# Mempelajari C#:
# Bahasa Pemrograman Modern

oleh

Agro Rachmatullah - gamemania1986@hotmail.com
Versi tertutup - tidak untuk umum

# Daftar Isi Singkat

# Daftar Isi

| | | |
|---|---|---|
| **4** | **Tipe-Tipe Data dan Operator** | |
| 4.1 | Integer | |
| 4.2 | Literal, Operator, dan Variabel | |
| 4.2.1 | Operator-Operator Aritmatika | |
| 4.2.2 | Penggunaan Variabel | |
| 4.2.3 | Operator = | |
| 4.2.4 | Operator op= | |
| 4.2.5 | Operator Increment dan Decrement | |
| 4.3 | Floating Point | |
| 4.4 | decimal | |
| 4.5 | char | |
| 4.6 | Literal string | |
| 4.7 | Jenis Literal | |
| 4.8 | Konversi Otomatis | |
| 4.9 | Konversi Manual (Cast) | |
| 4.10 | Perbedaan Type dalam Expression | |
| 4.11 | Type dari Class Library .NET | |
| 4.12 | Operasi-Operasi Ilegal | |
| 4.13 | Panduan Penamaan | |
| | Tes | |
| | | |
| **5** | **Input dan Output Console** | |
| 5.1 | Console | |
| 5.2 | String Format dalam Write() dan WriteLine() | |
| 5.2.1 | Indentasi | |
| 5.2.2 | Mengatur Output Bilangan Pecahan | |
| 5.2.3 | Format Lain-Lain | |
| 5.3 | Parse() | |
| 5.4 | ReadLine() | |
| | Tes | |
| | | |
| **6** | **Mengubah Alur Program** | |
| 6.1 | bool | |
| 6.2 | Operator-Operator Relational | |
| 6.3 | Statement if | |
| 6.3.1 | Statement if Kosong | |
| 6.4 | Penggunaan Blok | |
| 6.4.1 | Variabel dan Blok | |
| 6.5 | if di dalam if | |
| 6.6 | Statement if-else | |
| 6.6.1 | if-else di Dalam if | |
| 6.6.2 | if atau if-else di dalam if-else | |
| 6.7 | Operator-Operator Logical | |
| 6.7.1 | AND | |
| 6.7.2 | OR | |

| | | |
|---|---|---|
| 6.7.3 | XOR | |
| 6.7.4 | NOT | |
| 6.7.5 | Short circuit AND dan OR | |
| 6.8 | Operator Conditional | |
| 6.9 | Statement switch | |
| 6.10 | goto | |
| | Tes | |

# Pendahuluan

Microsoft mengembangkan Framework .NET sebagai platform modern untuk menghadapi tantangan dunia komputer saat ini. Kekayaan class library .NET dan konsep-konsep inovatifnya membuat platform ini "general purpose" namun "powerful". Platform yang baru ini secara perlahan-lahan akan menggeser platform Win32, sebagaimana dulu Win32 membunuh DOS.

C# diciptakan sebagai bahasa untuk mengeksploitasi kekuatan Framework .NET. C# mengambil dan menggabungkan elemen-elemen terbaik dari bahasa-bahasa sebelumnya, sambil menambahkan beberapa hal baru. Dengan sintaksnya yang mirip C, bahasa ini tidak akan terlihat asing bagi para pemrogram C, C++, dan Java.

Buku ini lebih menekankan pada C#, walaupun cukup banyak juga materi yang berkaitan dengan Framework .NET. Pembahasan akan dimulai dari sejarah dan latar belakang C#. Bab selanjutnya membahas mengenai hal-hal fundamental diantaranya cara mengcompile dan menjalankan program C#. Setelah menyelesaikan buku ini anda diharapkan menguasai seluruh unsur bahasa C#, mahir membuat program yang terstruktur baik dan berorientasi objek, serta memahami konsep-konsep dasar Framework .NET.

# Elemen-Elemen Buku Ini

Pembahasan akan dibagi ke dalam bab dan subbab. Pada akhir setiap bab terdapat latihan-latihan untuk menguji sejauh mana penguasaan materi anda. Jawaban dari latihan tersebut ada di Lampiran A. Ingat bahwa jawaban dari latihan pemrograman dan debugging hanyalah sebagai panduan. Source code program yang anda buat dapat saja berbeda, yang penting tujuan pembuatan program tercapai.

Ini adalah beberapa hal yang mungkin anda temui di sela-sela pembahasan:

## Catatan

Catatan berisi informasi-informasi tambahan atau hal-hal menarik yang berhubungan dengan topik yang sedang dibicarakan.

## Tanya Jawab

Tanya jawab memuat pertanyaan yang kemungkinan besar akan ditanyakan oleh pembaca, beserta jawabannya tentunya.

## Latihan Mini

Di setiap bab akan ada beberapa latihan mini yang disertai jawabannya.

## Proyek

Proyek-proyek pemrograman yang ada akan menerapkan hal-hal yang telah dipelajari.

## Target Pembaca

Jika anda tidak memiliki pengalaman pemrograman sebelumnya, anda tetap dapat menggunakan buku ini. Buku ini mengajarkan pemrograman C# tanpa asumsi bahwa anda telah mengerti C, C++, Java, atau bahasa pemrograman lainnya. Tentunya bagi yang telah memrogram menggunakan bahasa keluarga C, C# akan dapat dengan mudah dipelajari.

## Software yang Dibutuhkan

Untuk mengcompile dan menjalankan program-program C#, anda membutuhkan .NET Framework SDK yang ada di [[x]]. Bagi yang ingin bekerja dalam IDE (Integrated Development Environment), Visual Studio .NET dapat digunakan (http://www.microsoft.com/visualstudio). Pengguna Linux harus menginstall Mono (implementasi Framework .NET untuk Linux) dan compiler yang bersangkutan dari http://www.go-mono.com.

## Istilah Asing

Kebanyakan istilah penulis biarkan dalam bahasa Inggris sebab penulis rasa penerjemahannya ke bahasa Indonesia malah akan membuat bingung. Alasan lain adalah karena beberapa istilah tersebut merupakan kata-kata khusus bahasa C#. Anda dapat melihat Lampiran [[x]] jika terdapat kata-kata yang membingungkan.

## Referensi Waktu

Anda akan menemukan beberapa kalimat yang memuat kata-kata yang berhubungan dengan waktu relatif seperti "... akan segera keluar" dan "Saat ini ...". Acuan waktu dari kalimat-kalimat tersebut adalah saat buku ini diterbitkan [[x]].

# Bab 1: Mengenal C# dan Framework .NET

Sebelum "bermain air" dengan menulis kode-kode C# (baca: c sharp), sebaiknya kita mengenal latar belakang dan sejarahnya terlebih dahulu. Pengenalan terhadap C# sebenarnya cukup rumit sebab bahasa ini muncul bersamaan dengan Framework .NET (baca: dot net). Banyak orang yang salah paham, menganggap bahwa Framework .NET adalah C# atau malah sebaliknya.

Kita akan memulai dengan membahas beberapa bahasa pemrograman yang ada sebelum C#. Ini karena terciptanya suatu bahasa pemrograman tidak terlepas dari pengaruh bahasa-bahasa lain yang sudah ada. Apalagi C# hubungan kekerabatannya dekat dengan C, C++, dan Java.

# 1.1 Evolusi Dunia Pemrograman

Dunia pemrograman selalu berevolusi. Perubahan-perubahan yang terjadi tidak saja memudahkan para pemrogram tetapi juga secara langsung menguntungkan pemakai. Dengan bahasa, metode, dan platform yang semakin baik, pemrogram menjadi lebih produktif dan hasil akhirnya adalah program yang kualitasnya semakin baik dengan jangka waktu pembuatan yang semakin singkat. Pembahasan di subbab ini akan memberikan gambaran mengenai evolusi dunia pemrograman.

## 1.1.1 Assembly, Bahasa Pemrograman Zaman Purba

Salah satu bahasa pemrograman terdini adalah bahasa assembly, di mana pemrogram membuat programnya dengan menuliskan instruksi-instruksi prosesor tertentu (biasa disebut low level programming). Karena segala seluk-beluk program berada 100% di tangan pemrogram, maka pemrogram yang handal mampu menciptakan program yang efisiensi dan kecepatannya sangat tinggi.

Ada beberapa kelemahan yang dimiliki bahasa assembly. Keharusan untuk bermain langsung dengan instruksi prosesor, alamat-alamat memori, dan hardware-hardware lain menyebabkan bahasa ini susah dipelajari dan dipakai. Kelemahan lain dari bahasa assembly adalah produktifitas pemrogram yang sangat rendah dan susahnya mengelola program berskala menengah ke atas. Untuk melakukan hal yang paling sederhana sekalipun diperlukan berbaris-baris kode dalam bahasa assembly. Semakin besar ukuran program kita, kesulitan untuk mengelolanya akan meningkat secara eksponen. Karena itulah muncul bahasa-bahasa pemrograman lain yang tingkat abstraksinya lebih tinggi.

## 1.1.2 Kesuksesan C dan C++

Tidak bisa diragukan lagi bahwa C dan C++ adalah dua bahasa pemrograman yang paling sukses. C diciptakan oleh Dennis Ritchie pada tahun 70-an sebagai bahasa pemrograman yang mengiringi operating system UNIX, yang akhirnya distandarisasi oleh ANSI pada tahun 1980.

Bahasa C tingkat abstraksinya cukup tinggi namun konsep-konsep low level masih melekat padanya. C juga menanamkan konsep-konsep structured programming. Dasar dari structured programming adalah penstukturan program melalui penggunaan fungsi dan statement-statement pengatur alur program. Program yang terstruktur dengan baik akan menjadi lebih mudah dikelola. Bahasa structured seperti C memungkinkan

dibuatnya program-program yang berskala cukup besar dalam jangka waktu yang masuk akal.

Seiring dengan berlalunya waktu program-program yang dibuat pun menjadi semakin besar dan kompleks. C dengan structured programmingnya ternyata tidak begitu baik untuk dipakai menangani proyek-proyek yang berskala besar. Kode C yang terdiri dari beribu-ribu baris menjadi susah untuk dipahami dan dikelola. Dalam usaha memecahkan masalah ini, sebuah metode pemrograman baru muncul. Metode tersebut dinamakan Object Oriented Programming (OOP) atau pemrograman berorientasi objek.

[catatan]
Secara mudahnya, OOP membagi masalah ke dalam objek-objek. Perlu diketahui bahwa OOP bukanlah substitusi dari structrured programming. Praktek-praktek structured programming masih tetap dipakai dalam OOP. Kedua metode programming tersebut akan tercermin dalam contoh-contoh program di buku ini dan akan dibahas secara lebih mendalam di bab-bab berikutnya.
[/catatan]

Dengan menggunakan OOP, program-program yang besar menjadi jauh lebih mudah untuk ditangani dan dikembangkan. Sayangnya potensi penuh OOP tidak bisa dieksploitasi menggunakan C karena memang bahasa ini tidak dirancang untuk OOP.

Penambahan unsur-unsur OOP ke C dilakukan oleh Bjarne Stroustrup dari Bell Laboratories pada tahun 1979. Pada awalnya bahasa yang dia buat disebut C with Classes, namun pada tahun 1983 namanya diubah menjadi C++. Standar ANSI C++ keluar pada tahun 1998.

Dengan membuat C++ berfondasikan C, Bjarne menyedot banyak pemrogram C karena mereka tidak perlu mempelajari bahasa yang benar-benar baru. Pemrogram yang sudah mengenal C hanya perlu mempelajari fitur-fitur baru dan sedikit perubahan sebelum dapat memanfaatkan kemampuan OOP dari C++.

Awal mula kepopuleran C disebabkan oleh distribusinya yang luas bersama UNIX. Namun pada akhirnya C/C++ menjadi banyak pilihan pemrogram profesional karena kecepatan, kemampuan, dan fleksibilitasnya yang sangat tinggi. Konsep-konsep low levelnya bahkan menjadikan C/C++ banyak dipakai untuk membuat operating system dan driver.

### 1.1.3 Visual Basic

Bahasa BASIC dibuat di universitas Dartmouth pada tahun 1964 sebagai bahasa pemrograman yang mudah dipakai. Hal ini dapat terlihat mulai dari sintaksnya yang banyak menggunakan kata-kata dari bahasa Inggris, sementara keluarga C menggunakan simbol-simbol seperti kurung kurawal dan titik koma. Bahasa ini memang tidak banyak mengenalkan konsep-konsep sulit sehingga banyak penggunanya. Bahkan karena kemudahannya, bahasa ini biasa disarankan sebagai pintu masuk bagi orang yang ingin belajar membuat program.

Microsoft memilih BASIC sebagai bahasa untuk Rapid Application Development (RAD) di Windows. Implementasi BASIC Mircrosoft ini disebut Visual Basic. Dalam RAD yang dipentingkan adalah produktifitas pemrogram, dan Visual Basic memang

sangat cocok dalam hal ini. Pembuatan program Windows dengan Visual Basic dan sistem drag and drop jauh lebih mudah dibanding kalau kita mengkode segala sesuatunya dengan C/C++.

Walaupun begitu Visual Basic tidak banyak dipakai untuk membuat program-program komersial. Alasannya adalah karena program yang dihasilkan berkecepatan rendah dan ukuran file program yang dihasilkan sangat besar dibandingkan kalau program tersebut dibuat menggunakan C/C++. Alasan lain adalah karena Visual Basic bukan bahasa pemrograman yang sepenuhnya berorientasi objek. Ini tidak berarti bahwa para profesional tidak memakainya sama sekali. Beberapa pemrogram menggunakan Visual Basic untuk membuat prototip programnya dalam waktu yang singkat sebelum menulis kode C atau C++ nya.

### 1.1.4 Java yang Merajai Internet

Java dicipatakan oleh James Gosling di Sun Microsystems pada tahun 1991. Motif diciptakannya Java adalah portabilitas. Program-program Java berjalan di atas Java Virtual Machine (JVM) sehingga suatu program Java dapat dijalankan di komputer apapun asalkan di komputer yang bersangkutan terinstall JVM. Ini berbeda dengan program-program C, C++, dan Visual Basic yang terikat dengan operating system dan arsitektur hardware tertentu.

James Gosling mengikuti langkah Bjarne Stroustrup dengan tidak membuat bahasa yang sintaksnya benar-benar baru, dan itu adalah langkah yang sangat tepat. Bahasa Java sintaksnya mirip dengan C++, namun konsep-konsep sulit yang ada di C++ dibuat lebih mudah atau dihilangkan. Java juga sedikit lebih berorientasi objek dibandingkan C++. Kemiripan Java dengan keluarga C menarik perhatian banyak pemrogram C/C++ untuk mempelajarinya.

Sayangnya program Java yang portable jauh lebih lambat dibandingkan dengan program yang dibuat dengan C/C++ sehingga Java tidak banyak dipakai untuk membuat program-program komersial.

Daerah di mana Java merajalela adalah internet. Applet-applet dan script-script yang ada di berbagai web site kebanyakan menggunakan Java. Ini karena kebanyakan web site tidak membutuhkan kode yang kompleks, sehingga Java yang lambat tidak menjadi masalah. C/C++ tidak terlalu suskes di daerah ini karena implementasinya yang telat dan banyak yang menganggapnya terlalu rumit.

## 1.2 Munculnya Dunia Internet

Munculnya internet mengubah berbagai hal. Di zaman sekarang komputer-komputer yang berbeda saling terhubung dan berinteraksi. Java dengan JVM-nya mencoba mengatasi masalah portabilitas, namun seperti telah disebutkan hasilnya kurang memuaskan. Java juga tidak memiliki cross language interoperability, yaitu kemampuan sebuah program untuk berinteraksi secara mudah dengan program yang dibuat menggunakan bahasa lain. Kemampuan tersebut adalah hal yang sangat berguna di era internet.

[catatan]

Kata "program" di dalam buku ini artinya tidak hanya terbatas pada file yang bisa dijalankan (berakhiran **.exe**), namun juga mencakup library (file berakhiran **.dll**), module (file berakhiran **.netmodule**), dan sejenisnya.
[/catatan]

# 1.3 Framework .NET

Setelah DOS, platfrom unggulan Microsoft adalah Win32. Pengembangan platform ini dimulai pada era 80-an. Saat itu C adalah bahasa pilihan dan OOP belum banyak digunakan. Win32 pada akhirnya menjadi platform pilihan, menyebabkan Windows menjadi OS yang paling populer saat ini.

Kalau Win32 merupakan cerminan teknologi pengembangan software tahun 80-an, Framework .NET adalah platform yang merupakan perwujudan teknologi milenium ke dua. Framework .NET diciptakan untuk dapat memecahkan masalah yang banyak dihadapi dunia pemrograman masa kini secara lebih efisien. Metodologi terbaru seperti OOP dan konsep software sebagai komponen tertanam dengan kuat di Framework .NET.

[catatan]
Microsoft benar-benar serius dalam mengembangkan Framework .NET ini. Lebih dari 4 milyar dolar telah dialokasikan untuk penelitian dan pengembangan Framework .NET. Bahkan sekitar 80% dari seluruh personil R&D (Research and Development) Microsoft ditempatkan di bagian .NET pada tahun 2001-2002.
[/catatan]

Dua bagian penting dari Framework .NET adalah Common Language Runtime (CLR) dan class library .NET.

## 1.3.1 Common Language Runtime (CLR)

CLR adalah bagian dari Framework .NET yang mengelola program-program .NET yang dijalankan. CLR inilah yang mengatur hal-hal seperti pengalokasian memori, pengecekan type, dan keamanan. Program .NET biasa disebut program managed, sedangkan program klasik yang langsung berinteraksi dengan operating system disebut program unmanaged. Program .NET dengan CLR bisa dianalogikan seperti program Java dengan JVM.

## 1.3.2 Class Library .NET

Di dalam Framework .NET terdapat sangat banyak class library mulai dari class yang berisi fungsi-fungsi matematika, class-class yang berhubungan dengan keamanan, class-class untuk membuat program dengan GUI (Graphical User Interface, sebagaimana program-program Windows), sampai pada class-class yang berhubungan dengan input-output. Menguasai penggunaan class library .NET adalah salah satu hal yang vital karena pemrograman .NET tidak akan terlepas dari class-class librarynya.

### 1.3.3 Kelebihan Framework .NET

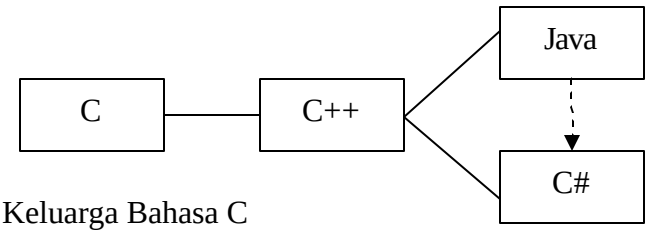
Sejak pertama kali diperkenalkan di tahun 2000, versi alpha Framework .NET sudah menyedot banyak pemrogram. Framework .NET versi 1 keluar pada akhir 2001 dan saat pembuatan buku ini sudah keluar Service Pack pertamanya. Banyak yang mengatakan bahwa Framework .NET memulai era pemrograman baru, bahkan Gartner Group memperkirakan dalam jangka waktu yang singkat dunia pemrograman akan didominasi Framework .NET. Di bawah ini akan diuraikan beberapa kelebihan Framework .NET:

- **Platform Independence:** Program-program yang dibuat untuk berjalan di atas Framework .NET dapat dijalankan di komputer apapun, asalkan di komputer yang bersangkutan terinstall implementasi Framework .NET. Saat penulisan buku ini Framework .NET baru tersedia untuk Windows (98/98SE/Me/NT4/2000/XP). Implementasi Framework .NET untuk Linux yang dinamakan Mono masih dalam pengembangan (http://www.go-mono.com). Microsoft sendiri akan mengeluarkan Framework .NET Compact (untuk peralatan seperti PDA) dalam jangka waktu yang singkat. Tidak tertutup kemungkinan dibuatnya Framework .NET untuk sistem lain misalnya untuk Mac.
- **Language Independence dan Cross Language Interoperability:** Tidak seperti platform Java (JVM) yang terkait dengan bahasa Java, platform .NET tidak terkait dengan bahasa apapun. Contoh-contoh bahasa yang bisa dipakai untuk pemrograman .NET adalah C#, Managed C++, Visual Basic .NET, Jscript .NET, Visual J++ .NET, COBOL .NET, Eiffel#, Phyton .NET, Pascal .NET, dan Perl .NET. Hal yang sangat menarik adalah kemampuan berinteraksi program yang dibuat dengan bahasa berbeda secara langsung! Program-program managed juga dapat berinteraksi dengan program-program unmanaged (dan sebaliknya), walaupun kecepatannya tidak maksimal.
- **Kecepatan:** Program-program .NET walaupun portable akan berjalan dengan kecepatan tinggi, bahkan dapat lebih cepat dari program-program unmanaged. Penjelasan mendetail mengenai hal ini akan dibahas pada bab 2.
- **Keamanan:** Karena semua program .NET berjalan di bawah pengawasan CLR, maka keamanan menjadi lebih terjamin.
- **Produktifitas Tinggi:** Konsep OOP yang tertanam kuat memungkinkan pembuatan program yang dapat dengan mudah dikembangkan. Kekayaan class library .NET juga sangat berperan dalam produktifitas.
- **Support untuk Network/Internet:** Framework .NET dirancang dengan internet "in mind". Interaksi komponen yang terletak di dua komputer yang berbeda dapat dilakukan secara mudah.
- **.NET adalah Platform yang Terbuka:** Siapapun boleh membuat bahasa dan compiler yang menghasilkan program .NET, bahkan membuat implementasi .NET itu sendiri. Spesifikasi-spesifikasi yang berhubungan dengan Framework .NET diserahkan oleh Microsoft kepada ECMA (suatu badan standar internasional), tidak seperti Sun Microsystems yang menolak untuk menyerahkan Java kepada badan standar walaupun sudah didesak banyak pihak. Framework .NET juga banyak menggunakan teknologi-tekonologi yang sudah dipakai secara luas seperti XML, Unicode, HTTP, dan TCP/IP.

# 1.4 Lahirnya C#

Microsoft membuat C# seiring dengan pembuatan Framework .NET. Chief Architect dalam pembuatan C# adalah Anders Hejlsberg yang sebelumnya berperan dalam pembuatan Borland Delphi dan Turbo Pascal. C# menjanjikan produktifitas dan kemudahan yang ada di Visual Basic dengan kemampuan dan fleksibilitas yang ada di C/C++.

Apa sebenarnya tujuan bahasa C#? Menurut spesifikasi bahasanya, *"C# (pronounced "C Sharp") is a simple, modern, object oriented, and type-safe programming language. It will immediately be familiar to C and C++ programmers. C# combines the high productivity of Rapid Application Development (RAD) languages and the raw power of C++."*. Untuk mencapai produktifitas tinggi ini konsep-konsep sulit C++ disederhanakan dan fitur-fitur baru ditambahkan. Hal ini mungkin terasa mirip dengan Java, karena itulah C# bisa dianggap sebagai sepupu Java. Diagram keluarga C bisa dilihat pada gambar 1.1.




Gambar 1.1 - Keluarga Bahasa C

C# berhasil distandarisasi oleh ECMA pada Desember 2001. Dengan standar tersebut siapa saja dapat membuat implementasi C#. Saat ini baru terdapat compiler C# buatan Microsoft dan compiler dari proyek Mono.

## 1.4.1 Perbandingan C# dengan Bahasa .NET Lain

C# adalah salah satu dari banyak bahasa yang bisa dipakai untuk pemrograman .NET. Kelebihan utama bahasa ini adalah sintaksnya yang mirip C, namun lebih mudah dan lebih bersih. Untuk perbandingan penulis cantumkan sedikit informasi mengenai Managed C++ dan Visual Basic .NET:

- **Managed C++:** Managed C++ adalah ekstensi terhadap C++ untuk membuat program .NET. Salah satu keunikan Managed C++ adalah kita bisa mencampur kode-kode managed dengan unmanaged dalam program kita. Ini akan sangat berguna bagi pihak-pihak yang sudah memiliki banyak kode C++ namun ingin bermigrasi ke platform .NET. Dalam pemrograman Managed C++ kita masih akan terikat dengan konsep-konsep sulit C++ sehingga produktifitas akan lebih rendah dibanding jika kita menggunakan C#.
- **Visual Basic .NET:** Perbedaan antara C# dengan Visual Basic .NET yang akan langsung terlihat adalah sintaksnya. C# memiliki beberapa fitur yang tidak ada di

Visual Basic .NET sehingga C# sedikit lebih fleksibel. Perlu diketahui bahwa Visual Basic .NET cukup berbeda dengan Visual Basic 6, sebab Visual Basic .NET adalah bahasa yang sepenuhnya berorientasi objek dan dibuat untuk pemrograman .NET.

Tentang kecepatan program yang dihasilkan, semua bahasa .NET menghasilkan program .NET yang berkecepatan tinggi. Perbedaan kecepatan yang ada sangat kecil bahkan pada umumnya bisa dianggap tidak ada.

### 1.4.2 Ruang Lingkup Pemakaian C# dan Masa Depan Framework .NET

C# sebagai bahasa pemrograman untuk Framework .NET memiliki ruang lingkup penggunaan yang sangat luas. Pembuatan program dengan user interface Windows maupun console dapat dilakukan dengan C#. Karena Framework .NET memberikan fasilitas untuk berinteraksi dengan kode yang unmanaged, penggunaan library seperti DirectX 8.1 dan OpenGL dapat dilakukan. C# juga dapat digunakan untuk pemrograman web site dan web service.

Perlu diingat bahwa semua ini baru permulaan. Microsoft sendiri akan terus mengembangkan Framework .NET dan mengintegrasikan produk-produknya dengan Framework .NET. Sebagai contoh, DirectX 9 akan memiliki komponen-komponen yang managed untuk menyedot para game developer ke dunia pemrograman .NET. Windows dua generasi setelah Windows XP yaitu Windows Blackcomb diisukan akan menjadi akhir dari era Win32 dengan menjadi operating system yang sepenuhnya berfondasikan Framework .NET.

Pembahasan mengenai Framework .NET secara menyeluruh dapat menjadi satu buku yang jauh lebih tebal dari buku ini. Dalam buku ini kita akan lebih memusatkan pada penggunaan bahasa C# untuk mengeksploitasi kekuatan Framework .NET. Selamat datang di era pemrograman baru!

[tes]

## A. Pertanyaan

1. Untuk membuat program-program .NET kita harus menggunakan C#. Benar atau salah?

2. Jelaskan mengenai CLR (Common Language Runtime)!

3. Apakah program-program .NET hanya dapat berjalan di Windows?

[/tes]

# Bab 2: Membuat Program Pertama Anda

Setelah pengenalan terhadap C# dan Framework .NET, kini kita akan langsung mencoba membuat sebuah program C# sederhana. Setelah mempelajari bab ini anda akan tahu cara untuk mengubah suatu file teks menjadi file program yang bisa dijalankan. Analisa program tersebut akan dilakukan di bab selanjutnya.

# 2.1 Proses Compilation

Sebelum melangkah lebih jauh anda perlu tahu mengenai proses compilation. Kode C# yang kita ketik akan disave sebagai file teks dengan akhiran **.cs** (misalnya **ProgramSaya.cs**). Isi file tersebut biasa disebut source code. Compilation adalah proses yang mengubah file teks menjadi file program yang bisa dijalankan (file dengan ahiran **.exe**). Program yang melakukannya disebut compiler.

[catatan]
Walaupun secara default hasil compile adalah file program (file berakhiran **.exe**), kita bisa memilih beberapa output lainnya misalnya file library (file dengan akhiran **.dll**).
[/catatan]

Program-program C# yang akan dibuat dapat diketik dengan menggunakan text editor seperti Notepad maupun dengan menggunakan IDE (Integrated Development Environment) seperti Visual Studio .NET (yang selanjutnya akan disebut VS.NET). Jika anda menggunakan IDE seperti VS.NET, maka program dapat dicompile dan dijalankan dengan mudah melalui tombol maupun menu yang ada. Pengguna Notepad atau text editor lainnya harus memanggil **csc.exe** dari console (command prompt) untuk mengcompile. Kedua cara tersebut akan dibahas dalam bab ini.

[catatan]
Pengguna Linux dapat memperoleh informasi mengenai cara mengcompile programnya di dokumentasi Mono (http://www.go-mono.com).
[/catatan]

## 2.1.1 Menginstall SDK untuk Framework .NET

Compiler untuk C# (**csc.exe**) merupakan bagian dari .NET Framework SDK (Software Development Kit). Anda bisa menginstall .NET Framework SDK dari [[x]]. Jika anda menggunakan VS.NET perlu diketahui bahwa .NET Framework SDK ikut terinstall saat anda menginstall VS.NET. Di dalam CD yang menyertai buku ini juga tersedia Service Pack 1 untuk Framework .NET ([[path/filename]]) yang sebaiknya anda install (baik bagi yang menginstall .NET Framework SDK secara manual maupun bagi pengguna VS.NET).

## 2.1.2 Menggunakan Text Editor dan csc.exe

Jalankan sebuah text editor misalnya Notepad, lalu ketik program di bawah ini:

[program lengkap]

```
// Program 2.1 – Program Pertama Anda

class Halo
{
    static void Main()
    {
        System.Console.WriteLine("Halo Dunia!");
    }
}
```
[/program lengkap]

Save teks tersebut dengan nama file **program-2-1.cs**. Jika anda menggunakan program seperti Microsoft Word pastikan anda mengesave file anda sebagai "plain text" atau "text only".

Langkah berikutnya adalah mengcompile file yang telah kita buat. Tuliskan perintah ini di console (tentunya anda harus berada di folder tempat **program-2-1.cs** berada):

[console]
```
c:\SuatuFolder>csc program-2-1.cs
```
[/console]

Jika tidak ada yang salah dalam langkah di atas maka akan terbuat file yang bernama **program-2-1.exe**. Program tersebut kini dapat anda jalankan dengan mengetik nama filenya:

[console]
```
c:\SuatuFolder>program-2-1
```
[/console]

Program tersebut akan berjalan dalam console dan menghasilkan output sebagai berikut di layar:

[console]
```
Halo Dunia!
```
[/console]

[catatan]
Jika program tersebut dijalankan misalnya dengan mengklik iconnya dua kali pada explorer, console yang terbuka akan langsung hilang setelah program selesai dijalankan (dalam hal ini setelah menampilkan **"Halo Dunia!"**). Ini jelas tidak menguntungkan sebab kita tidak bisa menganalisa output program kita. Karena itu disarankan agar program-program console dijalankan dengan memanggilnya melalui command prompt, kecuali kalau kita memberikan mekanisme semacam "Press any key to exit..." dalam program yang kita buat.
[/catatan]

Jadi secara umum ada tiga tahap yang akan selalu dilakukan dalam proses pembuatan program C#, yaitu membuat/mengetik program, mengcompile, dan menjalankan program.

### 2.1.3 Menggunakan Visual Studio .NET

Visual Studio 7 atau biasa disebut Visual Studio .NET dapat mengcompile program untuk platform .NET. Ikuti langkah-langkah di bawah ini untuk mengetik, mengcompile, dan menjalankan program C# melalui VS.NET (jika anda memiliki Visual Studio yang lebih baru mungkin caranya akan sedikit berbeda):

1. Buat proyek C# baru dengan memilih **File -> New -> Project**. Di panel kiri pilih **Visual C# Projects** dan di panel kanan pilih **Empty Project**.
2. Setelah terbuat proyek baru, klik kanan namanya pada **Solution Explorer**. Lalu pilih **Add -> New Item**.
3. Pada dialog yang muncul, pilih **Local Project Items** di panel kiri dan **Code File** di panel kanan.
4. Ketik program 2.1 yang telah ditampilkan sebelumnya.
5. Compile program yang telah anda ketik dengan memilih **Build -> Build Solution**.
6. Program dapat dijalankan dengan memilih **Debug -> Start Without Debugging**.

Anda tidak perlu membuat proyek baru setiap kali anda ingin mencoba contoh program di buku ini. Hapus saja file yang sebelumnya ada lewat **Solution Explorer** lalu tambahkan file baru (langkah 2). Setelah itu compile ulang (langkah 5) dan jalankan programnya (langkah 6).

IDE seperti VS.NET memiliki banyak kelebihan dibanding text editor biasa misalnya pewarnaan dan indenting kode secara otomatis. Tentunya penggunaan IDE akan mempermudah pengelolaan proyek yang berskala besar (misalnya proyek yang melibatkan banyak file). Pilihan untuk menggunakan **csc.exe** atau VS.NET ada di tangan anda.

[tanya jawab]
**Q:** Program yang saya buat tidak bisa dijalankan di komputer lain!!!
**A:** Seperti kita ketahui, program yang kita buat adalah program yang ditargetkan untuk platform .NET. Karena itu Framework .NET harus terinstall di komputer yang bersangkutan. .NET Framework Redistributable tersedia di CD yang menyertai buku ini ([[path/filename]]). Jangan lupa untuk menginstall Service Pack 1 untuk Framework .NET ([[path/filename]]).

**Q:** Apa beda antara .NET Framework SDK dengan .NET Framework Redistributable?
**A:** .NET Framework Redistributable hanya mencakup bagian untuk menjalankan program-program .NET, sedang .NET Framework SDK mencakup bagian untuk membuat program-program .NET (diantaranya compiler untuk C#, Managed C++, dan Visual Basic .NET).
[/tanya jawab]

[latihan mini]

- Jelaskan mengenai compilation secara singkat!
- Tuliskan perintah command prompt yang dipakai untuk mengcompile!
- Apakah Windows (98/98SE/Me/NT4/2000/XP) dan Linux sudah langsung menyediakan compiler C#?

- Compilation akan mengubah file teks menjadi file program yang bisa dijalankan.
- **csc** *[namafile]* .
- Tidak. .NET Framework SDK, VS.NET, atau compiler lain harus diinstall terlebih dahulu.

[/latihan mini]

### 2.1.4 Syntax Error

Jika ada kesalahan bahasa dalam program anda (yang biasanya karena salah ketik), maka compiler tidak akan menghasilkan output file apapun. Kesalahan yang ditemukan akan dilaporkan sehingga akan mempermudah kita untuk memperbaikinya. Contoh pesan yang dikeluarkan compiler adalah sebagai berikut:

[console]
```
program-2-1.cs(7,42): error CS1002: ; expected
```
[/console]

Ini menunjukkan terdapat kesalahan pada baris ke 7 di kolom/karakter ke 42. Jenis kesalahannya juga disebutkan. Pada contoh di atas jenis kesalahannya adalah kurangnya penulisan titik koma (;) di akhir statement.

## 2.2 IL dan JIT

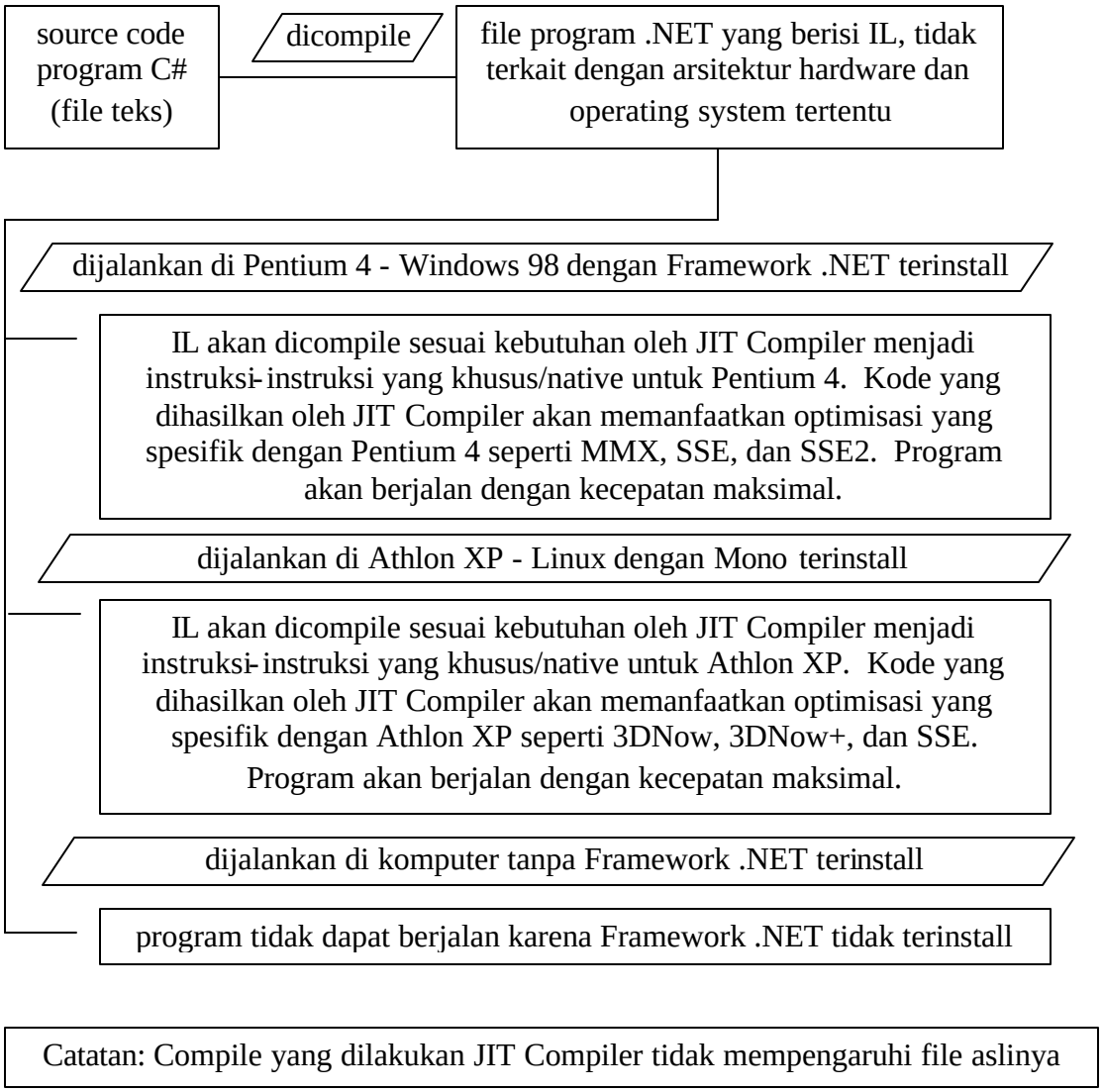
Jika kita mengcompile source code program unmanaged (misalnya source code program C++ yang tidak ditargetkan untuk .NET), maka hasilnya adalah sebuah program yang terikat dengan operating system dan arsitektur hardware tertentu. Program yang dicompile untuk Windows misalnya, tidak akan bisa berjalan di Linux dan Macintosh tanpa bantuan program-program emulator. Program yang dicompile untuk prosesor IA-64 (misalnya Intel Itanium) juga tidak bisa berjalan di komputer dengan prosesor x86 (misalnya Intel Pentium 4 dan AMD Athlon XP). Dengan demikian hasil programnya tidak portable.

Di sisi lain, hasil compile program-program .NET adalah file yang berisi IL (Intermedia Language). Instruksi-instruksi IL tidak terkait dengan operating system dan arsitektur prosesor apapun. Karena itu program-program .NET dapat dijalankan di mana saja, asalkan Framework .NET terinstall di komputernya.

Saat sebuah program .NET dijalankan, CLR akan memanggil JIT Compiler (Just In Time Compiler) untuk mengcompile IL dalam program tersebut agar menjadi kode yang native di komputer yang bersangkutan. Efeknya, program bisa berjalan dengan kecepatan tinggi. Keseluruhan program tidak dicompile oleh JIT Compiler sekaligus, melainkan sesuai kebutuhan. Jadi walaupun program-program .NET portable, kecepatan dan efisiensinya tetap tinggi karena pada akhirnya program tersebut dijalankan secara

native di komputer yang bersangkutan (lihat gambar 2.1). IL juga merupakan salah satu faktor yang memungkinkan tercapainya cross language interoperability.

Konsep-konsep portabilitas dan interoperability .NET yang dijelaskan di sini sebenarnya hanya kulitnya saja. Beberapa standar yang juga berperan seperti Common Type System (CTS) dan Common Language Specification (CLS) tidak dibahas di buku ini.




Gambar 2.1 - IL dan JIT

[tes]

## A. Pertanyaan

1. Apakah akhiran yang biasa dipakai untuk file yang berisi source code program C#?

2. File source code C# dapat langsung dijalankan, benar atau salah?

3. Apakah hasil compile suatu program C# berupa kode native?

4. Jelaskan mengenai JIT Compiler!

[/tes]

# Bab 3: Dasar-Dasar C#

[program lengkap]
```
// Program 2.1 – Program Pertama Anda

class Halo
{
    static void Main()
    {
        System.Console.WriteLine("Halo Dunia!");
    }
}
```
[/program lengkap]

Mari kita lirik kembali program yang telah kita buat pada bab 2 (dicantumkan di atas). Walaupun pendek, program tadi memuat hal-hal dasar yang ada di setiap program C#. Kita akan mulai dari nama filenya.

# 3.1 Nama File

Walaupun program yang baru kita save dengan nama **program-2-1.cs**, anda boleh saja memberinya nama lain misalnya **programpertama.cs** atau **xyz123.cs**.

Akhiran yang biasa dipakai untuk menandakan bahwa suatu file berisi source code program C# adalah **.cs**. Anda bisa saja memberi akhiran lain misalnya **.txt**, dan bagi compiler pun tidak ada masalah, namun pemberian akhiran selain **.cs** sangat tidak disarankan.

# 3.2 Comment

Program yang telah kita buat dimulai dengan baris ini:

[kutipan program]
```
// Program 2.1 – Program Pertama Anda
```
[/kutipan program]

Dalam dunia pemrograman hal di atas dinamakan comment. Comment tidak akan mempengaruhi program yang anda buat sedikit pun karena compiler akan menghiraukannya. Lalu apa gunanya comment? Comment dapat digunakan sebagai dokumentasi, untuk memberikan informasi-informasi mengenai program yang anda buat. Penggunaan comment yang baik dapat membantu seseorang dalam memahami suatu source code. Orang tersebut bisa saja anda sendiri yang perlu menengok kode yang sudah 6 bulan tidak anda sentuh.

## 3.2.1 Comment Single Line

Di dalam bahasa C# terdapat 3 jenis comment. Yang pertama adalah comment single line yang baru saja kita bahas. Comment single line diawali dengan // dan berakhir

pada baris yang bersangkutan. Walaupun begitu, tidak berarti bahwa comment single line harus dimulai pada awal baris, contohnya seperti berikut ini:

[kutipan program]
```
System.Console.WriteLine("Halo Dunia!"); // akan menuliskan pesan di layar
```
[/kutipan program]

Comment juga dapat digunakan untuk mematikan statement secara sementara, contohnya:

[kutipan program]
```
// System.Console.WriteLine("Halo Dunia!"); // akan menuliskan pesan di layar
```
[/kutipan program]

Seluruh baris di atas akan menjadi comment dan dihiraukan oleh compiler. Perhatikan juga bahwa // yang kedua tidak berpengaruh.

## 3.2.2 Comment Multi Line

Comment yang kedua adalah comment multi line. Comment multi line dimulai dengan **/*** dan diakhiri dengan ***/**. Sesuai namanya, satu set comment jenis ini dapat memenuhi lebih dari satu baris. Ini beberapa contoh penggunaannya:

[kutipan program]
```
/* Comment jenis ini
dapat menutupi lebih
dari satu baris */

/**********************
** comment sekaligus **
** sebagai dekorasi   **
**********************/
```
[/kutipan program]

Perlu diketahui bahwa comment multi line dapat disisipkan di antara dua elemen bahasa, tetapi tidak bisa disisipkan di dalam suatu elemen bahasa. Untuk lebih jelasnya perhatikan kutipan-kutipan program berikut:

[kutipan program]
```
static /* OK */ void Main()
{
    System./* OK */Console.WriteLine("Halo Dunia!");
}
```
[/kutipan program]

Kutipan program di atas dapat dicompile namun kutipan selanjutnya tidak:

[kutipan program]
```
System.Con/* Error!!! */sole.WriteLine("Halo Dunia!");
```

[/kutipan program]

Comment multi line tidak bisa disisipkan ke dalam comment multi line lainnya. Contohnya:

[kutipan program]
```
/* Comment 1
/* Comment 2. Sayangnya pembuka comment ini akan dihiraukan compiler (karena berada di dalam comment 1) dan penutup comment ini akan dianggap penutup comment 1. Sisa satu penutup di paling bawah akan menyebabkan kode ini tidak bisa dicompile */
*/
```
[/kutipan program]

[catatan]
Masih ada satu jenis comment lagi yaitu comment XML, namun baru akan kita bahas di bab [[x]].
[/catatan]

# 3.3 Method Main()

Kembali pada program 2.1, perhatikan baris berikut ini (baris sebelumnya sengaja penulis lompati):

[kutipan program]
```
static void Main()
```
[/kutipan program]

Baris di atas mendeklarasikan suatu method yang bernama **Main()** (di beberapa bahasa pemrograman lain, method disebut fungsi atau subroutine). Setiap program C# harus memiliki method ini. Anggaplah method **Main()** sebagai pintu masuk program anda. Kalau anda mencoba mengganti nama **Main()** dengan nama lain, compiler akan mengeluh bahwa tidak ada entry point atau pintu masuk. Perlu juga diketahui bahwa isi atau tubuh method harus diawali dengan **{** dan diakhiri dengan **}**. Ini adalah kutipan method **Main()** yang lengkap:

[kutipan program]
```
static void Main()
{
    // tubuh atau isi method
}
```
[/kutipan program]

**static** dan **void** akan diulas di bab-bab berikutnya.

# 3.4 Class

Method di C# tidak bisa berdiri sendiri, namun harus menjadi bagian dari suatu class. Di program 2.1 method **Main()** berada di dalam class yang bernama **Halo**.

[kutipan program]
```
class Halo
{
    // tubuh atau isi class
}
```
[/kutipan program]

Nama class yang menyelimuti **Main()** tidak menjadi masalah. Seperti method, tubuh suatu class dimulai dengan **{** dan diakhiri dengan **}**.

[latihan mini]
- Sebutkan jenis-jenis comment yang ada di C#!
- Method apakah yang dipanggil/dijalankan saat suatu program C# dimulai?
- Suatu method harus ada di dalam class. Benar atau salah?

- Comment single line, comment multi line, dan comment XML.
- Method **Main()**.
- Benar.

[/latihan mini]

# 3.5 System.Console.WriteLine()

Sekarang perhatikan baris berikut:

[kutipan program]
```
System.Console.WriteLine("Halo Dunia!");
```
[/kutipan program]

Statement di atas adalah melakukan sesuatu yang bisa kita lihat hasilnya, yaitu menuliskan kata-kata "**Halo Dunia!**" di layar. Suatu aturan yang harus diingat adalah setiap statement harus diakhiri dengan titik koma (**;**). Lupa memberikan titik koma pada akhir statement biasanya adalah kesalahan yang paling banyak dilakukan pemrogram pemula.

Sekarang kita analisa statement tersebut. Yang bekerja menuliskan "**Halo Dunia!**" ke layar adalah method yang bernama **WriteLine()**. Method **WriteLine()** tersebut berada di dalam class yang bernama **Console** (sebagaimana method **Main()** berada dalam suatu class). Class **Console** sendiri dikelompokkan ke dalam namespace yang bernama **System**. Namespace **System** menampung semua class library .NET (**Console** adalah salah satu dari class library .NET). Jadi kita memanggil method **WriteLine()** dengan menuliskan mulai dari namespacenya sampai ke method itu sendiri. "**Halo Dunia!**" dikatakan sebagai argument bagi method **WriteLine()**.

# 3.6 Namespace

Namespace digunakan untuk pengelompokan dan untuk menghindari konflik nama. Misalnya perusahaan A dan perusahaan B sama-sama membuat class yang bernama **Bank**. Kedua class yang namanya sama tersebut dapat digunakan di suatu program asalkan terletak di namespace yang berbeda. Kegunaan namespace untuk mengelompokan elemen-elemennya (misalnya class) bisa dimisalkan seperti kegunaan folder untuk mengelompokkan file-file.

## 3.6.1 Penggunaan using

Perhatikan contoh program di bawah ini:

[program lengkap]
```
// Program 3.1

class KalimatKalimat
{
    static void Main()
    {
        System.Console.WriteLine("Kalimat 1.");
        System.Console.WriteLine("Kalimat 2.");
        System.Console.WriteLine("Kalimat 3.");

    }
}
```
[/program lengkap]

Kita bisa mempersingkat program kita dengan menggunakan statement **using**. Penggunaannya ditunjukkan di bawah:

[program lengkap]
```
// Program 3.2  - Penggunaan using

using System; // statement using ditambahkan

class PenggunaanUsing
{
    static void Main()
    {
        Console.WriteLine("Kalimat 1.");
        Console.WriteLine("Kalimat 2.");
        Console.WriteLine("Kalimat 3.");

    }
}
```
[/program lengkap]

Dengan statement **using** di atas, semua elemen dari namespace **System** bisa kita akses tanpa menuliskan nama namespacenya. Penulisan nama lengkap (beserta

namespacenya) tetap dapat dipakai, namun biasanya hanya digunakan jika terdapat dua elemen namespace yang namanya sama (terletak di namespace yang berbeda). Statement-statement **using** seperti di atas hanya dapat dipakai di awal program.

# 3.7 WriteLine() vs Write()

Salah satu method lain dari dass **Console** adalah **Write()**. Method **WriteLine()** akan menyisipkan baris baru setelah menuliskan argument yang diberikan, sedangkan method **Write()** tidak. Anda bisa melihat perbedaannya jika menjalankan program di bawah ini:

[program lengkap]
```
// Program 3.3 - Write() dan WriteLine()

using System;

class WriteDanWriteLine
{
    static void Main()
    {
        Console.WriteLine("Kalimat 1.");
        Console.WriteLine("Kalimat 2.");
        Console.WriteLine(); // WriteLine() dipanggil tanpa argument
        Console.Write("Kalimat 3.");
        Console.Write("Kalimat 4.");
    }
}
```
[/program lengkap]

Outputnya adalah sebagai berikut:

[console]
```
Kalimat 1.
Kalimat 2.

Kalimat 3.Kalimat 4.
```
[/console]

Melalui contoh tersebut kita juga bisa melihat bahwa method **WriteLine()** dapat dipanggil tanpa argument apapun. Pemanggilan method **WriteLine()** tanpa argument akan menyisipkan baris baru atau biasa disebut new line.

[latihan mini]
- Apakah nama namespace yang memuat seluruh class library .NET?
- Apakah yang dilakukan method **WriteLine()**?
- Apakah yang salah pada kutipan program di bawah ini?

[kutipan program]

```
static void Main()
{
  using System;
  Console.WriteLine(“Apa yang salah???”);
}
```
[kutipan program]

- **System**.
- Menampilkan argument yang diberikan pada layar (console) dan menambahkan karakter new line.
- **using System** hanya dapat diletakkan di awal program.

[/latihan mini]

# 3.8 Spasi dan Indentasi

C# adalah bahasa yang free form, tidak perduli mengenai jumlah pemisah (spasi, new line, maupun tabulasi) yang menyusun suatu program. Kutipan program di bawah ini walaupun tidak enak dilihat, secara kebahasaan sah-sah saja:

[kutipan program]
```
Console.WriteLine(“Kalimat 1.”);Console.WriteLine(“Kalimat 2.”);
Console . WriteLine ( “Kalimat 3.” ) ;
```
[/kutipan program]

Ada beberapa aturan yang sebaiknya anda ikuti. Pertama, sebaiknya satu baris hanya terdiri dari satu statement. Kedua, setiap dimulai blok baru (misalnya saat memulai tubuh class atau method) sebaiknya indentasi dimajukan. Sebisa mungkin ikuti style penulisan yang terdapat pada program-program dalam buku ini.

# 3.9 Identifier dan Keyword

Nama yang kita berikan kepada namespace, class, method, variabel, dan benda-benda buatan kita lainnya disebut identifier. Ada beberapa aturan yang harus dipenuhi dalam membuat identifier. Identifier hanya dapat terdiri dari abjad, angka (**0-9**), dan garis bawah (_). Identifier tidak boleh dimulai dengan angka.

Ada kata-kata tertentu yang tidak bisa dipakai sebagai identifier secara normal. Kata-kata tersebut disebut keyword, yang dipakai sebagai element bahasa C# (lihat tabel 3.1).

| | | | | |
|---|---|---|---|---|
| absract | as | base | bool | break |
| byte | case | catch | char | checked |
| class | const | continue | decimal | default |
| delegate | do | double | else | enum |
| event | explicit | extern | false | finally |
| fixed | float | for | foreach | goto |
| if | implicit | in | int | interface |
| internal | is | lock | long | namespace |
| new | null | object | operator | out |

| | | | | |
|---|---|---|---|---|
| override | params | private | protected | public |
| readonly | ref | return | sbyte | sealed |
| short | sizeof | stackalloc | static | string |
| struct | switch | this | throw | true |
| try | typeof | uint | ulong | unchecked |
| unsafe | ushort | using | virtual | void |
| volatile | while | | | |

*Tabel 3.1 - Daftar Keyword C#*

Kalau memang diperlukan, keyword dapat digunakan sebagai identifier dengan mencantumkan **@** sebelumnya. Contohnya **@class** dapat digunakan sebagai identifier. Sebenarnya **@** dapat digunakan untuk mengawali identifier apapun (yang bukan keyword sekalipun), namun sebaiknya tidak digunakan kecuali jika diperlukan. Perlu diketahui bahwa huruf besar dibedakan dengan huruf kecil di C#, jadi **Alpha**, **aLpHa**, dan **alpha** adalah tiga identifier yang berbeda.

Sebaiknya identifier memiliki makna yang berarti. Tentunya penggunaan nama seperti **JariJari**, **Keliling**, dan **Luas** lebih baik dibanding jika kita menggunakan **Var01**, **Var01**, dan **Var02**.

[tanya jawab]
**Q:** Kenapa penggunaan keyword sebagai identifier diizinkan di C#?
**A:** Ini karena C# berada dalam komunitas bahasa-bahasa .NET yang keywordnya berbeda-beda. Misalnya di bahasa A tidak ada keyword **static**. Tentunya seseorang bisa saja membuat library yang berisi class bernama **static**. Tanpa mekanisme "keyword sebagai identifier" maka class yang bernama **static** tersebut tidak akan bisa dipakai di dalam suatu program C#. Bahasa-bahasa .NET lain juga menyediakan mekanisme "keyword sebagai identifier"walaupun caranya berbeda-beda.
[/tanya jawab]

[tes]
## A. Pertanyaan

1. Method apakah yang dipanggil saat suatu program C# dijalankan?

2. Apakan setiap program C# harus dimulai dengan statement di bawah ini?

   [kutipan program]
   ```
   using System;
   ```
   [/kutipan program]

3. Bagaimanakah cara membuat comment single line dan multi line? Dapatkan comment multi line dimasukkan ke dalam suatu comment multi line lain?

4. Manakah diantara identifier-identifier berikut yang tidak benar?

   double __byte 15Mobil Pagar# @class Kelas6SD

5. Apakah kegunaan namespace?

6. Karakter apakah yang digunakan untuk mengakhiri suatu statement?

## B. Membuat Program

1. Buatlah program yang menampilkan kata "**Hai**" di console! Method **Main()** dalam program tersebut harus berada di dalam class yang bernama **new**.

## C. Debugging

1. Program berikut bertujuan menampilkan kata "**Halo**" di console. Terdapat banyak kesalahan sehingga program tidak bisa dicompile. Betulkanlah!

[program lengkap]
Tes 2.C.1

```
Class DebuglahSaya
{
 // program dimulai dari method ini // static void main()
 {
      Console.WriteLine("Halo")
 }
}
```

[/program lengkap]
[/tes]

# Bab 4: Tipe-Tipe Data dan Operator

[catatan]

Dimulai bab ini beberapa pembahasan akan berhubungan dengan binary dan heksadesimal. Pembahasan mengenai materi tersebut dapat anda peroleh di Lampiran [[x]].

[/catatan]

C# adalah bahasa yang strongly typed. Setiap benda (data, misalnya variabel) yang ada di program C# memiliki type. Type mendifinisikan suatu benda, termasuk operasi/tindakan yang bisa dilakukan oleh/terhadap benda tersebut. Compiler akan melakukan pengecekan type sehingga kode yang berisi tindakan-tindakan ilegal tidak akan dapat dicompile.

Type di dalam C# terbagi menjadi dua yaitu value type dan reference type. Pembahasan reference type behubungan erat dengan class, karena itu di bab ini hanya value type sederhana yang akan dibahas dimulai dari integer.

# 4.1 Integer

C# mendifinisikan 9 jenis integer atau bilangan bulat yaitu **char**, **byte**, **sbyte**, **short**, **ushort**, **int**, **uint**, **long**, dan **ulong**. Yang membedakan di antara kesembilan jenis tersebut adalah batas maksimum dan minimumnya. **char** cukup spesial karena **char** digunakan untuk merepresentasikan karakter (huruf, tanda baca, dsb). **char** jarang digunakan untuk perhitungan angka, dan akan kita bahas tersendiri di bagian lain bab ini. Tabel berikut akan memberikan informasi mengenai integer yang ada di C#:

| Type | Jarak | Bit / Byte yang Digunakan |
|---|---|---|
| char | 0 sampai 65535 (tidak biasa digunakan untuk perhitungan) | 16 bit / 2 byte |
| byte | 0 sampai 255 | 8 bit / 1 byte |
| sbyte | -128 sampai 127 | 8 bit / 1 byte |
| short | -32,768 sampai 32,767 | 16 bit / 2 byte |
| ushort | 0 sampai 65,535 | 16 bit / 2 byte |
| int | -2,147,483,648 sampai 2,147,483,647 | 32 bit / 4 byte |
| uint | 0 sampai 4,294,967,295 | 32 bit / 4 byte |
| long | -9,223,372,036,854,775,808 sampai 9,223,372,036,854,775,807 | 64 bit / 8 byte |
| ulong | 0 sampai 18,446,744,073,709,551,615 | 64 bit / 8 byte |

[catatan]

Koma yang ada dalam angka misalnya 10,001 bukan berarti sepuluh koma nol nol satu. Koma tersebut digunakan sebagai penanda ribuan, ratus ribuan, dan seterusnya. Untuk pemisah antara bilangan bulat dengan pecahannya digunakan karakter titik. Misalnya 1.5 berarti satu koma lima dan 1,000.3 berarti seribu koma tiga. Ini dipakai agar buku ini konsisten dengan C# yang menggunakan titik untuk membagi antara bilangan bulat dengan bilangan fraksionalnya. Perlu diketahui juga bahwa di dalam program C# koma tidak dapat digunakan sebagai penanda, angka harus ditulis apa adanya.
[/catatan]

# 4.2 Literal, Operator, dan Variabel

Operasi-operasi seperti penjumlahan dan pengurangan dilakukan dengan menggunakan operator. Contoh operator yang ada adalah **+** untuk penjumlahan dan **/** untuk pembagian. Pada **1 + 3**, angka **1** dan **3** disebut operand.

Operator seperti **/** disebut operator binary karena membutuhkan dua operand. Operator yang hanya membutuhkan satu operand disebut operator unary. Beberapa operator dapat berfungsi sebagai unary dan binary. Misalnya operator **-** dalam **2 - 1** adalah binary, namun dalam **-3** adalah unary.

Angka-angka (dan beberapa bentuk data lain) yang tertulis dalam program anda disebut literal. Ini beberapa contoh literal integer:

`101` `-5` `1234567890` `+20` (sama dengan `20`) `0`

Literal nilainya pasti dan tetap/konstan. Jadi kalau tertulis **101** maka sudah pasti nilainya **101**. Literal berbeda dengan variabel misalnya variabel yang kita beri nama **x**. Nilai **x** bisa berubah-rubah.

Kita juga dapat menuliskan literal integer dalam bentuk heksadesimal atau bilangan berbasis **16**. Caranya ialah dengan menuliskan **0x** atau **0X** (angka nol diikuti huruf x) sebelum angkanya. Misalnya **0x10** adalah **10** dalam heksadesimal atau **16** dalam desimal, dan **0xb** adalah **11** dalam desimal. Karakter **a** - **f** atau **A** - **F** digunakan untuk melambangkan **10** sampai **15**.

[catatan]
Secara default type dari literal integer adalah **int**, kecuali pada kasus-kasus tertentu (akan dibahas di akhir bab ini).
[/catatan]

## 4.2.1 Operator-Operator Aritmatika

Beberapa operasi aritmatika beserta operatornya yang dapat digunakan untuk integer adalah:

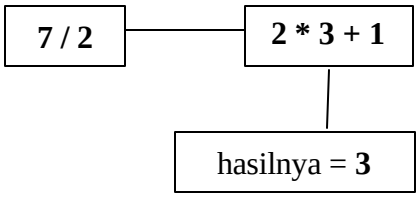
| Operasi | Operator |
|---|---|
| Tanda positif | + (unary) |
| Tanda negatif | - (unary) |
| Penjumlahan | + |

| | |
|---|---|
| Pengurangan | - |
| Perkalian | * |
| Pembagian | / |
| Modulus (sisa pembagian) | % |

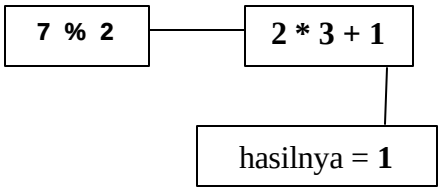
Perlu diketahui bahwa operasi binary yang kedua operandnya adalah integer akan menghasilkan integer juga. Jadi **7 / 2** hasilnya bukanlah **3.5**, namun hasilnya **3** (selalu dibulatkan ke bawah).

Operasi modulus menghasilkan sisa dari suatu pembagian yang hasil baginya bulat. Misalnya **7 % 2** menghasilkan **1**, sebab **7** adalah **2 * 3 + 1** (**3** adalah hasil bagi yang bulat dan **1** adalah sisanya). Gambaran yang lebih jelas dapat dilihat di bagan berikut:

Pembagian Integer




Operasi Modulus




Method **Write()** dan **WriteLine()** dapat menerima integer sebagai argument. Mari kita praktekkan hal-hal yang telah kita bahas!

[program lengkap]

```
// Program 4.1 - Operasi aritmatika

using System;

class OperasiAritmatika
{
    static void Main()
    {
        // semua integer yang digunakan di program ini adalah literal
        Console.WriteLine(10); // hasilnya 10
        Console.WriteLine(+10); // hasilnya 10
        Console.WriteLine(-10); // hasilnya -10

        Console.WriteLine(10 + 4); // hasilnya 14
        Console.WriteLine(10 - 4); // hasilnya 6
        Console.WriteLine(4 - 10); // hasilnya -6
        Console.WriteLine(10 * 4); // hasilnya 40
        Console.WriteLine(10 / 4); // hasilnya 2 (sisa pembagian diabaikan)
        Console.WriteLine(4 / 10); // hasilnya 0 (sisa pembagian diabaikan)
        Console.WriteLine(10 % 4); // hasilnya 2 (10 adalah 4 * 2 + 2)
        Console.WriteLine(4 % 10); // hasilnya 4 (4 adalah 10 * 0 + 4)

        Console.WriteLine(10 + 4 * 2); // hasilnya 18
    }
}
```

[/program lengkap]

Inilah output program di atas:

[console]
```
10
10
-10
14
6
-6
40
2
0
2
4
18
```
[/console]

Ada beberapa hal yang cukup menarik dari program di atas, salah satunya statement berikut:

[kutipan program]
```
Console.WriteLine(+10); // hasilnya 10
```
[/kutipan program]

Dari outputnya kita dapat mengetahui bahwa secara default tanda **+** tidak akan ditampilkan untuk bilangan positif.

Statement berikut memerlukan analisis lebih lanjut:

[kutipan program]
```
Console.WriteLine(10 + 4 * 2); // hasilnya 18
```
[/kutipan program]

Kalau yang dihitung lebih dulu adalah **10 + 4**, maka hasil akhirnya adalah **14 * 2** atau **28**. Ini bukan kenyataanya. Operator perkalian di C# memiliki prioritas yang lebih tinggi dibanding operator penjumlahan (sebagaimana aritmatika). Urutan prioritasnya ada di tabel berikut:

| Urutan/Prioritas | Operator | Pengerjaan |
|---|---|---|
| 1 | unary + - | dari kanan ke kiri |
| 2 | * / % | dari kiri ke kanan |
| 3 | + - | dari kiri ke kanan |

Pengerjaan dari kiri ke kanan maksudnya jika terdapat operasi **1 + 2 - 3 + 4** maka yang akan dihitung pertama kali adalah **1 + 2**, lalu hasilnya dikurangi **3**, dan terakhir ditambah **4**.

Prioritas pengerjaan dapat kita ubah dengan menggunakan kurung buka dan kurung tutup. Kita lihat contoh penggunaannya dalam program berikut:

[program lengkap]
```
// Program 4.2 - Penggunaan kurung untuk mengubah hasil suatu perhitungan

using System;

class Kurung
{
    static void Main()
    {
        Console.WriteLine(10 + 5 * 2  + 4); // hasilnya 24
        Console.WriteLine((10 + 5) * 2  + 4); // hasilnya 34
        Console.WriteLine(10 + 5 * (2  + 4)); // hasilnya 40
        Console.WriteLine((10 + 5) * (2  + 4)); // hasilnya 90
    }
}
```
[/program lengkap]

Outputnya adalah:

[console]
```
24
34
40
90
```
[/console]

Prioritas operator yang lengkap bisa dilihat di lampiran C.

[latihan mini]
- Mengapa **char** merupakan type integer yang spesial!
- Berapakah **-1 + 2 * 5**?
- Bagaimanakah cara mengontrol prioritas pengerjaan operator?

- Sebab **char** digunakan untuk merepresentasikan karakter.
- **9**.
- Dengan menggunakan pasangan kurung buka dan kurung tutup misalnya dalam **(1 + 2) * 5**.

[/latihan mini]

## 4.2.2 Penggunaan Variabel

Variabel sangatlah penting dalam pemrograman sebab variabel dapat digunakan untuk menyimpan data dan nilainya dapat berubah-rubah. Secara teknis variabel adalah suatu lokasi di memory yang digunakan untuk menyimpan data. Kita dapat dengan mudah mengaksesnya karena kita bisa memberikan nama untuk variabel. Suatu variabel harus memiliki nama dan type. Cara mendeklarasikan suatu variabel adalah sebagai berikut:

[fomat]
*[type] [nama]*;
[/fomat]

Misalnya jika kita ingin mendeklarasikan sebuah **int** (integer yang bisa menyimpan nilai -2,147,483,648 sampai 2,147,483,647) yang bernama **harga**, caranya adalah sebagai berikut:

[kutipan program]
```
int harga;
```
[/kutipan program]

Setalah dideklarasikan, barulah kita dapat menggunakannya di dalam program kita.

[catatan]
Nama variabel termasuk identifier, yang aturannya ada di 3.9.
[/catatan]

## 4.2.3 Operator =

Operator yang sangat penting dalam penggunaan variabel adalah operator = (biasa disebut operator assignment). Perlu diingat bahwa operator ini bukanlah seperti operator persamaan dalam matematika. Yang dilakukan operator ini adalah memasukkan nilai di kanan operator ke dalam variabel di sebelah kiri operator. Operand di sebelah kiri harus berupa sebuah variabel. Operand di sebelah kanan dapat berupa literal, variabel, perhitungan, pemanggilan fungsi, maupun kombinasinya.

Kita coba bermain-main dengan variabel di program berikut:

[program lengkap]
```
// Program 4.3 - Penggunaan variabel

using System;

class Variabel
{
    static void Main()
    {
        int permen; // mendeklarasikan variabel permen yang typenya int
        int anak; // mendeklarasikan variabel anak yang typenya int
        permen = 13; // memasukkan nilai 13 ke permen
        anak = 3; // memasukkan nilai 3 ke anak
        Console.Write("jumlah permen:");
        Console.WriteLine(permen);
        Console.Write("jumlah anak: ");
        Console.WriteLine(anak);
        Console.Write("permen untuk setiap anak: ");
        Console.WriteLine(permen / anak);
        Console.Write("sisa permen: ");
```

```
            Console.WriteLine(permen % anak);
        }
}
```
[/program lengkap]

Program di atas akan mengeluarkan output sebagai berikut:

[console]
```
jumlah permen:13
jumlah anak: 3
permen untuk setiap anak: 4
sisa permen: 1
```
[/console]

Sebelum sebuah variabel dapat dipakai, variabel teresebut harus dideklarasikan dan harus memiliki nilai. Jika kita mengcomment salah satu dari statement berikut:

[kutipan program]
```
int permen; // mendeklarasikan variabel permen yang typenya int
```
[/kutipan program]

[kutipan program]
```
permen = 13; // memasukkan nilai 13 ke permen
```
[/kutipan program]

... program tidak akan dapat dicompile.

Variabel-variabel yang typenya sama dapat dideklarasikan dalam satu statement dengan format sebagai berikut:

[fomat]
*[type] [nama1], [nama2], [nama3], /* ... , */ [nama-n];*
[/fomat]

Contohnya 2 statement pertama dalam program 4.3 dapat kita singkat menjadi:

[kutipan program]
```
int permen, anak;
```
[/kutipan program]

Dalam mendeklarasikan sebuah variabel kita juga dapat langsung memberikan nilai kepadanya. Formatnya adalah sebagai berikut:

[fomat]
*[type] [nama] = [nilai];*
[/fomat]

Contohnya:

[kutipan program]
```
int permen = 13;
```
[/kutipan program]

Pendeklarasian banyak variabel dan pemeberian nilai juga dapat berada dalam satu statement misalnya:

[kutipan program]
```
int permen = 13, anak = 3, permenTiapAnak = (permen / anak), sisa;
```
[/kutipan program]

Dalam kutipan di atas variabel **permen** dan **anak** diberi nilai menggunakan literal, variabel **permenTiapAnak** diberi nilai menggunakan perhitungan yang melibatkan variabel lain, sedangkan variabel **sisa** tidak diberi nilai. Penggunaan tanda kurung dalam kutipan di atas hanya untuk mempermudah pembacaan.

Operator = memiliki prioritas yang terendah di C#, jadi kalau ada statement semacam:

[kutipan program]
```
jumlahPeserta = pria + wanita;
```
[/kutipan program]

... maka yang pertama kali akan dilaksanakan adalah perhitungan **pria + wanita**.

Operator = dikerjakan dari kanan ke kiri yang bisa kita manfaatkan, contohnya dalam kutipan berikut:

[kutipan program]
```
pria = wanita = 25;
```
[/kutipan program]

Yang pertama kali dikerjakan adalah memasukkan nilai **25** ke **wanita**. Setelah itu nilai **wanita** (**25**) dimasukkan ke **pria**. Pada akhirnya **pria** dan **wanita** akan sama-sama bernilai **25**. Sebagai perbandingan statement di atas sama artinya dengan statement berikut:

[kutipan program]
```
pria = (wanita = 25);
```
[/kutipan program]

## 4.2.4 Operator op=

Dalam pemrograman kita akan sering mengubah nilai suatu variabel berdasarkan nilai variabel tersebut. Contohnya jika kita ingin menambahkan satu ke variabel kita, kita dapat menulis statement berikut:

[kutipan program]
```
int permen = 4;
```

```
permen = permen + 1; // menambahkan 1 ke dalam permen
```
[/kutipan program]

Ingat bahwa operator = di sini tidak berarti sebuah persamaan matematika! Yang akan dilakukan pada statement ke dua adalah mengambil nilai **permen** yaitu **4**, menambahkannya dengan **1** (sehingga hasilnya **5**), dan memasukkan hasilnya ke dalam variabel permen. Pada akhirnya permen akan bernilai **5**.

Kita bisa menyingkat statement ke dua menjadi:

[kutipan program]
```
permen += 1;
```
[/kutipan program]

Perlu diperhatikan bahwa diantara + dan = tidak boleh ada pemisah.

Kita bisa menggunakan **op=** untuk berbagai operator, inilah daftar lengkapnya (beberapa operator belum kita pelajari):

```
+=  -=  *=  /=  %=  &=  |=  ^=  <<=  >>=
```

Operator **op=** juga memiliki prioritas yang terendah (sama dengan operator =), jadi statement di bawah:

[kutipan program]
```
permen *= 2 + 1;
```
[/kutipan program]

... sama dengan:

[kutipan program]
```
permen *= (2 + 1); // atau: permen = permen * (2 + 1);
```
[/kutipan program]

## 4.2.5 Operator Increment dan Decrement

Kita akan sering menambah atau mengurangi nilai variabel kita dengan **1**, karena itu di C# terdapat operator yang khusus untuk hal tersebut yaitu operator increment dan decrement. Dalam kutipan program di bawah:

[kutipan program]
```
permen = permen + 1;
permen += 1;
++permen; // bentuk prefix, operator ditulis sebelum operand
permen++; // bentuk postfix, operator ditulis sebelum operand
```
[/kutipan program]

... kempat statement tersebut melakukan hal yang sama. Begitu juga dengan keempat statement di bawah ini:

[kutipan program]
```
permen = permen - 1;
permen -= 1;
--permen; // bentuk prefix, operator ditulis sebelum operand
permen--; // bentuk postfix, operator ditulis sebelum operand
```
[/kutipan program]

Terdapat sedikit perbedaan antara bentuk prefix dengan bentuk postfix dari operator ++ dan --. Keduanya akan mengubah nilai operandnya, namun bentuk prefix akan mengembalikan nilai operand setelah dilakukan operasi sedangkan bentuk postfix akan mengembalikan nilai operand sebelum dilakukan operasi. Nilai yang dikembalikan itu tidak dipakai dalam kutipan-kutipan program di atas, namun sekarang coba kita perhatikan program berikut:

[program lengkap]
```
// Program 4.4 - Prefix dan postfix

using System;

class PrefixPostfix
{
    static void Main()
    {
        int var1 = 5;
        int var2 = 5;

        ++var1; // menambahkan 1 ke var1
        var2++; // menambahkan 1 ke var2

        Console.WriteLine(var1); // output 6
        Console.WriteLine(var2); // output 6

        Console.WriteLine(++var1); // output 7
        Console.WriteLine(var2++); // output 6

        Console.WriteLine(var1); // output 7
        Console.WriteLine(var2); // output 7
    }
}
```
[/program lengkap]

Ini adalah outputnya:

[console]
```
6
6
7
6
7
7
```

[/console]

Dua statement ini perlu kita analisa lebih lanjut:

[kutipan program]
```
Console.WriteLine(++var1); // output 7
Console.WriteLine(var2++); // output 6
```
[/kutipan program]

Sebelumnya **var1** dan **var2** sama-sama bernilai **6**. Pada statement pertama **var1** nilainya ditambahkan **1** (menjadi **7**), dan **7** akan menjadi argument bagi **Console.WriteLine()**. Namun di statement ke dua walaupun nilai **var2** juga dinaikkan dari **6** menjadi **7**, yang dikembalikan untuk menjadi argument bagi **Console.WriteLine()** adalah nilai **var2** sebelum dilakukan penambahan yaitu **6**. Bukti bahwa pada akhirnya keduanya bernilai **7** ada di pada output.

Ini contoh lainnya:

[kutipan program]
```
int var1 = 5;
int var2 = ++var1 * 2; // var2 akan bernilai 6 * 2
int var3 = 5;
int var4 = var3++ * 2; // var4 akan bernilai 5 * 2
```
[/kutipan program]

Setelah statement-statement di atas dilaksanakan, **var1** dan **var3** akan bernilai **6**, **var2** akan bernilai **12**, dan **var4** akan bernilai **10**.

Penggunaan operator increment dan decrement sebaiknya tidak berlebihan sebab selain menyulitkan orang untuk membacanya, juga dapat membingungkan compiler! Misalnya kutipan program di bawah ini:

[kutipan program]
```
var3 = var1++ + ++var2;
```
[/kutipan program]

... tidak dapat dipahami compiler. Penggunaan kurung akan menyelesaikan masalah:

[kutipan program]
```
var3 = (var1++) + (++var2);
```
[/kutipan program]

Sekarang lihat contoh di bawah ini:

[kutipan program]
```
var2 = (var1++) + (++var1) * (var1++);
```
[/kutipan program]

Walaupun sah-sah saja, penggunaan incrementer dan decrementer sebaiknya dibatasi, sekali untuk setiap variabel dalam suatu statement. Kalau tidak kode kita akan menjadi sulit dimengerti, bahkan oleh kita sendiri!

[latihan mini]

- Tuliskan cara mendeklarasikan variabel bernama **foo** yang memiliki type **byte**!
- Adakah yang salah dalam statement berikut?

  [kutipan program]
  ```
  int a;
  Console.WriteLine(a = 100);
  ```
  [/kutipan program]

- Adakah yang salah dalam statement berikut?

  [kutipan program]
  ```
  Console.WriteLine(int a = 100);
  ```
  [/kutipan program]

- **byte foo;**
- Tidak ada. **100** akan dimasukkan ke **a** dan **a** akan menjadi argument bagi **Console.WriteLine()**.
- Ya. Kita tidak boleh mendeklarasikan variabel baru dalam pemanggilan suatu method.

[/latihan mini]

# 4.3 Floating Point

Type floating point dapat digunakan untuk merepresentasikan bilangan pecahan misalnya **1.5** dan **-0.001**. Terdapat dua type yang bisa kita pilih yaitu **float** dan **double**. Jika terdapat literal angka yang mengandung bilangan fraksional maka type literal tersebut secara default adalah **double**.

| Type | Jarak (positif dan negatif) | Bit / Byte yang Digunakan |
|---|---|---|
| float | $1.5 * 10^{-45}$ sampai $3.4 * 10^{38}$ | 32 bit / 4 byte |
| double | $5 * 10^{-324}$ sampai $1.7 * 10^{308}$ | 64 bit / 8 byte |

Batas terkecil di tabel tersebut maksudnya adalah fraksional terkecil yang bisa direpresentasikan. Tentu saja 0 juga bisa direpresentasikan oleh floating point.

Selain menuliskan literal floating point dalam bentuk biasanya, kita juga bisa menuliskannya dengan format mantisa + exponen. Untuk format tersebut karakter **e** atau **E** digunakan. Contohnya **1.01e2** sama dengan $\mathbf{1.01 * 10^2}$ atau **101**, dan **5e1** sama dengan $\mathbf{5 * 10^1}$ atau **50**. Tidak boleh ada pemisah antara mantisa, **e**, dan eksponennya.

Hal-hal yang telah kita bahas sebelumnya di bab ini yang berhubungan dengan inte berlaku juga untuk floating point. Mari kita pakai floating point dalam program berikut:

[program lengkap]
```
// Program 4.5 - Penggunaan floating point
```

```
using System;

class FloatingPoint
{
    static void Main()
    {
        double panjang = 100.4;
        double lebar = 32.3;
        double anak = 5;
        Console.WriteLine("Luas tanah: ");
        Console.WriteLine(panjang * lebar);
        Console.WriteLine("Luas tanah untuk tiap anak:");
        Console.WriteLine(panjang * lebar / anak);
    }
}
```
[/program lengkap]

Outputnya adalah sebagai berikut:

[console]
```
Luas tanah:
3242.92
Luas tanah untuk tiap anak:
648.584
```
[/console]

Ada beberapa hal baru yang bisa kita ketahui. Pertama, jika kedua operand dari operator binary adalah floating point, maka hasilnya juga adalah floating point. Kedua, Method **WriteLine()** dapat menerima floating point sebagai argumentnya (begitu juga dengan method **Write()**).

# 4.4 decimal

Type floating point memiliki beberapa kelemahan dalam ketepatannya yang berhubungan dengan cara kerja internalnya. Kadang-kadang kesalahan dapat terjadi, contohnya ada di program berikut:

[program lengkap]
```
// Program 4.6 - Memperagakan kelemahan floating point

using System;

class KelemahanFloating
{
    static void Main()
    {
        Console.WriteLine(1.1 - 1.09); // seharusnya 0.01
        Console.WriteLine(2.1 - 2.09); // seharusnya 0.01
```

```
        Console.WriteLine(1.1 + 0.1); // seharusnya 1.2
        Console.WriteLine(1.1 + 0.00000000000000001);
        // seharusnya 1.10000000000000001
    }
}
```
[/program lengkap]

Hasilnya tidak seperti yang kita harapkan, perhatikan baris ke 2 dan 4:

```
[console]
0.01
0.0100000000000002
1.2
1.1
[/console]
```

Kesalahan-kesalahan tersebut mungkin begitu kecil dan biasanya dapat ditoleransi, namun bagaimana kalau program yang kita buat berhubungan dengan perhitungan uang? Tentunya kesalahan sekecil apapun tidak dapat diterima.

Disitulah type **decimal** berperan. Type ini besarnya 128 bit atau 16 byte, dan dapat merepresentasikan positif/negatif **1 * $10^{-28}$** sampai **7.9 * $10^{28}$**. Jaraknya jauh lebih kecil dari **float** dan **double**, namun tidak akan ada pembulatan dan kesalahan-kesalahan perhitungan lain yang ada di type floating point. Untuk menulis literal yang typenya **decimal**, kita harus membubuhi **m** atau **M** di belakang angkanya. Program 4.6 akan kita tulis ulang menggunakan **decimal**:

[program lengkap]
```
// Program 4.7 - Menggunakan decimal

using System;

class MenggunakanDecimal
{
    static void Main()
    {
        Console.WriteLine(1.1m - 1.09m); // seharusnya 0.01
        Console.WriteLine(2.1m - 2.09m); // seharusnya 0.01

        Console.WriteLine(1.1m + 0.1m); // seharusnya 1.2
        Console.WriteLine(1.1m + 0.00000000000000001m);
        // seharusnya 1.10000000000000001
    }
}
```
[/program lengkap]

Hasilnya sempurna:

```
[console]
0.01
0.01
```

```
1.2
1.10000000000000001
```
[/console]

[latihan mini]

- Sebutkan 2 type floating point yang ada di C#!
- Sebutkan keunggulan dan kelemahan type **decimal** dibandingkan dengan **float** atau **double**!

- **float** dan **double**.
- Type **decimal** memiliki ketepatan yang sangat tinggi dan tidak akan ada pembulatan dalam operasi yang melibatkan type **decimal**. Di lain pihak type **decimal** membutuhkan 128 bit, jangkauannya lebih rendah, dan operasi yang melibatkan type decimal membutuhkan waktu yang lebih lama.

[/latihan mini]

# 4.5 char

char digunakan untuk merepresentasikan 1 karakter. C# tidak menggunakan standar ASCII tetapi menggunakan Unicode. Unicode besarnya 16 bit dan mendefinisikan semua karakter yang ada di berbagai bahasa manusia beserta macam-macam simbol. Untuk menulis literal karakter, kita harus mengurung karakternya dengan ‘ dan ‘. Lihat contoh berikut:

[program lengkap]
```
// Program 4.8 - Karakter

using System;

class Karakter
{
    static void Main()
    {
        char h = ‘h’;
        char a = ‘a’;
        char l = ‘l’;
        char o = ‘o’;
        Console.Write(h);
        Console.Write(a);
        Console.Write(l);
        Console.Write(o);
        Console.WriteLine(‘!’);

    }
}
```
[/program lengkap]

Ini outputnya:

[console]

halo!
[/console]

Baris baru (new line), huruf-huruf arab, dan banyak karakter lain secara umum cukup sulit untuk ditulis secara normal. Untuk menyelesaikan masalah tersebut kita bisa menggunakan escape sequence heksadesimal atau escape sequence unicode.

Cara menggunakan escape sequence heksadesimal adalah menuliskan **\x** diikuti kode Unicodenya dalam heksadesimal. Contohnya karakter **a** memiliki kode Unicode **97** atau **0x61**. Ini berarti kita bisa menulis **'\x61'** sebagai alternatif dari **'a'**.

Escape sequence Unicode dimulai dengan **\u** atau **\U** dan ditulis menggunakan salah satu format berikut:

[format]
\u*[angka][angka][angka][angka]*
\U*[angka][angka][angka][angka]*
\u*[angka][angka][angka][angka][angka][angka][angka][angka]*
\U*[angka][angka][angka][angka][angka][angka][angka][angka]*
[/format]

Contohnya adalah **'\u0061'** yang sama artinya dengan **'\x61'** dan **'a'**.

Escape sequence Unicode dapat digunakan untuk menuliskan identifier, sedangkan escape sequence heksadesimal tidak. Mekanisme "keyword sebagai identifier" juga dapat dilaksanakan dengan menggunakan escape sequence Unicode, seperti dalam kutipan berikut:

[kutipan program]
```
class cl\u0061ss // cl\u0061ss sama artinya dengan dengan @class
```
[/kutipan program]

[catatan]
Yang dimaksud angka dalam format penulisan escape sequence Unicode (dan heksadesimal) juga mencakup abjad **a**-**f** dan **A**-**F**.
[/catatan]

Beberapa karakter malah memiliki escape sequence spesial, seperti ditunjukkan dalam tabel berikut:

| **Escape Sequence** | **Kode Unicode** | **Arti** |
|---|---|---|
| \a | 0x7 atau 7 | Alert (membunyikan suara) |
| \b | 0x8 atau 8 | Backspace |
| \f | 0xc atau 12 | Form Feed |
| \n | 0xa atau 10 | New line (baris baru) |
| \r | 0xd atau 13 | Kembali ke awal baris |
| \t | 0x9 atau 9 | Tabulasi horizontal |
| \v | 0xb atau 11 | Tabulasi vertikal |
| \0 | 0x0 atau 0 | Null |

| | | |
|---|---|---|
| \' | 0x27 atau 39 | Tanda kutip |
| \" | 0x22 atau 34 | Tanda kutip ganda |
| \\ | 0x5c atau 92 | Backslash (garis miring kiri) |

Penggunaannya ada di program berikut:

[program lengkap]
```
// Program 4.9 - Escape sequence

using System;

class EscapeSequence
{
    static void Main()
    {
        char tandaPetik = '\"';
        char tandaKutip = '\'';
        char barisBaru = '\n';
        Console.Write(tandaPetik);
        Console.Write("halo.....");
        Console.Write(tandaPetik);
        Console.Write(barisBaru);
        Console.Write(tandaKutip);
        Console.Write("halo.....");
        Console.Write(tandaKutip);
    }
}
```
[/program lengkap]

Outputnya adalah sebagai berikut:

[console]
```
"halo....."
'halo.....'
```
[/console]

Daftar karakter-karakter Unicode beserta kodenya ada di [[x]].

## 4.6 Literal string

Apakah anda menyadari atau tidak, sebenarnya kita telah menggunakan literal sejak program 2.1. Literal yang kita gunakan adalah literal **string**, contohnya **"halo"**. Literal string dapat berisi nol atau lebih karakter, dan untuk menulisnya kita harus mengurungnya dengan **"** dan **"**. Ini beberapa contoh literal **string** yang sah:

```
" Halo "        "kalimat 1\nkalimat2"        "dia \"Dewi\""
```

Literal **string** dapat dituliskan dalam bentuk verbatim. Di literal **string** verbatim escape sequence tidak dikenal dan literal **string** verbatim dapat ditulis dalam lebih dari

satu baris. Untuk membuat literal **string** verbatim kita perlu menuliskan **@** sebelum stringnya. Untuk menuliskan tanda petik (**“**) dalam literal **string** verbatim, kita harus menulis tanda petik dua kali (**“”**). Lihat contoh program di bawah ini:

[program lengkap]
```
// Program 4.10 - Verbatim

using System;

class Verbatim
{
    static void Main()
    {
        // literal string verbatim digunakan dan memenuhi lebih dari 1 baris
        Console.WriteLine(@"Baris 1
Baris 2
Baris 3");
        // di string biasa kita harus menggunakan karakter ‘\n’
        Console.WriteLine("Baris 1\nBaris 2\nBaris 3");
        // ingat, di literal string verbatim escape sequence tak dikenal
        Console.WriteLine(@"Baris 1\nBaris 2\nBaris 3");
    }
}
```
[/program lengkap]

Ini adalah outputnya:

[console]
```
Baris 1
Baris 2
Baris 3
Baris 1
Baris 2
Baris 3
Baris 1\nBaris 2\nBaris 3
```
[/console]

**string** juga merupakan suatu type, namun pembahasan mendalamnya tidak akan dilakukan di bab ini karena **string** merupakan reference type.

[latihan mini]
- Apakah sistem/standar karakter yang digunakan oleh C#? Apa type yang digunakan untuk merepresentasikannya?
- Tuliskan literal karakter yang berisi new line!
- Apakah type dari literal berikut: **“yo!”**?

- C# menggunakan Unicode yang besarnya 16 bit. Type yang bersangkutan adalah **char**.
- **‘\n’**.
- string.

[/latihan mini]

# 4.7 Jenis Literal

Seperti telah disebutkan, literal interger misalnya **100** typenya secara default adalah **int**. Namun jika nilai dari literal integer itu di luar batas **int**, maka literal integer tersebut typenya adalah **uint**, **long**, atau **ulong**. Lihat kutipan program di bawah ini:

[kutipan program]
```
// 1 adalah literal int
Console.WriteLine(1);
// 2000000000 literal int juga
Console.WriteLine(2000000000);
// literal berikut di luar batas int tetapi di dalam batas uint, typenya adalah uint
Console.WriteLine(4000000000);
// literal berikut di luar batas int dan uint, typenya adalah long
Console.WriteLine(-4000000000);
// literal berikut juga long karena di luar batas int dan uint
Console.WriteLine(5000000000);
// literal berikut adalah ulong karena hanya ulong yang mampu menampungnya
Console.WriteLine(10000000000000000000);
```
[/kutipan program]

Ada cara untuk merubah tipe literal integer. Jika kita membubuhkan **u** atau **U** di akhir angka, maka typenya akan menjadi **uint**. Jika **l** atau **L** dibubuhkan typenya akan menjadi **long**. Pembubuhan **ul**, **UL**, **uL**, **Ul**, **lu**, **LU**, **lU**, atau **Lu** akan mengubah typenya menjadi **ulong**. Tentu saja batas type tetap tidak bisa dilanggar. Contohnya **5** adalah **int** tapi **5u** adalah **uint**.

Telah disebutkan juga bahwa literal floating point secara default memiliki type **double**. Untuk mengubah literal floating point dan integer menjadi **float** kita dapat membubuhkan **f** atau **F**. Literal integer dapat dipaksa menjadi **double** dengan membubuhkan **d** atau **D**.

[catatan]
Berbagai type angka yang ada merupakan suatu fleksibilitas. Misalnya jika kita memerlukan integer untuk menyimpan jumlah anak dalam suatu keluarga, maka **byte** dapat kita pakai sebab tidak mungkin suatu keluarga memiliki anak di bawah **0** dan kemungkinan jumlah anak suatu keluarga di atas **255** sangat kecil. Selain hanya menghabiskan memory sebesar 1 byte, operasi penjumlahan pada **byte** tentu akan lebih cepat dibanding jika kita melakukan operasi yang sama pada type yang lebih "berat" misalnya **long**. Penggunaan **float**, **double**, atau **decimal** untuk keperluan di atas dapat dianggap tidak perlu sebab tidak mungkin jumlah anak berupa bilangan yang memiliki fraksional.
[/catatan]

# 4.8 Konversi Otomatis

Coba perhatikan kutipan di bawah ini:

[kutipan program]
```
float x = 1;
double y = x;
```
[/kutipan program]

**1** adalah literal **int**, namun **x** adalah variabel **float**. **y** sendiri adalah **double**. Apakah program akan dapat dicompile? Jawabannya ya. Pada operasi assignment, C# dapat melakukan konversi otomatis asalkan kedua tipenya kompatibel dan tipe tujuan lebih besar. Sebagai contoh, **int** lebih besar dari **byte** (**int** dapat menyimpan **-2,147,483,648** sampai **2,147,483,647** sementara **byte** hanya mungkin menyimpan **0** sampai **255**) sedangkan **ulong** tidak lebih besar dari **sbyte** sebab **ulong** tidak dapat menyimpan angka-angka negatif.

Dalam kutipan program di atas literal **1** secara otomatis diubah menjadi **float** sebelum dimasukkan dalam **x** dan nilai dari **x** juga secara otomatis dikonversi menjadi **double** sebelum dimasukkan ke dalam **y**.

# 4.9 Konversi Manual (Cast)

Konversi manual atau cast dapat dilakukan walaupun ada beberapa konsekuensinya. Jika suatu floating point dikonversi menjadi integer, maka bagian fraksionalnya akan hilang. Lalu jika nilai yang ingin dikonversi berada diluar batas type tujuan, maka informasi juga akan hilang. Cara melakukan cast adalah dengan menuliskan type tujuan di dalam kurung. Kita lihat contoh program berikut:

[program lengkap]
```
// Program 4.11 - Cast

using System;

class Cast
{
    static void Main()
    {
        double x = 300.5;
        short y = (short) x;
        byte z = (byte) y;
        Console.WriteLine(x);
        Console.WriteLine(y);
        Console.WriteLine(z);
    }
}
```
[/program lengkap]

Inilah outputnya:

[console]

```
300.5
300
44
```
[/console]

Kita bisa melihat bahwa karena nilai **x** (**300**) berada di luar wilayah **byte** (0 sampai 255), maka hasil yang diperoleh pun kacau. Perlu diketahui semua konversi baik otomatis maupun melalui cast tidak akan mengubah nilai variabelnya sendiri. Buktinya nilai **x** saat ditulis ke layar tetap **300.5** dan nilai **y** tetap **300**.

[tanya jawab]
**Q:** OK, hasilnya kacau, tapi kenapa **300** menjadi **44**?
**A:** Ini berhubungan dengan bit-bit yang dihilangkan. **300** (**short**) dalam binary adalah **0000 0001 0010 1100**. Sementara short memakan 16 bit, **byte** hanya menggunakan 8 bit. Saat **short** dikonversi menjadi **byte**, 8 bit atasnya terpaksa dihilangkan sehingga **0000 0001 0010 1100** menjadi ~~0000 0001~~ **0010 1100** yang nilai desimalnya **44**.
[/tanya jawab]

Dengan konversi manual kita bisa mengubah char menjadi bilangan dan sebaliknya. Perhatikan contoh berikut:

[program lengkap]
```
// Program 4.11 - Cast menjadi/ke char

using System;

class CastChar
{
    static void Main()
    {
        Console.WriteLine("Kode Unicode 67 karakternya adalah:");
        Console.WriteLine((char)67);
        Console.WriteLine("Karakter 'b' kode Unicodenya adalah:");
        Console.WriteLine((int)'b');
    }
}
```
[/program lengkap]

Ini adalah outputnya:

[console]
```
Kode Unicode 67 karakternya adalah:
C
Karakter `b' kode Unicodenya adalah:
98
```
[/console]

Kita bisa lihat bahwa cast tidak hanya terbatas pada assignment saja.
Ada kasus-kasus khusus yang perlu diketahui:

[kutipan]
```
byte x = 1;
```
[/kutipan]

Walaupun **1** adalah literal yang typenya **int** dan **byte** lebih kecil dari **int**, kita tidak perlu melakukan cast. Compiler akan melakukan pengecekan untuk memastikan bahwa nilai yang ingin dimasukkan berada di dalam batas **byte**. Ini berlaku untuk assignment kepada **ulong**, **uint**, dan type lain yang lebih kecil dari **int**. Sekarang perhatikan contoh di bawah ini:

[kutipan program]
```
int x = 1
byte y = x; // harus menggunakan: byte y = (int) x;
```
[/kutipan program]

Program di atas tidak akan dapat dicompile karena **x** merupakan variabel **int** yang bisa berisi nilai di luar batas **byte**.

Integer dapat dikonversi secara otomatis menjadi **decimal** namun tidak sebaliknya. **float** dan **double** tidak kompatibel dengan **decimal** dan memerlukan cast jika ingin melakukan konversi. **char** walaupun secara internal merupakan sebuah integer, dianggap spesial dan tidak kompatibel dengan type lainnya.

[catatan]
Terdapat beberapa hal yang tidak konsisten dalam C#. Contohnya ...

[kutipan program]
```
float f = 1.0; // error! harus menggunakan cast atau menuliskan 1.0f
```
[/kutipan program]

... tidak diizinkan karena **1.0** typenya **double**, tetapi...

[kutipan program]
```
byte b = 1; // boleh asal nilainya berada di dalam batas byte
```
[/kutipan program]

... diizinkan padahal **1** typenya adalah **int** dan **byte** lebih kecil daripada **int**. Seharusnya ditentukan suatu aturan bahasa yang konsisten.
[/catatan]

# 4.10 Perbedaan Type dalam Expression

Operator, variabel, dan literal adalah elemen-elemen sebuah expression. Dalam sebuah expression kita diperbolehkan mencampuradukkan type asalkan mereka kompatibel. Contohnya **int** dengan **double** dapat berada di dalam suatu expression. Jika terdapat lebih dari dua type dalam suatu expression, maka akan terjadi pengkonversian nilai operand secara otomatis dengan aturan sebagai berikut:

1. Kalau salah satu operand adalah **decimal**, maka operand lain akan diubah menjadi **decimal** (jika operand lain adalah **float** atau **double** maka program tidak bisa dicompile).
2. Jika tidak, kalau salah satu operand adalah **double**, maka operand lain akan diubah menjadi **double**.
3. Jika tidak, kalau salah satu operand adalah **float**, maka operand lain akan diubah menjadi **float**.
4. Jika tidak, kalau salah satu operand adalah **ulong**, maka operand lain akan diubah menjadi **ulong** (jika operand lain adalah **sbyte**, **short**, **int**, atau **long** maka program tidak bisa dicompile).
5. Jika tidak, kalau salah satu operand adalah **long**, maka operand lain akan diubah menjadi **long**.
6. Jika tidak, kalau salah satu operand adalah **uint** dan operand lainnya adalah **sbyte**, **short**, atau **int**, maka kedua operand akan diubah menjadi **long**.
7. Jika tidak, kalau salah satu operand adalah **uint**, maka operand lain akan diubah menjadi **uint**
8. Jika tidak, Kedua operand akan diubah menjadi **int**.

Konversi tersebut akan dilakukan operasi per operasi. Untuk mencampur type yang tidak kompatibel dalam suate expression (misalnya antara **double** dan **decimal**), cast dapat dipakai.

Lalu perhatikan aturan terakhir. Berdasarkan aturan tersebut, operand **char**, **byte**, **sbyte**, **short**, dan **ushort** akan diubah menjadi **int**. Karena itu hasil terkecil dari sebuah expression pastilah **int**. Ini membutuhkan cast di beberapa situasi, contohnya pada kutipan program berikut:

[kutipan program]
```
byte a = 1;
byte b = 2;
byte c = (byte) (a + b);
```
[/kutipan program]

**a** dan **b** sama-sama berupa **byte**, namun dalam expression **a + b** keduanya akan diubah menjadi **int**. Karena **c** adalah **byte**, maka cast untuk mengembalikan **a + b** dari **int** menjadi **byte** diperlukan. Tanda kurung pada **(a + b)** diperlukan sebab tanpa tanda kurung tersebut yang dicast menjadi **byte** hanyalah variabel **a** (yang memang sudah **byte**).

Untuk mengakhiri subbab ini penulis akan memberikan suatu program lagi:

[program lengkap]
```
// Program 4.12 - Cast int ke double

using System;

class IntKeDouble
{
    static void Main()
```

```
    {
        int jeruk = 20;
        int anak = 9;
        Console.WriteLine("Jeruk untuk tiap anak:");
        Console.WriteLine((double) jeruk / anak);
    }
}
```
[/program lengkap]

Output:

[console]
```
Jeruk untuk tiap anak:
2.22222222222222
```
[/console]

Jika kita menuliskannya berbeda...

[kutipan program]
```
Console.WriteLine((double) (jeruk / anak));
```
[/kutipan program]

... maka hasilnya akan berbeda. Kenapa?

Jika kita mengurung **jeruk / anak**, maka **jeruk / anak** akan dihitung terlebih dahulu. Karena **jeruk** dan **anak** sama-sama **int** maka hasilnya adalah **int** (**2**). Selanjutnya **2** dicast ke **double**, hasilnya tetap saja **2** (walaupun typenya menjadi **double**).

Sebaliknya, tanpa kurung antara **jeruk / anak**, maka nilai **jeruk** akan dicast menjadi **double**. Karena salah satu operand pembagian tersebut adalah **double**, maka nilai dari **anak** secara otomatis akan dikonversi menjadi **double**, dan hasil pembagiannya juga **double** (**2.22222222222222**).

[catatan]
Konversi otomatis dapat terjadi jika kita menggunakan operator unary **-**. Operand yang lebih kecil dari **int** (**byte**, **sbyte**, **short**, **ushort**, dan **char**) akan diubah ke **int**. **uint** akan diubah menjadi **long**.
[/catatan]

[latihan mini]
- Deteksi hal yang tidak diperlukan dalam kutipan program berikut ini:

  [kutipan program]
  ```
  int int1 = 100, int2;
  char ch1 = 'a', ch2;
  int2 = (int) ch1;
  ch2 = (char) int1;
  ```
  [/kutipan program]

- Apakah type terkecil yang mungkin dari hasil operasi yang melibatkan integer?
- Apakah type dari hasil perhitungan berikut: **(byte) 100 / 2.5**?

- Pada statement ke 3, kita sebenarnya tidak perlu melakukan cast manual. **char** akan otomatis terkonversi menjadi **int**.
- **int**.
- Hasilnya adalah **double**.

[/latihan mini]

# 4.11 Type dari Class Library .NET

Sebenarnya dari tadi kita menggunakan type-type yang terdefinisi di class library .NET. Nama seperti **int**, **double**, dan yang lainnya hanyalah nama yang spesifik dengan C#. Tabel berikut akan menuliskan nama-nama asli dari type-type yang telah kita pelajari:

| Keyword C# | Nama di Class Library .NET |
|---|---|
| char | System.Char |
| byte | System.Byte |
| sbyte | System.SByte |
| short | System.Int16 |
| ushort | System.UInt16 |
| int | System.Int32 |
| uint | System.UInt32 |
| long | System.Int64 |
| ulong | System.UInt64 |
| float | System.Single |
| double | System.Double |
| decimal | System.Decimal |

[tanya jawab]
**Q:** Apakah type-type tersebut juga merupakan **class,** sebagaimana **System.Console**?
**A:** Type-type tersebut adalah **struct** yang mirip dengan **class** di banyak hal. Seperti **System.Console** yang memiliki method static **Write()** dan **WriteLine()**, type-type tersebut juga memiliki method-method yang sebentar lagi akan kita pelajari. Anda juga sekarang tahu bahwa **class**/**struct** dapat menyimpan data (misalnya **int** menyimpan data yang berupa angka).
[/tanya jawab]

Walaupun tidak biasa dilakukan, kita bisa menggunakan nama aslinya dalam program kita:

[program lengkap]
```
// Program 4.13 - Menggunakan nama-nama .NET

using System;

class NamaNamaDotNet
```

```
{
    static void Main()
    {
        int a = 50;
        Int32 b = 100;
        System.Int32 c = a + b; // menggunakan nama lengkap
        Console.WriteLine(c);
    }
}
```
[/program lengkap]

Ingat bahwa kita tidak perlu menuliskan nama namespacenya yaitu **System** karena di program tersebut sudah ada statement **using System**. Namun penulisan nama namespacenya tetap diperbolehkan.

# 4.12 Operasi-Operasi Ilegal

Jika anda melakukan operasi-operasi yang ilegal seperti membagi suatu integer atau decimal dengan **0**, maka program akan melempar exception. Jika anda mencoba melewati batas suatu variabel, maka akan terjadi overflow dan nilainya akan kembali dari batas lainnya. Lihat contoh program berikut:

[program lengkap]
```
// Program 4.14 - Program akan crash!!!

using System;

class OperasiIlegal
{
    static void Main()
    {
        byte x = 0; // 0 adalah batas minimum byte
        x - = 2; // karena melewati batas bawah, nilai x kembali dari atas
        Console.WriteLine(x); // output: 254
        x = 255; // 255 adalah batas maksimum byte
        x++; // karena melewati batas atas, nilai x kembali dari bawah
        Console.WriteLine(x); // output: 0
        int y = 1 / x; // mencoba membagi suatu angka dengan 0
        Console.WriteLine("saya tidak akan pernah tampil");
    }
}
```
[/program lengkap]

Ini outputnya:

[console]
```
254
0

Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
```

```
   at OperasiIllegal.Main()
```
[/console]

Kita bisa lihat bahwa statement terakhir tidak akan pernah dilaksanakan. Program yang keluar karena exception (dan sebab-sebab tak terduga lainnya) biasa dikatakan crash. Penanganan exception akan dibahas di bab tersendiri.

[catatan]
Untuk floating point, pembagian dengan **0** akan menjadikan nilainya NaN (not a number). Overflow pada floating point akan menyebabkan nilainya menjadi infinity.
[/catatan]

# 4.13 Panduan Penamaan

Dalam memberikan nama **class**, disarankan anda menggunakan pascal casing yaitu menggunakan huruf besar pada setiap karakter yang mengawali kata. Contoh identifier yang menggunakan pascal casing:

```
KendaraanBermotor    BinatangMelata    Buku    PapanPermainan
```

Untuk variabel yang berada dalam suatu method, anda disarankan menggunakan camel casing. Camel casing mirip dengan pascal casing namun karakter pertama menggunakan huruf kecil. Ini beberapa contoh camel casing:

```
permen    jumlahAnak    tingkatResiko    penggemarProsesorAMD    x
```

Ini sesuai dengan panduan penamaan yang terdapat di standar C#.

[tes]
## A. Pertanyaan

1. Sebutkan type dari literal-literal di bawah ini!

   ```
   0    'h'    "a"    50.98    100ul
   ```

2. Apakah kegunaan operator increment (**++**) dan decrement (**--**)? Apakah perbedaan antara penggunaan prefix (misal **++x**) dengan postfix (misal **x++**)?

3. Apakah arti escape sequence **\n** dan **\t**?

4. Dari berbagai type angka, manakah yang tidak kompatibel dengan decimal?

5. Apakah syarat sebelum suatu variabel dapat digunakan?

6. Apakah yang salah dengan kutipan program berikut?

[kutipan program]
```
int umur = 12, double berat = 30.5;
```
[/kutipan program]

7. Berapakah nilai akhir dari **var1**, **var2**, dab **var3** pada kutipan program berikut?

   [kutipan program]
   ```
   int var1 = 5, var2 = 10, var3 = 3;
   var1 += var2 * var3--;
   ```
   [/kutipan program]

8. Apakah konsekuensi mengecast **int** menjadi **double**? Bagaimana kalau sebaliknya?

## B. Membuat Program

1. Buatlah suatu program yang menghitung luas suatu persegi panjang! Panjang persegi panjang tersebut adalah **11** dan lebarnya **6.12**.

2. Buatlah sebuah program yang menghitung berapa kaki + inci yang ada dalam **193** inci! Contohnya **13** inci adalah **1** kaki + **1** inci. (**1** kaki sama dengan **12** inci)

## C. Debugging

1. Program di bawah ini bertujuan meghitung jumlah murid yang ada di sebuah sekolah. Hasilnya tidak sesuai dengan yang diharapkan. Betulkanlah!

   [program lengkap]
   ```
   // Tes 3.C.1

   using System;

   class JumlahMurid
   {
    static void Main()
    {
         byte kelas1 = 50;
         byte kelas2 = 42;
         byte kelas3 = 56;
         byte kelas4 = 47;
         byte kelas5 = 28;
         byte kelas6 = 55;
         // harus dicast menjadi byte sebab hasil penjumlahan ini adalah int
         byte totalMurid = (byte) kelas1 + kelas2 + kelas3 + kelas4 + kelas5 +
   kelas6;
         Console.WriteLine("Total Murid:");
         Console.WriteLine(totalMurid);
    }
   }
   ```
   [/program lengkap]

2. Program di bawah ini bertujuan untuk menghitung total biaya yang harus dibayar per tahun. Biaya listrik dan air tiap bulan masing-masing adalah **100** dan **30**. Program ini menampilkan jawaban **460** yang tidak mungkin adalah jawaban yang benar. Biaya listrik untuk setahun saja sudah **100 * 12 = 1000**. Cari kesalahannya dan betulkan!

[program lengkap]
```
// Tes 3.C.2

using System;

class Biaya
{
 static void Main()
 {
      int biayaListrikPerBulan = 100;
      int biayaAirPerBulan = 30;
      int biayaPerTahun = biayaListrikPerBulan + biayaAirPerBulan * 12;
      Console.WriteLine("Total biaya:");
      Console.WriteLine(biayaPerTahun);
 }
}
```
[/program lengkap]

3. Program di bawah ini bertujuan untuk menampilkan **'baris 1'**, **'baris2'**, dan **'baris3'** dalam 3 baris yang berbeda. Modifikasi program sampai mendapatkan hasil yang diinginkan!

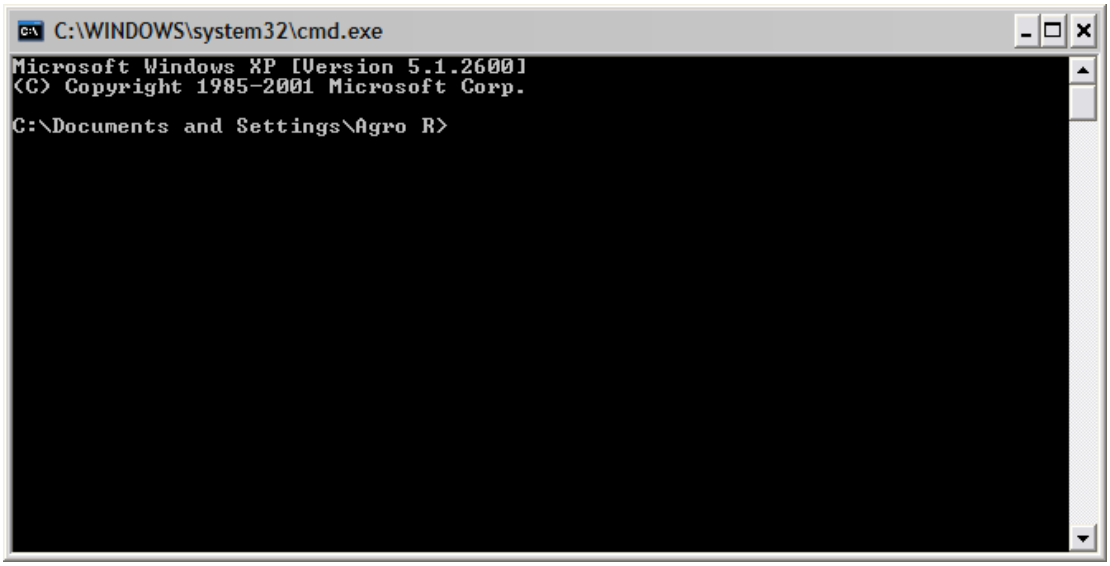
[program lengkap]
```
// Tes 3.C.3

using System;

class BarisBaris
{
 static void Main()
 {
      Console.WriteLine(@"'baris 1'\n'baris 2'\n'baris 3");
 }
}
```
[/program lengkap]
[/tes]

# Bab 5: Input dan Output Console

Di bab ini kita tidak akan mempelajari hal baru yang spesifik dengan C#. Kita akan lebih mendalami class **Console** yang merupakan bagian dari class library .NET. Kemahiran menggunakan class tersebut akan sangat membantu sebab kita akan membuat program-program console sampai akhir buku ini! Class ini juga akan tetap berguna saat kita membuat program dengan GUI, sebab console merupakan alat yang handal untuk debugging.

# 5.1 Console

Apakah sebenarnya console itu? Coba perhatikan gambar berikut:




Gambar 5.1 - Console

Window di atas memuat sebuah console yang pasti sudah tidak asing lagi di mata anda. Console adalah lingkungan di mana kita berinteraksi dengan teks. Program yang berjalan di dalam lingkungan console disebut program console.

Framework .NET menyediakan class yang bernama **Console** di dalam namespace **System**. Class **Console** tersebut dapat melakukan input/output console sederhana. Selama ini kita telah memepelajari cara penggunaan sederhana **Write()** dan **WriteLine()** untuk output. Di bab ini kita akan lebih mendalami kedua method tersebut dan mempelajari sebuah method baru yaitu **ReadLine()**.

# 5.2 String Format dalam Write() dan WriteLine()

Seringkali kita ingin menuliskan beberapa hal sekaligus di layar. Pemanggilan **Write()** atau **WriteLine()** berulang-ulang akan sangat tidak efisien. Kita dapat menggunakan string format untuk keperluan tersebut. Inilah contoh penggunaannya:

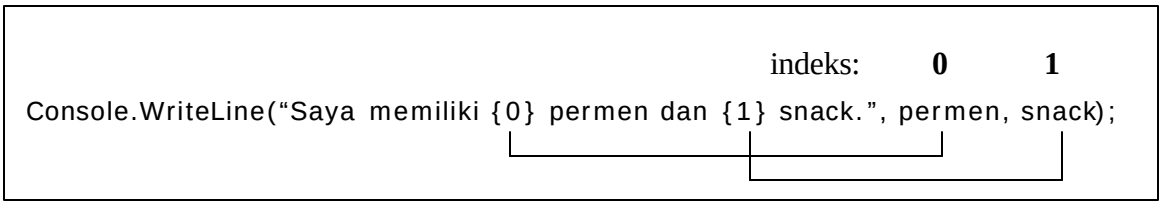
[kutipan program]

```
Console.WriteLine("Saya memiliki {0} permen dan {1} snack.", permen, snack);
```
[/kutipan program]

Kita bisa perhatikan bahwa **WriteLine()** dipanggil menggunakan tiga argument yang dipisahkan oleh koma. Argument pertama adalah sebuah literal **string** yang merupakan string format. Di dalam string format tersebut terdapat placeholder yaitu **"{0}"** dan **"{1}"**. Saat program dijalankan, **"{0}"** akan diganti dengan objek yang menempati indeks ke 0, dan **"{1}"** akan diganti dengan objek yang menempati indeks ke 1. Jika **permen** adalah **int** yang bernilai **10** dan **snack** adalah **int** yang bernilai **8**, output program tersebut adalah sebagai berikut:

[console]
```
Saya memiliki 10 permen dan 8 snack.
```
[/console]

Objek yang indeksnya 0 adalah objek yang menjadi argument ke 2. Lihat bagan berikut:




Ini contoh lain penggunaannya:

[kutipan program]
```
Console.WriteLine("{0} jeruk dan {1} permen harganya {2} rupiah.", jeruk, permen, (jeruk * hargaJeruk) + (permen * hargaPermen));
```
[/kutipan program]

Jika **jeruk**, **permen**, **hargaJeruk**, dan **hargaPermen** adalah variabel-variabel **int** yang nilainya **5**, **15**, **500**, dan **100**, maka outputnya adalah sebagai berikut:

[console]
```
5 jeruk dan 15 permen harganya 4000 rupiah.
```
[/console]

Dalam contoh di atas 4 argument digunakan. Argument ke 4 merupakan sebuah expression. Penggunaan tanda kurung dalam expression tersebut hanya untuk menjelaskan penulisan saja.

[catatan]
Jika string format tidak dalam bentuk yang benar maka exception akan dilempar. Contohnya adalah menuliskan indeks negatif.
[/catatan]

## 5.2.1 Indentasi

Perhatikan kutipan program di bawah ini:

[kutipan program]
```
Console.WriteLine("output: {0} ...", 101);
Console.WriteLine("output: {0} ...", 3);
Console.WriteLine("output: {0} ...", 55);
Console.WriteLine("output: {0} ...", 1024);
Console.WriteLine("output: {0} ...", 768);
```
[/kutipan program]

Outputnya adalah sebagai berikut:

[console ]
```
output: 101 ...
output: 3 ...
output: 55 ...
output: 1024 ...
output: 768 ...
```
[/console]

Kita dapat merapikannya dengan indentasi dalam string format. Cara penggunaan indentasi adalah seperti berikut:

[format]
{*[indeks]*,*[lebar]*}
[/format]

***[lebar]*** menunjukkan lebar minimum yang digunakan untuk indentasi. Inilah program sebelumnya namun dengan menggunakan indentasi:

[kutipan program]
```
Console.WriteLine("output: {0,4} ...", 101);
Console.WriteLine("output: {0,4} ...", 3);
Console.WriteLine("output: {0,4} ...", 55);
Console.WriteLine("output: {0,4} ...", 1024);
Console.WriteLine("output: {0,4} ...", 768);
```
[/kutipan program]

Outputnya adalah sebagai berikut (spasi sengaja diganti dengan garis bawah agar efeknya lebih tampak):

[console ]
```
output: _101 ...
output: ___3 ...
output: __55 ...
output: 1024 ...
output: _768 ...
```

[/console]

Jika ***[lebar]*** bernilai negatif, maka argument yang diformat akan diratakan di sebelah kiri. Ingat bahwa ***[lebar]*** menunjukkan nilai minimum. Kalau panjang argument yang diformat melebihi ***[lebar]***, argument tidak akan terpotong. Contohnya kutipan program berikut:

[kutipan program]
```
Console.WriteLine("{0,4} ...", 1);
Console.WriteLine("{0,4} ...", 1234567890); // melebihi lebar indentasi yaitu 4
```
[/kutipan program]

Ini outputnya:

[console ]
```
   1
1234567890
```
[/console]

## 5.2.2 Mengatur Output Bilangan Pecahan

Kadang-kadang kita ingin hanya menampilkan beberapa angka di belakang koma untuk bilangan-bilangan pecahan. Untuk keperluan tersebut kita gunakan salah satu dari format berikut:

[format]
```
{[indeks]:[0 sebanyak x kali].[0 sebanyak y kali]}
{[indeks]:#.[# sebanyak x kali]}
{[indeks]:[0 sebanyak x kali].[# sebanyak y kali]}
```
[/format]

Jika menggunakan format pertama, koma dan **0** yang tidak diperlukan akan selalu ditunjukkan. Format kedua hanya menampilkan koma dan bilangan fraksional bila diperlukan. Perlu diingat bahwa format kedua tidak akan menampilkan apa-apa bila bilangan bulatnya **0**. Format ketiga akan selalu menampilkan ***[x]*** digit bulat, namun hanya menampilkan komponen fraksional bila diperlukan. Perhatikan kutipan program berikut:

[kutipan program]
```
Console.WriteLine("{0:00.00}", 1.21854); // output 01.22
Console.WriteLine("{0:0.##}", 1.21854); // output 1.22
Console.WriteLine("{0:#.##}", 1.219854); // output 1.22
Console.WriteLine();
Console.WriteLine("{0:00.00}", 3.1); // output 03.10
Console.WriteLine("{0:0.##}", 3.1); // output 3.1
Console.WriteLine("{0:#.##}", 3.1); // output 3.1
Console.WriteLine();
Console.WriteLine("{0:0.00}", 0); // output 0.00
```

```
Console.WriteLine("{0:#.##}", 0); // output kosong
Console.WriteLine("{0:0.##}", 0); // output 0
Console.WriteLine();
Console.WriteLine("{0:00.00}", 0.12); // output 00.12
Console.WriteLine("{0:#.##}", 0.12); //output .12
Console.WriteLine("{0:0.##}", 0.12); //output 0.12
Console.WriteLine();
Console.WriteLine("{0:00.00}", 1999); // output 1999.00
Console.WriteLine("{0:0.##}", 1999); // output 1999
Console.WriteLine("{0:#.##}", 1999); // output 1999
Console.WriteLine();
Console.WriteLine("{0,10:#.##}", 1.29854); // menggunakan indentasi
```
[/kutipan program]

Ini outputnya:

[console]
```
01.22
1.22
1.22

03.10
3.1
3.1

0.00

0

00.12
.12
0.12

1999.00
1999
1999

      1.3
```
[/console]

## 5.2.3 Format Lain-Lain

Tabel berikut dapat dijadikan referensi untuk beberapa bentuk format lainnya. Ingat bahwa indentasi dapat dipakai bersamaan dengan format-format berikut.

| string format | argument ke 0 | hasil | catatan |
|---|---|---|---|
| "{0:x}" | 1022 | "3fe" | menampilkan bentuk heksadesimal suatu integer menggunakan huruf kecil |
| "{0:X}" | 1023 | "3FE" | menampilkan bentuk heksadesimal |

| | | | |
|---|---|---|---|
| | | | suatu integer menggunakan huruf besar |
| “{0:c}” atau “{0:C}” | 5040 | “$5,040.00” | menampilkan angka dalam bentuk uang |
| “{0:p}” atau “{0:P}” | 0.25 | “25.00 %” | menampilkan angka dalam bentuk persen |
| “{0:e}” | 125 | “1.250000e+002” | menampilkan angka dalam bentuk ilmiah menggunakan huruf **e** kecil |
| “{0:E}” | 125 | “1.250000E+002” | menampilkan angka dalam bentuk ilmiah menggunakan huruf **E** besar |
| “{0:e*[x]*}” contoh nyatanya “{0:e1}” | 125 | “1.3e+002” | menampilkan angka dalam bentuk ilmiah menggunakan huruf **e** kecil dan ***[x]*** angka di belakang koma |
| “{0:E*[x]*}” contoh nyatanya “{0:E2}” | 125 | “1.25E+002” | menampilkan angka dalam bentuk ilmiah menggunakan huruf **E** besar dan ***[x]*** angka di belakang koma |

Yang telah dibahas barulah sebagian kecil dari format yang dapat dibuat, tapi merupakan bentuk-bentuk yang paling sering dipakai.

[latihan mini]

- Dimulai dari angka berapakah indeks yang digunakan dalam string format?
- Apa yang terjadi kalau output melebihi lebar indentasi yang ditentukan?
- Apa output dari statement berikut?

  [kutipan program]
  ```
  Console.WriteLine("{2}{0}{1}{0}", 'a', 'y', 's');
  ```
  [/kutipan program]

- Dimulai dari **0**.
- Output tidak akan terpotong dan ditampilkan seperti biasanya.
- **“saya”**.

[/latihan mini]

# 5.3 Parse()

Pembahasan akan kita alihkan sejenak dari class **Console**. Seperti kita ketahui, type-type yang telah dipakai seperti **int** dan **double** juga merupakan sejenis **class**. Type-type dasar yang didefinisikan Framework .NET memiliki method static yang bernama **Parse()**. Method tersebut menerima sebuah argument **string** dan mengembalikan hasil olahannya. Hasil olahannya berupa type yang bersangkutan, dengan nilai yang terkandung di dalam **string** argument. Untuk lebih jelasnya perhatikan program berikut:

[program lengkap]
```
// Program 5.1 - Parse()
```

```
using System;

class PenggunaanParse
{
    static void Main()
    {
        // baris di bawah (dicomment) tidak sah karena
        // string tidak kompatibel dengan type lainnya
        // int a = "100";
        // cast juga tidak dapat dilakukan

        // Parse() akan mengembalikan 100 (int)
        int a = int.Parse("100");

        // Parse() akan mengembalikan 3.5 (double)
        double b = double.Parse("3.5");

        // Parse() akan mengembalikan 0.01 (decimal)
        decimal c = decimal.Parse("0.01");

        // Parse() akan mengembalikan 'a' (char)
        char d = char.Parse("a");

        Console.WriteLine(a);
        Console.WriteLine(b);
        Console.WriteLine(c);
        Console.WriteLine(d);
    }
}
```
[/program lengkap]

Inilah outputnya:

[console]
```
100
3.5
0.01
a
```
[/console]

Jika string yang diberikan tidak sah misalnya **“100???”**, exception akan dilempar.

# 5.4 ReadLine()

Method static **ReadLine()** dari class **Console** memungkinkan kita menuliskan input sampai enter ditekan. Input yang kita berikan akan dikembalikan ke program dalam bentuk **string**. **string** yang dikembalikan tersebut dapat kita jadikan argument bagi **Parse()**. Perhatikan contoh program berikut:

[program lengkap]
```
// Program 5.2 - Input

using System;

class Input
{
    static void Main()
    {
        int var1, var2;
        Console.WriteLine("Program Penjumlahan");
        Console.WriteLine();
        Console.Write("Masukkan angka pertama: ");
        var1 = int.Parse(Console.ReadLine());
        Console.Write("Masukkan angka kedua: ");
        var2 = int.Parse(Console.ReadLine());
        Console.WriteLine();
        Console.WriteLine("Jumlahnya adalah {0}.", var1 + var2);
    }
}
```
[/program lengkap]

**Console.ReadLine()** akan mengembalikan suatu **string**, kita anggap **string** itu adalah ***[suatu string]***. Method **int.Parse()** sendiri menerima **string** sebagai argument, contoh pemanggilannya **int.Parse(*[suatu string]*)**. Karena itu kita bisa menggabungkannya sebagai **int.Parse(Console.ReadLine())**. Ini contoh outputnya (input digarisbawahi):

[console]
```
Program Penjumlahan

Masukkan angka pertama: 100
Masukkan angka kedua: 30

Jumlahnya adalah 130.
```
[/console]

Dengan **ReadLine()** dan **Parse()**, kita dapat membuat program-program yang interaktif.

**ReadLine()** juga bisa kita gunakan sebagai cara untuk menghentikan alur program secara sementara. Perhatikan contoh berikut:

[program lengkap]
```
// Program 5.3 - Pause Menggunakan ReadLine()

using System;

class Pause
{
    static void Main()
```

```
    {
        Console.WriteLine("Pencet enter untuk melanjutkan...");
        Console.ReadLine(); // string yang dikembalikan tidak dipakai
        Console.WriteLine("Sampai jumpa!");
    }
}
```
[/program lengkap]

Inilah contoh outputnya:

[console ]
```
Pencet enter untuk melanjutkan...
kenapa harus enter?
Sampai jumpa!
```
[/console]

Dalam contoh tersebut, **string** yang dikembalikan yaitu **"kenapa harus enter?"** tidak dipakai lebih lanjut. Jika pengguna langsung memencet enter, maka **string** yang akan dikembalikan adalah **""** (juga tidak dipakai).

[tes]

## A. Pertanyaan

1. Class apakah yang digunakan untuk berinteraksi dengan console? Terletak di namespace manakah class tersebut? Apakah class tersebut merupakan bagian dari C#?

2. Apakah yang akan ditampilkan di layar bila statement di bawah dijalankan?

   [kutipan program]
   ```
   Console.WriteLine("{0}{0}{0}{0}", "\b\b\b123");
   ```
   [/kutipan program]

3. Apakah yang dikembalikan oleh method **ReadLine()**?

4. Bagaimanakah cara membuat type dasar berdasarkan nilai yang terkandung dalam suatu **string**?

## B. Membuat Program

1. Buatlah program yang menampilkan karakter dari sebuah kode Unicode beserta kode Unicodenya dalam heksadesimal. Misalnya jika pengguna menuliskan **97** maka program menampilkan **a** dan **61** (karakter **a** kode Unicodenya adalah **97**).

2. Buatlah program yang menanyakan 3 buah angka. Tampilkan jumlah dan rata-ratanya, dan gunakan hanya sebuah variabel dalam program ini!

3. Kecepatan cahaya adalah 300,000,000 m/s. Buatlah sebuah program yang menghitung jarak yang ditempuh cahaya dalam suatu selang waktu. Gunakan notasi ilmiah untuk outputnya dengan menunjukkan 3 angka di belakang koma pada mantisanya.

## C. Debugging

1. Program berikut bertujuan menampilkan jumlah pajak yang harus dibayar. Besarnya pajak adalah 5% dari penghasilan. Program tersebut tidak berjalan dengan seharusnya. Betulkan!

[program lengkap]

```
// Tes 5.C.1

using System;

class Pajak
{
 static void Main()
 {
      decimal penghasilan;
      decimal persentasePajak = 5 / 100;
      Console.WriteLine("Berapakah penghasilan anda per tahun?");
      penghasilan = decimal.Parse(Console.ReadLine());
      Console.WriteLine("Anda harus membayar pajak sebesar (0) Rupiah.",
penghasilan * persentasePajak);
 }
}
```

[/program lengkap]

[/tes]

# Bab 6: Mengubah Alur Program

Di program-program yang telah kita buat selama ini, statement-statement dilaksanakan berurutan. Program akan mulai dari statement pertama method **Main()**, dilanjutkan dengan statement ke dua, ke tiga, dan seterusnya. Di bab ini kita akan mempelajari beberapa konstruksi bahasa yang dapat mengubah alur program berdasarkan kondisi tertentu.

# 6.1 bool

**bool** atau **System.Boolean** merupakan type yang hanya dapat bernilai benar atau salah. Literal bool yang melambangkan benar adalah **true**, dan literal bool yang melambangkan salah adalah **false**. **bool** tidak dapat dikonversi ke type lainnya. Ini contoh program yang menggunakan **bool**:

[program lengkap]
```
// Program 6.1 - Penggunaan bool

using System;

class BenarDanSalah
{
    static void Main()
    {
        bool SukaCSharp = true;
        Console.WriteLine("Apakah program ini suka C#?");
        Console.WriteLine(SukaCSharp);
        Console.WriteLine("Apakah anda suka C# (true/false)?");
        SukaCSharp = bool.Parse(Console.ReadLine());
        Console.WriteLine("Anda menjawab {0}.", SukaCSharp);
    }
}
```
[/program lengkap]

Inilah contoh outputnya:

[console]
```
Apakah program ini suka C#?
True
Apakah anda suka C# (true/false)?
true
Anda menjawab True.
```
[/console]

# 6.2 Operator-Operator Relational

Untuk dapat membuat keputusan kita harus mengetahui kondisinya. Misalnya "jika nilai ulangan di atas 8, hadiah diberikan". Dalam kasus tersebut kita mengecek apakan nilai ulangan lebih besar dari 8. Jawaban dari perbandingan tersebut pasti ya atau tidak. Seperti itulah perbandingan bekerja di C#.

Untuk melakukan perbandingan kita menggunakan operator-operator relational. Semua operator tersebut sifatnya binary. Ada 6 operator relational yaitu:

| **Operator** | **Arti** |
|---|---|
| == | sama dengan |
| != | tidak sama dengan |
| > | lebih besar dari |
| < | lebih kecil dari |
| >= | lebih besar dari atau sama dengan |
| <= | lebih kecil dari atau sama dengan |

[catatan]
Untuk perbandingan kita menggunakan dua karakter sama dengan (==). Hati-hati sebab penggunaannya sering tertukar dengan assignment (=).
[/catatan]

Jika perbandingannya benar hasilnya adalah **true**, sendangkan jika perbandingannya salah hasilnya adalah **false**.

Misalnya **x** dan **y** adalah variabel integer. Contoh-contoh perbandingan yang dapat dilakukan adalah sebagai berikut:

| | | | | |
|---|---|---|---|---|
| x == y | x < y + 15 | x + y != 100 | 100 < x | x - 5 >= y / 2 |

Operator-operator relational prioritasnya lebih rendah dari operator-operator aritmatika sehingga dalam **x < y + 15**, yang lebih dahulu dievaluasi adalah **y + 15**. Mari kita gunakan beberapa operator relational dalam program berikut:

[program lengkap]
```
// Program 6.2 - Operator Relational

using System;

class OperatorRelational
{
    static void Main()
    {
        int nilai1, nilai2;
        Console.Write("Masukkan angka pertama: ");
        nilai1 = int.Parse(Console.ReadLine());
        Console.Write("Masukkan angka ke dua: ");
        nilai2 = int.Parse(Console.ReadLine());
        Console.WriteLine("Apakah {0} lebih kecil dari {1}?", nilai1, nilai2);
        Console.WriteLine(nilai1 < nilai2);
        Console.WriteLine("Apakah {0} sama dengan {1}?", nilai1, nilai2);
        Console.WriteLine(nilai1 == nilai2);
    }
}
```
[/program lengkap]

Ini contoh outputnya:

[console]
```
Masukkan angka pertama: 25
Masukkan angka ke dua: 10
Apakah 25 lebih kecil dari 10?
False
Apakah 25 sama dengan 10?
False
```
[/console]

Perbandingan yang melibatkan type angka lain juga dapat dilakukan. Untuk perbandingan **bool**, hanya operator == dan **!=** yang dapat digunakan.

[latihan mini]
- Apakah nilai yang dapat dimiliki oleh type **bool**?
- Apakah perbedaan antara operator = dan ==?
- Apakah nilai dari expression x > x?

- **true** dan **false**.
- Operator = adalah operator assignment, berfungsi untuk memasukkan nilai operand kanan ke operand kiri. Operator == digunakan untuk membandingkan kedua operandnya. Jika sama maka hasilnya adalah **true**, jika tidak hasilnya **false**.
- Nilai expressionnya **false** karena tidak mungkin **x** lebih besar dari **x**. Sudah pasti bahwa x sama dengan **x**.

[/latihan mini]

# 6.3 Statement if

Dengan menggunakan statement **if**, statement berikutnya hanya akan dijalankan jika kondisinya **true**. Inilah format statement **if**:

[format]
```
if([kondisi])
    [statement]
```
[/format]

[catatan]
Indentasi sebaiknya dimajukan saat menulis tubuh dari statement **if**.
[/catatan]

Simak kutipan program berikut:

[kutipan program]
```
if(nilai1 == nilai2)
    Console.WriteLine("Kedua angka sama nilainya.");
Console.WriteLine("Program selesai.");
```
[/kutipan program]

Jika **nilai1** dan **nilai2** sama, maka outputnya adalah seperti ini:

[console]
```
Kedua angka sama nilainya.
Program selesai.
```
[/console]

Jika tidak inilah outputnya:

[console]
```
Program selesai.
```
[/console]

### 6.3.1 Statement if Kosong

Anda tidak perlu memberi titik koma setelah menuliskan **if(*[kondisi]*)**. Jika anda memberinya, anda malah akan membuat statement **if** kosong! Contohnya, kutipan di bawah:

[kutipan program]
```
if(x = 100); // ada titik koma di sini
    Console.WriteLine("x nilainya 100");
```
[/kutipan program]

... sama dengan:

[kutipan program]
```
if(x = 100)
    ; // ada titik koma di sini, terbentuk statement if kosong
Console.WriteLine("x nilainya 100");
```
[/kutipan program]

Dengan statement **if** kosong seperti dicontohkan di atas, penulisan **"x nilainya 100"** akan selalu terjadi. Fenomena tersebut juga menjelaskan bahwa statement kosong termasuk statement yang sah, seperti kutipan berikut:

[kutipan program]
```
; ; ; ; ; // 5 statement kosong, sah-sah saja
```
[/kutipan program]

## 6.4 Penggunaan Blok

Kita bisa lihat bahwa statement **if** hanya mempengaruhi satu statement yang mengikutinya. Bagaimana kalau kita ingin statement **if** mempengaruhi lebih dari statu statement? Untuk keperluan tersebut kita harus membuat blok baru.

Blok baru dalam method dapat dibuat dengan menggunakan **{** dan **}**. Blok digunakan untuk mengelompokkan statement-statement menjadi suatu unit logis. Blok dapat berada di dalam blok lain. Ini contoh penggunaannya:

[program lengkap]
```
// Program 6.3 - Penggunaan blok
// (hanya untuk demonstrasi, tidak ada kegunaannya dalam program ini)

using System;

class PenggunaanBlok
{
    static void Main()
    {
        { // Blok 1 dimulai
            Console.WriteLine("Kalimat 1.");
            Console.WriteLine("Berada dalam blok 1.");
            { // Blok 2 (berada di dalam blok lain)
                Console.WriteLine("Kalimat 2.");
                Console.WriteLine("Di dalam blok 2.");
            } // Akhir blok 2
        } // Akhir dari blok 1
        Console.WriteLine("Di luar.");
    }
}
```
[/program lengkap]

Outputnya:

[console]
```
Kalimat 1.
Berada dalam blok 1.
Kalimat 2.
Di dalam blok 2.
Di luar.
```
[/console]

Blok dapat digunakan bersamaan dengan statement **if**. Contohnya adalah sebagai berikut:

[program lengkap]
```
// Program 6.4 - Penggunaan blok dengan if

using System;

class BlokDenganIf
{
    static void Main()
    {
        double kecepatan, jarak;
        Console.WriteLine("Berapakah kecepatan (dalam m/s)?");
```

```
            kecepatan = double.Parse(Console.ReadLine());
            if(kecepatan <= 0)
            {
                Console.WriteLine("Kecepatan akan diubah menjadi 0.1 m/s");
                kecepatan = 0.1;
            }
            Console.WriteLine("Berapakah jarak (dalam m)?");
            jarak = double.Parse(Console.ReadLine());
            if(jarak < 0)
            {
                Console.WriteLine("Jarak akan diubah menjadi 0 m");
                jarak = 0;
            }
            Console.WriteLine("Waktu yang dibutuhkan: {0:0.##} s", jarak /
kecepatan);
        }
}
```
[/program lengkap]

Jika angka yang dimasukkan baik, maka kedua statement **if** akan dilewati. Outputnya akan seperti ini:

[console ]
```
Berapakah kecepatan (dalam m/s)?
9.8
Berapakah jarak (dalam m)?
100
Waktu yang dibutuhkan: 10.2 s
```
[/console]

Kasus lainnya:

[console ]
```
Berapakah kecepatan (dalam m/s)?
10
Berapakah jarak (dalam m)?
-100
Jarak akan diubah menjadi 0 m
Waktu yang dibutuhkan: 0 s
```
[/console]

Karena user memasukkan nilai negatif untuk **jarak**, maka statement **if** berikut...

[kutipan program]
```
if(jarak < 0)
{
    Console.WriteLine("Jarak akan diubah menjadi 0 m");
    jarak = 0;
}
```
[/kutipan program]

... akan dimasuki. Semua statement yang ada di dalam blok akan dijalankan.

### 6.4.1 Variabel dan Blok

Variabel memiliki masa hidup tertentu, tergantung pada tempat variabel tersebut dideklarasikan. Jika ada blok baru dalam suatu method, maka semua variabel yang telah ada di blok sebelumnya juga dapat dipakai di blok baru tersebut. Variabel yang dideklarasikan di suatu blok tidak dikenal diluar blok tersebut. Untuk lebih jelasnya perhatikan kutipan program berikut:

[kutipan program]
```
static void Main()
{
    char a = 'a';
    // di sini hanya a yang bisa digunakan
    Console.WriteLine(a);
    //Console.WriteLine("{0} {1}", b, c); // error bila comment dihilangkan
    { // blok baru dimulai
        char b = 'b';
        // di sini a dan b bisa digunakan
        Console.WriteLine("{0} {1}", a, b);
        // Console.WriteLine(c); // error bila comment dihilangkan
    } // akhir blok, b mati tidak dikenal di luar blok ini
    char c = 'c';
    // di sini a dan c bisa digunakan
    Console.WriteLine("{0} {1}", a, c);
    // Console.WriteLine(b); // error bila comment dihilangkan
} // akhir method Main(), di luar method ini a dan c tidak dikenal
```
[/kutipan program]

Ouputnya, yang mungkin sudah dapat anda tebak:

[console]
```
a
a b
a c
```
[/console]

# 6.5 if di dalam if

Kita tinjau kembali format untuk statement **if**:

[format]
```
if([kondisi])
    [statement]
```
[/format]

***[statement]*** di sini bisa saja berupa statement **if** lagi. Kita lihat contoh berikut:

[program lengkap]
```
// Program 6.5 - if di dalam if

using System;

class IfDalamIf
{
    static void Main()
    {
        int sebuahAngka;
        Console.Write("Masukkan angka di antara 0 dan 100: ");
        sebuahAngka = int.Parse(Console.ReadLine());
        if(sebuahAngka > 0)
            if(sebuahAngka < 100)
                Console.WriteLine("Terima kasih...");
    }
}
```
[/program lengkap]

Ini contoh outputnya:

[console]
```
Masukkan angka di antara 0 dan 100: 80
Terima kasih...
```
[/console]

# 6.6 Statement if-else

Statement **if** yang selama ini kita buat tidak melaksanakan apa-apa bila ***[kondisi]*** nilainya **false**. Kita dapat menggunakan gabungan **else** untuk menentukan statement yang harus dilaksanakan bila ***[kondisi]*** nilainya **false**. Perhatikan contoh berikut:

[program lengkap]
```
// Program 6.6 - if dan else

using System;

class IfElse
{
    static void Main()
    {
        int sebuahAngka;
        Console.Write("Masukkan sebuah angka: ");
        sebuahAngka = int.Parse(Console.ReadLine());
        if(sebuahAngka % 2 == 0) //apakah sebuahAngka habis dibagi 2?
            Console.WriteLine("Angka genap!"); // jika ya ini dijalankan
        else
            Console.WriteLine("Angka ganjil!"); // jika tidak ini dijalankan
```

```
    }
}
```
[/program lengkap]

Ini contoh output dari program tersebut:

[console]
```
Masukkan sebuah angka: 31
Angka ganjil!
```
[/console]

Blok juga dapat digunakan dalam statement **if-else**.

## 6.6.1 if-else di Dalam if

Jika ada **else**, maka pasangan **if** nya adalah **if** bebas yang terdekat dengannya. Yang dimaksud bebas adalah **if** tersebut belum memiliki pasangan **else**. Perhatikan contoh berikut:

[kutipan program]
```
if(penghasilanOrtu < syarat1) // if pertama
    if(nilai > syarat 2) // if ke dua
        Console.WriteLine("Anda mendapatkan beasiswa!");
else
    Console.WriteLine("Beasiswa ini hanya untuk keluarga yang tak mampu...");
```
[/kutipan program]

Jika maksud pemrogram adalah memasangkan **else** dengan **if** pertama, maka programnya salah. **else** yang ada di program tersebut berpasangan dengan **if** ke dua sebab itulah **if** bebas yang terdekat dengannya. Blok dapat digunakan untuk mengubahnya:

[kutipan program]
```
if(penghasilanOrtu < syarat1) // if pertama
{
    if(nilai > syarat 2) // if ke dua
        Console.WriteLine("Anda mendapatkan beasiswa!");
}
else // kali ini berpasangan dengan if pertama
    Console.WriteLine("Beasiswa ini hanya untuk keluarga yang tak mampu...");
```
[/kutipan program]

Sayangnya tidak akan ada pesan yang ditampilkan bila nilai anak tidak memenuhi syarat. Program dapat kita sempurnakan:

[kutipan program]
```
if(penghasilanOrtu < syarat1)
{
    if(nilai > syarat 2) // if ke dua
```

```
            Console.WriteLine("Anda mendapatkan beasiswa!");
        else // berpasangan dengan if ke dua
            Console.WriteLine("Maaf, nilai anda tidak cukup...");
}
else
    Console.WriteLine("Beasiswa ini hanya untuk keluarga yang tak mampu...");
```
[/kutipan program]

Dalam program di atas penggunaan blok adalah opsional sebab **if** ke dua sudah mendapat pasangan yang seharusnya.

## 6.6.2 if atau if-else di dalam if-else

Statement **if** atau **if-else** dapat saja mengikuti **else**. Perhatikan contoh berikut:

[kutipan program]
```
if(nomor == 100232)
    Console.WriteLine("Anda mendapat mobil!");
else
    if(nomor == 302)
        Console.WriteLine("Anda mendapat motor!");
    else
        if(nomor == 994530243)
            Console.WriteLine("Anda mendapat radio!")
        else
            Console.WriteLine("Anda belum beruntung...")
```
[/kutipan program]

Jika kita melakukan banyak perbandingan, indentasi akan semakin ke kanan dan program menjadi cukup sulit dibaca. Inilah bentuk indentasi yang sering dipakai jika **if** langsung mengikuti **else**:

[kutipan program]
```
if(nomor == 100232)
    Console.WriteLine("Anda mendapat mobil!");
else if(nomor == 302)
    Console.WriteLine("Anda mendapat motor!");
else if(nomor == 994530243)
    Console.WriteLine("Anda mendapat radio!")
else
    Console.WriteLine("Anda belum beruntung...")
```
[/kutipan program]

[tanya jawab]
**Q:** Misalnya **kondisi1** dan **kondisi2** adalah bool, apakah beda antara:

[kutipan program]
```
if(kondisi1)
```

```
    /* statement 1 */;
else if(kondisi2)
    /* statement 2 */;
/* statement 3 */
```
[/kutipan program]

...dengan:

[kutipan program]
```
if(kondisi1)
    /* statement 1 */;
if(kondisi2) // tidak menggunakan else
    /* statement 2 */;
/* statement 3 */
```
[/kutipan program]

**A:** Di bentuk pertama, bila **kondisi1** benar maka setelah statement 1 dijalankan alur program akan melompat ke statement 3. **kondisi2** dicek hanya bila **kondisi1** salah. Di bentuk kedua, **kondisi1** dan **kondisi2** sama-sama dicek kebenarannya.
[/tanya jawab]

[latihan mini]
- Apakah type yang harus dipakai sebagai pengontrol statement **if**?
- Jika terdapat **else**, manakah **if** yang akan menjadi pasangannya?
- Apakah yang salah dari kutipan program berikut?

  [kutipan program]
  ```
  if(x > y)
  {
   int z = 100;
  }
  else
  {
   int z = 50;
  }
  Console.WriteLine(z);
  ```
  [/kutipan program]

- **bool**.
- Yang akan menjadi pasangannya adalah **if** terdekat yang belum mempunyai pasangan.
- **Console.WriteLine()** tidak dapat menggunakan **z** sebagai argument sebab **z** hanya hidup/dikenal di blok-blok yang bersangkutan.

[/latihan mini]

# 6.7 Operator-Operator Logical

Operator-operator logical memungkinkan kita melakukan banyak perbandingan dalam satu expression. Operand dari operator-operator tersebut adalah **bool**. Inilah jenis-jenisnya:

| **Operator** | **Arti** |
|---|---|
| & | AND |
| \| | OR |
| ^ | XOR (exclusive or) |
| ! | NOT |
| \|\| | Short circuit OR |
| && | Short circuit AND |

## 6.7.1 AND

Operator **&** akan menghasilkan **true** hanya jika kedua operandnya **true**. Contoh penggunaannya adalah sebagai berikut:

[kutipan program]
```
if(umur >= 13 & membawaUang) // membawaUang adalah bool
    Console.WriteLine("Anda diizinkan masuk.");
```
[/kutipan program]

**"Anda diizinkan masuk."** hanya akan ditulis ke layar jika variabel **umur** nilainya lebih besar dari atau sama dengan **13**, dan jika variabel **membawaUang** nilainya **true**.

Inilah tabel yang menunjukkan kemungkinan-kemungkinan yang terjadi:

| **operand1** | **operand2** | **operand1 & operand2** |
|---|---|---|
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

## 6.7.2 OR

Operator | akan menghasilkan **true** bila kedua operandnya **true** atau bila salah satu operandnya **true**. Ini contoh penggunaannya:

[kutipan program]
```
if(umur >= 13 | disertaiOrtu) // disertaiOrtu adalah bool
    Console.WriteLine("Anda diizinkan masuk."); // kita sebut [statement]
```
[/kutipan program]

Bila **umur >= 13** bernilai **true**, maka ***[statement]*** akan dijalankan. Bila variabel **disertaiOrtu** nilainya **true**, ***[statement]*** juga akan dijalankan. Hal yang sama terjadi bila kedua operand **true**. Perhatikan tabel berikut:

| **operand1** | **operand2** | **operand1 \| operand2** |
|---|---|---|

| | | |
|---|---|---|
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

## 6.7.3 XOR

Operator ^ akan mengembalikan **true** hanya jika salah satu operandnya **true**. Perhatikan tabel berikut:

| **operand1** | **operand2** | **operand1 \| operand2** |
|---|---|---|
| true | true | false |
| true | false | true |
| false | true | true |
| false | false | false |

## 6.7.4 NOT

Operator **!** bersifat unary, dan akan membalikkan nilai kebenaran. Perhatikan tabel berikut:

| **operand** | **!operand** |
|---|---|
| true | false |
| false | true |

Ini contoh pemakaiannya:

[kutipan program]
```
if(!(penyebut == 0)) // sama dengan if(penyebut != 0)
    hasil = pembilang / penyebut;
```
[/kutipan program]

## 6.7.5 Short circuit AND dan OR

Versi short circuit AND dan OR bekerja lebih efisien dari versi normalnya. Jika kita menggunakan short circuit AND dan operand pertama telah diketahui **false**, maka operand kedua tidak akan dievaluasi (dan hasilnya adalah **false**). Jika kita menggunakan AND biasa, kedua operand akan selalu dievaluasi sebelum menyimpulkan hasil akhirnya. Begitu pula dengan short circuit OR. Jika operand pertama nilainya **true**, maka operand kedua tidak akan dievaluasi dan nilai yang dikembalikan adalah **true**. Perhatikan tabel berikut:

| **operand1** | **operand2** | **operand1 && operand2** |
|---|---|---|
| true | true | true |
| true | false | false |
| false | *tidak dievaluasi* | false |

| operand1 | operand2 | operand1 \|\| operand2 |
|---|---|---|
| true | *tidak dievaluasi* | true |
| false | true | true |
| false | false | false |

Ini contoh penggunaannya yang menggabungkan berbagai operator logical:

[program lengkap]
```
// Program 6.7  - Operator Logical
// Hanya dapat meminjam buku bila jumlahnya di atas 0
// Anggota biasa hanya dapat meminjam paling banyak 10
// Anggota spesial dapat meminjam sebanyak-banyaknya

using System;

class OperatorLogical
{
    static void Main()
    {
        int buku;
        bool anggotaSpesial;
        Console.WriteLine("Berapa jumlah buku yang ingin anda pinjam?");
        buku = int.Parse(Console.ReadLine());
        Console.WriteLine("Apakah anda anggota spesial (true/false)?");
        anggotaSpesial = bool.Parse(Console.ReadLine());
        if((buku > 0) && ((buku <= 10) || anggotaSpesial))
            Console.WriteLine("Buku dapat anda pinjam");
        else
            Console.WriteLine("Maaf, anda tak dapat meminjam buku.");
    }
}
```
[/program lengkap]

Ini contoh outputnya:

[console]
```
Berapa jumlah buku yang ingin anda pinjam?
70
Apakah anda anggota spesial (true/false)?
true
Buku dapat anda pinjam
```
[/console]

Program di atas menggunakan beberapa operator logical short circuit. Dalam statement **if** yang ada terdapat expression yang cukup rumit. Inilah gerak lambatnya!

Pertama dilakukan pengecekan, apakah buku yang ingin dipinjam jumlahnya di atas **0**?

[kutipan program]

```
if((buku > 0) && ((buku <= 10) || anggotaSpesial))
```
[/kutipan program]

Expression tersebut merupakan operand bagi operator **&&**. Jika nilainya **false**, misalnya jika **buku** bernilai **-999**, maka operand ke dua tidak akan dievaluasi dan hasil keseluruhan adalah **false**. Jika **buku** nilainya di atas **0**, maka operand ke dua akan dievaluasi.

[kutipan program]
```
if([true] && ((buku <= 10) || anggotaSpesial))
```
[/kutipan program]

Operand ke dua sendiri merupakan expression yang melibatkan operator ||. Pertama dicek, apakah **buku** yang ingin dipinjam jumlahnya di bawah atau sama dengan **10**? Jika ya, maka jenis anggota peminjam tidak akan dicek sebab anggota biasa dan spesial sama-sama dapat meminjam buku yang jumlahnya kurang dari **11**. Jika **buku <= 10** nilainya **true**, hasil expression yang melibatkan || adalah **true**, dan ini berarti keseluruhan expression nilainya **true**.

Namun jika **buku** jumlahnya melebihi **10**, dilakukan pengecekan berikutnya:

[kutipan program]
```
if([true] && ([false] || anggotaSpesial))
```
[/kutipan program]

Apakah peminjam merupakan anggota spesial? Jika **anggotaSpesial** nilainya **true**, maka expression yang melibatkan || nilainya **true** dan keseluruhan expression bernilai **true**. Jika **anggotaSpesial** nilainya **false**, keseluruhan expression akan bernilai **false**.

Hal yang sama dapat dilakukan menggunakan **&** dan |, namun seluruh operand akan dievaluasi tanpa memperdulikan nilai operand yang telah dievaluasi. Perhatikan juga bahwa tanda kurung perlu digunakan dalam **((buku <= 10) || anggotaSpesial)** sebab operator **&&** prioritasnya lebih tinggi dibandingkan operator ||. Tanda kurung yang digunakan dalam **(buku > 0)** dan **(buku <= 10)** digunakan hanya untuk memudahkan pembacaan.

# 6.8 Operator Conditional

Di C# hanya ada sebuah operator ternary (membutuhkan 3 operand) yaitu operator conditional. Formatnya adalah sebagai berikut:

[format]
```
[kondisi] ? [expression 1] : [expression 2]
```
[/format]

Jika ***[kondisi]*** nilainya **true**, maka ***[expression 1]*** adalah nilai akhirnya. Jika ***[kondisi]*** nilainya **false**, ***[expression 2]*** yang akan menjadi nilai akhirnya. Perhatikan kutipan program berikut:

[kutipan program]
```
angkaTerbesar = angka1 > angka2 ? angka1 : angka2;
```
[/kutipan program]

Jika **angka1** > **angka2** bernilai **true**, maka nilai dari **angka1** akan dimasukkan ke dalam **angkaTerbesar**, dan sebaliknya. Kutipan di atas dapat dikonstuksi menggunakan **if-else** seperti berikut:

[kutipan program]
```
if(angka1 > angka2)
    angkaTerbesar = angka1;
else
    angkaTerbesar = angka2;
```
[/kutipan program]

Walaupun cukup sulit dibaca bagi yang belum terbiasa, operator conditional memungkinkan kita membuat program yang lebih singkat. Selain contoh yang telah diberikan, coba bandingkan dua alternatif berikut:

[kutipan program]
```
// menggunakan if-else
Console.Write("Angka terbesar: ");
if(angka1 > angka2)
    Console.WriteLine(angka1);
else
    Console.WriteLine(angka2);

// menggunakan operator conditional, lebih singkat
Console.WriteLine("Angka terbesar: {0}", angka1 > angka2 ? angka1 : angka2)
```
[/kutipan program]

Lebih dari satu operator conditional dapat digabungkan, seperti contoh berikut.

[kutipan program]
```
Console.WriteLine(angka1 > angka2 ? "angka1 terbesar" : (angka1 == angka2 ?
"nialinya sama" : "angka2 terbesar"));
```
[/kutipan program]

Dalam contoh tersebut digunakan tanda kurung ekstra untuk memudahkan pembacaan.

[catatan]
Type yang dikembalikan oleh operator conditional haruslah kompatibel. Misalnya kutipan program berikut sah:

[kutipan program]
```
nilai = kondisi ? 1 : 1.5; // 1 (int) akan diubah menjadi double
```

[/kutipan program]

... namun kutipan berikut tidak:

[kutipan program]
```
nilai = kondisi ? 1m : 1.5; // decimal dan double tidak kompatibel
```
[/kutipan program]
[/catatan]

[latihan mini]
- Apakah hasil dari expression (**false & true) | false**?
- Bagaimana format operator conditional?
- Persingkat kutipan program di bawah ini dengan menggunakan operator logical!

  [kutipan program]
  ```
  if(kecepatan > batasKecepatan)
    if(adaPolisi)
        Console.WriteLine("Anda ditilang!");
  ```
  [/kutipan program]

- Hasilnya adalah **false**.
- ***[kondisi] ? [expression 1] : [expression 2]***
- [kutipan program]
  ```
  if(kecepatan > batasKecepatan && adaPolisi)
    Console.WriteLine("Anda ditilang!");
  ```
  [/kutipan program]

[/latihan mini]

# 6.9 Statement switch

Kadang-kadang kita perlu membandingkan suatu hal dengan berbagai kemungkinan dan mengambil keputusan yang berbeda untuk tiap kemungkinan. Sebagai contoh adalah kutipan yang telah muncul sebelumnya:

[kutipan program]
```
if(nomor == 100232)
    Console.WriteLine("Anda mendapat mobil!");
else if(nomor == 302)
    Console.WriteLine("Anda mendapat motor!");
else if(nomor == 994530243)
    Console.WriteLine("Anda mendapat radio!")
else
    Console.WriteLine("Anda belum beruntung...")
```
[/kutipan program]

Sebagai alternatif, kita bisa menggunakan statement **switch**. Inilah formatnya:

[format]

```
switch([expression])
{
    case [expression konstan 1]:
        [statement-statement];
        break;
    case [expression konstan 2]:
        [statement-statement];
        break;
    case [expression konstan 3]:
        [statement-statement];
        break;
    case [expression konstan n]:
        [statement-statement];
        break;
    default: // opsional, boleh ada boleh tidak
        [statement-statement];
        break;
}
```
[/format]

Pertama-tama, ***[expression]*** akan dievaluasi. Nilainya akan dibandingkan dengan berbagai ***[expression konstan]*** yang ada. Jika nilainya sama, maka ***[statement-statement]*** yang bersangkutan akan dijalankan. Jika tidak ada yang cocok dan terdapat label **default**, maka ***[statement-statement]*** di bawah label **default** akan dijalankan. **break** digunakan untuk keluar dari statement **switch**.

Ada beberapa keterbatasan yang dimiliki statement **switch**. Hasil dari ***[expression]*** dan ***[expression konstan]*** hanya dapat berupa integer. Lalu ***[expression konstan]*** hanya dapat berupa nilai yang dapat diperhitungkan saat program dicompile, artinya tidak boleh ada unsur variabel dan pemanggilan method.

Inilah kutipan program yang telah kita bahas, dikonstruksi ulang menggunakan statement **switch**:

[kutipan program]
```
switch(nomor)
{
    case 100232:
        Console.WriteLine("Anda mendapat mobil!");
        break;
    case 302:
        Console.WriteLine("Anda mendapat motor!");
        break;
    case 994530243:
        Console.WriteLine("Anda mendapat radio!");
        break;
    default:
        Console.WriteLine("Anda belum beruntung...");
        break;
}
```
[/kutipan program]

Kita juga dapat membuat sebuah ***[statement-statement]*** yang digunakan oleh berbagai kondisi, caranya sebagai berikut:

[kutipan program]
```
switch(karakter)
{
    case ‘a’:
    case ‘i’:
    case ‘u’:
    case ‘e’:
    case ‘o’:
    case ‘A’:
    case ‘I’:
    case ‘U’:
    case ‘E’:
    case ‘O’:
        Console.WriteLine(“{0} adalah vokal.”, karakter);
        break;
    default:
        Console.WriteLine(“{0} bukan vokal.”, karakter);
        break;
}
```
[/kutipan program]

**switch** juga termasuk statement sehingga bisa berada di tempat-tempat sah statement. **switch** dapat digabung dengan **switch** lain, dengan **if-else**, dan dengan konstruksi-konstruksi bahasa lainnya.

# 6.10 goto

Kita dapat membuat label di program kita. Dengan statement **goto** kita dapat melompat ke label yang telah kita buat.

Format untuk membuat label adalah sebagai berikut:

[format]
*[label]* :
[/format]

Label merupakan identifier, sehingga terdapat beberapa aturan penamaan (bab 3.9). Ini adalah format untuk statement **goto**:

[format]
goto *[label]*;
[/format]

Contoh penggunaannya adalah sebagai berikut:

[kutipan program]

```
// program yang tak akan pernah selesai
int hitungMundur = 5;
Console.WriteLine("Bersiap-siaplah...");

Loop1 :
Console.ReadLine();
Console.WriteLine("{0}...", hitungMundur--); // nilai hitungMundur dikurangi 1
if(hitungMundur >= 0)
    goto Loop1;

Loop2 :
Console.WriteLine("Waaa!!!");
goto Loop2;
```
[/kutipan program]

Konstruksi bahasa menggunakan **goto** sangat tidak disarankan, sebab penggunaannya yang berlebihan membuat alur program susah diikuti. Secara teoritis, apapun yang dapat dibuat dengan **goto** dapat dibuat menggunakan konstruksi bahasa lain. Loop seperti contoh program di atas dapat dibuat dengan cara yang lebih baik (diajarkan di bab lain).

**goto** juga bisa digunakan untuk pergi ke label switch lain, seperti contoh berikut:

[kutipan program]
```
switch(favorit)
{
    case 1:
        Console.WriteLine("Saya suka Beckham...");
        goto case 2;
    case 2:
        Console.WriteLine("Saya suka Manchester United...");
        goto case 3;
    case 3:
        Console.WriteLine("Saya suka Inggris...");
        break;
    default:
        Console.WriteLine("Saya tidak suka sepak bola");
        break;
}
```
[/kutipan program]

Jika **favorit** nilainya **1**, maka **case 1**, **case 2**, dan **case 3** akan terkunjungi. Jika **favorit** nilainya **2**, hanya statement-statement dalam **case 2** dan **case 3** yang akan dijalankan. Jika nilai **favorit** adalah **3** yang akan dijalankan hanya statement-statement dalam **case 3**. Untuk nilai lainnya, statement-statement dalam **default** yang akan dijalankan.

[tes]

[/tes]

# Bab 7: Loop

Loop adalah konstruksi bahasa yang memungkinkan kita menjalankan satu atau lebih statement berulang-ulang. Terdapat beberapa jenis loop di C# yaitu **while**, **do-while**, **for**, dan **foreach**. foreach tidak akan dipelajari di bab ini.

# 7.1 while

Loop **while** terdiri dari dua bagian utama yaitu kondisi yang mengontrol jalannya loop dan sebuah statement yang menjadi tubuh loop tersebut. Kondisi harus merupakan sebuah bool. Statement dapat berupa statement biasa, statement kosong, maupun blok. Format dari loop **while** adalah sebagai berikut:

[format]
```
while([kondisi])
    [statement]
```
[/format]

Jika ***[kondisi]*** nilainya **true**, maka ***[statement]*** akan dijalankan. Setelah ***[statement]*** dijalankan, ***[kondisi]*** akan dicek kembali. Ini akan berlangsung terus sampai ***[kondisi]*** nilainya **false**. Dari penjelasan tersebut, jelas diperlukan mekanisme yang suatu saat membuat ***[kondisi]*** **false**. Jika tidak, loop akan berjalan selamanya! Perhatikan kutipan berikut:

[kutipan program]
```
int pengatur = 5;
Console.WriteLine("Akan memasuki loop...");
while(pengatur > 0)
{
    Console.WriteLine("Nilai pengatur: {0}", pengatur);
    pengatur--;
}
Console.WriteLine("Loop selesai...");
```
[/kutipan program]

Outputnya adalah sebagai berikut:

[console]
```
Akan memasuki loop...
Nilai pengatur: 5
Nilai pengatur: 4
Nilai pengatur: 3
Nilai pengatur: 2
Nilai pengatur: 1
Loop selesai...
```
[/console]

Nilai **pengatur** pada awalnya **5**, sehingga loop berikut:

[kutipan program]

```
while(pengatur > 0)
```
[/kutipan program]

... dapat dimasuki. Di dalam loop, nilai **pengatur** dikurangi dengan statement:

[kutipan program]
```
pengatur--;
```
[/kutipan program]

Suatu saat nilai **pengatur** akan menjadi **0**, sehingga **(pengatur > 0)** bernilai **false**. Saat itulah loop tidak akan dimasuki dan alur program berlanjut ke statement berikutnya:

[kutipan program]
```
Console.WriteLine("Loop selesai...");
```
[/kutipan program]

[proyek]

Selain untuk demonstrasi, program di atas tidak terlalu berguna. Sekarang kita akan mencoba membuat program yang menghitung faktorial dari sebuah angka. Faktorial sebuah bilangan bulat n (n!) didefinisikan sebagai n * (n - 1) * (n - 2) * (n - 3) * ... * 1. Misalnya 5! adalah 5 * 4 * 3 * 2 * 1 = 120. Sebagai catatan, didefinisikan bahwa 0! adalah 1. Program yang akan kita buat hanya akan memproses angka yang lebih besar atau sama dengan 0.

Kita akan menggunakan 2 variabel dalam program ini, yaitu **n** yang typenya **sbyte** dan **faktorial** yang typenya **double**. Nilai n! akan bertambah secara eksponensial jika **n** semakin besar, karena itu **sbyte** digunakan agar batas atas inputnya **127**. **double** cocok untuk menjadi type dari **faktorial** sebab seperti sudah dikatakan, nilai **faktorial** dapat menjadi besar dengan sangat mudah.

[kutipan program]
```
sbyte n;
double faktorial = 1;
```
[/kutipan program]

**faktorial** diberi nilai awal **1** agar dapat langsung dikalikan dengan angka lain di dalam loop.

Langkah selanjutnya adalah meminta input dari pemakai, namun kali ini akan kita lakukan agak berbeda. Input akan dilakukan di dalam statement **if**, yang akan sekaligus dicek nilainya.

[kutipan program]
```
if((n = sbyte.Parse(Console.ReadLine())) >= 0)
    ; // hitung faktorial
else
    Console.WriteLine("n tidak boleh negatif!");
```
[/kutipan program]

**Console.ReadLine()** akan memungkinkan pengguna menuliskan input. Input yang berupa **string** lalu akan diubah menjadi **sbyte** oleh method **sbyte.Parse()**. Nilainya akan dimasukkan ke **n**, lalu nilai **n** akan dibandingkan. Jika nilainya tidak negatif, maka faktorialnya akan dicari.

Penghitungan faktorialnya sendiri cukup mudah. Loop untuk menghitung faktorial hanya dapat dimasuki jika **n > 1**. Untuk **n == 0** atau **n == 1**, variabel **faktorial** tidak perlu diubah karena sudah bernilai **1**. Di dalam loop, **faktorial** akan dikalikan dengan **n**. Setelah itu nilai **n** akan dikurangi **1**. Ini akan diulang sampai nilai **n** menjadi **1**.

Inilah contoh program jadinya (terdapat dua statement yang dapat membantu memahami alur program):

[program lengkap]
```
// Proyek 7.1 - Menghitung Faktorial

using System;

class MenghitungFaktorial
{
    static void Main()
    {
        sbyte n;
        double faktorial = 1;
        Console.Write("Masukkan nilai n (paling kecil 0 dan paling besar 127): ");
        if((n = sbyte.Parse(Console.ReadLine())) >= 0)
        {
            while(n > 1)
            {
                // statement berikut dapat dihilangkan
                Console.WriteLine("Variabel faktorial akan dikalikan dengan {0}",
n);
                faktorial *= n--;
                // statement berikut juga dapat dihilangkan
                Console.WriteLine("Nilai faktorial sekarang {0}", faktorial);
            }
            Console.WriteLine("Faktorialnya adalah {0}", faktorial);
        }
        else
            Console.WriteLine("n tidak boleh negatif!");
    }
}
```
[/program lengkap]

Ini contoh outputnya:

[console]
```
Masukkan nilai n (paling kecil 0 dan paling besar 127): 4
Variabel faktorial akan dikalikan dengan 4
Nilai variabel faktorial sekarang 4
Variabel faktorial akan dikalikan dengan 3
Nilai variabel faktorial sekarang 12
Variabel faktorial akan dikalikan dengan 2
```

```
Nilai variabel faktorial sekarang 24
Faktorialnya adalah 24
```
[/console]
[/proyek]

# 7.2 do-while

Pada loop **do-while**, statement yang menjadi tubuh loop dijalankan sebelum kondisi dicek. Ini artinya statement yang bersangkutan paling tidak dijalankan sekali. Inilah formatnya:

[format]
```
do
    [statement]
while([kondisi]);
```
[/format]

Perlu diperhatikan bahwa setelah **while(*[kondisi]*)** harus ada titik koma.

**do-while** dapat dipakai jika diperlukan input yang berulang-ulang seperti dalam program berikut:

[program lengkap]
```
// Program 7.1 - do-while

using System;

class DoWhile
{
    static void Main()
    {
        char mauLagi;
        int banyaknyaAngka = 0;
        int jumlah = 0;
        do
        {
            Console.Write("Masukkan sebuah angka: ");
            jumlah += int.Parse(Console.ReadLine());
            banyaknyaAngka++;
            Console.Write("Lagi (y untuk ya)? ");
            mauLagi = char.Parse(Console.ReadLine());
        }while(mauLagi == 'y' || mauLagi == 'Y');
        Console.WriteLine("Jumlah : {0}", jumlah);
        Console.WriteLine("Rata-rata: {0}", (double)jumlah / banyaknyaAngka);
    }
}
```
[/program lengkap]

Contoh outputnya:

[console]
```
Masukkan sebuah angka: 9
Lagi (y untuk ya)? y
Masukkan sebuah angka: 2
Lagi (y untuk ya)? y
Masukkan sebuah angka: 7
Lagi (y untuk ya)? y
Masukkan sebuah angka: -1
Lagi (y untuk ya)? n
Jumlah : 17
Rata-rata: 4.25
```
[/console]

# 7.2 for

Loop jenis ini biasanya dipakai kalau kita sudah ?. Walaupun begitu loop ini bisa dipakai untuk tujuan apapun. Inilah formatnya:

[format]
for(*[expression 1]*$_{opt}$; *[kondisi]*$_{opt}$; *[expression 2]*$_{opt}$)
    *[statement]*
[/format]

***[expression 1]*** adalah yang akan pertama kali dijalankan, dan tidak akan pernah dijalankan lagi. Setelah itu ***[kondisi]*** dicek, dan jika **true** maka ***[statement]*** akan dijalankan. Setelah ***[statement]*** dijalankan, ***[expression 2]*** akan dijalankan, dan ***[kondisi]*** dicek kembali. Loop akan selesai kalau ***[kondisi]*** bernilai **false** saat dicek.

***[expression 1]*** dapat berupa pendeklarasian variabel, dan memang biasa digunakan untuk itu. Variabel yang dideklarasikan di bagian ini sebaiknya hanya yang akan berperan dalam pengontrolan loop. Sebagai contoh, **int i = 0** dapat digunakan sebagai ***[expression 1]***. ***[kondisi]*** harus berupa **bool**, misalnya **i < 5**. ***[expression 2]*** biasanya digunakan untuk memodifikasi variabel tertentu agar suatu saat nilai ***[kondisi]*** bernilai **false**, misalnya **i++**.

***opt*** dalam format tersebut berarti opsional, artinya tanpa bagian tersebut loop **for** tetap dapat berjalan. Loop **for** yang tidak lengkap akan dibahas di subbab lain.

Program berikut akan membuat dua loop yang melakukan hal yang sama. Loop pertama menggunakan **while** sedangkan loop kedua menggunakan **for**:

[kutipan program]
```
// ini adalah versi while
{
    int i = 0; // i dideklarasikan dalam blok
    while(i < 5)
    {
        Console.WriteLine("while: saya akan keluar 5 kali...");
        i++;
    }
}
// variabel i tidak dikenal di sini
```

```
Console.WriteLine();
// ini adalah versi for
for(int i = 0; i < 5; i++)
    Console.WriteLine("for: saya akan keluar 5 kali...");
// variabel i milik for tidak dikenal di sini
```
[/kutipan program]

Output kedua loop tersebut sama:

[console]
```
while: saya akan keluar 5 kali...
while: saya akan keluar 5 kali...
while: saya akan keluar 5 kali...
while: saya akan keluar 5 kali...
while: saya akan keluar 5 kali...

for: saya akan keluar 5 kali...
for: saya akan keluar 5 kali...
for: saya akan keluar 5 kali...
for: saya akan keluar 5 kali...
for: saya akan keluar 5 kali...
```
[/console]

Keuntungan penggunaan **for** yang pertama adalah program terlihat lebih singkat. Kedua, informasi yang berhubungan dengan pengontrolan loop bisa didapat di satu tempat. Dengan melihat loop **for** pada program di atas, kita akan tahu bahwa variabel **i** yang akan menjadi pengontrolnya, loop akan berjalan 5 kali, dan nilai **i** akan naik dari **0** sampai **4** (saat nilainya **5**, expression **i < 5** akan menghasilkan **false** dan loop akan selesai).

## 7.2.1 Loop for yang Tidak Lengkap

Ada kalanya kita ingin menggunakan variabel luar untuk menjadi pengontrol loop. Dalam kasus ini ***[statement 1]*** tidak perlu ditulis. Namun ingat bahwa titik koma tetap diperlukan. Ini contoh penggunaannya:

[kutipan program]
```
int i = 0;
for(;i < 5; i++)
    Console.WriteLine("saya akan keluar 5 kali...");
// variabel i tetap dapat dipakai di sini
```
[/kutipan program]

***[statement 3]*** juga dapat dihiraukan. Ini contoh for tanpa ***[statement 1]*** dan ***[statement 3]***:

[kutipan program]
```
int i = 0;
for(;i < 5;)
```

```
{
    Console.WriteLine("saya akan keluar 5 kali...");
    i++; // nilai i dinaikkan di dalam loop
}
```
[/kutipan program]

Tanpa ***[kondisi]***, loop akan berjalan selamanya. Untuk keluar dari loop semacam itu, **break** dapat digunakan.

# 7.3 break

Selain untuk keluar dari statement switch, break juga dapat digunakan untuk keluar dari loop. Loop yang dimaksud di sini adalah semua jenis loop yaitu **while**, **do-while**, dan **for**. Ini contoh penggunaannya:

[kutipan program]
```
int i = 0;
for(;;)
{
    Console.Write("Masukkan angka yang tidak negatif: ");
    i = int.Parse(Console.ReadLine());
    if(i >= 0)
        break;
    Console.WriteLine("Angka tidak sah!");
}
Console.WriteLine("Terima Kasih");
```
[/kutipan program]

Contoh outputnya:

[console]
```
Masukkan angka yang tidak negatif: -1
Angka tidak sah!
Masukkan angka yang tidak negatif: -5
Angka tidak sah!
Masukkan angka yang tidak negatif: 10
Terima kasih!
```
[/console]

Loop yang sama dapat dibuat dengan konstruksi **while** dan **do-while**. Tuliskan saja **true** sebagai kondisnya, misalnya:

[kutipan program]
```
while(true)
{
    // kode anda
}
```
[/kutipan program]

# 7.4 continue

Jika statement **continue** dalam sebuah loop dijalankan, maka loop akan berlanjut ke iterasi berikutnya. Format statement ini sangat sederhana:

[format]
```
continue;
```
[/format]

Contoh penggunaannya adalah sebagai berikut:

[kutipan program]
```
for(int i = -5; i < 6; i++)
{
    if(i == 0)
        continue;
    Console.WriteLine("1 dibagi {0} hasilnya {1}.", i, 1.0 / i);
}
```
[/kutipan program]

Inilah outputnya:

[console]
```
1 dibagi -5 hasilnya -0.2.
1 dibagi -4 hasilnya -0.25.
1 dibagi -3 hasilnya -0.333333333333333.
1 dibagi -2 hasilnya -0.5.
1 dibagi -1 hasilnya -1.
1 dibagi 1 hasilnya 1.
1 dibagi 2 hasilnya 0.5.
1 dibagi 3 hasilnya 0.333333333333333.
1 dibagi 4 hasilnya 0.25.
1 dibagi 5 hasilnya 0.2.
```
[/console]

Saat **i** nilainya **0**, maka statement continue akan dijalankan. Efeknya sama dengan melewati semua statement yang ada setelahnya. Kita bisa lihat bahwa tidak ada output saat **i** nilainya **0**.

# 7.5 Contoh-Contoh Tambahan

Kita dapat menyelesaikan hal-hal yang sederhana dengan loop **for** kosong. Contohnya adalah mencari jumlah seluruh angka dari **1** sampai angka tertentu. Perhatikan contoh berikut:

[kutipan program]
```
for(int i = 1; i <= maksimum; jumlah += i++) // sebelumnya jumlah bernilai 0
    ; // statement kosong
```

[/kutipan program]

Kita coba telusuri alur program kalau **maksimum** bernilai **3**. Saat loop dimasuki, maka akan tercipta variabel **i** yang bernilai **1**. Karena nilai **i** lebih kecil dari **maksimum**, maka **i <= maksimum** hasilnya **true**. Statement yang menjadi tubuh loop lalu dijalankan, yaitu statement kosong. Setelah itu **jumlah += i++** dijalankan. Pada statement tersebut expression **i++** menaikkan nilai **i** menjadi **2**. Karena yang digunakan operator increment postfix, maka **jumlah** ditambahkan dengan nilai **i** sebelumnya yaitu **1**. Untuk mudahnya expression tersebut efeknya sama dengan kutipan berikut:

[kutipan program]
```
jumlah += i;
i++;
```
[/kutipan program]

Loop akan berlanjut dan keluar saat **i** bernilai **4**. Nilai **jumlah** pada akhir loop adalah **1 + 2 + 3**. Sebagai contoh lain, jika nilai **maksimum 100** maka nilai **jumlah** pada akhir loop adalah **1 + 2 + 3 + ... + 99 + 100**.

Loop juga dapat berada di dalam loop lain. Untuk contoh penggunaannya ikuti proyek berikut!

[proyek]

Kali ini kita akan membuat program yang menampilkan tabel perkalian dari **1 * 1** sampai **10 * 10**. Penggunaan loop akan sangat berguna dibanding jika kita harus menulis tiap-tiap **Console.WriteLine()** yang bersangkutan.

Di dalam program ini kita akan menggunakan loop di dalam loop. Loop pertama akan mengiterasikan nilai sebuah variabel (misal **i**) dari **1** sampai **10**. Di dalam loop pertama ini akan terdapat sebuah loop yang mengiterasikan nilai variabel lain (misal **j**) dari **1** sampai **10** juga. Perkalian **i** dengan **j** akan dilakukan di dalam loop kedua. Inilah contoh program jadinya (penjelasan terdapat pada comment):

[program lengkap]
```
// Proyek 7.2 - Tabel Perkalian

using System;

class TabelPerkalian
{
    static void Main()
    {
        int kolom = 10; // untuk setiap baris akan ada i * 1 sampai i * kolom
        int baris = 10; // program akan menghitung 1 * 1 sampai baris * kolom

        // membuat puncak tabel
        // contoh outputnya jika kolom bernilai 4
        // .....|...1.|...2.|...3.|...4.|
        // (titik melambangkan spasi)
        Console.Write("     |");
        for(int i = 1; i <= kolom; i++)
```

```
            Console.Write(" {0,3} |", i);
        Console.WriteLine();

        // membuat garis pemisah
        // contoh output jika kolom bernilai 4
        // ----------------------------
        for(int i = 1; i <= kolom + 1; i++)
            Console.Write("------");
        Console.WriteLine();

        // loop ini akan memproses baris per baris
        for(int i = 1; i <= baris; i++)
        {
            // menuliskan identitas baris, misalnya
            // ...1.|
            // (titik melambangkan spasi)
            Console.Write(" {0,3} |", i);

            // loop ini akan memproses kolom per kolom
            // untuk baris yang bersangkutan
            for(int j = 1; j <= kolom; j++)
                Console.Write(" {0,3} |", i * j);

            // baris selesai diproses
            // output karakter new line ke console
            Console.WriteLine();
        }
    }
}
```
[/program lengkap]

Ini adalah output dari program di atas:

[console]
```
     |   1 |   2 |   3 |   4 |   5 |   6 |   7 |   8 |   9 |  10 |
-------------------------------------------------------------------
   1 |   1 |   2 |   3 |   4 |   5 |   6 |   7 |   8 |   9 |  10 |
   2 |   2 |   4 |   6 |   8 |  10 |  12 |  14 |  16 |  18 |  20 |
   3 |   3 |   6 |   9 |  12 |  15 |  18 |  21 |  24 |  27 |  30 |
   4 |   4 |   8 |  12 |  16 |  20 |  24 |  28 |  32 |  36 |  40 |
   5 |   5 |  10 |  15 |  20 |  25 |  30 |  35 |  40 |  45 |  50 |
   6 |   6 |  12 |  18 |  24 |  30 |  36 |  42 |  48 |  54 |  60 |
   7 |   7 |  14 |  21 |  28 |  35 |  42 |  49 |  56 |  63 |  70 |
   8 |   8 |  16 |  24 |  32 |  40 |  48 |  56 |  64 |  72 |  80 |
   9 |   9 |  18 |  27 |  36 |  45 |  54 |  63 |  72 |  81 |  90 |
  10 |  10 |  20 |  30 |  40 |  50 |  60 |  70 |  80 |  90 | 100 |
```
[/console]
[/proyek]

Ada hal lain yang unik dari loop for.  Perhatikan kembali formatnya:

[format]

for(*[expression 1]*$_{opt}$; *[kondisi]*$_{opt}$; *[expression 2]*$_{opt}$)
    *[statement]*
[/format]

***[expression 1]*** dan ***[expression 2]*** juga dapat berisi lebih dari 1 expression sah yang dipisahkan dengan koma. Contoh **for** di bawah ini sah:

[kutipan program]
```
// dalam contoh ini anggap a dan b sudah dideklarasikan di luar loop
// walaupun begitu dapat saja anda membuat versi di mana
// pendeklarasian a dan b menjadi bagian dari loop for
for(a = 1, b = 10; a < b; a++, b--)
    Console.WriteLine("a: {0}\tb: {1}", a, b);;
```
[/kutipan program]

# Lampiran A: Kunci Jawaban Tes

# Bab 1: Mengenal C# dan Framework .NET

## A. Pertanyaan

1. Untuk membuat program-program .NET kita harus menggunakan C#. Benar atau salah?

   Salah. Banyak bahasa yang bisa digunakan untuk pemrograman .NET diantaranya Managed C++ dan Visual Basic .NET. Yang lebih menarik lagi, program-program .NET dapat berinteraksi walaupun dibuat dengan bahasa yang berbeda.

2. Jelaskan mengenai CLR (Common Language Runtime)!

   CLR adalah bagian dari Framework .NET yang mengelola program-program .NET yang dijalankan.

3. Apakah program-program .NET (termasuk yang dibuat dengan C#) hanya dapat berjalan di Windows?

   Tidak. Program-program .NET dapat berjalan di komputer apapun asalkan terinstall Framework .NET di komputer yang bersangkutan.

# Bab 2: Membuat Program Pertama Anda

## A. Pertanyaan

1. Apakah akhiran yang biasa dipakai untuk file yang berisi source code program C#?

   Akhiran yang dipakai untuk menandakan bahwa file tersebut berisi source code program C# adalah .cs.

2. File source code C# dapat langsung dijalankan, benar atau salah?

   Salah. Source code tersebut harus dicompile terlebih dahulu.

3. Apakah hasil compile suatu program C# berupa kode native?

   Tidak. Hasilnya adalah file yang berisi IL (Intermediate Language).

4. Jelaskan mengenai JIT Compiler!

   JIT Compiler akan mengcompile IL menjadi kode native sesuai kebutuhan saat program dijalankan. Kode native tersebutlah yang sebenarnya akan dijalankan.

# Bab 3: Dasar-Dasar C#

## A. Pertanyaan

1. Method apakah yang dipanggil saat suatu program C# dijalankan?

   Method **Main()**.

2. Apakan setiap program C# harus dimulai dengan statement di bawah ini?

   [kutipan program]
   using System;
   [/kutipan program]

   Menggunakan **using System** bukanlah suatu keharusan, namun tanpa menuliskan statement tersebut kita harus selalu mencantumkan **System** setiap kali ingin mengakses class library .NET.

3. Bagaimanakah cara membuat comment single line dan multi line? Dapatkan comment multi line dimasukkan ke dalam suatu comment multi line lain?

   Comment single line diawali dengan **//** dan berakhir di baris yang bersangkutan. Comment multi line diawali dengan **/*** dan diakhiri dengan ***/**. Comment multi line tidak dapat berada di dalam comment multi line lain.

4. Manakah diantara identifier-identifier berikut yang tidak benar?

   double __byte 15Mobil Pagar# @class Kelas6SD

   Yang salah adalah: **double** (merupakan keyword yang digunakan tanpa **@**), **15Mobil** (diawali dengan angka), dan **Pagar#** (mengandung karakter **#** yang tidak diizinkan).

5. Apakah kegunaan namespace?

   Namespace digunakan untuk mengelompokkan elemen-elemen namespace (misalnya class) ke dalam suatu nama. Dengan namespace dapat dibuat suatu susunan hierarkis dan konflik nama dapat dihindari.

6. Karakter apakah yang digunakan untuk mengakhiri suatu statement?

   Karakter titik koma.

## B. Membuat Program

1. Buatlah program yang menampilkan kata "**Hai**" di console! Method **Main()** dalam program tersebut harus berada di dalam class yang bernama **new**.

Ingat, karena **new** merupakan keyword, maka kita harus menambahkan **@** jika ingin memakai nama **new** untuk class kita.

[program lengkap]
```
// Tes 2.B.1

class @new // Penggunaan keyword untuk identifier sangat tidak disarankan
{
 static void Main()
 {
      Console.WriteLine("Hai");
 }
}
```
[/program lengkap]

## C. Debugging

1. Program berikut bertujuan menampilkan kata "**Halo**" di console. Terdapat banyak kesalahan sehingga program tidak bisa dicompile. Betulkanlah!

[program lengkap]
```
Tes 2.C.1

Class DebuglahSaya
{
 // program dimulai dari method ini // static void main()
 {
      Console.WriteLine("Halo")
 }
}
```
[/program lengkap]

Perhatikan kutipan di bawah ini:

[kutipan program]
```
Tes 2.C.1
```
[/kutipan program]

Jika maksud kutipan di atas adalah untuk memberikan informasi tekstual mengenai programnya, maka informasi tersebut harus berada dalam comment. Comment single line maupun multi line dapat digunakan.

Lalu perhatikan kutipan ini:

[kutipan program]
```
Class DebuglahSaya
```
[/kutipan program]

C# membedakan antara huruf besar dengan huruf kecil. Untuk membuat class kita harus menggunakan keyword **class**, bukan **Class**.

Di baris berikut terdapat dua kesalahan:

[kutipan program]
```
// program dimulai dari method ini // static void main()
```
[/kutipan program]

Karena di awal baris terdapat //, maka seluruh baris tersebut menjadi comment. // yang kedua tidak berpengaruh apa-apa dalam baris ini. Masalah comment dapat diselesaikan dengan menggunakan comment multi line maupun dengan memulai comment single line di akhir baris. Commentnya dibuang pun sebenarnya tidak masalah.

Kesalahan kedua adalah kesalahan pemberian nama. Program C# memerlukan method **Main()** sebagai pintu masuk program. Di program yang sedang kita bahas hanya terdapat method **main()** (m huruf kecil).

Di baris bermasalah ini juga terdapat dua kesalahan:

[kutipan program]
```
Console.WriteLine("Halo")
```
[/kutipan program]

Statement di atas membutuhkan titik koma. Lalu karena tidak digunakan **using System** pada awal program maka namespace **System** harus ditulis saat memanggil **WriteLine()**. Alternatif lain adalah membubuhkan **using System** pada awal program.

Contoh program jadinya adalah sebagai berikut:

[program lengkap]
```
// Tes 2.C.1

using System;

class DebuglahSaya
{
 static void Main() // program dimulai dari method ini
 {
      Console.WriteLine("Halo");
 }
}
```
[/program lengkap]

# Bab 4: Tipe-Tipe Data dan Operator

## A. Pertanyaan

1. Sebutkan type dari literal-literal di bawah ini!

   `0` `'h'` `"a"` `50.98` `100ul`

   Secara berurutan, typenya adalah **int**, **char**, **string**, **double**, dan **ulong**.

2. Apakah kegunaan operator increment (++) dan decrement (--)? Apakah perbedaan antara penggunaan prefix (misal **++x**) dengan postfix (misal **x++**)?

   Increment akan menambah nilai variabel angka dengan **1** dan decrement akan mengurangi nilai variabel angka dengan **1**. Pada bentuk prefix nilai yang dikembalikan adalah nilai sesudah variabel yang bersangkutan diincrement/didecrement, sedangkan pada bentuk postfix nilai yang dikembalikan adalah nilai sebelumnya.

3. Apakah arti escape sequence **\n** dan **\t**?

   **\n** berarti new line dan **\t** berarti tabulasi horizontal.

4. Dari berbagai type angka, manakah yang tidak kompatibel dengan decimal?

   **double** dan **float**.

5. Apakah syarat sebelum suatu variabel dapat digunakan?

   Variabel teresebut harus sudah dideklarasikan dan memiliki nilai.

6. Apakah yang salah dengan kutipan program berikut?

   [kutipan program]
   ```
   int umur = 12, double berat = 30.5;
   ```
   [/kutipan program]

   Kita tidak bisa mendeklarasikan lebih dari satu type variabel dalam sebuah satement. Statement di atas mencoba mendeklarasikan **umur** yang typenya **int** dan **berat** yang typenya **double**.

7. Berapakah nilai akhir dari **var1**, **var2**, dab **var3** pada kutipan program berikut?

   [kutipan program]
   ```
   int var1 = 5, var2 = 10, var3 = 3;
   var1 += var2 * var3--;
   ```
   [/kutipan program]

Operator decrement postfix dikenakan pada **var3**, sehingga nilainya menjadi **2**. Namun karena decrement yang digunakan adalah bentuk postfix, maka nilai yang dikembalikan adalah nilai sebelumnya yaitu **3**. Hasil perkalian **var2** dengan **var3--** adalah **10 * 3** atau **30**. **var1** nilainya ditambah dengan hasil perkalian, sehingga nilai **var1** menjadi **5 + 30** atau **35**. **var2** hanya menjadi operand dari operasi perkalian sehingga nilainya tidak berubah.

8. Apakah konsekuensi mengecast **int** menjadi **double**? Bagaimana kalau sebaliknya?

   Tidak ada resiko dalam mengecast **int** menjadi **double** karena **double** lebih besar dari **int**. Saat mengecast **double** menjadi **int** nilai fraksionalnya akan hilang dan jika nilainya diluar batas **int** maka akan diperoleh nilai yang mungkin tidak diinginkan.

## B. Membuat Program

1. Buatlah suatu program yang menghitung luas suatu persegi panjang! Panjang persegi panjang tersebut adalah **11** dan lebarnya **6.12**.

   Kita akan menggunakan **double** karena **6.12** bukanlah bilangan bulat. **float** dan **decimal** juga dapat dipakai.

   [program lengkap]
   ```
   // Tes 3.B.1

   using System;

   class Luas
   {
    static void Main()
    {
         double panjang = 11;
         double lebar = 6.12;
         Console.WriteLine("Luas:");
         Console.WriteLine(panjang * lebar);
    }
   }
   ```
   [/program lengkap]

2. Buatlah sebuah program yang menghitung berapa kaki + inci yang ada dalam **193** inci! Contohnya **13** inci adalah **1** kaki + **1** inci. (**1** kaki sama dengan **12** inci)

   Kita akan menggunakan pembagian integer dan operasi modulus untuk mendapatkan jawabannya. Type variabel yang akan kita gunakan adalah **int**.

   [program lengkap]
   ```
   // Tes 3.B.2

   using System;
   ```

```
class KakiDanInci
{
 static void Main()
 {
      int panjangDalamInci = 193;
      Console.Write(panjangDalamInci);
      Console.WriteLine(“ inci sama dengan:”);
      Console.Write(panjangDalamInci / 12);  // hasilnya adalah int juga
      Console.WriteLine(“ kaki”);
      Console.WriteLine(“+”);
      Console.Write(panjangDalamInci % 12); // menghitung sisanya
      Console.WriteLine(“ inci”);
 }
}
```
[/program lengkap]

## C. Debugging

1. Program di bawah ini bertujuan meghitung jumlah murid yang ada di sebuah sekolah. Hasilnya tidak sesuai dengan yang diharapkan. Betulkanlah!

   [program lengkap]
   ```
   // Tes 3.C.1

   using System;

   class JumlahMurid
   {
    static void Main()
    {
         byte kelas1 = 50;
         byte kelas2 = 42;
         byte kelas3 = 56;
         byte kelas4 = 47;
         byte kelas5 = 28;
         byte kelas6 = 55;
         // harus dicast menjadi byte sebab hasil penjumlahan ini adalah int
         byte totalMurid = (byte) kelas1 + kelas2 + kelas3  + kelas4 + kelas5 +
   kelas6;
         Console.WriteLine(“Total Murid:”);
         Console.WriteLine(totalMurid);
    }
   }
   ```
   [/program lengkap]

   Kalau kita coba hitung sendiri, maka total murid seharusnya adalah **278**. Program kita menghasilkan **22**. Ini disebabkan karena hasil penjumlahan melebihi batas atas **byte** yaitu **255**. Solusinya adalah mengubah type **totalMurid** menjadi **short** atau yang lebih tinggi. Ini contoh penyelesaiannya:

[program lengkap]
```
// Tes 3.C.1

using System;

class JumlahMurid
{
 static void Main()
 {
      byte kelas1 = 50;
      byte kelas2 = 42;
      byte kelas3 = 56;
      byte kelas4 = 47;
      byte kelas5 = 28;
      byte kelas6 = 55;
      int totalMurid = kelas1 + kelas2 + kelas3 + kelas4 + kelas5 + kelas6;
      Console.WriteLine("Total Murid:");
      Console.WriteLine(totalMurid);
 }
}
```
[/program lengkap]

2. Program di bawah ini bertujuan untuk menghitung total biaya yang harus dibayar per tahun. Biaya listrik dan air tiap bulan masing-masing adalah **100** dan **30**. Program ini menampilkan jawaban **460** yang tidak mungkin adalah jawaban yang benar. Biaya listrik untuk setahun saja sudah **100 * 12 = 1000**. Cari kesalahannya dan betulkan!

[program lengkap]
```
// Tes 3.C.2

using System;

class Biaya
{
 static void Main()
 {
      int biayaListrikPerBulan = 100;
      int biayaAirPerBulan = 30;
      int biayaPerTahun = biayaListrikPerBulan + biayaAirPerBulan * 12;
      Console.WriteLine("Total biaya:");
      Console.WriteLine(biayaPerTahun);
 }
}
```
[/program lengkap]

Kesalahan terdapat dalam baris berikut:

[kutipan program]
```
int biayaPerTahun = biayaListrikPerBulan  + biayaAirPerBulan * 12;
```

[/kutipan program]

Yang akan dikalikan dengan **12** hanyalah **biayaAirPerBulan**, padahal untuk mendapatkan jawaban yang benar kita harus menghitung ***[totalBiayaPerBulan]* * 12**. Gunakan tanda kurung untuk memperbaiki program:

[program lengkap]
```
// Tes 3.C.2

using System;

class Biaya
{
 static void Main()
 {
      int biayaListrikPerBulan = 100;
      int biayaAirPerBulan = 30;
      int biayaPerTahun = (biayaListrikPerBulan + biayaAirPerBulan) * 12;
      Console.WriteLine("Total biaya:");
      Console.WriteLine(biayaPerTahun);
 }
}
```
[/program lengkap]

3. Program di bawah ini bertujuan untuk menampilkan **'baris 1'**, **'baris2'**, dan **'baris3'** dalam 3 baris yang berbeda. Modifikasi program sampai mendapatkan hasil yang diinginkan!

[program lengkap]
```
// Tes 3.C.3

using System;

class BarisBaris
{
 static void Main()
 {
      Console.WriteLine(@"'baris 1'\n'baris 2'\n'baris 3");
 }
}
```
[/program lengkap]

Pemrogram mencoba membuat baris baru dengan escape sequence **\n** padahal pemrogram menggunakan string verbatim. String verbatim tidak mengenal escape sequence! Untuk memecahkan masalahnya, kita dapat:

[kutipan program]
```
// menggunakan string verbatim yang memenuhi lebih dari 1 baris
Console.WriteLine(@"'baris 1'
'baris 2'
```

```
’baris 3”);
```
[/kutipan program]

[kutipan program]
```
// menggunakan string biasa
Console.WriteLine(“’baris 1’\n’baris 2’\n’baris 3”);
```
[/kutipan program]

[kutipan program]
```
// menggunakan string biasa dengan memanggil
// Console.WriteLine() lebih dari 1 kali
Console.WriteLine(“’baris 1’”);
Console.WriteLine(“’baris 2’”);
Console.WriteLine(“’baris 1’”);
```
[/kutipan program]

Ini contoh program jadinya (menggunakan solusi ke dua):

[program lengkap]
```
// Tes 3.C.3

using System;

class BarisBaris
{
 static void Main()
 {
      Console.WriteLine(“’baris 1’\n’baris 2’\n’baris 3”);
 }
}
```
[/program lengkap]

# Bab 5: Input dan Output Console

## A. Pertanyaan

1. Class apakah yang digunakan untuk berinteraksi dengan console? Terletak di namespace manakah class tersebut? Apakah class tersebut merupakan bagian dari C#?

   Class yang dimaksud bernama **Console** dan terletak di dalam namespace **System**. Class tersebut bukanlah merupakan bagian dari bahasa C#, tapi merupakan bagian dari class library .NET. Class **Console** bisa digunakan di semua bahasa pemrograman .NET, tidak hanya terbatas pada C#.

2. Apakah yang akan ditampilkan di layar bila statement di bawah dijalankan?

   [kutipan program]

```
Console.WriteLine("{0}{0}{0}{0}", "\b\b\b123");
```
[/kutipan program]

Yang akan tertulis di console adalah **“123”**. Ini karena string **“\b\b\b123”** berisi 3 backspace yang akan menghapus 3 karakter sebelumnya di baris yang bersangkutan.

3. Apakah yang dikembalikan oleh method **ReadLine()**?

   **string** yang berisi input yang diberikan.

4. Bagaimanakah cara membuat type dasar berdasarkan nilai yang terkandung dalam suatu **string**?

   Dengan menggunakan method static **Parse()** dari type yang bersangkutan.

## B. Membuat Program

1. Buatlah program yang menampilkan karakter dari sebuah kode Unicode beserta kode Unicodenya dalam heksadesimal. Misalnya jika pengguna menuliskan **97** maka program menampilkan **a** dan **61** (karakter **a** kode Unicodenya adalah **97**).

   Karena Unicode besarnya 16 bit dan tidak ada kode Unicode negatif, maka **ushort** akan kita gunakan. Ini contoh program jadinya:

   [program lengkap]
```
// Tes 5.B.1

using System;

class AngkaKeKarakter
{
 static void Main()
 {
      Console.WriteLine(“Tuliskan kode Unicode:”);
      char karakter = (char) ushort.Parse(Console.ReadLine());
      Console.WriteLine(“Karakternya adalah: {0}”, karakter);
      Console.WriteLine(“Kode Unicodenya dalam hex: {0:x}”, (ushort)
karakter);
 }
}
```
   [/program lengkap]

   [catatan]
   Beberapa karakter-karakter spesial mungkin tidak dapat ditampilkan di console anda atau ditampilkan sebagai tanda tanya.
   [/catatan]

2. Buatlah program yang menanyakan 3 buah angka. Tampilkan jumlah dan rata-ratanya, dan gunakan hanya sebuah variabel dalam program ini!

   Operator += akan kita gunakan untuk penjumlahan. Ini contoh program jadinya:

   [program lengkap]
```
// Tes 5.B.2

using System;

class TigaAngka
{
 static void Main()
 {
      double jumlah = 0;
      Console.Write("Angka pertama? ");
      jumlah += double.Parse(Console.ReadLine());
      Console.Write("Angka kedua? ");
      jumlah += double.Parse(Console.ReadLine());
      Console.Write("Angka ketiga? ");
      jumlah += double.Parse(Console.ReadLine());
      Console.WriteLine("Jumlah: {0}", jumlah);
      Console.WriteLine("Rata-rata: {0}", jumlah / 3);
 }
}
```
   [/program lengkap]

3. Kecepatan cahaya adalah 300,000,000 m/s. Buatlah sebuah program yang menghitung jarak yang ditempuh cahaya dalam suatu selang waktu. Gunakan notasi ilmiah untuk outputnya dengan menunjukkan 3 angka di belakang koma pada mantisanya.

   [program lengkap]
```
// Tes 5.B.3

using System;

class Cahaya
{
 static void Main()
 {
      double waktu;
      Console.Write("Tuliskan selang waktu: ");
      waktu = double.Parse(Console.ReadLine());
      Console.WriteLine("Jarak yang ditempuh cahaya: {0:e3}", waktu * 3e8);
// 3e8 sama dengan (3 * (10 pangkat 8)) atau 300000000
 }
}
```
   [/program lengkap]

## C. Debugging

1. Program berikut bertujuan menampilkan jumlah pajak yang harus dibayar. Besarnya pajak adalah 5% dari penghasilan. Program tersebut tidak berjalan dengan seharusnya. Betulkan!

[program lengkap]
```
// Tes 5.C.1

using System;

class Pajak
{
 static void Main()
 {
      decimal penghasilan;
      decimal persentasePajak = 5 / 100;
      Console.WriteLine("Berapakah penghasilan anda per tahun?");
      penghasilan = decimal.Parse(Console.ReadLine());
      Console.WriteLine("Anda harus membayar pajak sebesar (0) Rupiah.",
penghasilan * persentasePajak);
 }
}
```
[/program lengkap]

Kita coba analisa outputnya:

[console]
```
Berapakah penghasilan anda per tahun?
50000000
Anda harus membayar pajak sebesar (0) Rupiah.
```
[/console]

Bisa kita lihat bahwa ada yang salah dalam string formatnya. **"(0)"** harus diganti dengan **"{0}"**. Inilah statement yang telah diperbaiki:

[kutipan program]
```
Console.WriteLine("Anda harus membayar pajak sebesar {0} Rupiah.",
penghasilan * persentasePajak);
```
[/kutipan program]

Namun ternyata outputnya tetap kacau:

[console]
```
Berapakah penghasilan anda per tahun?
50000000
Anda harus membayar pajak sebesar 0 Rupiah.
```
[/console]

Berapapun angka yang kita masukkan, hasilnya selalu **0**. Untuk mengetahui mengapa hal tersebut bisa terjadi, mari kita cek nilai-nilai variabel yang bersangkutan. Tambahkan statement-statement berikut pada akhir program:

[kutipan program]
```
Console.WriteLine("nilai penghasilan: {0}", penghasilan);
Console.WriteLine("nilai persentasePajak: {0}", persentasePajak);
```
[/kutipan program]

Program kita jalankan sekali lagi:

[console]
```
Berapakah penghasilan anda per tahun?
50000000
Anda harus membayar pajak sebesar 0 Rupiah.
nilai penghasilan: 50000000
nilai persentasePajak: 0
```
[/console]

Kini masalah utamanya berhasil kita temukan. persentasePajak nilainya **0**! Mengapa itu bisa terjadi? Perhatikan statement berikut:

[kutipan program]
```
decimal persentasePajak = 5 / 100;
```
[/kutipan program]

**5** dan **100** typenya sama-sama **int**, karena itu hasil pembagiannya juga **int** yaitu **0**! Masalah dapat kita atasi dengan membuat salah satu atau kedua operandnya **decimal**. Ini contoh program jadinya:

[program lengkap]
```
// Tes 5.C.1

using System;

class Pajak
{
 static void Main()
 {
      decimal penghasilan;
      decimal persentasePajak = 5m / 100m;
      Console.WriteLine("Berapakah penghasilan anda per tahun?");
      penghasilan = decimal.Parse(Console.ReadLine());
      Console.WriteLine("Anda harus membayar pajak sebesar {0} Rupiah.",
penghasilan * persentasePajak);
 }
}
```
[/program lengkap]

Output:

[console]
Berapakah penghasilan anda per tahun?
50000000
Anda harus membayar pajak sebesar 2500000 Rupiah.
[/console]

# Bab 6: Mengubah Alur Program

## A. Pertanyaan

1. Perhatikan kutipan program berikut:

   [kutipan program]
   ```
   if(a == 0)
     if(b == 100)
     {
           Console.WriteLine("Jawaban benar...");
           nilai++;
     }
   else
     Console.WriteLine("Jawaban salah...");
   ```
   [/kutipan program]

   Manakah **if** yang merupakan pasangan **else**?

   Yang merupakan pasangannya adalah **if(b == 100)** sebab itu adalah **if** terdekat yang belum memiliki pasangan.

2. Apakah perbedaan operator logical biasa dengan operator logical short circuit?

   Versi short circuit AND dan OR bekerja lebih efisien dari versi normalnya. Operator logical biasa akan mengevaluasi semua operandnya, sedangkan dalam beberapa kondisi operator logical short circuit dapat memperhitungkan hasil operasi sebelum semua operandnya dievaluasi.

3. Apakah hasil dari expression **(false ? nilai1 : nilai2)**?

   Karena kondisi **false**, maka hasilnya adalah **nilai2**.

4. Apakah type yang dapat menjadi pengontrol statement **switch**?

   Type-type integer seperti **byte**, **int**, dan **char**.

5. Apakah kegunaan **break** dalam statement **switch**?

Untuk keluar dari statement **switch** tersebut.

6. Mengapa penggunaan **goto** tidak dianjurkan?

   Sebab penggunaan **goto** yang berlebihan akan menyebabkan alur program susah dipahami. Lagipula banyak konstruksi bahasa lain yang bisa dijadikan alternatif.

7. Jika **a = 1**, **b = 10**, dan **variabelBool = true**, apakah nilai dari expression-expression berikut?

| | | |
|---|---|---|
| a > 0 && b != 0 | !(a == b) && variabelBool | a == 2 \|\| b == 1 \|\| variabelBool |
| !variabelBool | variabelBool && (a > b ? true : false) | |

| | | |
|---|---|---|
| true | true | true |
| false | false | |

## B. Membuat Program

1. Buatlah program yang meminta pengguna memasukkan 2 angka positif.
   - Jika angka yang dimasukkan tidak sah (lebih kecil atau sama dengan **0**), tampilkan **"Anda memasukkan angka yang tidak sah!"** atau yang sejenisnya.
   - Jika angka yang dimasukkan sama, tampilkan **"Anda memasukkan angka yang sama."** atau yang sejenisnya.
   - Jika kedua kasus di atas tidak terjadi, cek mana angka yang lebih besar. Lalu beritahu pengguna apakah angka yang lebih besar habis dibagi angka yang lebih kecil (sisa pembagian **0**).

   [program lengkap]

```
// Tes 6.B.1

using System;

class HabisDibagi
{
 static void Main()
 {
      int var1, var2;
      Console.WriteLine("Masukkan 2 angka yang lebih besar dari 0!");
      Console.Write("Angka 1: ");
      var1 = int.Parse(Console.ReadLine());
      Console.Write("Angka 2: ");
      var2 = int.Parse(Console.ReadLine());
      if(var1 > 0 && var2 > 0)
      {
           if(var1 > var2 && var1 % var2 == 0)
                Console.WriteLine("Angka yang lebih besar ({0}) habis dibagi
angka yang lebih kecil ({1}).", var1, var2);
```

```
            else if(var1 < var2 && var2 % var1 == 0)
                Console.WriteLine("Angka yang lebih besar ({0}) habis dibagi
angka yang lebih kecil ({1}).", var2, var1);
            else if(var1 == var2)
                Console.WriteLine("Anda memasukkan angka yang sama.");
            else
                Console.WriteLine("Angka yang lebih besar ({0}) tidak habis
dibagi angka yang lebih kecil ({1}).", (var1 > var2 ? var1 : var2), (var1 < var2 ?
var1 : var2));
        }
        else
            Console.WriteLine("Anda memasukkan angka yang tidak sah!");
    }
}
```
[/program lengkap]

Kemungkinan besar konstruksi **if-else** yang anda buat berbeda dengan contoh jawaban di atas. Itu tidak menjadi masalah, yang penting program berjalan dengan baik. Untuk mengetes program anda, coba masukkan input-input di bawah ini:

| input 1 | input 2 | hasil yang seharusnya |
|---|---|---|
| 1 | 0 | angka tidak sah |
| 0 | 1 | |
| 1 | -1 | |
| -1 | 1 | |
| -1 | -1 | |
| 0 | 0 | |
| 0 | -1 | |
| -1 | 0 | |
| 5 | 5 | angka sama |
| 10 | 5 | angka besar habis dibagi angka kecil |
| 5 | 10 | |
| 8 | 9 | angka besar tidak habis dibagi angka kecil |
| 9 | 8 | |

2. Buatlah program yang menanyakan 3 buah angka. Tampilkan jumlah dan rata-ratanya, dan gunakan hanya sebuah variabel dalam program ini!

   Operator += akan kita gunakan untuk penjumlahan. Ini contoh program jadinya:

   [program lengkap]
   ```
   // Tes 5.B.2

   using System;

   class TigaAngka
   {
    static void Main()
    {
        double jumlah = 0;
   ```

```
        Console.Write("Angka pertama? ");
        jumlah += double.Parse(Console.ReadLine());
        Console.Write("Angka kedua? ");
        jumlah += double.Parse(Console.ReadLine());
        Console.Write("Angka ketiga? ");
        jumlah += double.Parse(Console.ReadLine());
        Console.WriteLine("Jumlah: {0}", jumlah);
        Console.WriteLine("Rata-rata: {0}", jumlah / 3);
    }
}
```
[/program lengkap]

3. Kecepatan cahaya adalah 300,000,000 m/s. Buatlah sebuah program yang menghitung jarak yang ditempuh cahaya dalam suatu selang waktu. Gunakan notasi ilmiah untuk outputnya dengan menunjukkan 3 angka di belakang koma pada mantisanya.

[program lengkap]
```
// Tes 5.B.3

using System;

class Cahaya
{
 static void Main()
 {
        double waktu;
        Console.Write("Tuliskan selang waktu: ");
        waktu = double.Parse(Console.ReadLine());
        Console.WriteLine("Jarak yang ditempuh cahaya: {0:e3}", waktu * 3e8);
// 3e8 sama dengan (3 * (10 pangkat 8)) atau 300000000
 }
}
```
[/program lengkap]

## C. Debugging

1. Program berikut bertujuan menampilkan jumlah pajak yang harus dibayar. Besarnya pajak adalah 5% dari penghasilan. Program tersebut tidak berjalan dengan seharusnya. Betulkan!

[program lengkap]
```
// Tes 5.C.1

using System;

class Pajak
{
 static void Main()
 {
```

```
        decimal penghasilan;
        decimal persentasePajak = 5 / 100;
        Console.WriteLine("Berapakah penghasilan anda per tahun?");
        penghasilan = decimal.Parse(Console.ReadLine());
        Console.WriteLine("Anda harus membayar pajak sebesar (0) Rupiah.",
penghasilan * persentasePajak);
    }
}
```
[/program lengkap]

Kita coba analisa outputnya:

[console]
```
Berapakah penghasilan anda per tahun?
50000000
Anda harus membayar pajak sebesar (0) Rupiah.
```
[/console]

Bisa kita lihat bahwa ada yang salah dalam string formatnya. **"(0)"** harus diganti dengan **"{0}"**. Inilah statement yang telah diperbaiki:

[kutipan program]
```
Console.WriteLine("Anda harus membayar pajak sebesar {0} Rupiah.",
penghasilan * persentasePajak);
```
[/kutipan program]

Namun ternyata outputnya tetap kacau:

[console]
```
Berapakah penghasilan anda per tahun?
50000000
Anda harus membayar pajak sebesar 0 Rupiah.
```
[/console]

Berapapun angka yang kita masukkan, hasilnya selalu **0**. Untuk mengetahui mengapa hal tersebut bisa terjadi, mari kita cek nilai-nilai variabel yang bersangkutan. Tambahkan statement-statement berikut pada akhir program:

[kutipan program]
```
Console.WriteLine("nilai penghasilan: {0}", penghasilan);
Console.WriteLine("nilai persentasePajak: {0}", persentasePajak);
```
[/kutipan program]

Program kita jalankan sekali lagi:

[console]
```
Berapakah penghasilan anda per tahun?
50000000
Anda harus membayar pajak sebesar 0 Rupiah.
```

```
nilai penghasilan: 50000000
nilai persentasePajak: 0
```
[/console]

Kini masalah utamanya berhasil kita temukan. persentasePajak nilainya **0**! Mengapa itu bisa terjadi? Perhatikan statement berikut:

[kutipan program]
```
decimal persentasePajak = 5 / 100;
```
[/kutipan program]

**5** dan **100** typenya sama-sama **int**, karena itu hasil pembagiannya juga **int** yaitu **0**! Masalah dapat kita atasi dengan membuat salah satu atau kedua operandnya **decimal**. Ini contoh program jadinya:

[program lengkap]
```
// Tes 5.C.1

using System;

class Pajak
{
 static void Main()
 {
      decimal penghasilan;
      decimal persentasePajak = 5m / 100m;
      Console.WriteLine("Berapakah penghasilan anda per tahun?");
      penghasilan = decimal.Parse(Console.ReadLine());
      Console.WriteLine("Anda harus membayar pajak sebesar {0} Rupiah.",
penghasilan * persentasePajak);
 }
}
```
[/program lengkap]

Output:

[console]
```
Berapakah penghasilan anda per tahun?
50000000
Anda harus membayar pajak sebesar 2500000 Rupiah.
```
[/console]

# Lampiran B: Binary dan Heksadesimal

Manusia dalam kehidupan sehari-hari menggunakan sistem angka berbasis sepuluh atau biasa disebut sistem desimal. Di dalam sistem desimal angka nol sampai sembilan dituliskan dengan simbol **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, dan **9**. Untuk menuliskan angka selain **0** sampai **9**, kombinasi angka-angka tersebut digunakan. Perhatikan contoh angka di bawah ini dan penjabarannya:

$$637 = 600 + 30 + 7$$
$$= (6 * 100) + (3 * 10) + (7 * 1)$$
$$= (6 * 10^2) + (3 * 10^1) + (7 * 10^0)$$

[catatan]
Ingat bahwa $\mathbf{10^0}$ adalah **1** (sebuah bilangan **x** jika dipangkatkan **0** hasilnya adalah **1**, dengan catatan **x** bukan **0**).
[/catatan]

Pada bilangan **637**, **7** biasa disebut sebagai komponen satuan, **3** sebagai komponen puluhan, dan **6** sebagai komponen ratusan. Ini contoh lainnya:

$$1304 = 1000 + 300 + 0 + 4$$
$$= (1 * 1000) + (3 * 100) + (0 * 10) + (4 * 1)$$
$$= (1 * 10^3) + (3 * 10^2) + (0 * 10^1) + (4 * 10^0)$$

Komputer pada dasarnya adalah mesin yang hanya dapat memahami **0** dan **1**. Karena itu, angka dan data-data lainnya harus disimpan dengan menggunakan kombinasi **0** dan **1** tersebut. Sistem angka yang hanya menggunakan **0** dan **1** disebut sistem angka berbasis dua atau sistem binary. Inilah beberapa angka dalam bentuk desimal dan binarynya:

| **Desimal** | **Binary** | **Penjabaran ke Desimal** |
|---|---|---|
| 0 | 0 | $0 * 2^0$ |
| 1 | 1 | $1 * 2^0$ |
| 2 | 10 | $(1 * 2^1) + (0 * 2^0)$ |
| 3 | 11 | $(1 * 2^1) + (1 * 2^0)$ |
| 4 | 100 | $(1 * 2^2) + (0 * 2^1) + (0 * 2^0)$ |
| 5 | 101 | $(1 * 2^2) + (0 * 2^1) + (1 * 2^0)$ |
| 6 | 110 | $(1 * 2^2) + (1 * 2^1) + (0 * 2^0)$ |
| 7 | 111 | $(1 * 2^2) + (1 * 2^1) + (1 * 2^0)$ |
| 8 | 1000 | $(1 * 2^3) + (0 * 2^2) + (0 * 2^1) + (0 * 2^0)$ |
| 9 | 1001 | $(1 * 2^3) + (0 * 2^2) + (0 * 2^1) + (1 * 2^0)$ |
| 10 | 1010 | $(1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0)$ |

Jika di sistem desimal terdapat satuan, puluhan, ratusan, dan seterusnya, maka di sistem binary kita akan menjumpai satuan, duaan, empatan, delapanan, dan seterusnya.

Digit suatu bilangan binary disebut bit. Misalnya bilangan binary berikut:

0001 1000

... terdiri dari 8 bit atau 8 digit. Dengan bit yang lebih banyak tentunya kita bisa melambangkan bilangan yang lebih besar. Misalnya dengan 2 bit, angka terbesar yang bisa kita representasikan hanyalah adalah **$11_{(binary)}$** atau **3**. Dengan 8 bit angka terbesar yang bisa kita representasikan adalah **$1111\ 1111_{(binary)}$** atau **255**.

[catatan]
8 bit sama dengan 1 byte.
[/catatan]

Penjumlahan binary dapat dengan mudah dilakukan baik oleh kita maupun oleh komputer sebab angka yang terlibat hanyalah **0** dan **1**. Contoh berikut memperlihatkan penjumlahan 2 bilangan binary (8 bit) menggunakan cara yang biasa kita pakai:

```
  binary      desimal

0000 1010       10
0000 1011       11
----------- +   --- +
0001 0101       21
   ∪ ∪
```

Sayangnya sistem binary dapat memusingkan mata orang yang melihatnya apabila banyak bit yang terlibat, seperti di contoh berikut ini:

0110 0001 1010 1001 1111 0011

Untuk mempermudah penggunaan sistem binary bagi manusia, dipakailah sistem heksadesimal atau bilangan berbasis 16. Untuk melambangkan bilangan **10** sampai **15**, abjad **a - f** atau **A - F** digunakan. Perhatikan tabel berikut:

| **Heksadesimal** | **Desimal** | **Binary** |
|---|---|---|
| 0 | 0 | 0000 0000 |
| 1 | 1 | 0000 0001 |
| 2 | 2 | 0000 0010 |
| … | … | … |
| 9 | 9 | 0000 1001 |
| a | 10 | 0000 1010 |
| b | 11 | 0000 1011 |
| c | 12 | 0000 1100 |
| d | 13 | 0000 1101 |
| e | 14 | 0000 1110 |
| f | 15 | 0000 1111 |
| 10 | 16 | 0001 1000 |

Kita bisa melihat bahwa satu digit heksadesimal bisa melambangkan **0** sampai **15**. Ini sama persis dengan jangkauan 4 bit binary! Bilangan binary dapat dengan mudah dikonversi ke bentuk heksadesimal dengan memilahnya menjadi bagian yang terdiri dari 4 bit. Contohnya bilangan binary berikut:

0110 0001 1010 1001 1111 0011

... dalam heksadesimal bentuknya menjadi:

6 1 a 9 f 3

Penjabaran bilangan heksadesimal tersebut ke dalam desimal adalah sebagai berikut:

$(6 * 16^5) + (1 * 16^4) + (10 * 16^3) + (9 * 16^2) + (15 * 16^1) + (3 * 16^0)$

... yang hasil akhirnya adalah **6400499** (sama dengan nilai bilangan binarynya, anda dapat membuktikannya sendiri).

[tanya jawab]
**Q:** Bagaimana dengan sistem oktal atau bilangan berbasis 8?
**A:** Pada sistem oktal, 3 bit binary dapat diubah menjadi 1 digit bilangan oktal dengan mudah. Sekarang sistem ini sangat jarang dipakai karena orang cenderung menggunakan sistem heksadesimal.
[/tanya jawab]

Pembahasan terakhir pada lampiran ini adalah mengenai bilangan negatif. Sejauh ini kita mengasumsikan bahwa bilangan negatif tidak ada. Contohnya pada bilangan binary 8 bit seperti:

0000 0000

... kita berasumsi semua bit digunakan untuk melambangkan bilangan positif (biasa disebut unsigned).

Jika kita ingin dapat melambangkan bilangan negatif, maka sebuah bit harus digunakan sebagai penanda positif/negatif. Tipe bilangan binary seperti ini disebut signed. Bit penanda adalah bit terkiri/teratas yang biasa disebut sign bit. Jika bit penanda bernilai **1**, maka bilangan tersebut negatif, sedangkan jika nilainya **0** maka bilangan tersebut positif. Dalam hal ini jumlah bit harus ditentukan (misalnya 8 bit). Bit penanda digarisbawahi pada bilangan binary 8 bit berikut:

1111 1000

Ini adalah langkah-langkah untuk menerjemahkan bilangan tersebut ke dalam desimal:

1. Bit penanda bernilai **1**, berarti bilangan tersebut negatif.
2. Hilangkan sign bit, kita dapatkan **111 1000**.
3. Ganti **0** dengan **1** dan sebaliknya, kita dapatkan **000 0111**.
4. **000 0111** dalam desimal nilainya **4 + 2 + 1 = 7**.
5. Jumlahkan **1**, kita dapatkan **8**.
6. Artinya **1111 1000$_{\text{(signed binary)}}$** bernilai **-8**.

Untuk membuktikan kebenarannya, kita coba jumlahkan dengan **0000 1001**$_{\textbf{(signed binary)}}$ atau **9**.

```
signed binary       desimal

1111 1000           -8
0000 1001            9
------------ +      --- +
0000 0001            1
```

Karena bit yang ditentukan hanya 8, maka kelebihan bit dibuang/dihiraukan. Terbukti bahwa hasilnya benar yaitu **0000 0001**$_{\textbf{(signed binary)}}$ atau **1**.

Sebagai pelengkap penulis cantumkan cara mengubah bilangan desimal negatif ke bentuk binarynya:

1. Misalnya kita ingin mengetahui bentuk binary dari **-1**. Dalam contoh ini kita gunakan bilangan binary 8 bit.
2. Kita cari dulu binary dari **1**. Kita dapatkan **0000 0001**.
3. Ganti **0** dengan **1** dan sebaliknya, kita dapatkan **1111 1110**.
4. Jumlahkan **1** ke dalamnya, kita dapatkan **1111 1111**.
5. **1111 1111**$_{\textbf{(singed binary)}}$ adalah **-1**.

[catatan]
Pada jumlah bit yang sama, jarak yang ditawarkan bilangan binary singed dan unsigned berbeda. Contohnya bilangan binary 8 bit unsigned bisa melambangkan **0** sampai **255**, sedangkan bilangan binary 8 bit signed jaraknya adalah **-128** sampai **127**.
[/catatan]

# Lampiran C: Prioritas Operator

Urutan operator yang dilaksanakan dalam suatu expression didasarkan pada prioritas operator. Operator yang memiliki prioritas lebih tinggi akan dilaksanakan sebelum operator yang memiliki prioritas lebih rendah. Tabel berikut akan menampilkan operator-operator dari yang prioritasnya paling tinggi ke yang paling rendah.

| **Operator-Operator dari Prioritas Tertinggi ke Prioritas Terendah** |
|---|
| x.y f(x) a[x] x++ x-- new typeof checked unchecked |
| + - ! ~ ++x --x (T)x |
| * / % |
| + - |
| << >> |
| < > <= >= is as |
| == != |
| & |
| ^ |
| \| |
| && |
| \|\| |
| ?: |
| = *= /= %= += -= <<= >>= &= ^= \|= |

Operator-operator dalam baris yang sama memiliki prioritas yang sama. Jika terdapat dua operator yang prioritasnya sama dalam suatu expression, maka urutan pelaksanaannya adalah:

- Operator-operator assignment (**= *= /= %= += -= <<= >>= &= ^= |=**) dan operator conditional (**?:**) dilaksanakan dari kanan ke kiri. Contohnya **x = y = z** dilaksanakan sebagaimana **x = (y = z)**.
- Operator-operator lainnya dilaksanakan dari kiri ke kanan. Contohnya **1 + 2 - 3** dilaksanakan sebagaimana **(1 + 2) - 3**.

Kita dapat menggunakan kurung untuk mengubah urutan pelaksanaan. Misalnya **1 + 2 * 3** akan mengalikan **2** dengan **3**, lalu menjumlahkan hasilnya dengan **1**, namun **(1 + 2) * 3** akan menjumlahkan **1** dengan **2**, lalu mengalikan hasilnya dengan **3**.

# Daftar Pustaka

[Bal] E. Balagurusamy: Object Oriented Programming with C++, Tata McGraw-Hill, 1998
[Den] Dennis M. Ritchie: The Development of the C Language, ACM, 1993
[ECMA] Standard ECMA 334 - C# Language Specification, ECMA, 2001
[Herb] Herbert Schildt: C# - A Begginer's Guide, Osborne/McGraw-Hill, 2001
[Ivor] Ivor Horton: Ivor Horton's Beginning C++ - The Complete Language, Wrox, 1998
[NET] Microsoft .NET Framework SDK Documentation, Microsoft, 2001
[VS] Microsoft Visual Studio .NET Documentation, Microsoft, 2001
[Sau] Saurabh Nandu: www.mastercsharp.com
[Tan] Tan Soei Tien: Bahasa C# untuk Pemrograman Berorientasi Objek, Elex Media Komputindo, 2001