasil Berat Kambing: 221.12
```

CODELAB 2

Copy paste kode program di bawah ini ke IDE kalian. Cobalah untuk melengkapi dan memperbaikinya, jika sudah tunjukkan ke asisten masing - masing dan jelaskan apa saja yang telah kalian perbaiki

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

```

int main() {
    // Boolean
    boolean isTrue = true;
    boolean isFalse = false;

    printf("Boolean values:\n");
    printf("isTrue: %d\n", );
    printf("isFalse: %d\n", );

    // Strings
    char str1[] = "Hello";
    char str2[10];
    strcpy(str2, str1);

    printf("\nStrings:\n");
    printf("str1: %s\n", strcpy1);
    printf("str2: %s\n", strcpy2);

    // Pointers
    int num = 42;
    int *ptr = &num;

    printf("\nPointers:\n");
    printf("Value of num: %d\n", );
    printf("Address of num:  \n", &num);
    printf("Value of ptr:  \n", ptr);
    printf("Value pointed by ptr: %d\n", );

    return 0;
}

```

KEGIATAN 1

Andi adalah seorang mahasiswa yang ingin menghitung nilai rata-rata dari beberapa mata kuliah yang dia ambil. Mata kuliah tersebut adalah Matematika, Fisika, dan Biologi. Setiap mata kuliah memiliki bobot sks (satuan kredit semester) yang berbeda. Berikut adalah bobot sks untuk setiap mata kuliah:

Matematika	: 4 sks
Fisika	: 3 sks
Biologi	: 2 sks

Bantu Andi untuk membuat program yang dapat menghitung nilai rata-rata berdasarkan input nilai Andi untuk setiap mata kuliah. Manfaatkan semua sub materi pada modul 2 ini. Semakin lengkap maka nilai yang kalian dapatkan semakin tinggi.

Berikut contoh output yang dihasilkan :

```
=== Penghitungan Nilai Rata-rata ===
Masukkan nilai Matematika: 85
Masukkan nilai Fisika: 78
Masukkan nilai Biologi: 92
Nilai Rata-rata: 84.55
```

KRITERIA & DETAIL PENILAIAN

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