**Tugas 1**

|  |  |  |
| --- | --- | --- |
| **Nama & NPM** | **Topik:** | **Tanggal:** |
| **Fikri irwansyah**  **G1F024073** | **Operator Aritmatika: +, -, \*, /** | **06-09-2024** |
| **[1.1,1.2,1.3] Identifikasi Masalah:** | | |
| * 1. Tambahkan baris System.out.println("a + b = "  + (a + b));  Ubahlah operator ( + ) dengan tanda ( -, \*, /, %)   2. Analisa perhitungan matematika yang terjadi! | | |
| **Penyusunan Algoritma dan Kode Program** | | |
| 1. public class OperatorAritmatika{  public static void main(String[] args) {  // deklarasi nilai  int a = 20, b = 3;  //operator aritmatika  System.out.println("a: " +a);  System.out.println("b: " +b);  System.out.println("a + b = " + (a - b));  System.out.println("a + b = " + (a - b));  } }  Luaran  a: 20  b: 3  a + b = 17  a + b = 17  === Code Execution Successful ===  Hasil yang terjadi adalah tanda (+) untuk pertambahan, tanda (-) untuk pengurangan, tanda (\*) untuk perkalian, tanda (/) untuk pembagian, dan tanda (%) untuk sisa | | |
| **Kesimpulan** | | |
| Kesimpulannya adalah operator aritmatika digunakan untuk melakukan perhitungan matematika. | | |
| **Refleksi**   **Akurasi dalam Labeling**: Penting untuk memastikan label output sesuai dengan operasi yang dilakukan untuk mencegah kebingungan.   **Perhatikan Operator**: Selalu periksa operator yang digunakan dalam operasi aritmatika untuk memastikan hasil sesuai dengan harapan. | | |
|  | | |

**Tugas 2**

|  |  |  |
| --- | --- | --- |
| **Nama & NPM** | **Topik:** | **Tanggal:** |
| Fikri irwansyah  G1F024073 | **Operator Penugasan  ( = )** | **06-09-2024** |
| **[2.1,2.2] Identifikasi Masalah:** | | |
| 2.1 Bandingkan hasil Contoh 1 dengan Contoh 2!  public class OperatorAritmatika { public static void main(String[] args) { // deklarasi nilai int a = 20, b = 3; // operator aritmatika System.out.println("a: " + a); System.out.println("b: " + b); System.out.println("a + b = " + (a - b)); } }  public class OperatorPenugasan { public static void main(String[] args) { // deklarasi nilai int a = 20, b = 3; // operator penugasan b += a; // b = b + a System.out.println("Penambahan : " + b); // 3 + 20 = 23 b -= a; // b = b - a System.out.println("Pengurangan : " + b); // 23 - 20 = 3 b \*= a; // b = b \* a System.out.println("Perkalian : " + b); // 3 \* 20 = 60 b /= a; // b = b / a System.out.println("Pembagian : " + b); // 60 / 20 = 3 b %= a; // b = b % a System.out.println("Sisa Bagi: " + b); // 3 % 20 = 3 } } | | |
| **Penyusunan Algoritma dan Kode Program** | | |
| 2.1  Pada kode pertama, hanya operator aritmatika (+, -) yang digunakan untuk melakukan  operasi, dan nilainya tidak dimodifikasi.   Pada kode kedua, operator penugasan (+=, -=, \*=, dll.) digunakan untuk melakukan operasi  sekaligus memperbarui nilai variabel yang terlibat.   Pada kode pertama, nilai variabel tidak diubah setelah operasi (variabel a dan b tetap  sama).   Pada kode kedua, nilai variabel b diubah pada setiap operasi.   Kode pertama lebih fokus pada operasi aritmatika sederhana tanpa mempengaruhi  nilai variabel.   Kode kedua memanfaatkan operator penugasan untuk memodifikasi variabel secara  dinamis dan menghemat kode. | | |
| **Kesimpulan** | | |
| operator aritmatika lebih cocok digunakan ketika kita hanya ingin menghitung sesuatu tanpa mengubah nilai variabel. Sebaliknya, operator penugasan lebih efisien digunakan saat kita ingin melakukan operasi dan sekaligus memperbarui variabel. | | |
| **Refleksi**  penugasan ini mengajarkan kita tentang pentingnya memilih pendekatan yang tepat tergantung pada kebutuhan spesifik program yang sedang dikembangkan. Selain itu, ini juga menekankan pentingnya pemahaman dasar konsep matematika, karena mereka adalah fondasi dari banyak logika pemrograman. | | |
|  | | |
|  | | |

**Tugas 3**

|  |  |  |
| --- | --- | --- |
| **Nama & NPM** | **Topik:** | **Tanggal:** |
| **FIKRI IRWANSYAH**  **G1F024073** | **Operator Relasional:   <, >, <=, >=, =, ==, !=** | **06-09-2024** |
| **[2.1,2.2] Identifikasi Masalah:** | | |
| 3.1.  Ubahlah nilai A = 4 dan B = 4. Analisa perubahan yang terjadi!  3.2   Bandingkan bagaimana perbedaan nilai A dan B mempengaruhi nilai luaran! | | |
| **Penyusunan Algoritma dan Kode Program** | | |
| 3.1 public class OperatorRealasional {     public static void main(String[] args) {         int nilaiA = 4;         int nilaiB = 4;         boolean hasil;          System.out.println(" A = " + nilaiA + "\n B = " + nilaiB);         // apakah A lebih besar dari B?         hasil = nilaiA > nilaiB;         System.out.println("Hasil A > B = "+ hasil);          // apakah A lebih kecil dari B?         hasil = nilaiA < nilaiB;         System.out.println("Hasil A < B = "+ hasil);          // apakah A lebih besar samadengan B?         hasil = nilaiA >= nilaiB;         System.out.println("Hasil A >= B = "+ hasil);          // apakah A lebih kecil samadengan B?         hasil = nilaiA <= nilaiB;         System.out.println("Hasil A <= B = "+ hasil);          // apakah nilai A sama dengan B?         hasil = nilaiA == nilaiB;         System.out.println("Hasil A == B = "+ hasil);          // apakah nilai A tidak samadengan B?         hasil = nilaiA != nilaiB;         System.out.println("Hasil A != B = "+ hasil);     } }  Luaran ;  A = 4  B = 4  Hasil A > B = false  Hasil A < B = false  Hasil A >= B = true  Hasil A <= B = true  Hasil A == B = true  Hasil A != B = false  === Code Execution Successful ===  3.2 Ketika nilai **A** dan **B** sama, semua perbandingan "lebih besar" (>) dan "lebih kecil" (<) akan menghasilkan **false**, karena tidak ada perbedaan antara nilai A dan B. Sebaliknya, perbandingan yang melibatkan kesamaan (>=, <=, ==) akan menghasilkan **true**, karena kedua nilai sama. | | |
| **Kesimpulan** | | |
| Kesimpulannya perubahan ini menunjukkan bahwa ketika nilai dari dua variabel sama, hasil dari operasi relasional yang melibatkan perbandingan "lebih besar" dan "lebih kecil" akan selalu **false**, sementara operasi yang melibatkan kesamaan akan selalu **true**. | | |
| **Refleksi**  Mempelajari Secara umum, memahami bagaimana operator relasional bekerja dengan nilai yang sama dan berbeda akan membantu kita menulis program yang lebih kuat, jelas, dan bebas dari kesalahan logika. | | |
|  | | |
|  | | |

**Tugas 4**

|  |  |  |
| --- | --- | --- |
| **Nama & NPM** | **Topik:** | **Tanggal:** |
| **Fikri irwansyah**  **G1F024073** | **Operator Increment dan Decrement: ++, --** | **06-09-2024** |
| **[4.1] Identifikasi Masalah:** | | |
| 4.1.  Berdasarkan luaran program Contoh 4, bandingkan hasil Post dan Pre untuk Increment dan Decrement! | | |
| **Penyusunan Algoritma dan Kode Program** | | |
| 4.1   Post-Increment/Decrement (a++/a--): Operasi increment atau decrement dilakukan setelah nilai  digunakan atau ditampilkan. Karena itu, nilai yang ditampilkan pada kali pertama akan tetap  sama dengan nilai awal.   Pre-Increment/Decrement (++b/--b): Operasi increment atau decrement dilakukan sebelum nilai  digunakan atau ditampilkan. Ini berarti nilai yang ditampilkan setelah operator akan  mencerminkan hasil perubahan. | | |
| **Kesimpulan** | | |
| Kesimpulannya adalah Post digunakan ketika kita ingin nilai asli diproses terlebih dahulu sebelum perubahan, sedangkan pre digunakan saat perubahan harus terjadi sebelum nilai digunakan. | | |
| **Refleksi**  Pemahaman tentang perbedaan post dan pre increment/decrement penting dalam pemrograman, terutama saat mengelola alur logika dan perhitungan yang kompleks. Penggunaan yang tepat akan membantu mencegah kesalahan logika dan menghasilkan kode yang lebih efisien dan dapat diprediksi. | | |
|  | | |
|  | | |

**Tugas 5**

|  |  |  |
| --- | --- | --- |
| **Nama & NPM** | **Topik:** | **Tanggal:** |
| **Fikri irwansyah**  **G1F024073** | **Operator Logika:  &&, ||, !** | **06-09-2024** |
| **[4.1] Identifikasi Masalah:** | | |
| 5.1.  Tambahkan baris kode untuk memeriksa a || b. 5.2.  Ubahlah nilai a = false dan b = false. Analisa perubahan dan perbedaan boolean yang terjadi! 5.2.  Apabila diketahui pernyataan a || b && a || !b. Uraikan urutan logika yang akan dikerjakan!  Analisa luaran true atau false dari pernyataan tersebut! | | |
| **Penyusunan Algoritma dan Kode Program** | | |
| 5.1.  public class OperatorLogika {  public static void main (String [] args) {  boolean a = true;  boolean b = false;  boolean c;    c = a && b;  System.out.println("true && false = " + c);    c = a || b;  System.out.println("true || false = " + c);  }  }  5.2 public class OperatorLogika {  public static void main (String [] args) {  boolean a = false;  boolean b = false;  boolean c;    c = a && b;  System.out.println("false && false = " + c);    c = a || b;  System.out.println("false || false = " + c);  }  }  Luaran :  false && false = false  false || false = false  === Code Execution Successful ===  AND (&&) menghasilkan true hanya jika kedua operand true dan OR (||) menghasilkan false hanya jika kedua operand false.  5.3  Operator Negasi (!) : Negasi memiliki prioritas tertinggi, sehingga akan dihitung terlebih dahulu.  Operator AND (&&): Operator AND memiliki prioritas lebih tinggi daripada OR (||), sehingga  dihitung setelah negasi tetapi sebelum OR.  Operator OR (||) : Operator OR dihitung terakhir.  Urutan operasi adalah ; !b, lalu b && a, terakhir || (b && a) || !b.  Dengan nilai a = true dan b = false, pernyataan a || b && a || !b menghasilkan **true**. | | |
| **Kesimpulan** | | |
| Kesimpulannya adalah Negasi (!) Mengubah nilai boolean menjadi kebalikannya, AND (&&) Menghasilkan true hanya jika kedua operand adalah true, dan OR (||) Menghasilkan true jika salah satu operand adalah true. | | |
| **Refleksi**  Memahami urutan operasi logika sangat penting dalam pemrograman untuk memastikan bahwa ekspresi dievaluasi dengan benar sesuai yang diinginkan. | | |
|  | | |
|  | | |

**Tugas 6**

|  |  |  |
| --- | --- | --- |
| **Nama & NPM** | **Topik:** | **Tanggal:** |
| **Fikri irwansyah**  **G1F024073** | **Operator Kondisional (Ternary):  ?:** | **06-09-2024** |
| **[4.1] Identifikasi Masalah:** | | |
| 6.1.  Berdasarkan Contoh 6, ubahlah nilai = 60. Analisis hasil dan proses yang terjadi! | | |
| **Penyusunan Algoritma dan Kode Program** | | |
| 6.1 public class OperatorKondisi {  public static void main(String[] args) {  String status = "";  int nilai = 60;  status = (nilai > 60) ? "Lulus" : "Gagal";  System.out.println(status);  }  }  Luaran :  Gagal  === Code Execution Successful ===  Output menjadi gagal karena status lulus terjadi jika nilainya >60 | | |
| **Kesimpulan** | | |
| Kesimpulannya Ketika nilai = 60, output dari pernyataan kondisional (nilai > 60) ? "Lulus" : "Gagal" adalah "Gagal". | | |
| **Refleksi**  Memahami bagaimana kondisi dievaluasi sangat penting dalam menentukan hasil dari pernyataan kondisional. | | |
|  | | |
|  | | |

**Tugas 7**

|  |  |  |
| --- | --- | --- |
| **Nama & NPM** | **Topik:** | **Tanggal:** |
| **Fikri irwansyah**  **G1F024073** | **Operator Bitwise: &, |, ^, ~, <<, >>, >>>** | **06-09-2024** |
| **[4.1] Identifikasi Masalah:** | | |
| 7.1.  Pilihlah 3 perhitungan Contoh 7, kemudian uraikan perhitungan biner! Simpulkan hasilnya! | | |
| **Penyusunan Algoritma dan Kode Program** | | |
| 7.1  1. Bitwise AND (a & b)  Operasi: 10 & 7  Representasi Biner:  a (10) = 00001010  b (7) = 00000111  Operasi Bitwise AND:  00001010  00000111  AND: 00000010 (Hanya bit yang sama-sama 1 yang akan tetap 1)  Hasil:  00000010 = 2 dalam desimal.  2. Bitwise OR (a | b)  Operasi: 10 | 7  Representasi Biner:  a (10) = 00001010  b (7) = 00000111  Operasi Bitwise OR:  00001010  00000111  OR: 00001111 (Bit yang ada di salah satu atau kedua operand 1, maka hasilnya 1)  Hasil:  00001111 = 15 dalam desimal.  3. Bitwise XOR (a ^ b)  Operasi: 10 ^ 7  Representasi Biner:  a (10) = 00001010  b (7) = 00000111  Operasi Bitwise XOR:  00001010  00000111  XOR: 00001101 (Bit yang berbeda antara operand akan menjadi 1)  Hasil:  00001101 = 13 dalam desimal. | | |
| **Kesimpulan** | | |
| Kesimpulannya operasi bitwise sangat berguna untuk manipulasi bit-level dalam pemrograman dan membantu dalam berbagai aplikasi seperti enkripsi, pengaturan bit flag, dan operasi logika kompleks. | | |
| **Refleksi**  Dengan memahami dan menguasai operasi ini, Anda dapat menulis kode yang lebih efisien dan memahami lebih baik bagaimana data diproses pada level terendah.diprediksi. | | |
|  | | |
|  | | |