Ekspresi dan Masukan/Keluaran

Tim Olimpiade Komputer Indonesia

Pendahuluan

Melalui dokumen ini, kalian akan:

- Mengenal ekspresi.
- Mengenal masukan dan keluaran untuk program.

Kilas Balik: Assignment

- Membosankan sekali jika kita hanya bisa mengisi variabel dengan nilai yang pasti.
- Kadang-kadang dibutuhkan hal yang lebih ekspresif. Contoh:

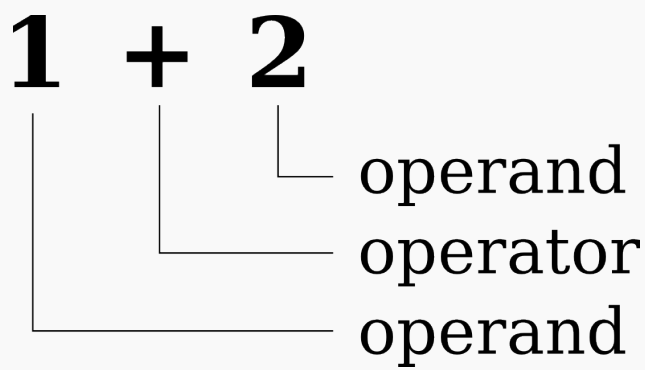
```
a := 5;
b := 2;
jumlah := a + b;
```

- Kenyataannya, hal ini dapat diwujudkan pada pemrograman!
- Perintah "a + b" biasa disebut sebagai ekspresi.

Bagian 1

Ekspresi

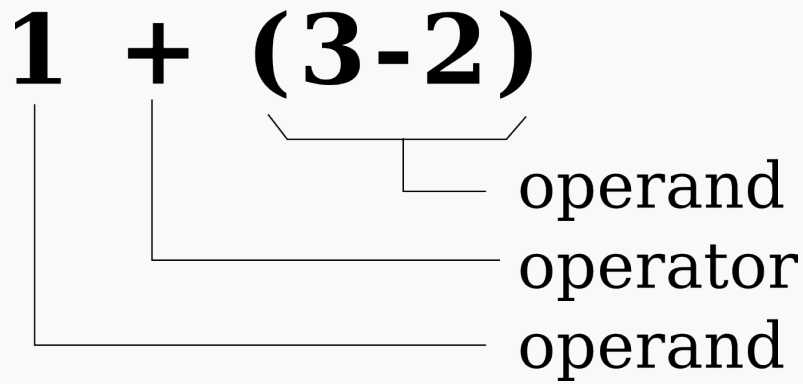
Mengenal Ekspresi



- Ekspresi terdiri dari dua komponen: operator dan operand.
- Operand menyatakan nilai yang akan dioperasikan, misalnya bilangan atau suatu ekspresi lagi.
- Operator menyatakan bagaimana operand akan dioperasikan, apakah ditambah, dikali, atau dibagi?

Mengenal Ekspresi (lanj.)

Bisa juga dibentuk ekspresi bersarang, yaitu ekspresi yang operand-nya merupakan ekspresi lagi:



Operasi Numerik

- Operasi pada bilangan yang dapat dilakukan adalah penjumlahan (+), pengurangan (-), perkalian (*), pembagian (/), pembagian *integer* (div), dan modulo (mod).
- Jika kedua operand merupakan bilangan bulat, hasil pengoperasian selalu bilangan bulat juga, kecuali untuk operasi pembagian yang selalu menghasilkan *floating point*. Contoh:
  - 3 - 1 = 2
  - 10 / 5 = 2.0000000
  - 7 / 2 = 3.5000000
- Ketika setidaknya salah satu dari operand ada yang bertipe data *floating point*, pengoperasian akan selalu menghasilkan *floating point*.

Operasi Numerik (lanj.)

- Operasi pembagian *integer* didefinisikan sebagai: membagi, lalu dibulatkan ke bawah. Contoh:
  - $7 \text{ div } 2 = 3$
  - $10 \text{ div } 2 = 5$
  - $3 \text{ div } 5 = 0$
- Operasi modulo adalah mengambil sisa bagi dari operand pertama terhadap operand kedua. Contoh:
  - $7 \text{ mod } 2 = 1$
  - $10 \text{ mod } 2 = 0$
  - $3 \text{ mod } 5 = 3$
  - $8 \text{ mod } 3 = 2$
- Operasi div dan mod hanya bisa dilakukan apabila kedua operand memiliki tipe data bilangan bulat.

Contoh Program: kuadrat.pas

- Setelah memahami tentang operasi numerik, coba perhatikan program berikut dan cari tahu apa keluarannya!

```
var
  a, b, c, x, hasil: longint;
begin
  a := 1;
  b := 3;
  c := -2;
  x := 2;

  hasil := a*x*x + b*x + c;
  writeln('ax^2 + bx + c = ', hasil);
end.
```

Prioritas Pengerjaan

- Seperti pada ilmu matematika, ada juga prioritas pengerjaan pada ekspresi numerik. Tabel berikut menunjukkan prioritasnya:

Prioritas	Operasi
1	*,/,div,mod
2	+,-

- Jika ada beberapa operasi bersebelahan yang memiliki prioritas sama, operasi yang terletak di posisi lebih kiri akan dikerjakan lebih dahulu.

Contoh Program: numerik.pas

- Kita juga bisa menggunakan tanda kurung untuk mengatur prioritas pengerjaan suatu ekspresi.
- Perhatikan contoh berikut dan coba jalankan programnya:

```
var
  hasil1, hasil2: longint;
begin
  hasil1 := 3+5 div 4;
  hasil2 := (3+5) div 4;
  writeln(hasil1);
  writeln(hasil2);
end.
```

- Isi dari variabel hasil1 adalah 4, karena operasi "5 div 4" memiliki prioritas yang lebih tinggi untuk dikerjakan, dan menghasilkan nilai 1. Barulah "3 + 1" dilaksanakan kemudian.

Fungsi Dasar Numerik

Berikut fungsi dasar yang disediakan Pascal untuk membantu perhitungan:

- **trunc**: mengambil bagian depan penanda desimal dari suatu bilangan pecahan. Contoh: **trunc(3.14)** akan menghasilkan 3 (sebuah *integer*).
- **frac**: mengambil bagian belakang penanda desimal dari suatu bilangan pecahan. Contoh: **frac(3.14)** akan menghasilkan 0.14.

Fungsi Dasar Numerik (lanj.)

- **round**: membulatkan suatu bilangan pecahan bilangan bulat terdekat (hasilnya adalah bilangan bertipe *integer*). Contoh: **round(1.2)** akan menghasilkan 1, sementara **round(1.87)** akan menghasilkan 2.
- **sqr**: mengkuadratkan suatu bilangan. Contoh: **sqr(3)** akan menghasilkan 9.
- **sqrt**: mendapatkan akar kuadrat dari suatu bilangan. Contoh: **sqrt(9)** akan menghasilkan 3.00, dan **sqrt(3)** akan menghasilkan 1.73205....

Fungsi Dasar Numerik (lanj.)

Catatan khusus untuk fungsi **round**

Jika angka yang diberikan memiliki pecahan tepat setengah, pembulatan ditentukan dari angka sebelum penanda desimal. Jika ganjil, dilakukan pembulatan ke atas. Jika genap, dilakukan pembulatan ke bawah.
Contoh:

- **round(1.5)** $= 2$
- **round(2.5)** $= 2$
- **round(3.5)** $= 4$

Contoh Program: kuadrat2.pas

- Program kuadrat1.pas bisa ditulis juga dengan menggunakan fungsi **sqr**, sehingga menjadi:

```
var
  a, b, c, x, hasil: longint;
begin
  a := 1;
  b := 3;
  c := -2;
  x := 2;

  hasil := a*sqr(x) + b*x + c;
  writeln('ax^2 + bx + c = ', hasil);
end.
```

Operasi Relasional

- Kita juga bisa melakukan operasi relasional, yaitu:
  - kurang dari ($<$)
  - lebih dari ($>$)
  - sama dengan ($=$)
  - kurang dari atau sama dengan ($<=$)
  - lebih dari atau sama dengan ($>=$)
  - tidak sama dengan ($<>$)
- Operasi relasional harus melibatkan dua operand (ingat bahwa operand bisa jadi berupa ekspresi lagi), dan menghasilkan sebuah nilai kebenaran.
- Pada Pascal, nilai kebenaran dinyatakan dengan tipe data **boolean**.

Contoh Program: relasional.pas

- Perhatikan contoh berikut dan coba jalankan programnya:

```
begin
  writeln(2 > 1);
  writeln(2 < 1);
  writeln(2 = 1);
  writeln(2 >= 1);
  writeln(1 = 1);
  writeln(1 <> 1);
  writeln(1 <> 2);
end.
```

Operasi Relasional pada Floating Point

- Karena komputer tidak dapat secara sempurna menyimpan nilai *floating point*, Anda perlu hati-hati saat membandingkan dua bilangan riil.
- Ekspresi berikut mungkin saja bernilai **FALSE**:

```
(0.1 + 0.2) = 0.3
```

- Sebab 0.1 + 0.2 bisa saja bernilai **0.30000000000000001**

Operasi Relasional pada Floating Point (lanj.)

- Untuk memeriksa kesamaan antara dua nilai *floating point*, biasanya dilibatkan suatu nilai toleransi.
- Misalnya, kedua nilai dianggap sama apabila selisih mereka kurang dari 10^{-8}.

Operasi Relasional (lanj.)

- Operasi relasional dapat dilakukan pada setiap tipe data ordinal, sehingga bisa juga diterapkan pada **char**.
- Perbandingan karakter dilakukan dengan membandingkan kode ASCII mereka, sehingga menjadi seperti membandingkan angka biasa.
- Contoh:
  - 'a' $<$ 'b' akan bernilai **TRUE**
  - 'a' $>$ 'z' bernilai **FALSE**
  - 'A' $<$ 'a' akan bernilai **TRUE**

Operasi Relasional (string)

- Lebih jauh lagi, **string** sebenarnya merupakan untaian **char**. Operasi relasional juga bisa diterapkan pada **string** (meskipun **string** bukan tipe data ordinal).
- Pascal akan membandingkan karakter demi karakter dari kiri ke kanan. Begitu ditemukan ada perbedaan karakter, lebih kecil atau tidaknya suatu string ditentukan oleh karakter tersebut.
  - Contohnya, 'aa' $<$ 'ab' akan bernilai **TRUE**.
- Jika sampai salah satu string habis dan tidak ditemukan ada perbedaan karakter, maka stirng yang lebih pendek dianggap lebih kecil.
  - Contohnya 'a' $<$ 'aa' bernilai **TRUE**.

Contoh Program: relasional2.pas

- Perhatikan contoh berikut dan coba jalankan programnya:

```
begin
  writeln('a' > 'A');
  writeln('a' < 'A');
  writeln('a' >= 'A');
  writeln('a' = 'A');

  writeln('a' < 'aa');
  writeln('abcb' > 'abca');
  writeln('abc' = 'abc');
  writeln('abc' <= 'abc');
end.
```

Operasi Boolean

- Operasi **boolean** merupakan operasi yang hanya melibatkan nilai-nilai kebenaran. Terdiri atas: **not**, **and**, **or**, **xor**.
- Operasi-operasi ini sesuai dengan sebuah cabang ilmu matematika yang bernama "aljabar boolean".
- Operasi not merupakan operasi *unary*, artinya hanya melibatkan satu operand. Gunanya untuk membalik nilai kebenaran.
- Tabel berikut menunjukkan efek dari penggunaan **not**.

a	not a
TRUE	FALSE
FALSE	TRUE

Operasi Boolean (lanj.)

- Operasi **boolean** yang lainnya merupakan operasi *binary*, yang artinya melibatkan dua operand.
- Tabel berikut menunjukkan efek dari penggunaan operator-operator tersebut:

a	b	a and b	a or b	a xor b
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE

Operasi Boolean (lanj.)

- Prioritas pengerjaan dari operator **boolean** secara berurutan adalah: **not**, **and**, **or**, **xor**.
- Tanda kurung juga bisa digunakan untuk menentukan operasi mana yang perlu dijalankan terlebih dahulu. Bahkan sangat disarankan untuk selalu menggunakan tanda kurung untuk kejelasan.

Contoh Program: relasional3.pas

- Perhatikan contoh berikut dan coba jalankan programnya:

```
begin
  writeln(2 > 1);
  writeln(not (2 > 1));
  writeln((2 > 1) and (3 > 1));
  writeln(((2 > 1) or (3 < 1)) and (1 = 1));
  writeln((1 <> 1) xor not (1 <> 1));
end.
```

- Perhatikan bahwa tanda kurung diperlukan dalam ekspresi "not $(2 > 1)$". Dengan tanda kurung, "$2 > 1$" akan dievaluasi terlebih dahulu, menghasilkan nilai **boolean**. Barulah operator **not** bisa mengolah nilai **boolean** tersebut.

Bagian 2

Masukan dan Keluaran

Kilas Balik: kuadrat2.pas

- Sekarang coba lihat kembali program kuadrat2.pas:

```
var
  a, b, c, x, hasil: longint;
begin
  a := 1;
  b := 3;
  c := -2;
  x := 2;

  hasil := a*sqr(x) + b*x + c;
  writeln('ax^2 + bx + c = ', hasil);
end.
```

- Jika kita ingin mengganti nilai **x**, kode harus diganti, dikompilasi ulang, baru dijalankan kembali.
- Untuk menghasilkan keluaran yang bervariasi, perlu ada masukan dari luar program.

Membaca Masukan

- Diperlukan mekanisme untuk melakukan pembacaan masukan dari luar program.
- Masukan bagi suatu program bisa berasal dari berbagai sumber, misalnya *standard input* atau *file*.
- Pada Pascal, dikenal dua fungsi yang umum untuk membaca masukan: **read** dan **readln**.

Membaca Masukan: readln

- Lakukan modifikasi pada bagian **x := 2** menjadi **readln(x)**:

```
var
  a, b, c, x, hasil: longint;
begin
  a := 1;
  b := 3;
  c := -2;
  readln(x);

  hasil := a*sqr(x) + b*x + c;
  writeln('ax^2 + bx + c = ', hasil);
end.
```

- Kompilasi, dan jalankan program. Kemudian ketikkan angka 2, dan tekan enter.
- Selamat! Kalian berhasil membaca masukan!

Fungsi readln

- Fungsi **readln** berguna untuk membaca masukan, dan nilainya dapat di-*assign* ke dalam variabel.
- Cara kerja readln: pada berkas masukan, cari *token* yang dapat dibaca berikutnya, lalu baca ambil nilainya.
- Yang dimaksud *token* adalah serangkaian karakter non-spasi, misalnya huruf atau angka.
- Pada contoh sebelumnya, *token* yang dimaksud adalah bilangan yang akan menjadi nilai variabel **x**.

Membaca Masukan: read

- Sekarang ganti **readln(x)** menjadi **read(x)**:

```
var
  a, b, c, x, hasil: longint;
begin
  a := 1;
  b := 3;
  c := -2;
  read(x);

  hasil := a*sqr(x) + b*x + c;
  writeln('ax^2 + bx + c = ', hasil);
end.
```

- Kompilasi, dan jalankan program. Kemudian ketikkan angka 2, dan tekan enter.
- Hasilnya tetap sama. Lalu apa yang membedakan **readln** dengan **read**?

Perbedaan readln dengan read

Setelah **read** atau **readln** mencari *token* berikutnya dan membacanya, yang terjadi selanjutnya adalah:

readln

Program disiapkan untuk membaca masukan selanjutnya di baris berikutnya.

read

Program tidak disiapkan untuk membaca masukan selanjutnya di baris berikutnya.

Perbedaan readln dengan read (lanj.)

- Artinya, jika kita ingin membaca masukan dengan format:
  - Baris pertama berisi tiga bilangan bulat dipisahkan dengan sebuah spasi, yaitu a, b, dan c.
  - Baris kedua berisi sebuah bilangan bulat, yaitu x.
- Dapat digunakan:

```
read(a);
read(b);
readln(c);
readln(x);
```

Perbedaan readln dengan read (lanj.)

- Boleh juga:

```
read(a);
read(b);
read(c);
read(x);
```

- Ingat bahwa baik **read** maupun **readln** akan mencari *token* berikutnya untuk dibaca, tanpa peduli di baris mana *token* itu berada.

Perbedaan readln dengan read (lanj.)

- Namun hal ini tidak bisa dilakukan:

```
readln(a);
readln(b);
readln(c);
readln(x);
```

- Sebab pada **readln**, setelah membaca *token* pada suatu baris, sisa *token* yang ada pada baris yang sama diabaikan dan langsung berpindah ke baris berikutnya.
- Akibatnya, *token* yang seharusnya menjadi masukan bagi variabel **b** dan **c** terabaikan.

Perbedaan readln dengan read (lanj.)

- Sebagai alternatif, **readln** dan **read** juga bisa digunakan untuk membaca beberapa masukan pada suatu baris secara bersamaan.
- Contoh penggunaan yang benar:

```
readln(a,b,c);
readln(x);
```

atau

```
read(a,b);
readln(c);
readln(x);
```

Kegunaan Lain readln

- Fungsi **readln** dapat digunakan untuk memaksa program untuk berpindah baris dalam membaca masukan.
- Caranya dengan mengosongkan variabel di dalam pemanggilan fungsi **readln**.
- Pada contoh yang sebelumnya, boleh juga digunakan cara membaca seperti ini:

```
read(a,b,c);
readln;
readln(x);
```

Membaca string

- Khusus untuk membaca *string*, baik **read** maupun **readln** akan membaca satu baris penuh.
- Perhatikan program berikut:

```
var
  s: string;
begin
  readln(s);
  writeln('s = ', s);
end.
```

Membaca string (lanj.)

- Jika program tersebut diberi masukan:

```
bebek 123
```

- maka keluaran dari program tersebut adalah:

```
s = bebek 123
```

- Dari contoh ini, diketahui bahwa variabel **s** berisi "bebek 123".

Membaca string (lanj.)

- Bagaimana jika kita ingin **s** hanya berisi "bebek", sementara 123 disimpan pada variabel lainnya?
- Apakah hal berikut bisa dilakukan?

```
var
  s: string;
  nilai: longint;
begin
  readln(s, nilai);
  writeln('s = ', s, ' nilai = ', nilai);
end.
```

- Sayangnya tidak bisa. Variabel **s** langsung berisi seluruh karakter yang ada pada baris input, sehingga variabel **nilai** tidak menerima masukan apa-apa.
- Penanganan kasus sejenis ini dipelajari pada materi yang akan datang.

Mencetak Keluaran

- Seperti masukan, keluaran juga bisa disajikan dalam bentuk langsung ke *standard output* atau ke *file*.
- Pada Pascal, fungsi untuk mencetak keluaran yang umum adalah **writeln** dan **write**.
- Sejauh ini, kita sudah menggunakan **writeln**.

Perbedaan writeln dengan write

writeln

Seperti yang sudah kita gunakan selama ini, **writeln** akan mencetak seluruh **string**, karakter, angka, atau variabel yang diberikan, lalu mencetak sebuah baris baru.

write

Fungsi **write** memiliki kegunaan yang sama, tetapi tidak mencetak baris baru di akhir percetakan,

Perbedaan writeln dengan write (lanj.)

- Dengan demikian, untuk mencetak teks berikut:

```
1 2 3
4 5 6
```

- Bisa digunakan:

```
writeln(1, ' ', 2, ' ', 3);
writeln(4, ' ', 5, ' ', 6);
```

atau

```
write(1, ' ', 2, ' ', 3);
writeln;
write(4, ' ', 5, ' ', 6);
writeln;
```

Contoh Program: jumlah.pas

- Coba ketikkan dan jalankan program berikut:

```
var
  a, b: longint;
begin
  write('masukkan nilai a: ');
  readln(a);
  write('masukkan nilai b: ');
  readln(b);
  writeln('hasil dari penjumlahan a dan b: ', a+b);
end.
```

- Pada program tersebut, dicetak terlebih dahulu apa yang perlu dimasukkan. Tentu saja, program seperti ini sangat ramah terhadap pengguna (*user-friendly*).
- Namun dalam kontes pemrograman OSN/IOI, hal seperti ini tidak perlu dilakukan. Bahkan, tidak boleh dilakukan.

Bagian 3

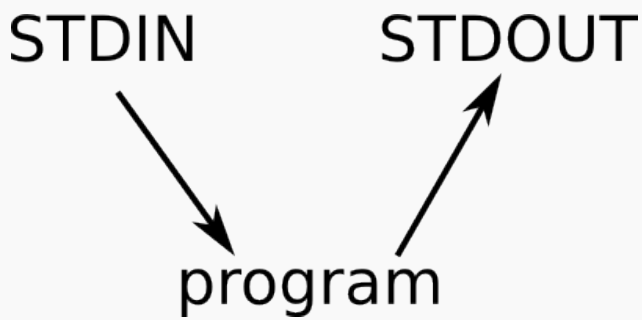
Standard Input Output

Penjelasan Tentang STDIO

- Tempat kalian selama ini mengisikan masukan dan melihat keluaran biasa disebut sebagai *standard input output*, atau **STDIO**.
- **STDIO** memiliki dua saluran yang berbeda, yaitu *input* (**STDIN**) dan *output* (**STDOUT**).

Penjelasan Tentang STDIO (lanj.)

- Masukan yang kalian masukkan, akan melewati saluran STDIN.
- Keluaran yang kalian lihat, sebenarnya datang lewat saluran STDOUT.
- Namun, pada *command line* keduanya terlihat seperti menyatu, seakan-akan keduanya melewati jalur yang sama.



Penjelasan Tentang STDIO (lanj.)

- Untuk lebih memahami tentang hal ini, coba buat sebuah berkas bernama input.txt pada *folder* yang sama dengan program jumlah.pas, dan berisi:

```
1
2
```

- Kemudian pada *command line*, saat menjalankan program jumlah.pas, ketikkan perintah:

```
jumlah < input.txt > output.txt
```

- Buka output.txt dan perhatikan apa yang tercetak!

Penjelasan Tentang STDIO (lanj.)

- Isi dari output.txt adalah:

```
masukkan nilai a:
masukkan nilai b:
hasil dari penjumlahan a dan b: 3
```

- Tulisan "masukkan nilai ..." juga ikut tercetak, karena pada kasus ini, **STDOUT** merupakan berkas output.txt. Segala yang dicetak lewat saluran **STDOUT** akan dicetak ke output.txt.
- Dengan pemahaman yang sama, seluruh masukan yang diberikan adalah lewat **STDIN**, yang merupakan input.txt. Sehingga masukannya perlu dimasukkan ke input.txt terlebih dahulu.

Masukan dan Keluaran pada OSN/IOI

- Setelah kalian memahami tentang **STDIN** dan **STDOUT**, mungkin kalian sudah bisa menebak kenapa pada OSN/IOI tidak boleh mencetak informasi masukan seperti "masukkan nilai ...".
- Hal ini dikarenakan tulisan itu akan ikut tercetak sebagai keluaran, yang mana mengakibatkan ada keluaran yang tidak sesuai spesifikasi soal. Hasilnya, program akan dinilai *wrong answer*, alias menghasilkan jawaban yang tidak sesuai.

Selanjutnya...

- Kini kalian sudah mempelajari tentang variabel, ekspresi, dan masukan/keluaran.
- Artinya, sudah waktunya untuk menulis program-program sederhana.