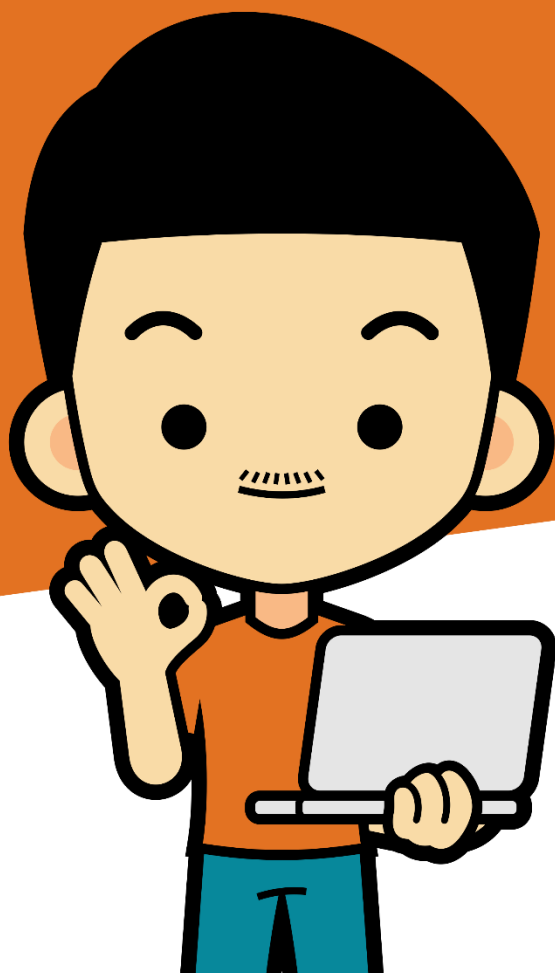


# Mudah Belajar Pemrograman C#

sekalipun Anda **awam** di bidang IT



dilengkapi dengan  
**praktikum**  
untuk memperdalam  
pemahaman Anda

Oleh:  
**Dian Nandiwardhana**

# Daftar Isi

<b>1</b>	<b>MENGAPA BELAJAR PEMROGRAMAN C#</b>	<b>1</b>
1.1	Sekilas Sejarah C#	2
1.2	3 Alasan Mengapa Bahasa C# Layak Anda Pelajari	2
<b>2</b>	<b>SEKILAS TENTANG .NET</b>	<b>14</b>
2.1	Apa itu .NET?	15
2.2	Mengkompilasi program .NET	16
<b>3</b>	<b>MENGENAL ELEMEN-ELEMEN DASAR BAHASA C#</b>	<b>19</b>
3.1	Membuat program Hello World	20
3.2	Elemen-elemen pemrograman C#	23
3.2.1	Namespace	23
3.2.2	Class	25
3.2.3	Method	27
3.2.4	Variabel	28
<b>4</b>	<b>KONSEP-KONSEP DASAR BAHASA C#</b>	<b>29</b>
4.1	Tipe data & variabel	30
4.2	Jenis-jenis Tipe Data	32
4.3	Statement / Pernyataan dalam bahasa C#	36

4.4	Identifier / Pengenal dalam bahasa C#	37
4.5	Operator	40
4.6	Konversi Tipe Data	42
<b>5</b>	<b>KONSEP MEMORI DALAM PEMROGRAMAN</b>	<b>46</b>
5.1	Alokasi memori	47
<b>6</b>	<b>CAKUPAN VARIABEL</b>	<b>52</b>
<b>7</b>	<b>MEMBERI KOMENTAR DI C#</b>	<b>57</b>
7.1	Komentar	58
7.2	Menggunakan komentar	58
7.3	Komentar baris-tunggal	59
7.4	Komentar multi-baris	60
7.5	Komentar dokumentasi XML	61
<b>8</b>	<b>PENGAMBILAN KEPUTUSAN</b>	<b>63</b>
8.1	Pengenalan	64
8.2	Pernyataan if	64
8.3	Pernyataan if-else	65
8.4	Pernyataan if-else if	66
8.5	Operator Ternary	67
8.6	Pernyataan switch	68

<b>9</b>	<b>PENGULANGAN / ITERASI</b>	<b>71</b>
9.1	Pengenalan	72
9.2	for loop	72
9.3	foreach loop	73
9.4	while loop	74
9.5	do-while loop	77
<b>10</b>	<b>METHOD</b>	<b>79</b>
10.1	Apa itu method	80
10.2	Mendeklarasikan sebuah method	80
10.3	Memanggil sebuah method	83
10.4	Memberi parameter pada sebuah method	85
10.5	Argumen opsional	87
10.6	Argumen bernama	88
10.7	Melewatkan Argumen	90
10.8	Method Overloading	95
	<b>PRAKTIKUM 1 : MENDEKLARASIKAN VARIABEL</b>	<b>98</b>
	<b>PRAKTIKUM 2 : MENGGUNAKAN PERNYATAAN IF</b>	<b>101</b>
	<b>PRAKTIKUM 3 : MENGGUNAKAN PERNYATAAN SWITCH</b>	<b>103</b>

<b>PRAKTIKUM 4 : MENGGUNAKAN FOR LOOP</b>	<b>106</b>
<b>PRAKTIKUM 5 : MENGGUNAKAN WHILE DAN DO-WHILE LOOP</b>	<b>109</b>
<b>PRAKTIKUM 6 : MENDEKLARASIKAN DAN MENGGUNAKAN METHOD</b>	<b>111</b>

## Tentang Penulis



### Dian Nandiwardhana

Nandi adalah seorang "*Self taught software developer*" yang memiliki keinginan untuk membantu Anda untuk menjadi seorang *software developer* profesional, tidak peduli apapun latar belakang Anda.

Mempelajari bahasa pemrograman secara otodidak sejak lulus SMA, membuat beliau bisa bekerja di Jepang sebagai *software developer* sejak tahun 2011 meskipun lulus S1 dari Jurusan Teknik Fisika hanya dengan IPK 2,74.

Sejak tinggal di Jepang, memasak yang semula menjadi aktifitas wajib, kini menjadi kegemaran beliau. Bagi beliau, melihat orang lain bisa menikmati masakannya, menjadi kepuasan tersendiri.

Apakah Anda ingin menguasai *skill* pemrograman agar siap menghadapi revolusi industri 4.0? Jika iya, Anda juga bisa mengikuti kursus online yang beliau sediakan secara gratis di kanal Youtube : "[Koding Bersama Nandi](#)".

# 1 Mengapa Belajar Pemrograman C#

## 1.1 Sekilas Sejarah C#

C# adalah bahasa pemrograman berbasis obyek modern yang dikembangkan pada tahun 2000 oleh Anders Hejlsber di Microsoft sebagai rival pemrograman Java. Pada saat itu, Sun (sekarang Oracle) tidak ingin Microsoft melakukan perubahan pada Java, sehingga Microsoft memutuskan untuk mengembangkan bahasa pemrogramannya sendiri.

Sejak pertama kali dikembangkan, dengan dukungan yang luas dari Microsoft, C# telah berkembang dengan sangat cepat. Sekarang C# adalah salah satu dari bahasa pemrograman paling populer di dunia.

## 1.2 3 Alasan Mengapa Bahasa C# Layak Anda Pelajari

Pastinya Anda mempunyai motivasi tersendiri mengapa Anda ingin belajar pemrograman terutama dengan bahasa C#. Kalau tidak, Anda tidak mungkin membaca buku ini, bukan?

Tapi, untuk lebih meyakinkan Anda mengapa mempelajari bahasa C# akan sangat bermanfaat bagi Anda, saya sudah merangkum beberapa alasan yang saya harap bisa lebih memotivasi Anda selama belajar pemrograman C# dari buku ini.

Sebenarnya ada cukup banyak alasan mengapa bahasa C# adalah bahasa pemrograman yang layak Anda pelajari. Namun, saya hanya akan menunjukkan 3 alasan yang menurut pendapat saya akan membuat Anda semakin termotivasi untuk mempelajari bahasa C#.



## 1. Bahasa pemrograman yang populer di kalangan developer profesional

Menurut *survey* yang dilakukan oleh **Stackoverflow** tiap tahunnya, dari tahun 2013 sampai dengan tahun 2017, bahasa C# selalu menempati urutan ke-4 sebagai bahasa pemrograman paling populer.

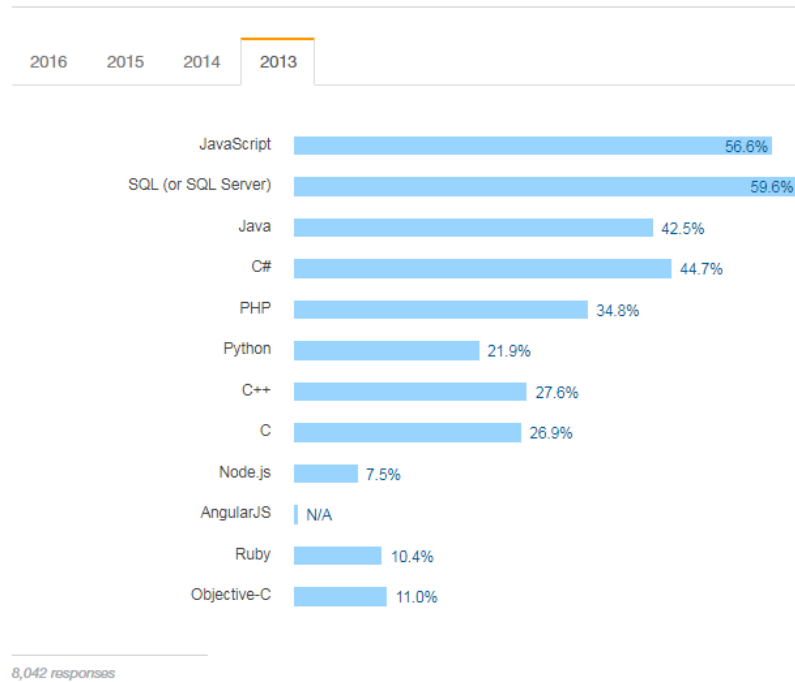
Pada *survey* yang diadakan di tahun 2013, 44.7% dari 8,042 responden memilih C# sebagai bahasa pemrograman yang digunakan. Di tahun 2014, 37.6% dari 6,537 responden memilih C#. Tahun 2015, 31.6% dari 21,982 responden memilih C#. Tahun 2016 30.9% dari 49,397 responden memilih C#. Sedangkan di tahun 2017, 34.1% dari 36,625 responden memilih C#.

Dalam *survey* yang dilakukan pada tahun 2018 lalu, bahasa C# memang mengalami penurunan peringkat sebagai bahasa pemrograman paling populer. Dari peringkat ke-4 menjadi peringkat ke-8. Itupun dikarenakan pada survey kali ini, Stackoverflow menambahkan bahasa skrip seperti Bash/Shell, bahasa markup seperti HTML dan bahkan bahasa style sheet seperti css ke dalam pertanyaan survey.

Selain itu, dengan semakin populernya teknologi *computer vision* dan *artificial intelligence*, bahasa Python yang biasanya selalu berada satu atau dua peringkat di bawah C# pada *survey-survey* sebelumnya, naik satu peringkat di atas C#.

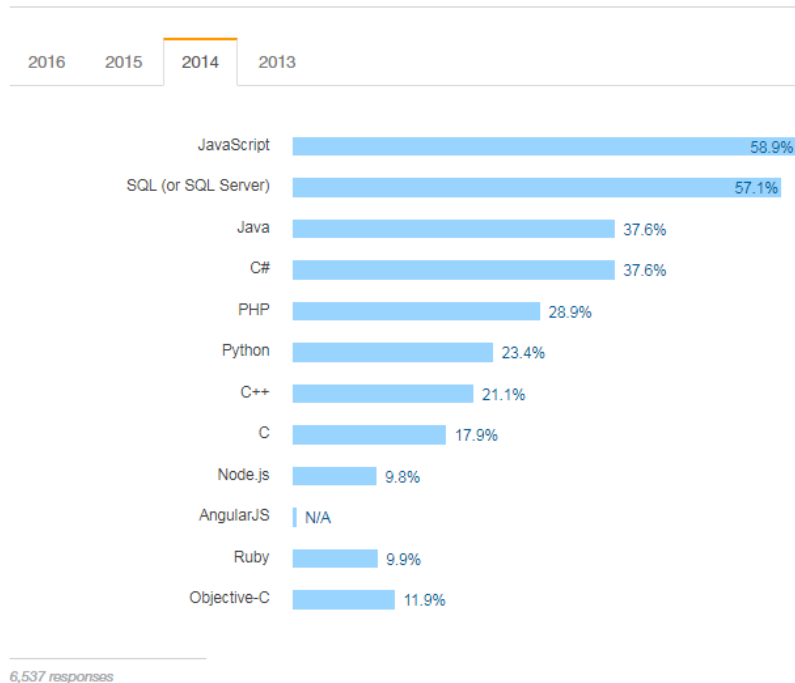
Namun demikian, 34.4% dari 78,334 responden masih menggunakan bahasa C# sebagai bahasa pemrograman mereka.

## I. Most Popular Technologies



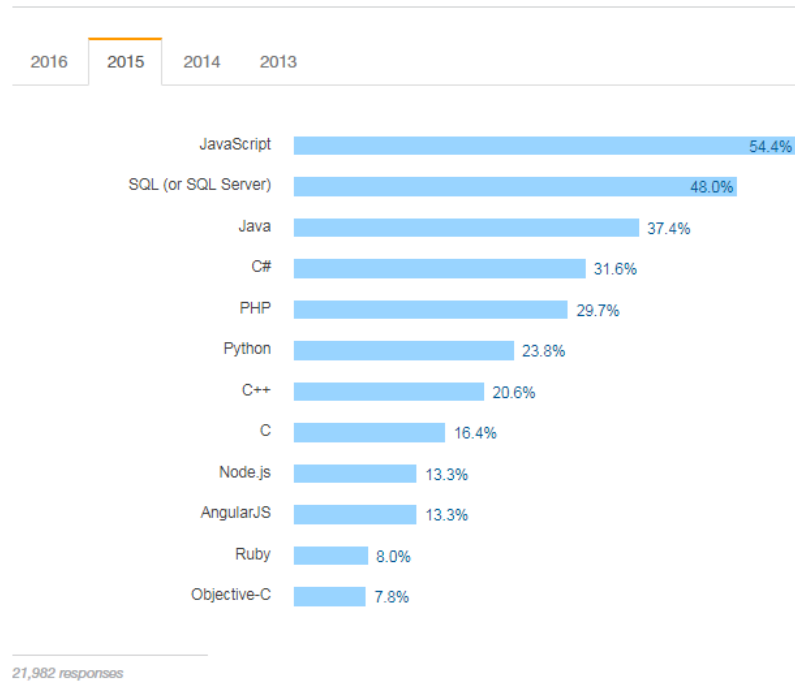
**Gambar 1.1**

## I. Most Popular Technologies



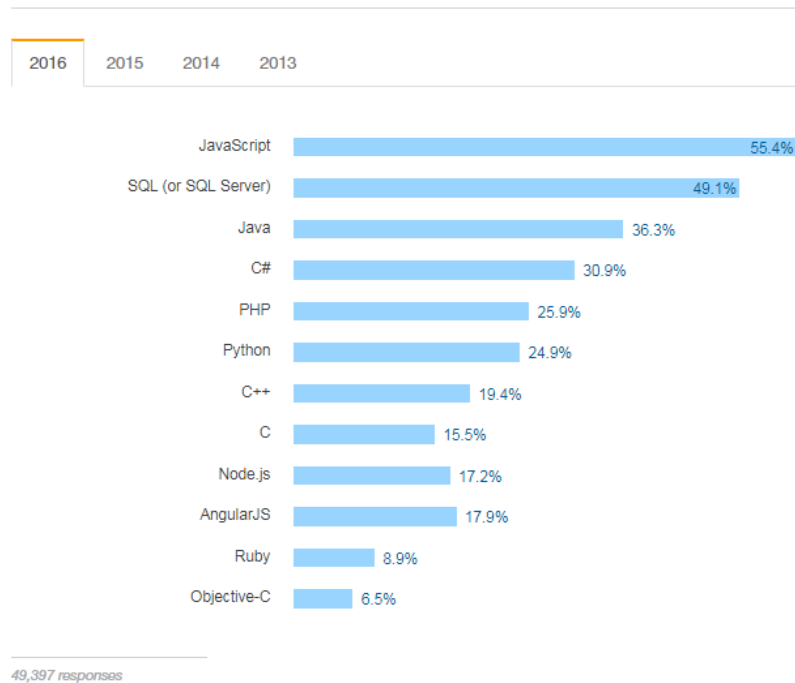
**Gambar 1.2**

## I. Most Popular Technologies



**Gambar 1.3**

## I. Most Popular Technologies

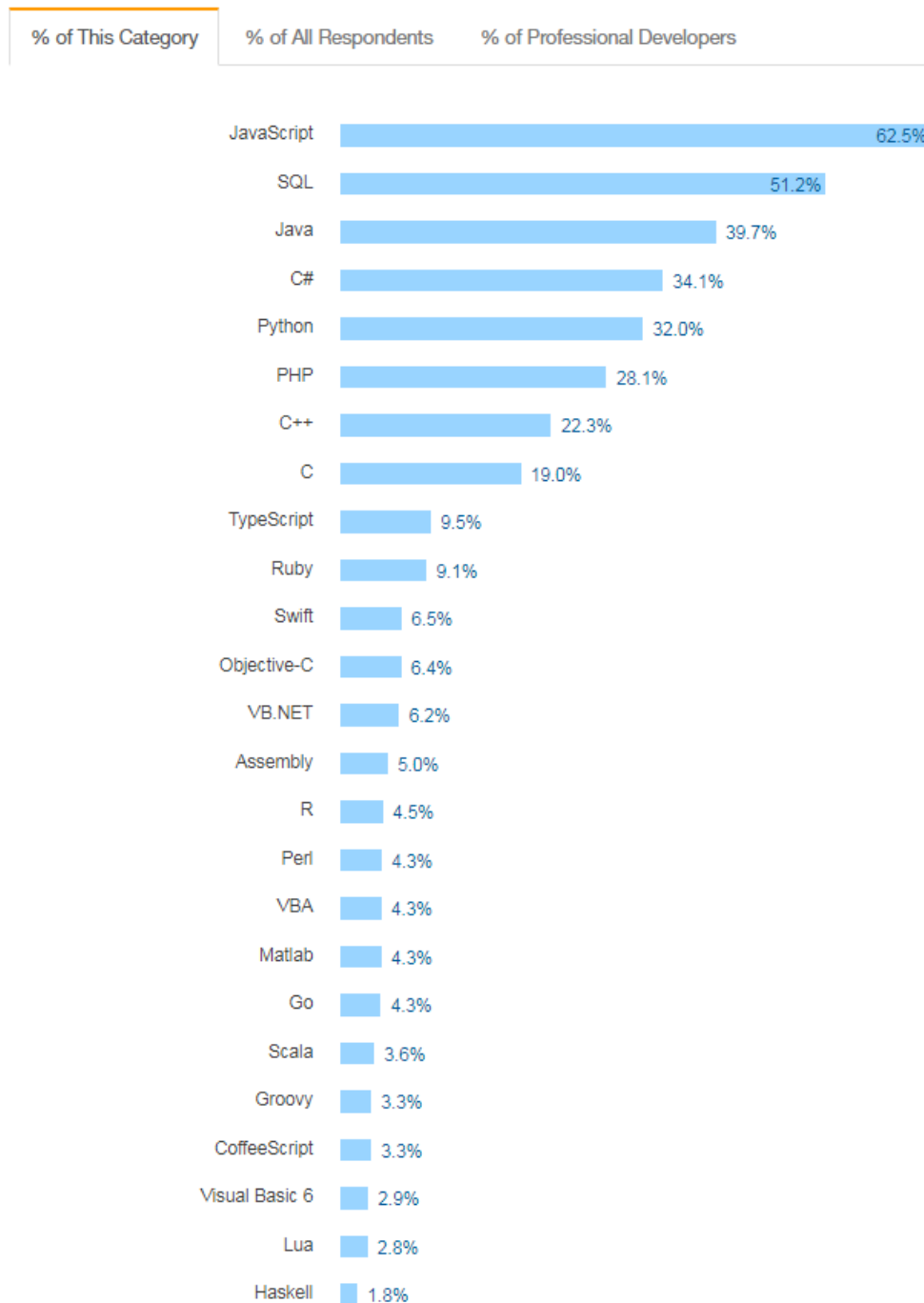


**Gambar 1.4**



## Most Popular Technologies

### Programming Languages



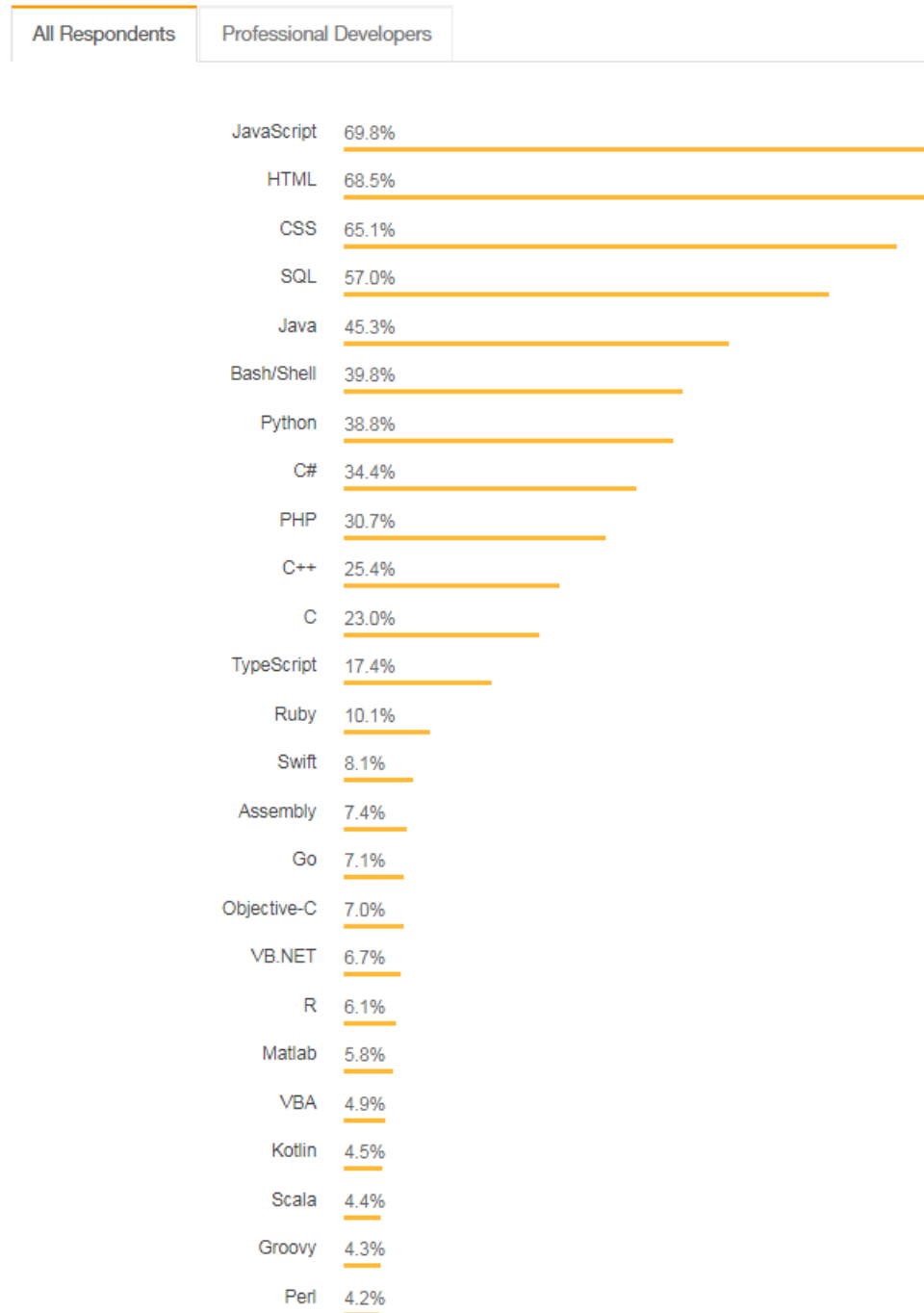
36,625 responses; select all that apply

**Gambar 1.5**



## Most Popular Technologies

### Programming, Scripting, and Markup Languages



78,334 responses; select all that apply

**Gambar 1.6**

## 2. Dapat digunakan untuk pengembangan aplikasi mobile

Memang pada awalnya, bahasa C# dikembangkan oleh *Microsoft* hanya untuk mengembangkan aplikasi di *platform* Windows saja. Namun, seiring berjalannya waktu, dengan menjadikan .NET menjadi *framework* yang *open source*, penggunaan bahasa C# menjadi lebih luas.

Sekarang, tidak sebatas hanya bisa digunakan untuk pengembangan aplikasi *desktop* maupun aplikasi layanan berbasis web saja, dengan menggunakan bahasa C#, Anda bahkan juga bisa mengembangkan aplikasi *mobile*. Ya, aplikasi di *smartphone* Anda!

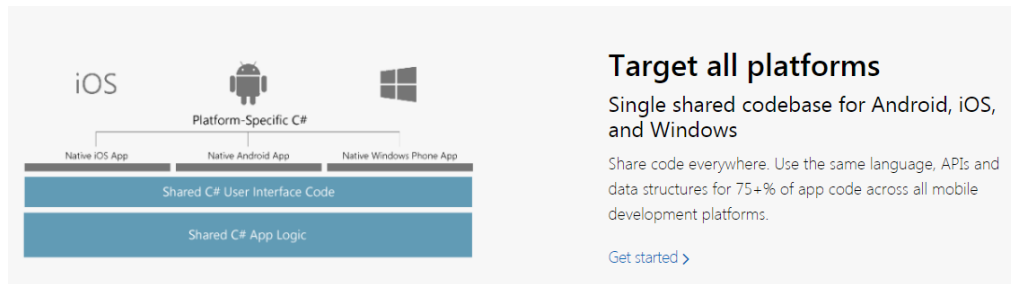
Ini berkat tools dari *Microsoft* yang bernama **Xamarin**. Xamarin sendiri diakuisisi oleh *Microsoft* pada tahun 2016 dan membuatnya menjadi *tool* yang *open source*.



**Gambar 1.7**

Meskipun Xamarin merupakan bagian dari *Microsoft*, Xamarin tidak hanya bisa digunakan untuk pengembangan aplikasi *mobile* di platform Windows Phone saja. Namun, bisa juga digunakan untuk mengembangkan aplikasi *mobile* baik untuk platform Android, maupun iOS.

Dengan begitu, Anda bisa menggunakan lebih dari 75% kode pemrograman dengan bahasa pemrograman, API, serta struktur data yang sama untuk pengembangan aplikasi *mobile* di semua *platform*.




**Gambar 1.8**

Selain itu, dengan menggunakan Xamarin, Anda bisa melakukan pengembangan aplikasi baik di *platform* Windows maupun Mac. Jadi, kalau Anda adalah pengguna Mac, ini bukan jadi halangan untuk bisa menggunakan Xamarin.

Dengan fleksibilitas seperti ini, banyak perusahaan yang memilih mengembangkan aplikasi *mobile* dengan Xamarin karena bisa mempercepat waktu pengembangan aplikasi dan menurunkan biaya *engineering*. Contohnya, seperti yang terlihat di halaman depan situs Xamarin, ada perusahaan besar seperti **Fox Sport** dan juga **UPS** yang mengembangkan aplikasi mobile mereka menggunakan Xamarin.

**What customers say**


**FOX Sports**



"With Visual Studio Tools for Xamarin, we deliver a native app experience, so it is easy and fun to use."

[Read their story >](#)

**UPS**



"With Visual Studio Tools for Xamarin, we can add a new feature across all devices in weeks or days. In the past, it would take us months to achieve feature parity."

[Read their story >](#)

**Gambar 1.9**

### 3. Dapat digunakan untuk pengembangan game, augmented reality, dan virtual reality

Alasan yang ketiga ini mungkin akan memotivasi Anda yang ingin mencoba terjun di dunia pengembangan game, *augmented reality*, dan juga *virtual reality*.

Selama ini, mungkin Anda membayangkan bahwa mengembangkan sebuah game adalah proses yang sangat rumit. Namun, dengan bantuan *game engine*, yang salah satunya bernama **Unity3d**, proses pengembangan sebuah game bisa disederhanakan.

#### Apa itu *game engine*?

*Game engine* merupakan sebuah sistem perangkat lunak atau *software* yang dirancang untuk pembuatan dan pengembangan suatu *video game*.

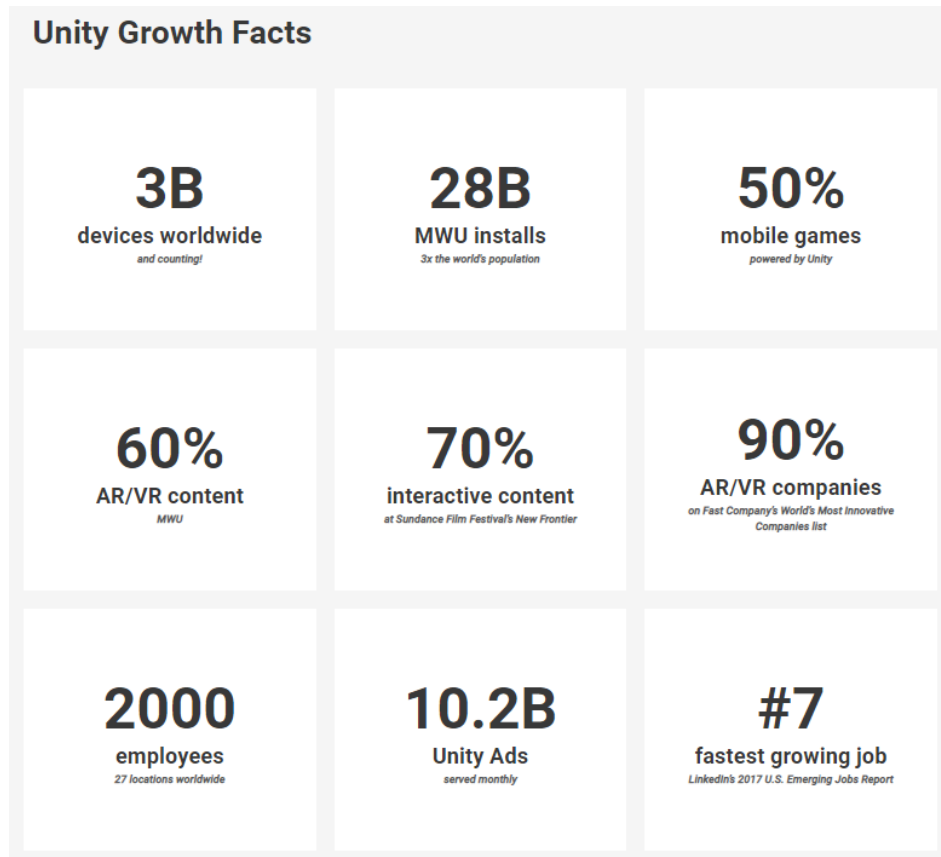
*Game engine* memberikan kemudahan dalam menciptakan konsep sebuah game yang akan dibuat. Mulai dari sistem *rendering* grafik, fisik, arsitektur, suara, *scripting*, kecerdasan buatan atau *A.I.*, dan bahkan sistem *networking*. *Game engine* dapat dikatakan sebagai jiwa dari seluruh aspek sebuah game.

Dari data angka yang dipublikasikan di situs web milik **Unity**, *game engine* ini sudah dipakai sekitar 50% aplikasi *mobile game* yang pernah dibuat.

Selain untuk pengembangan *mobile game*, 60% konten dari *augmented reality* dan *Virtual reality* juga dibuat dengan menggunakan *game engine*



unity, sedangkan 90% perusahaan AR dan VR menggunakan **Unity** sebagai *game engine*.



Gambar 1.10

### Apa itu *augmented reality* dan *virtual reality*?

*Augmented reality*, atau realitas bertambah dalam bahasa Indonesia, adalah teknologi yang menggabungkan benda maya dua dimensi dan ataupun tiga dimensi ke dalam sebuah lingkungan nyata tiga dimensi.

Salah satu contoh dari game *augmented reality* yang dikembangkan menggunakan bahasa C# adalah **Pokemon Go** yang sempat menjadi hit di tahun 2016 dan 2017.



**Gambar 1.11**

Sedangkan *virtual reality*, secara harfiah bisa diartikan sebagai hal-hal maya yang terasa nyata. Dan secara luas, *Virtual Reality* adalah proses penghapusan dunia nyata di sekeliling Anda untuk kemudian digiring ke dunia virtual yang baru.



**Gambar 1.12**

Sebagian besar konten *virtual reality* untuk **Samsung Gear VR** dikembangkan dengan menggunakan *game engine Unity*.

Itulah tadi 3 alasan, yang menurut saya, mengapa mempelajari bahasa C# akan menjadi modal yang besar buat Anda untuk bisa bersaing di industri pengembangan *software*. Tentunya dengan luasnya aplikasi yang bisa dikembangkan menggunakan bahasa C#, peluang kerjanya juga semakin luas.

Mudah Belajar Pemrograman C#

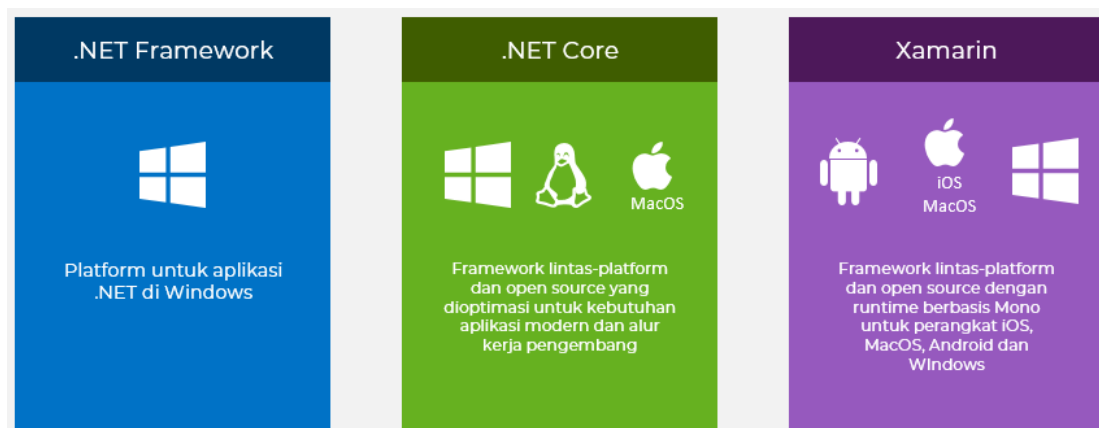
Bagaimana, sudah lebih bersemangat untuk memulai belajar pemrograman C#?

## 2 Sekilas Tentang .NET

## 2.1 Apa itu .NET?

.NET adalah sebuah *platform open source* untuk mengembangkan dan menjalankan berbagai jenis aplikasi lintas-platform. .NET sendiri tersedia dan dapat digunakan secara gratis.

Dengan .NET, Anda dapat menggunakan beberapa bahasa pemrograman, *editor*, dan *library* untuk membangun aplikasi web, mobile, desktop, game, dan bahkan Internet of Things atau IoT.



**Gambar 2.1**

Saat ini, ada tiga implementasi .NET untuk mendukung pemrograman lintas-platform yaitu,

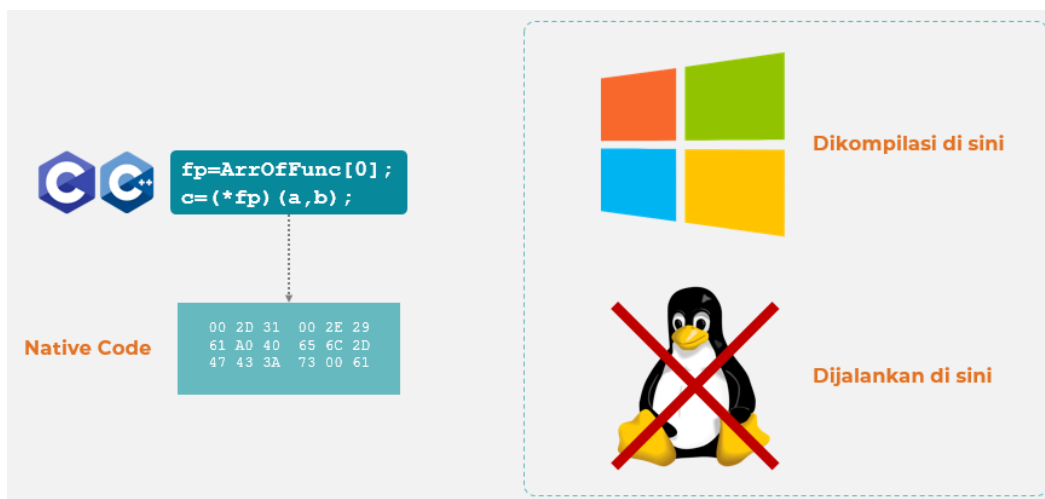
- **.NET Framework** yang mendukung website, service, aplikasi desktop, dan lainnya di platform Windows.
- **.NET Core** yang merupakan implementasi .NET lintas platform untuk aplikasi web, server, dan juga aplikasi konsol di macOS, Windows, dan Linux.

- **Xamarin / Mono** yang merupakan implementasi .NET untuk menjalankan aplikasi di semua sistem operasi mobile utama seperti iOS, Android, dan Windows phone.

## 2.2 Mengkompilasi program .NET

Sebelum Anda bisa menjalankan kode program yang Anda tulis, kode program tersebut perlu terlebih dulu diterjemahkan ke dalam kode yang bisa dimengerti oleh CPU. Proses ini biasa disebut dengan proses kompilasi.

Sebelum adanya C#, ada dua bahasa pemrograman dalam keluarga C, yaitu C dan C++.



**Gambar 2.2**

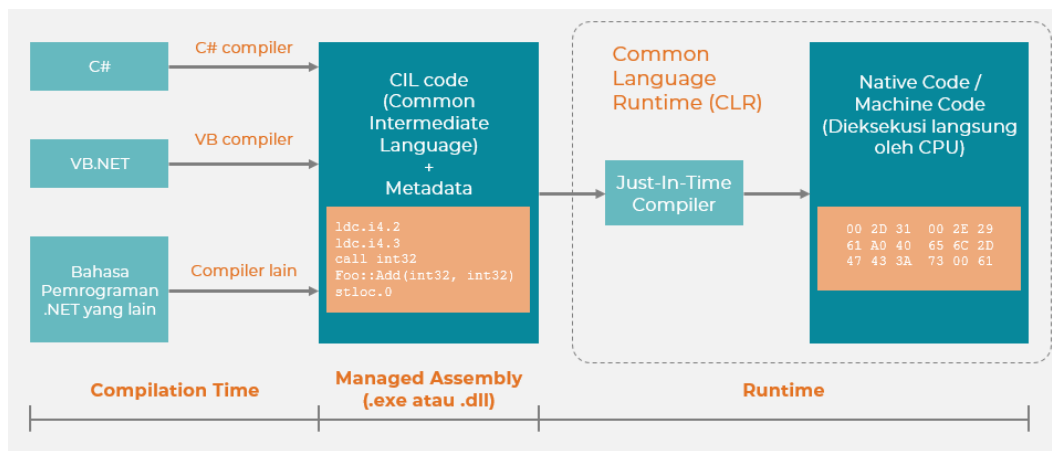
Pada proses kompilasi kedua bahasa pemrograman ini, *compiler* akan menerjemahkan kode program yang kita tulis menjadi *native code* atau bahasa khusus untuk mesin di mana proses kompilasi ini berlangsung.

Artinya, ketika Anda menulis program C++ di sistem operasi Windows dengan arsitektur prosesor x86 atau 32-bit, *compiler* akan menerjemahkan kode program tersebut hanya untuk mesin Windows dengan arsitektur prosesor x86.

Sehingga, ketika Anda mencoba menjalankan aplikasi tersebut di sistem operasi dan atau arsitektur prosesor yang berbeda, aplikasi tersebut tidak bisa dijalankan.

Berbeda dengan proses kompilasi pada umumnya, Proses kompilasi untuk program .NET tidak langsung menghasilkan native code.

Lalu, apa yang sebenarnya terjadi ketika kita mengkompilasi sebuah program .NET?



**Gambar 2.3**

- File .exe atau .dll (biasa disebut dengan *managed assembly*) yang dibuat tidak memuat kode yang dapat langsung dieksekusi oleh CPU (*native code*), melainkan memuat kode ***Common Intermediate Language*** atau biasa disingkat CIL dan *metadata*.

- Ketika kita menjalankan file .exe tersebut, sebuah lingkungan *runtime* khusus atau biasa disebut **Common Language Runtime (CLR)** akan diluncurkan. Instruksi-instruksi IL (*Intermediate language*) selanjutnya diterjemahkan oleh CLR menjadi Bahasa mesin (*native code*).
- CLR dilengkapi dengan **Just In Time (JIT) compiler** yang bertugas menerjemahkan IL ke *native code*.

Dengan arsitektur seperti ini, Anda tidak perlu lagi khawatir menulis program dengan bahasa C# di Windows dan mencoba menjalankannya di platform lain. Selama platform tersebut memiliki CLR, aplikasi tersebut akan tetap bisa dijalankan.



## **3 Mengetahui Elemen-Elemen Dasar**

### **Bahasa C#**

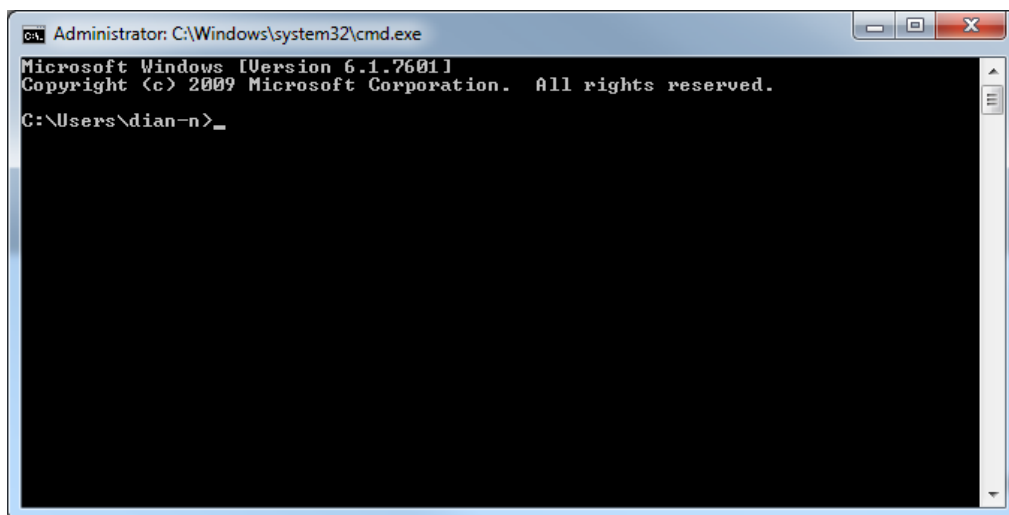
### 3.1 Membuat program Hello World

Sebelum melanjutkan materi kali ini, pastikan Anda sudah menginstall visual studio di komputer Anda.

Saya tidak akan memberikan cara mengunduh ataupun cara menginstall visual studio di dalam buku ini karena Anda bisa menemukan banyak sekali tutorial instalasi visual studio di Internet.

Apabila Visual Studio sudah terinstall di komputer Anda, mari kita mulai materi kali ini dengan membuat sebuah aplikasi *console* sederhana dengan Visual Studio.

Aplikasi *console* sendiri adalah aplikasi yang sama seperti ketika kita membuka aplikasi *command prompt* di sistem operasi windows.



**Gambar 2.4**

Ikuti langkah-langkah di bawah ini untuk membuat program C# pertama Anda.

1. Buka visual studio.

2. Setelah visual studio terbuka, Anda akan melihat halaman *start page* di layar monitor.
3. Untuk membuat sebuah aplikasi baru, klik *new project*, dengan demikian dialog *new project* akan tampil di layar monitor.
4. Pilih visual C# di jendela *explorer* pada dialog, lalu pilih *console application*.
5. Pada kolom nama, beri aplikasi Anda dengan nama BelajarCSharp, lalu tekan tombol OK.
6. Sekarang Anda bisa melihat halaman editor di layar monitor Anda dengan beberapa kode yang sudah tersedia secara default.

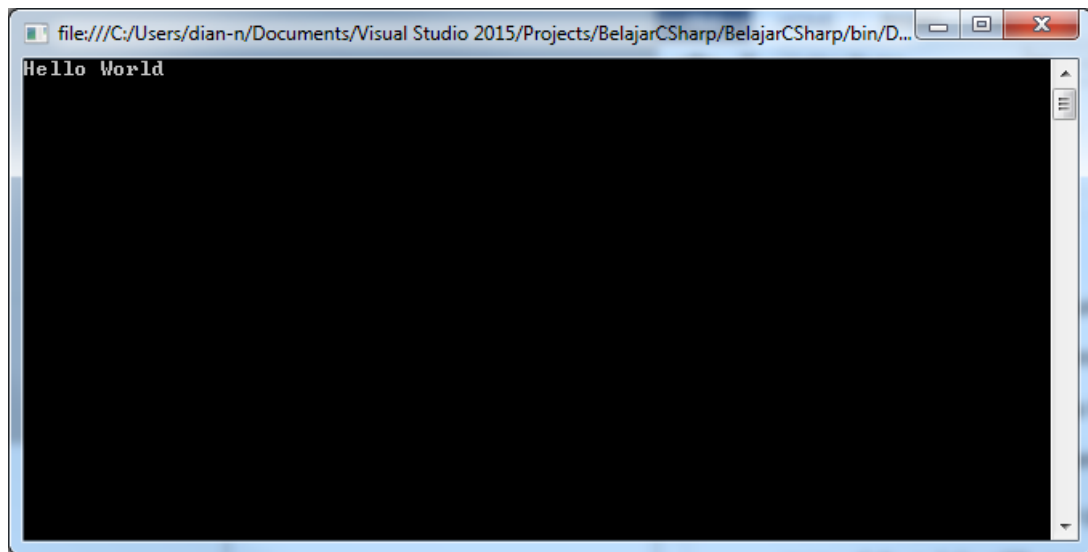
*Perhatikan bagian atas kode program yang tertulis di halaman editor, Anda bisa melihat beberapa kata kunci **using** digunakan di sini, bukan? Lalu apa maksud dari baris kode ini?*

*Setiap aplikasi .NET membutuhkan referensi ke beberapa namespace yang telah disediakan oleh .NET Framework. Misal, ketika Anda ingin menampilkan sebuah pesan di layar console, Anda memerlukan sebuah class bernama **Console** yang berada di dalam namespace **System** milik .NET Framework.*

7. Di dalam method Main, tulis kode berikut.

```
string pesan = "Hello World";  
Console.WriteLine(pesan);
```

8. Jalankan program yang baru saja Anda buat dengan cara memilih menu debug, lalu pilih *run without debugging*. Atau, Anda bisa langsung menekan tombol ctrl + F5.
9. Apabila tidak ada eror, tulisan “Hello World” akan tertampil di layar console seperti ini.



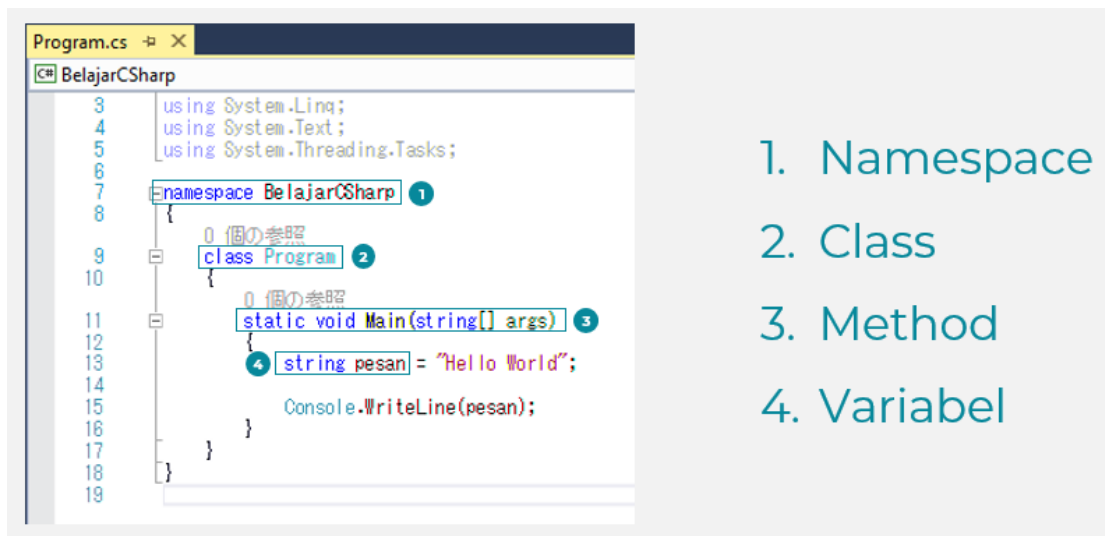
**Gambar 2.5**

Tentunya aplikasi seperti ini sangat tidak menarik, tapi ini adalah titik awal bagi Anda untuk membuat aplikasi yang lebih kompleks dan pastinya lebih keren.

## 3.2 Elemen-elemen pemrograman C#

Seperti yang sudah Anda lihat barusan pada saat membuat program Hello World, ketika Anda membuat aplikasi baru di Visual Studio, secara default, Visual Studio akan menyediakan beberapa elemen dasar untuk Anda.

Elemen-elemen tersebut adalah *namespace*, *class*, dan *method*.



**Gambar 2.6**

Selain tiga elemen di atas, dalam pemrograman C#, ada satu elemen yang meskipun tidak disediakan secara default oleh Visual Studio, namun pemakaiannya tidak bisa Anda hindari. Elemen itu bernama *variabel*.

### 3.2.1 Namespace

.NET framework menggunakan namespace sebagai cara untuk memisahkan file-file class kedalam kategori yang terkait.

Selain itu, penggunaan namespace juga bisa membantu menghindari bentroknya penamaan dalam aplikasi yang mungkin berisi class-class dengan nama yang sama.

Untuk lebih memahaminya, coba bayangkan Anda akan membuat sebuah simulator atau game bertema transportasi. Pasti, di dalam simulator tersebut, Anda membutuhkan moda transportasinya seperti mobil, pesawat, kereta, dan lain-lain. Selain itu, Anda juga mungkin membutuhkan bangunan-bangunan terminal seperti bandara, stasiun kereta, dan lain-lain.

Dengan konsep seperti itu, Anda kemudian memutuskan untuk membuat dua namespace di dalam aplikasi simulator tersebut, yaitu **ModaTransportasi** dan **BangunanTerminal**.



**Gambar 2.7**

Pada namespace **ModaTransportasi**, Anda kemudian mendeklarasikan beberapa class seperti, mobil, kapal, motor, kereta, sepeda, dan pesawat.

Sedangkan pada namespace **BangunanTerminal**, Anda mendeklarasikan class seperti pelabuhan, terminal bus, bandara, dan stasiun kereta.

Dengan memisah-misahkan class ke dalam kategori yang terkait seperti ini, kode program yang Anda buat nantinya akan lebih rapi dan terstruktur. Sehingga nantinya, Anda atau tim Anda akan lebih mudah melakukan *maintenance* atau perubahan bila diperlukan.

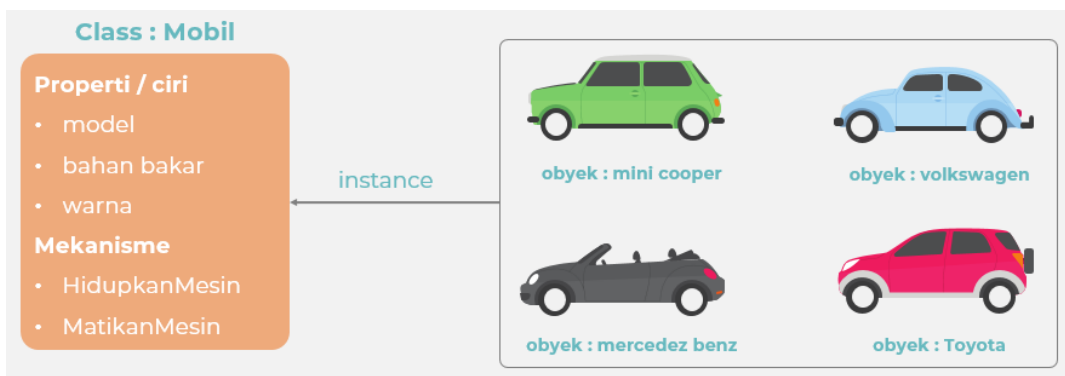
Pada contoh di atas, kebetulan tidak ada nama class yang sama untuk dua namespace ini, misal pun ada, karena masing-masing class berada di namespace yang berbeda, maka tidak akan menjadi masalah apabila kita terpaksa untuk memberi nama yang sama pada dua buah class.

### 3.2.2 Class

Ketika menulis sebuah program dengan bahasa C#, Anda akan sering sekali mendeklarasikan dan menggunakan class. Lalu apa itu class dan apa fungsinya?

Sebuah Class sejatinya adalah cetak biru (*blueprint*) dari sebuah obyek.

Maksudnya, sebuah Class nantinya akan menentukan struktur yang akan digunakan oleh sebuah obyek ketika kita membuat Instance dari Class tersebut.



Gambar 2.8

Untuk lebih memahaminya, anggap Anda memiliki sebuah class bernama mobil. Class ini memiliki properti seperti, apa modelnya, apa bahan bakarnya, dan apa warnanya. Lalu, class ini juga memiliki mekanisme seperti menghidupkan dan mematikan mesin.

Dengan membuat sebuah class, Anda seperti membuat sebuah *blueprint* tentang bagaimana sebuah mobil nantinya dibuat. Seperti apa properti atau cirinya dan apa mekanisme yang bisa dilakukan oleh sebuah mobil nantinya.

Sehingga, ketika Anda membuat beberapa obyek dari *blueprint* tersebut, obyek-obyek tersebut akan memiliki mekanisme yang sama dengan mekanisme yang disediakan oleh *blueprint* tersebut. Obyek-obyek tersebut juga bisa dibedakan ciri-cirinya berdasarkan properti yang disediakan oleh *blueprint* yang Anda buat sebelumnya.

Contohnya, ketika Anda membuat beberapa obyek dari class mobil seperti obyek *mini cooper*, *volkswagen*, *mercedez benz*, dan *toyota*, masing-masing obyek tersebut akan memiliki mekanisme yang sama, yaitu sama-sama bisa menghidupkan dan mematikan mesin.

Untuk membedakan satu obyek dengan obyek yang lain, Anda dapat memberikan nilai properti atau ciri-ciri yang berbeda. Misal, ketika Anda membuat obyek *mini cooper*, Anda menentukan properti modelnya adalah *hatchback* dan properti warnanya adalah hijau. Sementara, untuk obyek *toyota*, Anda menentukan properti modelnya adalah *SUV* dan properti warnanya adalah merah.



Materi tentang class ini akan kita pelajari lebih mendalam di buku

***Pemrograman Berorientasi Obyek***. Untuk sementara, paling tidak, Anda sudah mengetahui apa itu class.

### 3.2.3 Method

Pada saat membuat aplikasi *console* di visual studio tadi, Anda menuliskan kode program di dalam method **Main()**, bukan? Lalu apa itu method?

Sederhananya, sebuah method adalah blok kode program yang terdiri dari serangkaian instruksi.

Dengan memanggil sebuah method, instruksi-instruksi di dalamnya akan dieksekusi.

Dalam bahasa C#, method **Main()** tadi adalah titik masuk atau *entry point* dari aplikasi yang kita buat. Maksudnya, ketika kita menjalankan aplikasi tersebut, **Main()** adalah method yang pertama kali dipanggil.

Mudahnya, Anda dapat menganggap method sebagai sebuah mekanisme atau prosedur untuk melakukan sebuah operasi.

Contohnya, ketika Anda ingin membuat segelas jus buah, tentunya ada prosedur-prosedur yang harus Anda lakukan, dan tentunya Anda tidak bisa melakukannya dengan acak.. harus berurutan.

Misal Anda ingin membuat sebuah prosedur untuk membuat segelas jus buah. Katakanlah prosedur ini Anda beri nama **“membuat jus buah”**. Lalu, di dalam prosedur ini Anda tuliskan instruksi-instruksi yang harus dilakukan untuk membuat segelas jus buah secara berurutan, seperti berikut.

1. Pertama, cuci buah terlebih dulu.
2. Lalu, potong-potong dengan ukuran sedang.
3. Setelah itu, masukkan buah kedalam mixer.
4. Tambahkan gula.
5. Lalu tambahkan air.
6. Terakhir, baru jalankan mixer.

Kira-kira seperti itulah analogi sederhana sebuah method. Anda akan mempelajari method lebih detail lagi di bab selanjutnya di dalam buku ini. Sampai di sini, paling tidak Anda sudah tau apa itu method.

### **3.2.4 Variabel**

Elemen berikutnya adalah variabel, apa itu variabel?

Variabel adalah suatu tempat untuk menyimpan data berupa nilai atau referensi ke sebuah obyek.

Kita tidak akan membahas Variabel pada materi kali ini, karena kita akan mempelajari variabel lebih detail di bab selanjutnya.

## 4 Konsep-Konsep Dasar Bahasa C#

## 4.1 Tipe data & variabel

Kalau Anda diminta untuk membedakan sebuah entitas, bagaimana cara Anda membedakannya?

Tentu saja, mudah buat Anda untuk membedakan atau mengkategorikan mana entitas yang termasuk angka dan mana entitas yang termasuk karakter.

Seperti yang terlihat pada gambar di bawah ini, dengan mudahnya Anda bisa mengatakan bahwa angka 1256 adalah bilangan bulat, tulisan “pemrograman” adalah deret karakter, dan angka 126.98 adalah bilangan desimal.



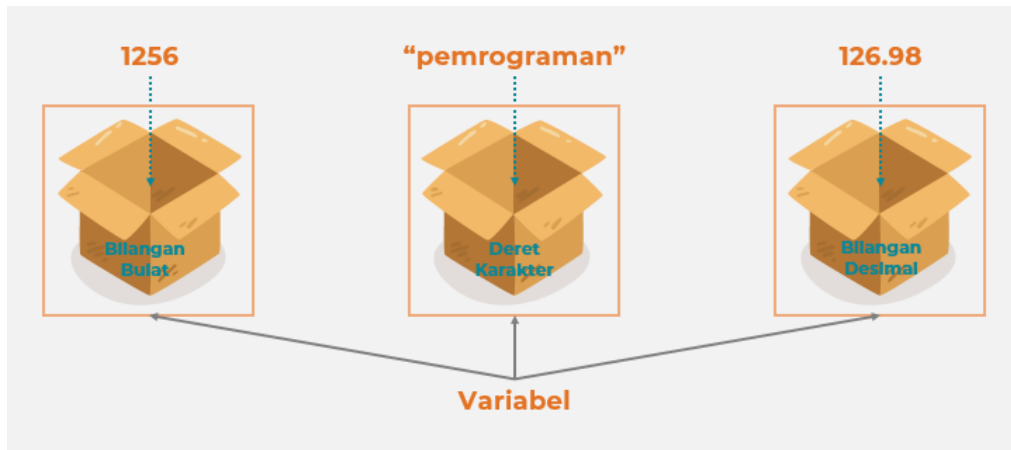
**Gambar 4.1**

Tidak seperti Anda, sebuah komputer tidak dapat membedakan mana entitas yang termasuk angka dan mana yang termasuk huruf.

Lalu pertanyaan yang muncul adalah bagaimana caranya agar komputer dapat membedakan sebuah nilai yang diberikan kepadanya, seperti angka **1256**, tulisan **“pemrograman”**, dan angka **126.98**?

Caranya adalah dengan memasukkan nilai-nilai tersebut ke dalam sebuah kontainer atau box yang terlebih dahulu kita labeli dengan jenis entitas yang kita inginkan untuk menyimpan nilai-nilai tersebut.

Contohnya, untuk menampung angka **1256**, kita memberi label bilangan bulat pada kontainer tersebut. Untuk menampung tulisan “**pemrograman**”, kita beri label deret karakter. Lalu, untuk menampung angka **126.98**, kontainer tersebut kita beri label bilangan desimal.



**Gambar 4.2**

Dalam dunia pemrograman, kontainer atau box ini biasa disebut dengan **variabel**. Sementara label yang kita berikan pada kontainer tersebut biasa disebut dengan **tipe data**. Dan proses pelabelan kontainer ini biasa disebut dengan **deklarasi variabel**.

Dalam bahasa C#, deklarasi variabel ditulis dengan cara seperti ini.

```
BilanganBulat box1 = 1256;
```

```
DeretKarakter box2 = "pemrograman";
```

```
BilanganDesimal box3 = 126.98;
```

**BilanganBulat**, **DeretKarakter**, dan **BilanganDesimal** di sini merupakan tipe data yang digunakan, sementara **box1**, **box2**, **box3** adalah

nama variabelnya, lalu **1256**, **“pemrograman”**, dan **126.98** adalah nilai yang disimpan ke dalam masing-masing variabel yang kita deklarasikan.

Jadi, ketika Anda membuat sebuah variabel di C#, Anda harus menentukan tipe data untuk variabel tersebut terlebih dulu. Tipe data berfungsi untuk memberitahu *compiler* dan *syntax checker*, jenis entitas seperti apa yang hendak Anda simpan ke dalam variabel tersebut.

Jika Anda mencoba menetapkan sebuah nilai yang bukan dari jenis tipe data tersebut, sebuah peringatan (*warning*) atau galat (*error*) akan memberi tahu Anda tentang kesalahan tersebut. Contohnya apabila Anda mencoba memasukkan angka **126.98** yang merupakan bilangan desimal ke dalam variabel **box1** yang bertipe data **BilanganBulat**, maka *compiler* akan memberi Anda sebuah peringatan atau eror.

Variabel ini nantinya akan disimpan di dalam memori untuk selanjutnya digunakan untuk melakukan operasi atau kalkulasi berdasarkan tipe data yang diberikan.

## 4.2 Jenis-jenis Tipe Data

Setelah Anda memahami apa itu tipe data dan apa itu variabel, selanjutnya Anda perlu mengetahui jenis-jenis tipe data dalam bahasa C#.

C# mendukung dua macam tipe data yang digunakan untuk merepresentasikan informasi di dunia nyata. Yaitu, **value types** atau tipe nilai dan **reference type** atau tipe referensi.

## 1. Value Types

Tipe ini memuat nilai sebenarnya dari data yang tersimpan. Maksudnya, jika Anda memiliki variabel bilangan bulat yang digunakan untuk menyimpan angka 3, angka 3 ini lah yang disimpan di dalam variabel yang Anda deklarasikan. Value type ini tidak dapat dibiarkan kosong atau bernilai null.

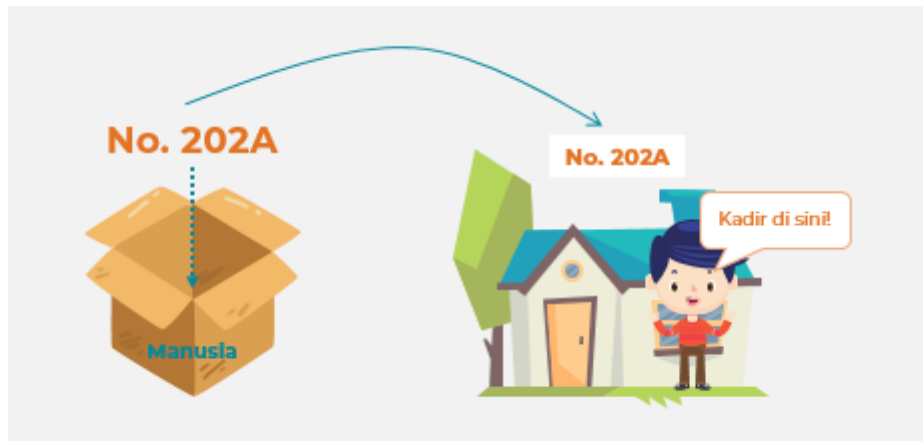


Gambar 4.3

## 2. Reference Types

Berbeda dengan *value type* yang menyimpan nilai sebenarnya di dalam variabel yang dideklarasikan, *reference type* hanya menyimpan alamat yang menunjuk sebuah lokasi di mana nilai atau obyek sebenarnya disimpan.

Contohnya ketika Anda membuat sebuah obyek dari class manusia bernama kadir. Variabel yang Anda deklarasikan tersebut hanya akan menyimpan alamat di mana obyek kadir ini sebenarnya berada. Sementara obyek kadir sendiri disimpan atau berada di lokasi yang berbeda.



**Gambar 4.4**

Lalu kenapa obyek kadir ini tidak langsung dimasukkan ke dalam kardus daripada harus repot-repot melakukan referensi? Kebayangkan ukuran obyek kadir ini? Karena ukuran kadir terlalu besar untuk dimasukkan ke kardus, alternatifnya adalah menyimpan obyek kadir ini di dalam ruangan yang lebih besar.

*Reference type* ini dimungkinkan untuk bernilai null. Maksudnya, tidak ada referensi alamat yang tersimpan di dalam variabel yang Anda deklarasikan.

Pada bab selanjutnya di dalam buku ini, saya akan menjelaskan bagaimana kedua tipe ini dialokasikan dan disimpan di dalam memori. Setelah mempelajari konsep pengalokasian memori, Anda mungkin akan lebih mudah memahami perbedaan antara *value type* dan *reference type*.

Tabel berikut mencantumkan tipe data yang tersedia (pre-defined types) di C# bersama dengan rentang nilai yang dimungkinkan untuk setiap tipe data:



Alias	.NET Type	Deskripsi	Ukuran (bytes)	Rentang Nilai
int	System.Int32	Bilangan bulat	4	-2,147,483,648 ~ 2,147,483,647
long	System.Int64	Bilangan bulat (rentang nilai lebih besar)	8	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
float	System.Single	Bilangan dengan nilai pecahan	4	$\pm 3.4 \times 10^{38}$
double	System.Double	Bilangan dengan nilai pecahan dengan presisi ganda (lebih akurat)	8	$\pm 1.7 \times 10^{308}$
decimal	System.Decimal	Nilai mata uang	16	28 angka penting
char	System.Char	Karakter tunggal	2	Tidak tersedia
bool	System.Boolean	Boolean	1	True atau false
DateTime	System.DateTime	Waktu pada tanggal	8	0:00:00 pada 01/01/0001 sampai 23:59:59 pada 12/31/9999
string	System.String	Rentetan karakter	2 per karakter	Tidak tersedia

Pada tabel tipe data di atas, kolom pertama adalah nama alias dari tipe data sedangkan kolom kedua adalah nama tipe data .NET yang sebenarnya. Sebagai contoh, *int* adalah nama alias dari sebuah struktur (*struct*) bernama Int32 yang

didefinisikan di dalam *namespace* bernama *System* milik .NET. Struct dapat Anda pelajari di buku ***Algoritma dan Struktur Data di Pemrograman C#***.

### 4.3 Statement / Pernyataan dalam bahasa C#

Dalam pemrograman C#, sebuah pernyataan (*statement*) bisa dianggap sebagai sebuah instruksi. *Statement* mengerjakan beberapa operasi di dalam kode program kita seperti memanggil sebuah method atau melakukan sebuah perhitungan. *Statement* juga digunakan untuk mendeklarasikan sebuah variabel dan menetapkan nilai ke dalamnya.

Sebuah *statement* biasanya tersusun dari beberapa token. Token-token tersebut bisa berupa *keywords* (kata kunci), *identifiers* (pengenal), *operators*, dan *statement terminator* yaitu titik koma atau *semicolon* (;). Semua *statement* di dalam pemrograman C# harus diakhiri dengan titik koma.

Contoh:

```
int thnPembuatan = 2018;
```

Pada contoh pernyataan di atas, token-tokennya adalah :

- **int**
- **thnPembuatan**
- **=**
- **2018**
- **;**

**Int** adalah tipe data yang digunakan untuk variabel **thnPembuatan**.

Simbol '=' adalah operator penugasan (*assignment operator*) dan digunakan untuk menetapkan nilai variabel **thnPembuatan** menjadi bernilai **2018**.

Angka **2018** dikenal sebagai nilai literal. Literal maksudnya, harfiah atau apa adanya. Angka 2018 tidak dapat berupa apa pun kecuali angka 2018. Kita tidak bisa menetapkan sebuah nilai ke angka 2018. Namun, kita bisa menetapkan angka 2018 ke sebuah variabel, dan itulah yang dilakukan oleh pernyataan C# di atas.

Terakhir, pernyataan berakhir dengan **titik koma (;)**.

#### 4.4 Identifier / Pengenal dalam bahasa C#

Seperti yang telah Anda pelajari sebelumnya, bahasa C# memiliki beberapa elemen seperti namespace, class, method, dan variabel.

Agar dapat menggunakan elemen-elemen tersebut dalam kode program yang Anda buat, Anda harus memberi pengenal atau sederhananya memberikan nama untuk setiap elemen yang hendak kita gunakan dalam kode program tersebut.

```
Namespace Transportasi{}
```

```
Class Mobil{}
```

```
Method MatikanMesin{}
```

```
int thnPembuatan = 2018;
```

Seperti contoh penamaan elemen-elemen di atas, Transportasi adalah nama yang diberikan untuk elemen Namespace, Mobil untuk elemen Class, MatikanMesin untuk elemen Method, dan thnPembuatan untuk elemen Variabel bertipe data int.

Ada beberapa aturan yang diberikan oleh C# dalam memberi nama sebuah identifier. Jadi, meskipun Anda bebas menentukan apapun nama untuk sebuah elemen dalam pemrograman C#, Anda tetap harus mengikuti batasan-batasan yang diberikan oleh C# agar kode program yang Anda buat bisa dijalankan.

Berikut ini adalah batasan-batasan apa saja yang perlu Anda perhatikan untuk menamai sebuah identifier.

- *Identifiers* bersifat *case-sensitive*, yaitu huruf kapital dan huruf kecil akan dikenali sebagai huruf yang berbeda, hal ini dikarenakan C# sendiri adalah bahasa pemrograman yang *case-sensitive*. Konsekuensinya, *identifier* seperti **myVar**, **\_myVar**, dan **myvar**, akan dianggap sebagai *identifier* yang berbeda.
- *Identifiers* hanya bisa terdiri dari huruf, angka, dan karakter garis bawah atau *underscore* (`_`). Penamaan identifier hanya bisa dimulai dengan huruf atau karakter *underscore*. Anda tidak bisa memberi nama identifier dengan dimulai oleh angka. **myVar** dan **\_myVar** adalah penamaan yang valid sedangkan **2Vars** bukanlah penamaan yang valid.
- C# memiliki sebuah kumpulan kata kunci cadangan (*reserved keywords*) yang sudah dipakai dalam bahasa pemrograman itu sendiri. Kita tidak

diperkenankan untuk menggunakan kata-kata kunci tersebut sebagai nama *identifier* dalam kode program kita.

Tabel berikut berisi kata kunci cadangan dari bahasa C#.

abstract	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	in (generic modifier)	int	interface internal
is	lock	long namespace	new
null	object operator	out	out (generic modifier)
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

## 4.5 Operator

Ketika menulis kode program C#, Anda akan sering menggunakan operator.

Operator adalah token yang diperuntukkan untuk operasi pada satu atau lebih *operand* dalam sebuah ekspresi. Ekspresi sendiri bisa menjadi bagian atau keseluruhan dari sebuah pernyataan.

### Penggunaan operator pada dua operand

**3 + 4** – sebuah ekspresi yang akan menghasilkan nilai literal 4 yang ditambahkan ke nilai literal 3. Angka 3 dan 4 adalah *operand* sementara tanda plus (karakter +) adalah operator. 3 + 4 adalah ekspresi yang menjadi bagian dari sebuah pernyataan, misalnya [x = 3 + 4;].

### Penggunaan operator pada satu operand

**counter++** – sebuah ekspresi yang akan menghasilkan nilai variabel ‘counter’ menjadi bertambah satu (*incremented by one*). Variabel ‘counter’ adalah *operand* sementara tanda plus plus merupakan operator kenaikan (*increment*

*operator*). Ekspresi `counter++` sendiri merupakan keseluruhan dari sebuah pernyataan di C#.

Tidak semua operator sesuai untuk semua tipe data di C#. Sebagai contoh, dalam contoh di atas operator (+) digunakan untuk menjumlahkan dua angka. Anda bisa menggunakan operator yang sama untuk menggabungkan dua *string* menjadi satu (*string concatenation*), seperti:

“Pemrograman” + “C#” akan menghasilkan sebuah nilai *string* baru, “PemrogramanC#”.

Namun, Anda tidak bisa menggunakan *increment operator* (++) pada *string*. Dengan kata lain, contoh berikut akan menghasilkan error di C#.

“Pemrograman”++

Tabel berikut mencantumkan operator C# menurut tipe.

Tipe	Operator
Arithmetic	+, -, *, /, %
Increment, decrement	++, --
Comparison	==, !=, <, >, <=, >=, is
String concatenation	+
Logical/bitwise operations	&,  , ^, !, ~, &&,

Indexing (counting starts from element 0)	[ ]
Casting	( ), as
Assignment	=, +=, -=, *=, /=, %=, &=,  =, ^=, <<=, >>=, ??
Bit shift	<<, >>
Type information	sizeof, typeof
Delegate concatenation and removal	+, -
Overflow exception control	checked, unchecked
Indirection and Address (unsafe code only)	*, ->, [ ], &
Conditional (ternary operator)	?:

## 4.6 Konversi Tipe Data

C# mendukung dua jenis konversi tipe data atau kasting (*casting*), yaitu konversi secara implisit dan eksplisit. C# akan menggunakan konversi implisit selama melakukan konversi tersebut dimungkinkan. Konversi secara implisit ini kebanyakan terjadi pada saat sebuah konversi tidak akan menghasilkan data



yang hilang (*lost of data*) atau bisa juga terjadi ketika melakukan konversi dari tipe data yang sepadan (*compatible*).

Berikut ini adalah contoh dari sebuah konversi data secara implisit. Yaitu, mengkonversi dari tipe integral yang lebih kecil ke tipe integral yang lebih besar.

```
int myInt = 2147483647;
long myLong = myInt;
```

Pada contoh di atas, tipe data **long** memiliki ukuran memori 64-bit sedangkan tipe data **int** menggunakan 32-bit. Oleh karena itu, variabel bertipe-data **long** bisa dengan mudah mengakomodasi berapapun nilai yang disimpan di variabel bertipe-data **int**. Namun, sebaliknya ada kemungkinan terjadinya data hilang (*data loss*) apabila konversi data dilakukan dari tipe **long** ke tipe **int**.

Anda sebaiknya menggunakan kasting eksplisit (*explicit casting*) jika kira-kira Anda tahu bahwa ada kemungkinan terjadinya data yang hilang dan yakin bahwa dengan menggunakan konversi jenis ini kode program Anda tidak akan terdampak.

Kasting eksplisit dilakukan dengan salah satu dari dua cara seperti didemonstrasikan oleh contoh kode program di bawah ini.

```
double myDouble = 1234.6;
int myInt;
// Mengkasting double ke int dengan cara menempatkan pengubah tipe (type modifier) tepat
// di depan tipe yang akan dikonversi, di dalam kurung.
myInt = (int)myDouble;
```

Opsi kedua adalah menggunakan *method* yang disediakan di dalam .NET framework.

```
double myDouble = 1234.6;
int myInt;
// Mengkasting double ke int dengan menggunakan class Convert dan method.ToInt32.
// Ini mengkonversi nilai double ke 32-bit signed integer.
myInt = Convert.ToInt32(myDouble);
```

Nilai dari variabel **myInt** untuk kedua metode kasting eksplisit di atas berturut-turut adalah 1234 dan 1235. Artinya data yang dihasilkan setelah proses kasting menjadi tidak utuh atau bisa dikatakan ada bagian data yang hilang karena terjadi proses pembulatan pada saat konversi.

C# juga menyediakan mekanisme lain untuk mengkasting tipe data.

Penggunaan method **TryParse()** dan **Parse()** juga bisa membantu kita melakukan kasting. Mendemonstrasikannya dengan sebuah contoh akan membantu Anda untuk lebih memahaminya.

```
// Contoh penggunaan method TryParse()

bool result = Int32.TryParse(value, out number);

// Contoh penggunaan method Parse()

int number = Int32.Parse(value);
```

Pada contoh penggunaan **TryParse()**, method tersebut akan menghasilkan (*return*) sebuah nilai Boolean (*True* atau *False*) yang mengindikasikan apakah

konversinya berhasil atau tidak. Pada contoh penggunaan **Parse()**, jika konversinya tidak berjalan sukses, sebuah eksepsi akan terjadi.

## **5 Konsep memori dalam pemrograman**

## 5.1 Alokasi memori

Semua aplikasi menyimpan dan memanipulasi data di dalam memori komputer. Ketika kita mendeklarasikan sebuah variabel, sebuah potongan memori akan dialokasikan untuk menyimpan dan memanipulasi variabel tersebut. Potongan memori ini menyimpan tiga buah informasi, yaitu nama variabel, tipe data dan nilai dari variabel tersebut.



**Gambar 5.1**

Ada dua jenis pengalokasian memori berdasarkan dari tipe data yang disimpan, yaitu memori **stack** di mana data akan disimpan dalam alamat memori secara berurutan, serta memori **heap** di mana data akan disimpan tanpa mempertimbangkan urutan alamat memorinya.

Untuk mempermudah pemahaman Anda mengenai dua jenis pengalokasian memori tersebut, coba Anda bayangkan memori *stack* seperti sebuah tumpukan kardus dan memori *heap* seperti sebuah loker.

Ketika menumpuk sebuah kardus, Anda biasanya memulainya dengan menumpuk dari bawah, kemudian menumpuk kardus yang lain di atasnya.

Ketika ingin mengambilnya, Anda harus memulainya dari tumpukan paling atas dan seterusnya. Konsep ini dikenal dengan LIFO atau *last in first out*.



**Gambar 5.2**

Berbeda ketika Anda ingin menyimpan barang ke dalam sebuah loker, Anda bebas memilih pintu loker yang mana saja selama loker tersebut kosong. Ketika ingin mengeluarkan barang, Anda juga bisa langsung menuju pintu loker tersebut tanpa harus mengeluarkan barang yang disimpan di dalam pintu loker yang lain.



**Gambar 5.3**

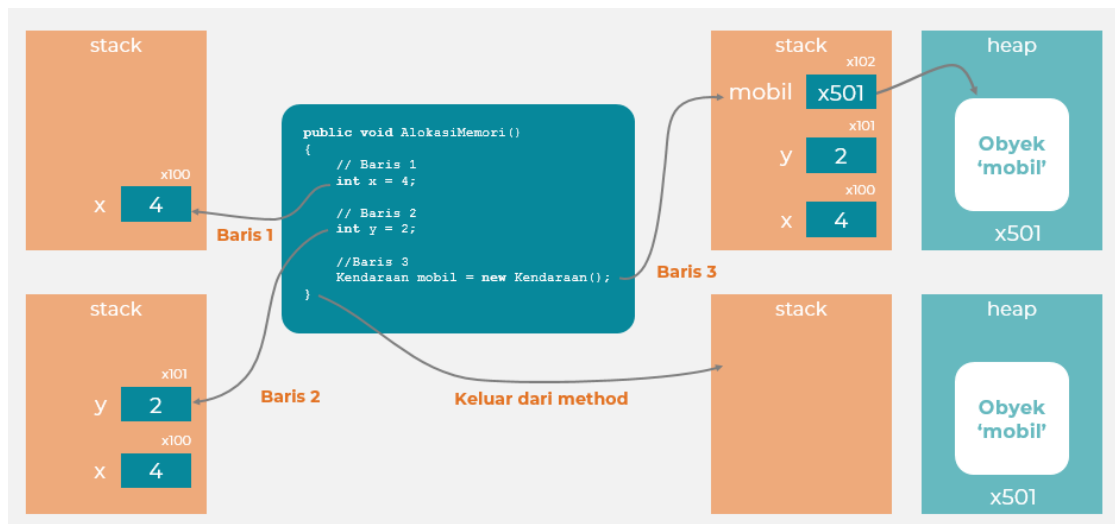
Untuk memahami konsep memori stack dan heap mari kita telusuri apa yang terjadi ketika potongan kode program berikut ini dieksekusi.

```
public void AlokasiMemori()
{
    // Baris 1
    int x = 4;
    // Baris 2
    int y = 2;
    //Baris 3
    Kendaraan mobil = new Kendaraan();
}
```

Mari kita pahami bagaimana kode program di atas dieksekusi baris demi baris.

- **Baris 1** : Ketika baris kode program ini dieksekusi, kompilator akan mengalokasikan 4 byte atau 32 bit memori di stack untuk variabel 'x'. (Lihat kembali tabel tipe data, variabel int memiliki ukuran 4 byte atau 32 bit)
- **Baris 2** : Baris kedua selanjutnya dieksekusi. Kompilator akan mengalokasikan 4 byte untuk variabel 'y' dan diletakkan di "atas" alokasi memori untuk variabel 'x'. Itulah mengapa memori ini bernama stack atau tumpukan.
- **Baris 3** : Pada eksekusi baris ketiga, kita membuat sebuah *instance* bernama 'mobil' dari sebuah *class* bernama 'Kendaraan'. Ketika baris ini dieksekusi, sebuah pointer (referensi) akan dibuat di memori stack dan obyek yang sebenarnya akan disimpan di tipe memori yang berbeda bernama "Heap". Pointer yang disimpan di memori stack, berisi alamat yang menunjukkan lokasi object di memori heap.

- **Keluar dari method 'AlokasiMemori'** : Ketika kontrol eksekusi keluar dari method di atas, semua variabel bertipe int yang disimpan di memori stack, termasuk referensi ke lokasi obyek 'mobil' akan dihapus satu per satu (didealokasi) dengan konsep LIFO (Last In First Out). Namun demikian, memori heap yang menyimpan obyek 'mobil' yang sebenarnya tidak akan didealokasi. Nantinya, 'mobil' akan didealokasi oleh *Garbage Collector* yang akan Anda bahas dalam kursus **Pemrograman Berorientasi Obyek Dengan Bahasa C#** tentang manajemen memori.



Gambar 5.4

### Kapan sebaiknya menggunakan memori heap?

Seperti yang telah dijelaskan di Bab 4 mengenai tipe referensi dimana kita memerlukan kapasitas memori yang dapat menyimpan data yang sangat besar (seperti obyek) atau menginginkan data kita tersimpan dalam waktu yang lama (variabel global), maka kita perlu menyimpan data di memori heap.



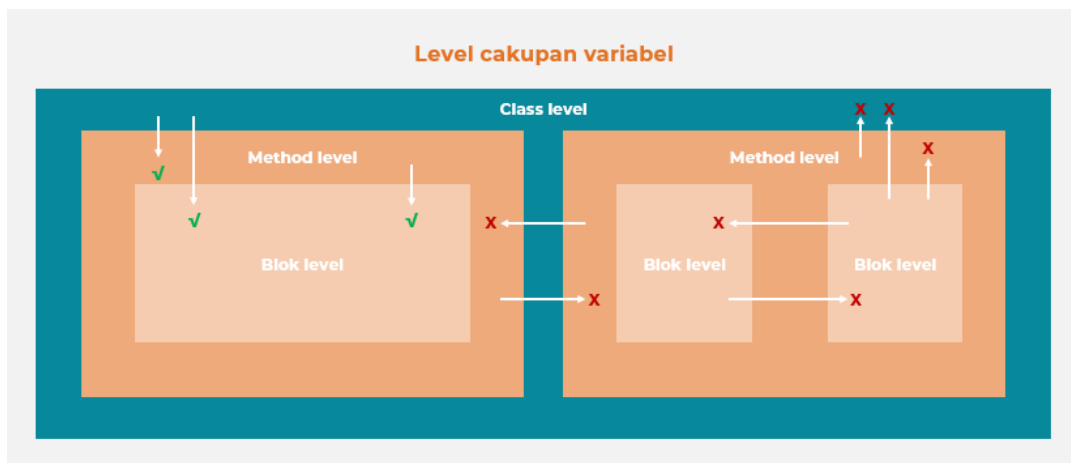
Ukuran dari memori stack sangat terbatas (1MB ~ 4MB) tergantung dari *Operating System* yang digunakan. Sehingga kita akan menghabiskan banyak memori ketika data Anda yang besar dialokasikan ke memori stack.

## 6 Cakupan Variabel

Pada saat mendeklarasikan sebuah variabel, Anda harus benar-benar memperhatikan pada level apa saja variabel yang kita deklarasikan tersebut bisa diakses dan digunakan oleh sebuah blok kode program yang Anda tulis nantinya.

Dalam pemrograman C#, paling tidak ada tiga level cakupan variabel. Sesuai dengan urutan dari level tertinggi sampai level terendahnya, yaitu :

1. Level class,
2. Level method,
3. dan Level blok.



**Gambar 6.1**

Ketika Anda mendeklarasikan sebuah variabel pada level yang lebih tinggi, variabel tersebut akan dapat diakses dan digunakan oleh blok kode program pada level yang lebih rendah.

Contohnya, apabila Anda mendeklarasikan sebuah variabel pada level class, variabel tersebut akan dapat diakses dan digunakan baik oleh kode program pada level method maupun pada level blok.

Begitu pula ketika Anda mendeklarasikan sebuah variabel pada level method, maka variabel tersebut dapat dilihat dan diakses oleh kode program pada level blok.

Namun sebaliknya, apabila Anda mendeklarasikan sebuah variabel pada level blok, variabel tersebut tidak akan bisa diakses oleh kode program yang ditulis pada level method apalagi pada level class. Variabel yang dideklarasikan pada sebuah blok tidak akan bisa diakses oleh blok yang lain, meskipun kedua blok tersebut berada di dalam method yang sama.

Begitu pula apabila Anda mendeklarasikan sebuah variabel pada level method, variabel tersebut tidak akan bisa diakses oleh kode program yang ditulis pada level class. Konsekuensinya, variabel yang kita deklarasikan pada sebuah method, tidak dapat diakses oleh method yang lain, meskipun method-method tersebut berada di dalam class yang sama.

Berikut ini adalah contoh kode program untuk membantu Anda agar dapat lebih memahami cakupan variabel,

```
static void Main(string[] args)
{
    int nilai; // variabel nilai ini dideklarasikan di blok method Main
    nilai = 50; // digunakan di blok yang sama
    if (nilai >= 50)
    {
        // digunakan di blok baru yang terletak di dalam blok method Main
    }
}
```

```
        Console.WriteLine("Nilai yang bagus : {0}", nilai);  
    }  
    else  
    {  
        // digunakan di blok baru yang terletak di dalam blok method Main  
        Console.WriteLine("Nilai yang kurang bagus : {0}", nilai);  
    }  
}
```

Contoh kode program di atas terdiri dari empat blok. Yaitu,

- Blok dalam cakupan method Main.
- Blok dalam cakupan pernyataan if-else.
- Blok dalam cakupan pernyataan if.
- Blok dalam cakupan klausa else.

Seperti yang dapat Anda lihat, variabel “nilai” dideklarasikan di dalam blok method Main. Kemudian, variabel “nilai” ini digunakan di dalam blok yang sama pada saat menetapkan angka 50 ke dalam variabel tersebut.

Di baris berikutnya, variabel “nilai” ini kembali digunakan oleh pernyataan if-else sebagai ekspresi pengkondisian. Karena blok pernyataan if-else tersebut dibuat di dalam blok method Main, maka blok pernyataan if-else ini masih berada dalam cakupan blok method Main. Sehingga, blok pernyataan if tersebut masih bisa mengakses variabel “nilai”.

Perhatikan contoh kode program berikut ini,

```
static void Main(string[] args)
```

```
{  
    int nilai; // variabel nilai ini dideklarasikan di blok pada level method Main  
    nilai = 50; // digunakan di blok yang sama  
    if (nilai >= 50)  
    {  
        // mendeklarasikan variabel di blok pernyataan if  
        string pesan = "Nilai yang bagus";  
    }  
    else  
    {  
        // mendeklarasikan variabel di blok klausa else  
        string pesan = "Nilai yang kurang bagus";  
    }  
    Console.WriteLine(pesan); // akan menghasilkan error  
}
```

Kali ini, kita mendeklarasikan variabel “pesan” di dalam **blok pernyataan if** dan di dalam **blok klausa else**. Apabila kita mencoba menggunakan variabel tersebut di luar dua blok tersebut, program tersebut akan menghasilkan error apabila dijalankan.

Hal ini dikarenakan blok method Main berada di luar cakupan baik dari blok pernyataan if maupun blok klausa else. Sehingga, blok method Main tidak dapat melihat atau mengakses variabel yang dideklarasikan di dalam kedua blok tersebut. Cakupan variabel “pesan” ini hanya berada di dalam **blok pernyataan if** dan di dalam **blok klausa else** di mana variabel tersebut dideklarasikan.

## 7 Memberi Komentar Di C#

## 7.1 Komentar

Di materi-materi sebelumnya, Anda mungkin sudah memperhatikan adanya beberapa baris pada contoh kode program yang dimulai dengan dua buah garis miring. Dalam pemrograman, ini disebut dengan komentar.

Komentar digunakan di dalam sebuah program untuk membantu Anda memahami sebuah potongan kode program. Komentar adalah kata-kata yang dapat dipahami oleh manusia, yang dimaksudkan untuk memberi penjelasan mengenai operasi seperti apa yang dilakukan sebuah pernyataan di dalam program.

Dalam bahasa C#, ada tiga tipe komentar:

1. Komentar baris-tunggal / single line (//)
2. Komentar multi-baris / multiline (/\* \*/)
3. Komentar XML(///)

*Catatan : komentar tidak akan dikompilasi oleh kompilator.*

## 7.2 Menggunakan komentar

Komentar bisa digunakan untuk banyak alasan. Oleh karena itu, Ada kemungkinan besar kita menggunakan komentar secara berlebihan, sehingga kita perlu memastikan bahwa komentar yang kita tulis benar-benar berguna.

Berikut ini adalah beberapa hal yang paling sering disertai dengan komentar,

- **Method.** Anda bisa menulis komentar pada method/function atau sub-rutin di dalam program. Hal ini biasanya dilakukan untuk menjelaskan



operasi apa yang dilakukan dan informasi apa yang dikembalikan oleh method tersebut.

- **Operasi yang rumit.** Pada saat Anda menerapkan sebuah algoritma yang rumit dan susah untuk bisa segera dimengerti, sebaiknya Anda menulis komentar untuk menjelaskan operasi tersebut.
- **Perubahan kode program.** Apabila Anda bekerja dalam sebuah tim, menulis komentar pada saat merubah beberapa baris kode, akan sangat membantu orang lain di tim Anda. Anda bisa mencantumkan nama Anda sendiri atau siapapun yang melakukan perubahan pada baris kode tersebut, tanggal di mana Anda melakukan perubahan, dan nomer versi dari software tersebut pada saat menulis komentar untuk menginformasikan perubahan pada sebuah kode program.

### 7.3 Komentar baris-tunggal

Cara penulisan komentar baris-tunggal adalah dengan menggunakan dua garis miring (*double slash*) “//”. *Compiler* mengabaikan apapun setelah “//” sampai akhir dari baris tersebut. Perhatikan contoh berikut ini,

```
int x = 4 + 8; // Menambahkan angka 4 dan 8
```

Pada contoh di atas, “*Menambahkan angka 4 dan 8*” adalah sebuah komentar.

Komentar baris-tunggal dapat ditulis dalam baris terpisah atau pada baris yang sama dengan kode program yang Anda tulis. Contoh di atas menunjukkan bagaimana menulis sebuah komentar pada baris yang sama dengan kode program yang diberi penjelasan.

Namun demikian, menulis sebuah komentar di baris yang berbeda biasanya lebih disarankan. Seperti pada contoh berikut ini,

```
// Menambahkan angka 4 dan 8  
int x = 4 + 8;
```

## 7.4 Komentar multi-baris

Komentar multi-baris dimulai dengan tanda “/\*” dan diakhiri dengan tanda “\*/”. Dengan menggunakan komentar multi-baris, apapun yang terletak di antara tanda “/\*” dan “\*/” akan dianggap sebagai komentar dan tidak akan dikompilasi oleh *compiler*. Perhatikan contoh penulisan komentar multi-baris berikut ini,

```
/*  
    Ini adalah program Hello World di C#  
    Program ini menampilkan tulisan Hello World di jendela konsol.  
*/  
using System;  
namespace HelloWorld  
{  
    class Program  
    {  
        public static void Main(string[] args)  
        {  
            /* Menampilkan Hello World */  
            Console.WriteLine("Hello World!");  
        }  
    }  
}
```

Pada contoh di atas, mungkin Anda memperhatikan adanya penggunaan komentar multi-baris untuk menulis sebuah komentar yang terdiri dari satu

baris saja. Tentu saja, Anda tidak diharuskan menggunakan cara penulisan komentar ini hanya untuk komentar yang terdiri dari banyak baris saja. Anda juga bebas menggunakannya untuk menulis komentar yang terdiri dari satu baris dengan cara ini.

## 7.5 Komentar dokumentasi XML

Penulisan komentar ini merupakan sebuah fitur spesial dari C#. Komentar ini dimulai dengan tiga buah garis miring (*triple slash*) “`///`” dan digunakan untuk mendeskripsikan sepotong kode secara kategoris. Ini dilakukan dengan menggunakan tag XML dalam komentar. Komentar ini nantinya digunakan untuk membuat file dokumentasi XML yang terpisah.

Pembahasan mengenai XML di luar dari cakupan materi dalam buku ini. Apabila Anda tertarik mempelajari XML lebih lanjut, Anda bisa mencari artikel-artikel yang banyak tersedia di Internet. Materi ini hanya memberi gambaran kepada Anda, bagaimana menggunakan komentar dokumentasi XML di dalam program C# Anda.

Berikut ini adalah contoh dari penggunaan komentar XML,

```
1220
1221    /// <summary>
1222    /// ストリップ検査処理 (Strip inspection process)
1223    /// </summary>
1224    1 個の参照
1225    public bool StripInspect(int pos, Bitmap input, out InspectData result)
1226    {
1227        // 結果データ新規作成
1228        result = new InspectData();
1229        if (Program.mIni.GetProfileString((int)Param.ImageSaveArea) == "1")
1230        {
1231            result.AreaImage = (Bitmap)input.Clone();
1232        }
1233    }
```

### Gambar 7.1

Komentar XML di atas digunakan untuk mendeskripsikan method bernama “*StripInspect*”. Perhatikan gambar berikut ini, Anda bisa melihat diskripsi dari method “*StripInspect*” yang ditulis menggunakan komentar dokumentasi XML ketika Anda mengarahkan kursor di atas pemanggilan method tersebut.

```
bool ret = true;
// 検査処理
if (mMastData.Mode == Program.MODE_STRIP)
{
    ret = StripInspect(cd.CameraPos, input, out result.ResultData);
}
else if (mMastData.Mode == Program.MODE_STRIP_CAMERA_CONTROL_EX)
{
    bool ThreadCameraControlEx.StripInspect(int pos, Bitmap input, out InspectData result)
    {
        // ストリップ検査処理 (Strip inspection process)
        ret = SealInspect(cd.CameraPos, input, out result.ResultData);
    }
}
```

### Gambar 7.2

## 8 Pengambilan Keputusan

## 8.1 Pengenalan

Struktur pengambilan keputusan dalam bahasa C# menyediakan perhitungan logika yang memungkinkan eksekusi pada beberapa bagian kode program yang berbeda tergantung pada status data dalam suatu aplikasi.

Misalnya Anda ingin menanyakan berapa usia pengguna aplikasi Anda. Berdasarkan nilai masukan yang diberikan oleh pengguna, Anda bisa menentukan apa yang harus dikerjakan oleh aplikasi Anda.

Pengambilan keputusan dalam bahasa C# utamanya adalah pernyataan *if*. Alternatif terhadap pernyataan *if* adalah pernyataan *switch*. Anda mungkin nantinya akan lebih memilih menggunakan pernyataan *switch* untuk pengambilan keputusan yang lebih rumit.

## 8.2 Pernyataan if

Dalam pemrograman C#, pernyataan *if* erat hubungannya dengan logika *Boolean*. Apabila pernyataan *if* menghasilkan nilai *true*, maka baris kode program yang berkaitan akan dieksekusi. Sebaliknya, apabila menghasilkan nilai *false*, maka kode program tersebut akan diabaikan.

Contoh kode program di bawah ini mendemonstrasikan sebuah pernyataan *if* untuk melihat status dari variabel *respon* apakah memuat nilai “Ya”.

```
//Pernyataan if
string respon = "Ya";
if (respon == "Ya")
{
    // pernyataan atau baris program yang akan dieksekusi apabila nilai variable respon
    // bernilai Ya, diletakkan di sini.
```

```
}
```

Perhatikan tanda kurung kurawal pada contoh kode program di atas. Anda bisa menghapus tanda kurung kurawal tersebut apabila baris kode program yang dieksekusi hanya terdiri dari satu baris. C# mengerti apabila tidak ada kurung kurawal yang digunakan, baris kode yang terletak segera setelah pernyataan *if(kondisi)* akan dieksekusi, jika '*kondisi*' menghasilkan nilai *true*. Jika tidak, baris kode tersebut akan diabaikan.

Untuk menghindari kebingungan mengenai baris mana yang akan dieksekusi ketika kondisi bernilai *true*, praktik penulisan kode yang disarankan adalah selalu menggunakan tanda kurung kurawal untuk pernyataan *if*.

### 8.3 Pernyataan if-else

Dalam bahasa pemrograman C#, pernyataan *if* bisa disertai dengan klausa *else*. Ketika pernyataan *if* menghasilkan nilai *false*, maka baris kode program setelah klausa *else* yang akan dieksekusi.

Contoh kode pemrograman berikut memperlihatkan bagaimana menggunakan pernyataan *if-else* untuk mengeksekusi sebuah atau beberapa baris kode program ketika kondisinya bernilai *false*.

```
//Pernyataan if else
string respon;
if (respon == "koneksi_gagal")
{
    // Baris ini dieksekusi ketika nilai dari variabel repon adalah "koneksi_gagal".
}
else
{
```

```
// Baris ini dieksekusi ketika nilai dari variabel repon tidak sama dengan
"koneksi_gagal".
}
```

## 8.4 Pernyataan if-else if

Pernyataan *if* juga bisa disertai dengan klausa *else if*. Klausa ini dievaluasi kondisinya sesuai dengan urutan penulisannya setelah pernyataan *if*. Jika satu dari klausa yang ada menghasilkan nilai *true*, maka baris kode program yang berkaitan dengan klausa tersebut yang akan dieksekusi.

Contoh kode program berikut memperlihatkan bagaimana menggunakan pernyataan *if* dengan sebuah klausa *else if*.

```
//else if Statements
string response;
if (response == "koneksi_gagal")
{
    // Baris ini dieksekusi ketika nilai dari variabel repon adalah "koneksi_gagal".
}
else if (response == "koneksi_error")
{
    // Baris ini dieksekusi ketika nilai dari variabel repon adalah "koneksi_error".
}
else
{
    // Baris ini dieksekusi ketika nilai dari variabel repon tidak sama dengan respon-
    respon di atas
}
```

Anda dapat membuat blok *else if* sebanyak-banyaknya sesuai dengan logika keputusan yang Anda butuhkan. Namun, ketika Anda memerlukan lebih dari lima klausa *else if*, mungkin lebih baik bagi kita untuk mempertimbangkan penggunaan pernyataan *switch*, yang akan dijelaskan di Bab selanjutnya.



## 8.5 Operator Ternary

Operator Ternary (`? :`) adalah operator pengambilan keputusan yang biasanya digunakan sebagai pengganti dari pernyataan *if-else* dalam bahasa pemrograman C#. Dengan menggunakan Operator Ternary, Anda dapat mengganti beberapa baris kode pernyataan *if-else* ke dalam satu baris kode program saja.

Operator ternary akan selalu menggunakan 3 *operand*. Berikut ini adalah sintaks yang mendefinisikan Operator Ternary dalam bahasa pemrograman C#.

```
ekspresi_pengkondisian ? ekspresi_pertama : ekspresi_kedua;
```

Jika Anda amati sintaks di atas, operator kondisional (`?:`) hanya akan mengembalikan satu nilai dari ekspresi yang didefinisikan, bisa `ekspresi_pertama` ataupun `ekspresi_kedua`, tergantung nilai dari kondisinya.

Dalam pemrograman C#, Operator ternary bekerja seperti berikut,

- Ekspresi pengkondisian akan dievaluasi apakah menghasilkan nilai `true` atau `false`. Apabila kondisinya bernilai `true`, `ekspresi_pertama` akan dikembalikan oleh operator ternary tersebut.
- Pada kasus di mana kondisinya bernilai `false`, hasil yang dikembalikan oleh operator ternary adalah `ekspresi_kedua`.

Seperti yang telah disebutkan sebelumnya, operator ternary adalah pengganti dari pernyataan *if-else*. Sebagai contoh, kita bisa mengganti pernyataan *if-else* berikut dengan operator ternary seperti yang didemonstrasikan dalam contoh berikut.

```

int x = 5, y = 20;
string result;
// Pernyataan if-else
if (x > y)
{
    result = "x lebih besar dari y";
}
else
{
    result = "x lebih kecil dari y";
}
//Pernyataan menggunakan Operator Ternary (?:)
result = (x > y) ? "x lebih besar dari y" : "x less than y";

```

## 8.6 Pernyataan switch

Apabila terlalu banyak menggunakan pernyataan *else if*, kode program Anda bisa menjadi terlihat berantakan dan susah dibaca. Pada skenario seperti ini, solusi yang lebih baik adalah menggunakan pernyataan *switch*. Pernyataan *switch* pada dasarnya hanyalah menggantikan beberapa pernyataan *else if*.

Contoh kode program berikut menunjukkan bagaimana menggunakan pernyataan *switch* untuk menggantikan kumpulan klausa *else if*.

```

//Pernyataan switch
string respon;
switch (respon)
{
    case "koneksi_gagal":
        // Baris ini dieksekusi ketika nilai dari variabel repon adalah "koneksi_gagal".
        break;
    case "koneksi_sukses":
        // Baris ini dieksekusi ketika nilai dari variabel repon adalah "koneksi_sukses".
        break;
}

```

```

    case "koneksi_error":
        // Baris ini dieksekusi ketika nilai dari variabel repon adalah "koneksi_error".
        break;
    default:
        // Baris ini dieksekusi ketika tidak ada kondisi di atas yang terpenuhi.
        break;
}

```

Pada setiap pernyataan *case* di atas, perhatikan penggunaan kata kunci *break*. Setelah baris kode program pada pernyataan *case* diproses, kata kunci ini menyebabkan kontrol eksekusi melompat ke bagian akhir dari pernyataan *switch*. Jika Anda menghilangkan kata kunci tersebut, kode Anda akan gagal dikompilasi dan menghasilkan eror “*Control cannot fall through from one case label*”.

Apabila Anda ingin menangani beberapa kasus (*case*) dengan segmen kode program yang sama, Anda bisa menggunakan cara yang biasa disebut *fall-through setup* seperti pada contoh kode program di bawah ini.

```

//Pernyataan switch
string respon;
switch (respon)
{
    case "koneksi_sukses":
        // Baris ini dieksekusi ketika nilai dari variabel repon adalah "koneksi_sukses".
        break;
    case "koneksi_gagal":
    case "koneksi_error":
        // Baris ini dieksekusi ketika nilai dari variabel repon adalah "koneksi_error".
        // atau "koneksi_gagal".
        break;
    default:
        // Baris ini dieksekusi ketika tidak ada kondisi di atas yang terpenuhi.
}

```

```
        break;  
    }
```

Jika Anda pernah mempelajari bahasa pemrograman lain, seperti bahasa C yang juga menggunakan pernyataan *switch*, mungkin Anda menyadari bahwa dalam bahasa C#, Anda bisa menggunakan nilai string di dalam pernyataan *switch* dan tidak harus menggunakan integer atau tipe enum.

Pernyataan switch di C# mendukung tipe data berikut untuk ekspresinya:

- sbyte
- byte
- short
- ushort
- int
- uint
- long
- ulong
- char
- string
- enumerations

## 9 Pengulangan / Iterasi

## 9.1 Pengenalan

Pengulangan pada dasarnya adalah sebuah konsep untuk melakukan suatu hal dengan cara berulang. Dalam pemrograman, Anda biasanya menggunakan pengulangan untuk mengiterasi item di dalam sebuah *collection* atau elemen di dalam sebuah *array*. Atau, Anda biasanya juga menggunakannya untuk melakukan tugas yang sama berulang kali agar menghasilkan efek yang diinginkan dalam program Anda. (*catatan : collection dan array akan Anda pelajari di buku **Algoritma dan Struktur Data di Pemrograman C#***)

C # menyediakan sejumlah konstruksi standar yang dikenal sebagai “loop” yang bisa Anda gunakan untuk mengimplementasikan logika iterasi. Jika Anda pernah mempelajari bahasa pemrograman lain, Anda mungkin sudah mengenal beberapa metode pengulangan seperti *for loop*, *while loop*, dan *do loop*. C# mendukung ketiga pernyataan iterasi tersebut.

## 9.2 for loop

Fungsi dari *for loop* adalah untuk mengeksekusi blok kode program berulang kali hingga ekspresi yang ditentukan bernilai *false*. Anda dapat mendefinisikan *for loop* sebagai berikut.

```
for ([penginisialisasi]; [kondisi]; [iterator])
{
    // kode yang diiterasi diletakkan di sini
}
```

Bagian [penginisialisasi] digunakan untuk menginisialisasi suatu nilai ke pencacah (*counter*) iterasi. Iterasi akan terus dijalankan sampai bagian [kondisi] terpenuhi. Selama iterasi dijalankan, kode program di dalam loop

akan terus dieksekusi. Di setiap akhir iterasi, `bagian[iterator]` bertugas untuk menaikkan nilai dari pencacah.

Contoh kode program di bawah menunjukkan bagaimana menggunakan *for loop* untuk mengeksekusi sebuah blok kode program sebanyak 10 kali.

```
//for Loop
for (int i = 0; i < 10; i++)
{
    // Kode program yang dieksekusi.
}
```

Pada contoh di atas, `i = 0;` adalah penginisialisasinya, `i < 10;` adalah kondisinya, dan `i++` adalah iteratornya.

### 9.3 foreach loop

Meskipun *for loop* mudah digunakan, metode iterasi ini bisa menghadirkan beberapa tantangan, tergantung situasinya. Sebagai contoh, anggap Anda perlu mengiterasi item-item di dalam sebuah *collection* atau elemen-elemen di dalam sebuah *array*, Anda perlu mengetahui berapa banyak elemen yang ada di dalam *collection* atau *array* tersebut. Memang dalam banyak kasus, Anda sudah menentukan ukuran atau banyaknya elemen yang membentuk sebuah *collection* atau *array*. Namun, kadang-kadang Anda harus menghadapi *collection* atau *array* yang dinamis. Artinya, ukuran dari *collection* dan *array* tersebut baru akan ditetapkan pada saat kompilasi.

Jika ukuran dari *collection* atau *array* berubah pada saat *runtime*, menggunakan *foreach loop* menjadi pilihan yang lebih baik.

Contoh kode program berikut ini menunjukkan bagaimana menggunakan *foreach loop* untuk mengiterasi sebuah *string array*.

```
//foreach Loop
string[] names = new string[10];

// Proses setiap name di dalam array.
foreach (string name in names)
{
    // Kode yang dieksekusi.
}
```

C # menangani penentuan berapa banyak item dalam *array*, iterasi akan berhenti mengeksekusi sebuah loop ketika akhir dari loop tersebut tercapai. Penggunaan *foreach loop* dapat membantu mencegah error *index out of bounds* pada array. Yaitu, error yang terjadi ketika Anda mencoba mengakses elemen dengan indeks yang melebihi ukuran dari array tersebut. Misalnya, ketika Anda mencoba mengakses elemen ke-10 dari sebuah array, sementara array tersebut hanya berukuran sembilan elemen, error tersebut akan terjadi.

Untuk saat ini, Anda tidak perlu bingung dengan apa itu *collection* dan *array*. Anda akan mempelajarinya di kursus ***Algoritma dan Struktur Data di Pemrograman C#***. Setelah Anda sudah mengenal dan terbiasa dengan *collection* dan *array*, Anda mungkin perlu kembali ke bagian ini untuk mengingat kembali bagaimana menggunakan *foreach loop* untuk melakukan iterasi pada dua struktur data tersebut.

## 9.4 while loop

Menggunakan *while loop* memungkinkan Anda untuk mengeksekusi sebuah blok kode program secara terus-menerus selama kondisi yang diberikan



bernilai *true*. Sebagai contoh, Anda bisa menggunakan *while loop* untuk memproses data masukan dari pengguna sampai pengguna menyatakan bahwa tidak ada lagi data yang dimasukkan. Loop tersebut dapat terus meminta data dari pengguna sampai mereka memutuskan untuk mengakhiri interaksi dengan memasukkan nilai sentinel. Nilai sentinel tersebut bertanggung jawab untuk mengakhiri loop.

Kode program di bawah ini menunjukkan bagaimana menggunakan perulangan *while loop* dengan nilai sentinel “Keluar”.

```
//while Loop
string respon = InputData();
while (respon != "Keluar")
{
    // Proses Data.
    respon = InputData();
}
```

Dari contoh kode program di atas, perlu diperhatikan bahwa baris kode `respon = InputData();` tidak boleh tidak dimasukkan ke dalam badan loop. Jika tidak, loop tersebut akan menjadi *infinite loop* atau perulangan yang tidak akan pernah selesai. Hal ini dikarenakan nilai sentinel tidak pernah bisa diubah.

Untuk lebih memahami bagaimana *while loop* bekerja, perhatikan contoh kode program berikut.

```
//while Loop
int i = 1;
while (i<=5)
```

```
{  
    Console.WriteLine("C# while Loop: Iterasi {0}", i);  
    i++;  
}
```

Jika kode program di atas Anda jalankan, nilai keluarannya akan seperti berikut.

```
C# while Loop: Iterasi 1  
C# while Loop: Iterasi 2  
C# while Loop: Iterasi 3  
C# while Loop: Iterasi 4  
C# while Loop: Iterasi 5
```

- Awalnya ‘i’ bernilai 1.

Ketika program memasuki pernyataan *while loop*,

- Ekspresi pengkondisian  $i \leq 5$  dievaluasi. Karena ‘i’ bernilai 1 dan  $1 \leq 5$  adalah bernilai *true*, baris kode program di dalam badan *while loop* dieksekusi. Di sini, tulisan “**C# while Loop: Iterasi 1**” akan ditampilkan di layar konsol. Lalu, dibaris selanjutnya nilai ‘i’ dinaikkan menjadi 2.
- Sekarang, ekspresi pengkondisian  $i \leq 5$  dievaluasi kembali. Kali ini ekspresi tersebut juga menghasilkan nilai *true* karena  $2 \leq 5$ , jadi tulisan “**C# while Loop: Iterasi 2**” ditampilkan di layar dan nilai dari ‘i’ sekarang adalah 3.
- Perulangan di atas akan terus terjadi sampai nilai ‘i’ menjadi 6. Pada titik ini, ekspresi  $i \leq 5$  akan menghasilkan nilai *false* karena angka 6 tidak lebih kecil atau sama dengan 5. Oleh karena itu, loop akan berhenti.

## 9.5 do-while loop

Perulangan *do-while loop* hampir sama dengan perulangan *while loop*, hanya saja ada satu perbedaan mendasar di antara kedua perulangan tersebut.

Pada *while loop*, pengkondisian dicek terlebih dahulu sebelum baris kode program di dalam badan loop tersebut dieksekusi. Kebalikannya, *do-while loop* mengeksekusi baris kode program di dalam badan loop terlebih dahulu sebelum mengecek pengkondisian.

Itulah mengapa, baris kode program di dalam badan *do-while loop* akan dieksekusi paling tidak sekali tanpa memperhatikan ekspresi pengkondisian pada loop tersebut apakah bernilai *true* atau *false*.

Untuk lebih memahami bagaimana while loop bekerja, perhatikan contoh kode program berikut.

```
//do-while Loop
int i = 1, n = 5, hasilKali;
do
{
    hasilKali = n * i;
    Console.WriteLine("{0} * {1} = {2}", n, i, hasilKali);
    i++;
} while (i <= 10);
```

Jika kode program di atas Anda jalankan, nilai keluarannya akan seperti berikut.

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
```

```
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Seperti yang bisa Anda lihat, program diatas menampilkan tabel perkalian dari angka 5.

- Awalnya 'i' bernilai 1. Program mengeksekusi baris kode program di dalam badan *do-while loop* tanpa mengecek kondisi apapun (berkebalikan dengan *while loop*).
- Di dalam badan *do-while loop*, hasil kali dihitung dan ditampilkan di layar konsol. Nilai dari 'i' kemudian menjadi 2.
- Setelah eksekusi terhadap baris kode program di dalam badan loop, ekspresi pengkondisian  $i \leq 10$  dievaluasi. Totalnya, *do-while loop* ini akan dijalankan selama 10 kali.
- Terakhir, ketika nilai dari 'i' adalah 11, ekspresi pengkondisian akan bernilai *false* karena angka 11 tidak lebih kecil atau sama dengan 10. Oleh karena itu, loop akan berhenti.

## 10 Method

## 10.1 Apa itu method

Method adalah sebuah blok kode program yang berisi serangkaian pernyataan atau instruksi. Selain method bawaan dari .NET seperti ***Parse()***, ***TryParse()***, ***WriteLine()***, ***ReadLine()*** dan masih banyak lainnya, Anda juga bisa mendefinisikan method Anda sendiri.

Kelebihan menggunakan method untuk instruksi-instruksi program Anda, antara lain:

- Kode program dapat digunakan kembali (*reusable code*).
- Mudah untuk dites.
- Memodifikasi sebuah *method* tidak akan mempengaruhi bagian kode program yang memanggil *method* tersebut.
- Satu *method* dapat menerima banyak input yang berbeda.

Setiap program C# yang valid, pasti memiliki paling tidak satu buah method. Yaitu, *method* **Main()** yang merupakan *entry point* dari program yang Anda buat.

## 10.2 Mendeklarasikan sebuah method

Ketika Anda memiliki serangkaian instruksi untuk sebuah operasi, sebaiknya Anda mendeklarasikan sebuah method untuk mendefinisikan operasi tersebut.

Contohnya, ketika Anda memerlukan operasi matematika yang sama berkali-kali pada kode program yang Anda tulis, Anda bisa mendeklarasikan sebuah

method, lalu memanggil method tersebut pada bagian kode program yang membutuhkan operasi matematika tersebut.

Deklarasi sebuah method biasanya mencakup komponen-komponen (*signature*) seperti berikut:

- Tipe *return* (tipe data dari nilai yang dikembalikan oleh sebuah method),
- Nama method,
- dan Daftar parameter (bersifat opsional).

Sementara, *syntax* untuk mendeklarasikan sebuah method adalah seperti berikut ini,

```
<tipe return> NamaMethod(type1 parameter1, type2 parameter2, ... , typeN parameterN)
{
    Daftar pernyataan
}
```

Tipe *return* dari sebuah method dideklarasikan sebelum namanya. Tipe return sendiri mengindikasikan bahwa method yang kita deklarasikan akan mengembalikan atau menghasilkan sebuah nilai yang nantinya bisa kita gunakan untuk menentukan nilai sebuah variabel pada pernyataan penugasaan atau *assignment statement*.

Agar sebuah method bisa mengembalikan sebuah nilai, method tersebut harus menyertakan sebuah pernyataan *return*.

Sebagai contoh, Ketika Anda ingin mendeklarasikan sebuah method untuk mendefinisikan sebuah operasi perkalian kuadrat, Anda menentukan bahwa

method ini memiliki tipe *return* **int** karena hasil dari operasi perkalian kuadrat akan bertipe data integer. Kemudian, Anda menamainya dengan “Kuadrat” dan memutuskan untuk menambahkan satu parameter bernama “x” yang bertipe data **int** untuk angka yang ingin Anda kuadratkan.

Lalu, Anda menggunakan nilai kembalian dari method “Kuadrat” tersebut untuk menentukan nilai variabel bernama “hasilKuadrat” yang bertipe data **int**.

```
int Kuadrat(int x)
{
    int hasil = x*x; //instruksi untuk melakukan operasi perkalian kuadrat
    return hasil; //nilai yang dikembalikan oleh method “Kuadrat”
}

void Main(string[] args)
{
    int hasilKuadrat;
    hasilKuadrat = Kuadrat(2); //Pernyataan penugasan atau assignment statement
}
```

Namun terkadang, sebuah method menjalankan suatu operasi tanpa mengembalikan sebuah nilai. Method seperti ini memiliki tipe return **void**. Method dengan tipe *return* **void** tidak bisa dipanggil sebagai bagian dari pernyataan penugasan seperti pada method yang Anda buat sebelumnya.

**Void** adalah tipe data dasar yang mendefinisikan sebuah status tanpa nilai (*valueless state*).

Sebagai contoh, kode program berikut ini mendefinisikan *method* yang tidak mengembalikan sebuah nilai apapun, dan sekedar mencetak baris teks ke layar.



```
void MenyapaHai ()
{
    Console.WriteLine("Halo!");
}
```

### 10.3 Memanggil sebuah method

Untuk mengeksekusi sebuah *method*, Anda cukup memanggil *method* tersebut dengan menggunakan **nama** dan **parameter** yang diperlukan dalam sebuah pernyataan. Seperti pada contoh sebelumnya, Anda memanggil method **Kuadrat()** pada saat menentukan nilai dari variabel “hasilKuadrat” dengan memberi angka “2” sebagai parameternya.

Parameter ini sebenarnya bersifat opsional. Maksudnya, Anda bisa memiliki sebuah method tanpa parameter apapun. Perhatikan contoh kode program berikut ini,

```
static void MenyapaHai ()
{
    Console.WriteLine("Halo!");
}
static void Main(string[] args)
{
    MenyapaHai ();
}
//Menghasilkan keluaran "Halo!" di layar
```

Pada contoh kode program di atas, Anda mendeklarasikan sebuah method untuk menampilkan tulisan “Halo!” di layar monitor, lalu memanggil method tersebut di dalam method **Main()**. Karena method yang Anda deklarasikan tersebut tidak memiliki parameter, maka Anda tidak perlu memberi parameter

apapun di dalam tanda kurung setelah nama method pada saat memanggil method **MenyapaHai()**.

*(Perhatikan juga penulisan method MenyapaHai pada contoh di halaman sebelumnya. Kali ini Anda menambahkan kata kunci **static** pada deklarasi method tersebut. Secara default, method **Main()** pada program C# dideklarasikan sebagai **static**. Karena method yang dideklarasikan sebagai static hanya bisa mengakses elemen static, maka method **MenyapaHai()** harus dideklarasikan dengan kata kunci static. Kata kunci static akan dijelaskan di buku **Pemrograman Berorientasi Obyek Dengan Bahasa C#**.)*

Anda bisa memanggil method yang sama lebih dari satu kali. Pada contoh berikut ini, Anda memanggil method **MenyapaHai** tiga kali di dalam method **Main**.

```
static void Main(string[] args)
{
    MenyapaHai ();
    MenyapaHai ();
    MenyapaHai ();
}
//keluaran di layar
//halo!
//halo!
//halo!
```

Ketika program di atas Anda jalankan, tulisan “Halo!” akan ditampilkan sebanyak tiga baris di jendela konsol.

## 10.4 Memberi parameter pada sebuah method

Pada saat mendeklarasikan sebuah method, Anda dapat menentukan daftar parameter yang diperlukan untuk perhitungan atau operasi yang dilakukan oleh method tersebut.

Parameter sendiri sebenarnya adalah variabel yang menerima sebuah nilai yang dilewatkan atau diberikan ke sebuah method pada saat method tersebut dipanggil. Sebagai contoh,

```
void Cetak(int x)
{
    Console.WriteLine(x);
}
```

Contoh kode program di atas mendefinisikan sebuah method yang diberi nama “Cetak” yang mengambil satu parameter bernama ‘x’ yang bertipe data **int** dan kemudian menampilkan nilainya ke layar monitor.

Kemudian, pada saat Anda memanggil method **Cetak()** di atas, Anda perlu melewatkan atau memberikan sebuah nilai untuk parameternya seperti pada contoh di bawah ini.

```
static void Cetak(int x)
{
    Console.WriteLine(x);
}

static void Main(string[] args)
{
    Print(56);
}
```

Nilai yang dilewatkan atau diberikan ke sebuah method biasa disebut juga dengan **argumen**. Anda bisa memberikan argumen yang berbeda ke method yang sama selama nilai dari argumen tersebut masih dari tipe data yang sama. Sebagai contoh:

```
static void KaliDua(int x)
{
    Console.WriteLine(x * 2);
}
static void Main(string[] args)
{
    KaliDua(5);
    //Menghasilkan 10
    KaliDua(12);
    //Menghasilkan 24
    KaliDua(42);
    //Menghasilkan 84
}
// Argumen 5, 12, dan 84 masih dari tipe data yang sama, yaitu tipe int.
```

Anda dapat memiliki parameter sebanyak yang diperlukan untuk suatu method dengan memisahkannya menggunakan koma pada saat mendefinisikan sebuah method.

Contoh kode program berikut memperlihatkan bagaimana membuat sebuah method sederhana yang mengembalikan nilai penjumlahan dari dua parameter:

```
int Penjumlahan(int x, int y)
{
    return x + y;
}
```

Method di atas menerima dua argumen bertipe integer dan mengembalikan nilai penjumlahannya. Karena hasil penjumlahan dua bilangan integer menghasilkan bilangan integer, Method **Penjumlahan()** di atas harus menggunakan **int** sebagai tipe *return*-nya.

## 10.5 Argumen opsional

Ketika mendefinisikan sebuah method, Anda dapat menyediakan parameter yang bersifat opsional. Artinya, nantinya pada saat method tersebut dipanggil, Anda tidak diharuskan memberi argumen untuk parameter tersebut. Method tersebut akan menggunakan nilai default yang sebelumnya sudah Anda tentukan.

Untuk membuat parameter menjadi bersifat opsional, tentukan sebuah nilai default untuk parameter tersebut pada saat mendefinisikannya. Untuk lebih memahaminya, perhatikan contoh kode program berikut.

```
static int Pangkat(int x, int y = 2)
{
    int hasil = 1;
    for (int i = 0; i < y; i++)
    {
        hasil *= x;
    }
    return hasil;
}
```

Perhatikan bahwa parameter opsional harus didefinisikan setelah parameter yang diperlukan. Maksudnya, pada contoh di atas, parameter 'y' yang merupakan parameter opsional, harus didefinisikan setelah parameter 'x' yang

merupakan parameter yang nantinya harus diberi argumen pada saat memanggil method **Pangkat()**.

Pada contoh method **Pangkat()** tadi, parameter 'y' memiliki nilai default 2. Nantinya, ketika Anda memanggil method tersebut tanpa memberikan argumen untuk parameter 'y', nilai default dari parameter tersebut akan digunakan.

Namun, apabila Anda memberikan argumen yang bernilai selain 2 untuk parameter 'y', method Pangkat() akan menggunakan nilai argumen yang Anda berikan dan akan mengabaikan nilai default yang diberikan kepada method tersebut.

Perhatikan contoh berikut,

```
static void Main(string[] args)
{
    Console.WriteLine(Pangkat(6));
    //Argumen untuk parameter 'y' tidak diberikan pada saat memanggil method Pangkat()
    //Method Pangkat() akan menggunakan nilai default untuk parameter 'y'
    //Output 36 (6 pangkat 2)
    Console.WriteLine(Pangkat(3, 4));
    //Memberikan nilai 4 untuk parameter 'y' pada saat memanggil method Pangkat()
    //Method Pangkat() akan mengabaikan nilai default untuk parameter 'y'
    //Output 81 (3 pangkat 4)
}
```

## 10.6 Argumen bernama

Pada saat memberikan argumen untuk sebuah method, Anda harus memperhatikan urutan definisi parameter pada deklarasi method tersebut. Contohnya pada method Pangkat() sebelumnya, parameter 'x' didefinisikan

sebelum parameter 'y'. Maka, pada saat Anda memberikan argumen ketika memanggil method tersebut, argumen yang pertama akan digunakan oleh parameter 'x' dan argumen kedua akan digunakan oleh parameter 'y'.

Ada kalanya ketika Anda terbalik memberikan argumen untuk sebuah method. Misalnya, Anda ingin menghitung nilai dari  $3^4$ , namun karena Anda memberikan argumen secara terbalik, perhitungannya malah menjadi  $4^3$  dan menghasilkan nilai diluar yang Anda inginkan.

Dengan menggunakan argumen bernama, Anda dibebaskan dari kebutuhan untuk mengingat urutan parameter dalam pemanggilan sebuah method. Setiap argumen dapat ditentukan dengan nama parameter yang bersesuaian.

Sebagai contoh, method berikut menghitung luas persegi panjang dengan nilai tinggi dan lebarnya:

```
static int Luas(int t, int l)
{
    return t * l;
}
```

Saat memanggil method di atas, Anda dapat menggunakan nama parameternya pada saat memberikan argumen. Dengan demikian, Anda bisa memberikan argumen dalam urutan apa pun yang Anda suka dengan cara seperti berikut:

```
static void Main(string[] args)
{
    int hasil = Luas(l: 8, t: 5);
    Console.WriteLine(hasil);
    //Output 40
}
```

Meskipun parameter 't' didefinisikan sebelum parameter 'l', namun karena Anda menggunakan nama parameternya masing-masing pada saat melewati argumen, terbalikpun, method Luas() akan mengerti argumen yang mana untuk parameter apa.

Untuk membuat argumen bernama, Anda dapat menggunakan nama parameternya diikuti oleh titik dua dan nilainya.

## 10.7 Melewatkan Argumen

Ada tiga cara untuk melewati argumen ke suatu method ketika method tersebut dipanggil. Yaitu, berdasarkan nilai (*by value*), berdasarkan referensi (*by Reference*), dan sebagai *Output*. Secara default, C # menggunakan pemanggilan berdasarkan nilai atau biasa disebut dengan *call by value* untuk melewati argumen.

### Melewatkan argumen berdasarkan nilai (*pass by value*)

Melewatkan argumen berdasarkan nilai (*pass by value*), hanya akan menyalin nilai dari sebuah argumen ke dalam parameter formal milik method tersebut. Maksudnya, ketika Anda melewati atau memberikan sebuah variabel sebagai argumen untuk sebuah method, **nilai** dari variabelnya saja yang disalin dan kemudian diberikan kepada parameter dari method tersebut.

Dengan demikian, pada saat sebuah method melakukan manipulasi pada argumen yang diberikan kepadanya, nilai dari variabel yang diberikan sebagai argumen untuk method tersebut tidak akan berubah.

Untuk lebih memahami konsep *pass by value*, coba perhatikan potongan kode program berikut ini,



```

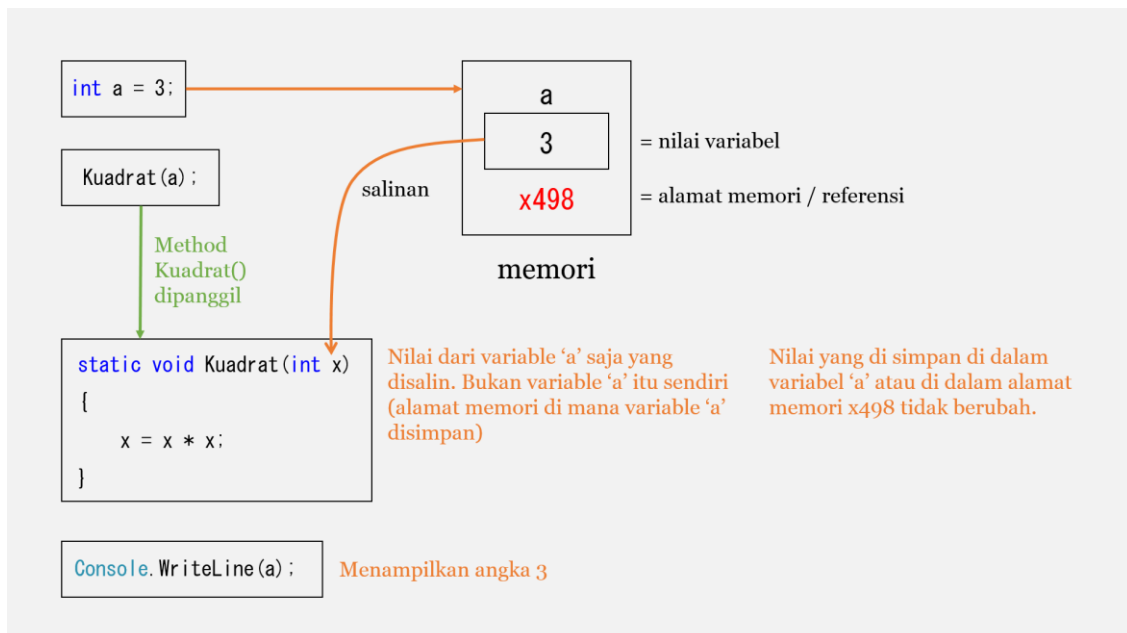
static void Kuadrat(int x)
{
    x = x * x;
}

static void Main(string[] args)
{
    int a = 3;
    Kuadrat(a);
    Console.WriteLine(a); // Output 3
}

```

Pada contoh di atas, 'x' adalah parameter dari method **Kuadrat()** dan merupakan argumen aktual yang dilewatkan ke dalam method tersebut.

Mungkin Anda akan mengira bahwa kode program tersebut akan menampilkan angka 9 di jendela konsol. Namun, apabila dijalankan, kode program di atas tidak akan menghasilkan nilai 9, melainkan menghasilkan nilai 3. Mengapa?



**Gambar 10.1**

Pada saat *method* **Kuadrat()** dipanggil, hanya nilai dari variabel 'a' saja, yaitu nilai 3, yang **disalin** ke parameter 'x' milik *method* **Kuadrat()**. Kemudian, salinan dari nilai 3 itu lah yang digunakan untuk melakukan perhitungan terhadap nilai 'x'.

Dari hasil perhitungan tersebut, kini parameter 'x' bernilai 9. Karena tadi yang dilewatkan adalah **hanya salinan nilai** dari variable 'a' dan **bukan variable 'a' itu sendiri**, ketika Anda mencoba menampilkan nilai variabel a, yang ditampilkan adalah angka 3 dan bukan angka 9. Karena nilai variabel 'a' sendiri tidak pernah berubah.

### **Melewatkan argumen berdasarkan referensi (*pass by reference*)**

Melewatkan argumen berdasarkan referensi, menyalin alamat memori dari sebuah variabel, yang dilewatkan ke suatu *method* sebagai argumen, ke dalam parameter formal milik *method* tersebut.

Di dalam *method*, alamat ini digunakan untuk mengakses argumen aktual (variabel) yang diberikan atau dilewatkan pada saat pemanggilan *method*. Ini berarti bahwa apabila sebuah *method* melakukan manipulasi terhadap parameternya, variabel yang diberikan atau dilewatkan kepada *method* tersebut juga akan berubah.

Untuk melewati argumen berdasarkan referensinya, kata kunci referensi digunakan baik dalam pemanggilan maupun definisi *method*:

```
static void Kuadrat(ref int x)
{
    x = x * x;
}
```

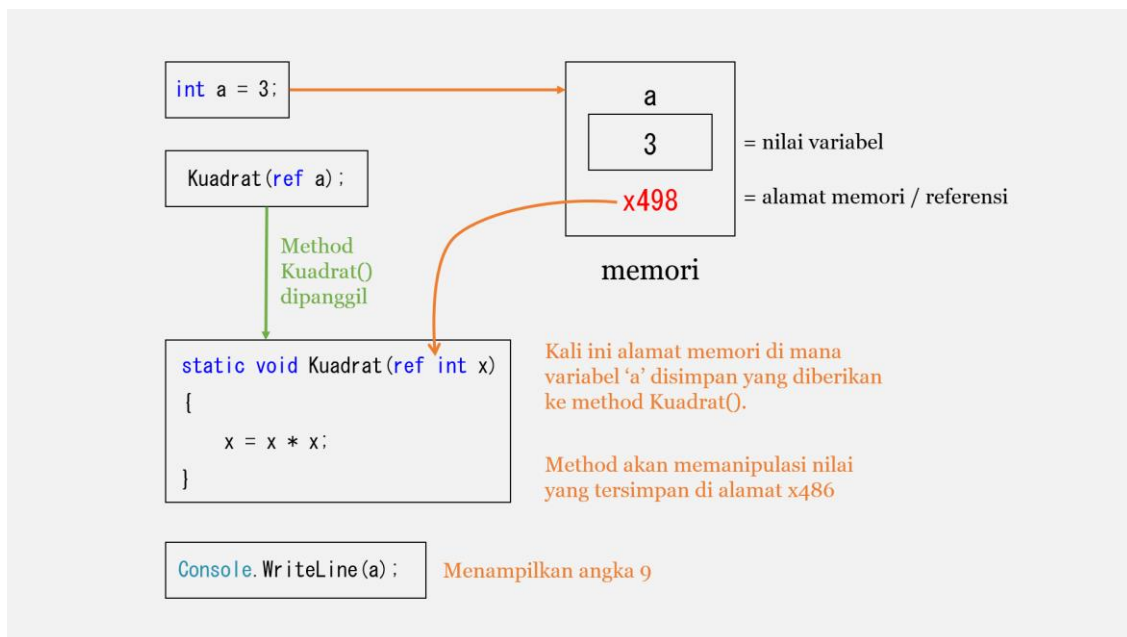
```

}
static void Main(string[] args)
{
    int a = 3;
    Kuadrat(ref a);
    Console.WriteLine(a); // Output 9
}

```

Kata kunci *ref* melewati alamat memori ke parameter sebuah method, yang memungkinkan method tersebut beroperasi pada variabel yang sebenarnya dan bukan salinan nilai variabelnya. Apabila dijalankan, contoh kode program di atas menghasilkan keluaran angka 9.

Mengapa keluarannya berbeda dengan kode program sebelumnya?



**Gambar 10.2**

Kali ini, pada saat pemanggilan method **Kuadrat()**, alamat dari variabel 'a' lah yang dilewatkan ke method tersebut sehingga parameter 'x' memiliki akses ke alamat memori yang menyimpan variabel 'a'. Hal ini menyebabkan setiap

manipulasi yang dilakukan oleh method **Kuadrat()** terhadap nilai yang tersimpan di dalam memori tersebut, akan mempengaruhi nilai dari variabel 'a'.

### Melewatkan argumen dengan kata kunci Output

Parameter Output sedikit mirip dengan parameter referensi. Bedanya, parameter ini lebih digunakan untuk mentransfer keluar sebuah nilai yang terdapat di dalam sebuah method. Anda tidak bisa melewati atau memberikan sebuah argumen untuk parameter output.

Parameter output didefinisikan dengan menggunakan kata kunci *out*.

Pada saat anda memanggil sebuah method dengan parameter output, variabel yang Anda deklarasikan untuk menyimpan nilai dari parameter output tersebut tidak perlu diinisialisasi karena nilai dari variabel tersebut akan didapatkan setelah pemanggilan terhadap method tersebut.

Untuk lebih memahaminya, coba perhatikan contoh kode program berikut ini yang memperlihatkan bagaimana mendefinisikan sebuah method dengan parameter output.

```
static void DapatkanNilai(out int x, out int y)
{
    x = 5;
    y = 42;
}
static void Main(string[] args)
{
    int a, b;
    // variabel a dan b tidak perlu diinisialisasi
    // karena nilai a dan b akan di dapat setelah melakukan
```

```
// pemanggilan terhadap method DapatkanNilai
DapatkanNilai(out a, out b);
Console.WriteLine(a); //5
Console.WriteLine(b); //42
}
```

Berbeda dengan contoh tipe referensi sebelumnya, di mana method **Kuadrat()** mendapatkan sebuah argumen berupa alamat memori yang menyimpan nilai 3 dan kemudian mengubah nilai yang tersimpan menjadi 9, parameter output hanya mentransfer sebuah nilai yang terdapat di dalam sebuah method (5 dan 42 dalam contoh di atas).

## 10.8 Method Overloading

*Method overloading* adalah ketika beberapa method memiliki nama yang sama, namun memiliki parameter yang berbeda.

Misalnya, Anda memiliki method **Cetak()** seperti kode program di bawah ini yang menampilkan nilai dari parameternya ke jendela konsol.

```
void Cetak(int a)
{
    Console.WriteLine("Nilai: " + a);
}
```

Method di atas **hanya** bisa menerima argumen dari tipe integer.

Apabila Anda ingin melewatkan argumen dari tipe berbeda, overloading adalah solusinya. Dengan melakukan overloading, method **Cetak()** akan tersedia untuk tipe data yang lain, misalnya *double*.

Coba perhatikan method **Cetak()** berikut ini,

```
void Cetak(double a)
{
    Console.WriteLine("Nilai: " + a);
}
```

Ketika ada method yang di-*overload*, method yang dipanggil didasarkan pada argumen yang dilewatkan. Melewatkan argumen dari tipe integer akan memanggil implementasi method yang bisa menerima parameter integer. Melewatkan argumen dari tipe double akan memanggil implementasi method yang bisa menerima parameter double. Melewatkan lebih dari satu argumen akan memanggil implementasi yang bisa menerima jumlah argumen yang sama.

```
static void Cetak(int a) {
    Console.WriteLine("Nilai: " + a);
}
static void Cetak(double a) {
    Console.WriteLine("Nilai: " + a);
}
static void Cetak(string label, double a) { // dua argumen
    Console.WriteLine(label + a);
}
static void Main(string[] args) {
    // memanggil method Cetak() yang bisa menerima argumen bertipe data integer
    Cetak(11);
    // memanggil method Cetak() yang bisa menerima argumen bertipe data double
    Cetak(4.13);
    // memanggil method Cetak() yang bisa menerima dua argumen dengan tipe data string
    // dan tipe data double
    Cetak("Rata-rata: ", 7.57);
}
```

# Praktikum

## Praktikum 1 : Mendeklarasikan Variabel

Dalam praktikum kali ini, Anda akan:

- Mempraktekkan bagaimana mendeklarasikan variabel dan menetapkan sebuah nilai ke variabel tersebut dengan benar.
- Mencoba menampilkan nilai variabel tersebut di jendela konsol (*console window*), dengan menggunakan method `WriteLine` yang terdapat pada class `Console`.

Praktikum ini menggunakan Visual Studio Community 2017. Jika Anda menggunakan tool pemrograman lain, sesuaikan langkah-langkah di bawah dengan tool yang Anda pakai.

1. Buka Visual Studio
2. Pilih File, New, Project
3. Dari Templates section, pilih Visual C#
4. Pilih Console App
5. Beri nama project Anda, seperti Modul1\_Lab
6. Pilih lokasi penyimpanan project Anda.
7. Klik tombol OK dan Visual Studio akan membuat sebuah proyek aplikasi konsol C# baru dan membuka Program.cs untuk Anda di jendela penyuntingan (*editor window*).



8. Arahkan kursor Anda tepat setelah kurung kurawal pembuka di method **Main**, kemudian tekan enter untuk membuat baris baru.

9. Masukkan kode pemrograman di bawah ini:

```
//Buat beberapa variabel dengan tipe data yang berbeda
//inisialisasi dengan nilai "default"
string jenisKendaraan = "";
string modelKendaraan = "";
string platNomor = "";
string noRegistrasi = "";
int thnPembuatan = 0;

//Tetapkan nilai dari variable yang telah dideklarasikan
jenisKendaraan = "Mobil";
modelKendaraan = "BMW i8 Roadster";
platNomor = "XY 3849 ABC";
noRegistrasi = "RG-0567319";
thnPembuatan = 2017;

//Tampilkan di jendela konsol
//Menggunakan nama variabel
Console.WriteLine(jenisKendaraan);
Console.WriteLine(modelKendaraan);

//menggunakan placeholder atau string format
Console.WriteLine("Diproduksi tahun {0}.", thnPembuatan);

//Menggunakan penggabungan string
Console.WriteLine("Plat Nomor : " + platNomor + "-- No Registrasi " + noRegistrasi);
```

10. Tekan tombol **CTRL+F5** untuk memulai aplikasi tanpa melakukan debugging. Atau, Anda bisa menggunakan Menu. Pilih **Debug -> Start Without Debugging**.

11. Visual Studio akan mengkompilasi kode program anda dan menjalankan aplikasi. Sebuah jendela konsol akan terbuka dan menampilkan hasil kompilasi apabila tidak terjadi eror.
12. Lakukan percobaan dengan menambahkan beberapa variabel baru, kemudian coba untuk menampilkannya di jendela konsol.

## Praktikum 2 : Menggunakan pernyataan if

1. Buka Visual Studio
2. Pilih File, New, Project
3. Dari Templates section, pilih Visual C#
4. Pilih Console App
5. Beri nama project Anda, seperti Modul2\_Praktik1
6. Pilih lokasi penyimpanan project Anda.
7. Klik tombol OK dan Visual Studio akan membuat sebuah proyek aplikasi konsol C# baru dan membuka Program.cs untuk Anda di jendela penyuntingan (*editor window*).
8. Arahkan kursor Anda tepat setelah kurung kurawal pembuka di method **Main**, kemudian tekan enter untuk membuat baris baru.
9. Masukkan kode pemrograman di bawah ini:

```
// Buat sebuah blok pengambil keputusan dengan pernyataan if
// gunakan blok pernyataan if ini untuk mengecek bilangan genap atau ganjil
// Minta pengguna untuk memasukkan sebuah nilai
Console.WriteLine("Masukkan bilangan bulat dan tekan Enter");

// masukkan nilai yang diberikan oleh pengguna ke dalam variabel input
// konversi nilai masukan yang didapat dari pengguna ke integer
int input = Int32.Parse(Console.ReadLine());

// Cek untuk melihat apakah nilainya genap atau ganjil
// Di sini Anda menggunakan perhitungan sederhana dengan menggunakan modulus
```

```
// Operator modulus atau (%) menghasilkan nilai sisa pembagian bilangan bulat
// Jika sisa pembagian adalah sama dengan 0, maka nilai tersebut habis dibagi dengan 2
// artinya bilangan tersebut adalah biangn bulat.
if(input % 2 == 0)
{
    Console.WriteLine("Nilai yang Anda masukkan adalah bilangan genap.");
}
else // sisa hasil pembagian tidak sama dengan 0, maka nilai yang dimasukkan adalah
bilangan ganjil.
{
    Console.WriteLine("Nilai yang Anda masukkan adalah bilangan ganjil");
}
```

10. Tekan tombol **CTRL+F5** untuk memulai aplikasi tanpa melakukan debugging. Atau, Anda bisa menggunakan Menu. Pilih **Debug -> Start Without Debugging**.
11. Visual Studio akan mengkompilasi kode pemrograman anda dan menjalankan aplikasi. Sebuah jendela konsol akan terbuka dan menampilkan hasil kompilasi apabila tidak terjadi eror.
12. Lakukan percobaan dengan nilai input yang berbeda untuk melihat seperti apa keluaran yang dihasilkan.

## Praktikum 3 : Menggunakan pernyataan switch

1. Buka Visual Studio
2. Pilih File, New, Project
3. Dari Templates section, pilih Visual C#
4. Pilih Console App
5. Beri nama project Anda, seperti Modul2\_ Praktik2
6. Pilih lokasi penyimpanan project Anda.
7. Klik tombol OK dan Visual Studio akan membuat sebuah proyek aplikasi konsol C# baru dan membuka Program.cs untuk Anda di jendela penyuntingan (*editor window*).
8. Arahkan kursor Anda tepat setelah kurung kurawal pembuka di method **Main**, kemudian tekan enter untuk membuat baris baru.
9. Masukkan kode pemrograman di bawah ini:

```
// Membuat blok pernyataan switch
Console.WriteLine("Ukuran Kopi: 1=short 2=tall 3=grande 4=venti");
Console.Write("Masukkan pilihan Anda: ");
string str = Console.ReadLine();
int harga = 0;

switch (str)
{
    case "1":
    case "short":
        harga += 30000;
```

```
        break;
    case "2":
    case "tall":
        harga += 35000;
        break;
    case "3":
    case "grande":
        harga += 40000;
        break;
    case "4":
    case "venti":
        harga += 45000;
        break;
    default:
        Console.WriteLine("Pilihan Anda tidak valid. Pilih 1, 2, 3, atau 4.");
        break;
}

if (harga != 0)
{
    Console.WriteLine("Silahkan masukkan Rp. {0} ", harga);
}

Console.WriteLine("Terima kasih atas pembelian Anda");
```

10. Tekan tombol **CTRL+F5** untuk memulai aplikasi tanpa melakukan debugging. Atau, Anda bisa menggunakan Menu. Pilih **Debug -> Start Without Debugging**.
11. Visual Studio akan mengkompil kode pemrograman anda dan menjalankan aplikasi. Sebuah jendela konsol akan terbuka dan menampilkan hasil kompilasi apabila tidak terjadi eror.

12. Lakukan percobaan dengan nilai input yang berbeda untuk melihat seperti apa keluaran yang dihasilkan.
13. Lakukan beberapa modifikasi untuk lebih memahami kode program di atas.

## Praktikum 4 : Menggunakan for loop

1. Buka Visual Studio
2. Pilih File, New, Project
3. Dari Templates section, pilih Visual C#
4. Pilih Console App
5. Beri nama project Anda, seperti Modul2\_ Praktikum3
6. Pilih lokasi penyimpanan project Anda.
7. Klik tombol OK dan Visual Studio akan membuat sebuah proyek aplikasi konsol C# baru dan membuka Program.cs untuk Anda di jendela penyuntingan (*editor window*).
8. Arahkan kursor Anda tepat setelah kurung kurawal pembuka di method **Main**, kemudian tekan enter untuk membuat baris baru.
9. Masukkan kode pemrograman di bawah ini:

```
// Membuat for loop sederhana yang menampilkan nilai dari pencacah
// Method WriteLine di sini mendemonstrasikan penggunaan interpolasi string ($) di C#
// sebagai cara untuk menghasilkan keluaran nilai string literal dengan menggunakan
// nilai variabel yang ditempatkan di tanda kurung kurawal.
// Ini adalah cara yang direkomendasikan untuk menghasilkan keluaran nilai string.
// Di praktikum modul 1, anda sudah mempraktekan cara menghasilkan keluaran string
// menggunakan placeholder atau string format.
for(int pencacah = 0; pencacah < 10; pencacah++)
{
    Console.WriteLine($"Nilai pencacah saat ini: {counter}");
}
```



10. Tekan tombol **CTRL+F5** untuk memulai aplikasi tanpa melakukan debugging. Atau, Anda bisa menggunakan Menu. Pilih **Debug -> Start Without Debugging**.
11. Visual Studio akan mengkompil kode pemrograman anda dan menjalankan aplikasi. Sebuah jendela konsol akan terbuka dan menampilkan hasil kompilasi apabila tidak terjadi galat (error).
12. Tutup jendela konsol dan kembali ke Visual Studio.
13. Masukkan kode program berikut untuk mengimplementasikan loop bersarang (*nested loop*).

```
// membuat loop bersarang (nested loop)
// Contoh ini menggunakan loop bersarang untuk menemukan bilangan prima
// yang kurang dari 100
int outer;
int inner;
for (outer = 2; outer < 100; outer++)
{
    for (inner = 2; inner <= (outer / inner); inner++)
        if ((outer % inner) == 0) break; // jika faktor ditemukan, bukan bilangan prima

    if (inner > (outer / inner))
        Console.WriteLine("{0} adalah bilangan prima", outer);
}
```

14. Tekan tombol **CTRL+F5** untuk memulai aplikasi tanpa melakukan debugging. Atau, Anda bisa menggunakan Menu. Pilih **Debug -> Start Without Debugging**.

15. Visual Studio akan mengkompil kode pemrograman anda dan menjalankan aplikasi. Sebuah jendela konsol akan terbuka dan menampilkan hasil kompilasi apabila tidak terjadi eror.

## Praktikum 5 : Menggunakan while dan do-while loop

1. Buka Visual Studio
2. Pilih File, New, Project
3. Dari Templates section, pilih Visual C#
4. Pilih Console App
5. Beri nama project Anda, seperti Modul2\_ Praktik4
6. Pilih lokasi penyimpanan project Anda.
7. Klik tombol OK dan Visual Studio akan membuat sebuah proyek aplikasi konsol C# baru dan membuka Program.cs untuk Anda di jendela penyuntingan (*editor window*).
8. Arahkan kursor Anda tepat setelah kurung kurawal pembuka di method **Main**, kemudian tekan enter untuk membuat baris baru.
9. Masukkan kode pemrograman di bawah ini:

```
// Membuat while loop
int n = 1;
while (n < 10)
{
    Console.WriteLine($"Nilai saat ini adalah {n}");
    n++;
}
```

10. Tekan tombol **CTRL+F5** untuk memulai aplikasi tanpa melakukan debugging. Atau, Anda bisa menggunakan Menu. Pilih **Debug -> Start Without Debugging**.
11. Visual Studio akan mengkompil kode pemrograman anda dan menjalankan aplikasi. Sebuah jendela konsol akan terbuka dan menampilkan hasil kompilasi apabila tidak terjadi error.
12. Tutup jendela konsol dan kembali ke Visual Studio.
13. Masukkan kode program berikut untuk mengimplementasikan *do-while* loop.

```
// Membuat do-while loop
// lakukan percobaan dengan memodifikasi nilai variable x
// amati setiap keluaran yang dihasilkan untuk bilai variabel x yang berbeda
int x = 0;
do
{
    Console.WriteLine(x);
    x++;
} while (x < 5);
```

14. Tekan tombol **CTRL+F5** untuk memulai aplikasi tanpa melakukan debugging. Atau, Anda bisa menggunakan Menu. Pilih **Debug -> Start Without Debugging**.
15. Visual Studio akan mengkompil kode pemrograman anda dan menjalankan aplikasi. Sebuah jendela konsol akan terbuka dan menampilkan hasil kompilasi apabila tidak terjadi error.

## Praktikum 6 : Mendeklarasikan dan menggunakan method

1. Buka Visual Studio
2. Pilih File, New, Project
3. Dari Templates section, pilih Visual C#
4. Pilih Console App
5. Beri nama project Anda, seperti Modul2\_Lab1
6. Pilih lokasi penyimpanan project Anda.
7. Klik tombol OK dan Visual Studio akan membuat sebuah proyek aplikasi konsol C# baru dan membuka Program.cs untuk Anda di jendela penyuntingan (*editor window*).
8. Arahkan kursor Anda tepat setelah kurung kurawal pembuka di method **Main**, kemudian tekan enter untuk membuat baris baru.
9. Masukkan kode pemrograman di bawah ini:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Mod3_Lab1
{
    class Program
    {
```

```
static void Main(string[] args)
{
    Penjumlahan(20, 40);
}

// Method Penjumlahan() method yang menerima dua argumen integer dan
// menjumlahkannya.
// Method ini tidak mengembalikan nilai apapun,
// itulah mengapa Anda menggunakan void.
// Anda juga harus menggunakan kata kunci static pada signature method
// karena Main adalah static.
// Anda tidak bisa memanggil non-static method dari sebuah static method.
// static method akan dipelajari di kursus pemrograman berbasis obyek dengan
// bahasa C#
static void Penjumlahan(int pertama, int kedua)
{
    int jumlah = pertama + kedua;
    Console.WriteLine($"Penjumlahan {pertama} dan {kedua} adalah: {jumlah}");
}
}
```

10. Tekan tombol **CTRL+F5** untuk memulai aplikasi tanpa melakukan debugging. Atau, Anda bisa menggunakan Menu. Pilih **Debug -> Start Without Debugging**.
11. Method yang Anda buat di atas tidak mengembalikan nilai apapun. Anda sekarang akan memodifikasi method tersebut supaya dapat mengembalikan sebuah nilai ke method pemanggilnya.
12. Modifikasi method Penjumlahan() menjadi seperti kode program berikut.

```
static int Penjumlahan(int pertama, int kedua)
{
```

```
    int jumlah = first + second;  
    return jumlah;  
}
```

13. Lalu modifikasi pemanggilan method Penjumlahan() di Main.

```
static void Main(string[] args)  
{  
    int hasil = Penjumlahan(20, 40);  
    Console.WriteLine($"Penjumlahan 20 dan 40 adalah {hasil}");  
}
```

14. Jalankan program dan amati nilai keluaran yang ditampilkan di jendela konsol.

15. Saat ini Anda sudah memiliki sebuah method yang mengembalikan nilai integer dan menerima dua argumen integer.

16. Tambahkan lagi dua method dengan nama yang sama, namun memiliki parameter yang berbeda untuk melakukan overloading terhadap method yang sudah Anda buat.

17. Method pertama mengembalikan nilai integer dan menerima tiga argumen integer. Method kedua mengembalikan nilai double dan menerima dua argumen double.

# **Anda Punya Pertanyaan Seputar Materi Di Buku Ini?**



Kirimkan pertanyaan Anda ke [dian.nandiwardhana@gmail.com](mailto:dian.nandiwardhana@gmail.com)