Tugas 1

Nama & NPM	**Topik:**	**Tanggal:**
Abdullah Hasyim Syauqi G1F024019	**Operator Aritmatika: +, -, *, /**	**05-09-2024**

[1.1,1.2,1.3] Identifikasi Masalah:

1.1 Tambahkan baris System.out.println("a + b = " + (a + b)); Ubahlah operator (+) dengan tanda (-, *, /, %)

1.2 Analisa perhitungan matematika yang terjadi!

Penyusunan Algoritma dan Kode Program

1.1
```
public class TugasHasyim{
   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));
  System.out.println("a * b = " + (a * b));
  System.out.println("a / b = " + (a / b));
  System.out.println("a % b = " + (a % b));
}   }
```

1.2 Hasil yang terjadi (Luaran) :

```
a: 20
b: 3
a + b = 23
a - b = 17
a * b = 60
a / b = 6
a % b = 2
```

Hasil yang terjadi adalah tanda (+) untuk pertambahan, tanda (-) untuk pengurangan, tanda (*) untuk perkalian, tanda (/) untuk pembagian, dan tanda (%) untuk sisa

Kesimpulan

Kesimpulannya adalah dengan menggunakan operator seperti penjumlahan, pengurangan, perkalian, pembagian, dan sisa bagi, kita dapat memahami bagaimana program memproses nilai-nilai numerik dan menghasilkan hasil yang sesuai berdasarkan operasi yang diterapkan.

Refleksi

Dengan memahami dasar-dasar operasi ini, kita dapat lebih baik memahami dan mengaplikasikan konsep matematika dalam berbagai konteks, termasuk dalam analisis data, pemrograman, dan pengembangan teknologi.

Tugas 2

Nama & NPM	**Topik:**	**Tanggal:**
Abdullah Hasyim Syauqi G1F024019	**Operator Penugasan (=)**	**05-09-2024**

[2.1,2.2] Identifikasi Masalah:

2.1 Bandingkan hasil Contoh 1 dengan Contoh 2!

Penyusunan Algoritma dan Kode Program

2.1

1. Pada kode pertama, hanya operator aritmatika (+, -) yang digunakan untuk melakukan operasi, dan nilainya tidak dimodifikasi.
2. Pada kode kedua, operator penugasan (+=, -=, *=, dll.) digunakan untuk melakukan operasi sekaligus memperbarui nilai variabel yang terlibat.
3. Pada kode pertama, nilai variabel tidak diubah setelah operasi (variabel `a` dan `b` tetap sama).
4. Pada kode kedua, nilai variabel `b` diubah pada setiap operasi.
5. Kode pertama lebih fokus pada operasi aritmatika sederhana tanpa mempengaruhi nilai variabel.
6. Kode kedua memanfaatkan operator penugasan untuk memodifikasi variabel secara dinamis dan menghemat kode.

Kesimpulan

Kesimpulannya adalah operator penugasan lebih efisien digunakan ketika kita ingin melakukan operasi matematika sekaligus memperbarui nilai variabel. Sebaliknya, operator aritmatika lebih cocok digunakan saat kita hanya perlu menghitung hasil tanpa mengubah nilai variabel yang ada. Dan dengan memahami perbedaan ini, kita dapat memilih operator yang paling sesuai untuk kebutuhan spesifik dalam pemrograman.

Refleksi

penugasan ini mengajarkan kita tentang pentingnya memilih kode yang tepat tergantung pada kebutuhan spesifik program yang sedang dikembangkan dan menekankan pentingnya pemahaman dasar konsep matematika.

Tugas 3

Nama & NPM	Topik:	Tanggal:
Abdullah Hasyim Syauqi G1F024019	**Operator Relasional: <, >, <=, >=, =, ==, !=**	**05-09-2024**

[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 TugasHasyim {
   public static void main(String[] args) {
      int nilaiA = 4;
      int nilaiB = 4;
      boolean hasil;
      System.out.println(" A = " + nilaiA + "\n B = " + nilaiB);

      hasil = nilaiA > nilaiB;
      System.out.println("Hasil A > B = "+ hasil);

      hasil = nilaiA < nilaiB;
      System.out.println("Hasil A < B = "+ hasil);

      hasil = nilaiA >= nilaiB;
      System.out.println("Hasil A >= B = "+ hasil);

      hasil = nilaiA <= nilaiB;
      System.out.println("Hasil A <= B = "+ hasil);

      hasil = nilaiA == nilaiB;
      System.out.println("Hasil A == B = "+ hasil);

      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

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 mengungkapkan bahwa ketika dua variabel memiliki nilai yang sama, hasil dari operasi relasional yang membandingkan "lebih besar" dan "lebih kecil" akan selalu false, sedangkan operasi yang memeriksa kesamaan akan selalu true. Ini menunjukkan bagaimana hasil dari perbandingan relasional bergantung pada nilai-nilai yang terlibat.
Refleksi Kita 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:**
Abdullah Hasyim Syauqi G1F024019	**Operator Increment dan Decrement: ++, --**	**05-09-2024**

[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, operator post digunakan ketika kita ingin memproses nilai asli terlebih dahulu sebelum melakukan perubahan, sementara operator pre digunakan ketika perubahan harus diterapkan sebelum nilai digunakan. Dengan memahami perbedaan ini, kita dapat mengelola urutan operasi dalam pemrograman secara lebih efektif.

Refleksi

Memahami perbedaan antara post dan pre increment/decrement sangat penting dalam pemrograman, terutama ketika mengelola alur logika dan perhitungan yang kompleks. Penggunaan yang tepat dari operator-operator ini dapat mencegah kesalahan logika dan menghasilkan kode yang lebih efisien serta dapat diprediksi.

Tugas 5

Nama & NPM	Topik:	Tanggal:
Abdullah Hasyim Syauqi G1F024019	**Operator Logika: &&, ||, !**	**05-09-2024**

[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 TugasHasyim {
   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

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 dengan yang diinginkan. Dengan mengetahui urutan evaluasi ini, kita dapat menghindari kesalahan dan memastikan hasil yang akurat dalam logika program.

Tugas 6

Nama & NPM	**Topik:**	**Tanggal:**
Abdullah Hasyim Syauqi G1F024019	**Operator Kondisional (Ternary): ?:**	**05-09-2024**

[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 TugasHasyim {
    public static void main(String[] args) {
        String status = "";
        int nilai = 60;
        status = (nilai > 60) ? "Lulus" : "Gagal";
        System.out.println(status);
    }
}
```

Luaran :
Gagal

Output menjadi gagal karena status lulus terjadi jika nilainya >60

Kesimpulan

Kesimpulannya, ketika nilai adalah 60, output dari pernyataan kondisional (nilai > 60) ? "Lulus" : "Gagal" adalah "Gagal".

Refleksi

Memahami bagaimana kondisi dievaluasi sangat penting dalam menentukan hasil dari pernyataan kondisional. Evaluasi kondisi yang tepat memastikan bahwa hasil yang dihasilkan sesuai dengan logika yang diinginkan dalam program.

Tugas 7

Nama & NPM	Topik:	Tanggal:
Abdullah Hasyim Syauqi G1F024019	**Operator Bitwise: &, |, ^, ~, <<, >>, >>>**	**05-09-2024**

[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. Mereka memainkan peran penting dalam berbagai aplikasi, seperti enkripsi, pengaturan bit flag, dan operasi logika kompleks, memungkinkan kontrol yang lebih presisi dan efisien terhadap data biner.

Refleksi Dengan memahami dan menguasai operasi bitwise, kita dapat menulis kode yang lebih efisien dan mendapatkan pemahaman yang lebih baik tentang bagaimana data diproses pada tingkat paling dasar.