

Materi 1: Pengenalan .NET Framework

(namespace, class, metode, variabel, tipe data dan operator)

Silabus

Pertemuan	Materi
1	Pengenalan .NET Framework (namespace,class,method,variabel,tipe data dan operator)
2	Math ,String dan Percabangan (If...else , Switch Case)
3	Perulangan (While , Do While , For , For Each)
4	Array
5	Method
6	Form
7	Event
8	UTS
9	OOP (class & object,constructor,access modifier)
10	OOP (property,inheritance,polymorphism, enum)
11	Exception (try catch,finally,trow keyword)
12	Koneksi Database (Konsep MVC,koneksi database,create,read)
13	Koneksi Database (update dan delete)
14	Membuat tampilan responsive dan validasi
15	Persiapan UAS dan sharing-sharing
16	Projek

Daftar isi

- **Materi 1: Pengenalan .NET Framework (namespace, class, metode, variabel, tipe data dan operator)**
- Silabus
- Daftar isi
- Apa itu C#?
- .NET itu apa?
- Tujuan .NET Framework
 - 1. Common Language Runtime (CLR)
 - 2. Base Class Library (BCL)
- Sejarah Singkat C#
- Versi dan Perkembangan C#
- Memahami Sintaks Dasar Bahasa Pemrograman C#
 - 1. Struktur Dasar Program C#
 - 1. Bagian Deklarasi Pustaka
 - 2. Bagian Class
 - 3. Bagian Fungsi
 - 2. Penulisan Statement dan Ekspresi
 - 3. Penulisan Blok Kode
 - 4. Penulisan Komentar
 - 5. Gaya Case yang digunakan C#
 - 6. Penulisan String dan Angka
 - 7. Reserved Keywords
- Mengenal Fungsi Output dan Input pada C#
 - Fungsi Output pada C#
 - Tanda Petik Tunggal Vs Tanda Petik Ganda
 - Perbedaan `WriteLine()` dengan `Write()`
 - Fungsi Input pada C#
- Mengenal Variabel, Konstanta, dan Tipe Data
 - Apa itu Variabel dan Tipe Data?

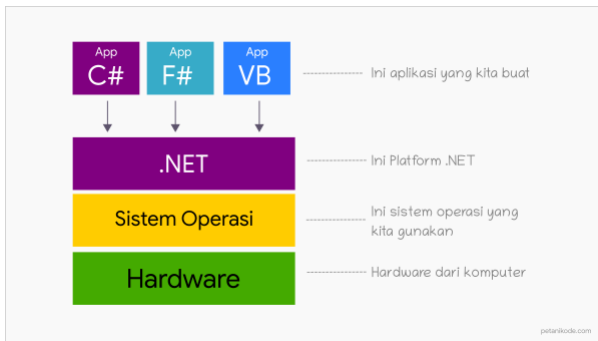
- [Macam-macam Tipe Data pada C#](#)
 - [Cara Membuat Variabel pada C#](#)
- [Konversi Tipe Data pada C#](#)
- [Mengenal Operator Dasar pada C#](#)
 - [1. Opearator Aritmatika](#)
 - [2. Operator Penugasan](#)
 - [3. Opearator Perbandingan](#)
 - [4. Operator Logika](#)
 - [6. Conditional Operator \(Ternary\)](#)

Apa itu C#?

C# (dibaca see sharp) adalah bahasa pemrograman yang dibuat oleh Microsoft dan ditargetkan berjalan di atas platform dotnet (.NET).

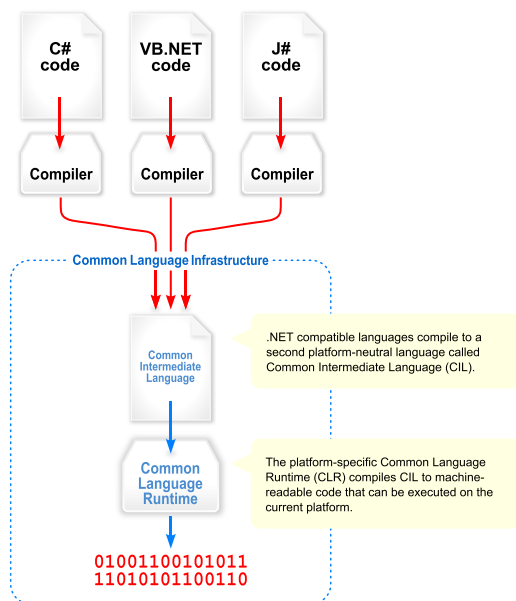
.NET itu apa?

.NET itu semacam mesin virtual yang tugasnya untuk menjalankan program C#, F#, VB.NET dan program lainnya. Selain itu, .NET juga menyediakan tools, library, dan API yang kita butuhkan untuk membuat program C#. Sehingga kadang .NET disebut juga .NET Framework.



Program C# tidak seperti program C dan C++ yang di-compile menjadi bahasa assembly dan bisa dieksekusi langsung oleh processor.

Program C# di-compile menjadi CIL (Common Intermediate Language). CIL adalah bahasa yang dipahami oleh .NET.



Semua program yang ingin dijalankan di atas .NET, haruslah di-compile menjadi CIL. Kalau tidak, ya tidak akan bisa dijalankan.

Jadi:

Tanpa adanya .NET, kita tidak akan bisa menjalankan program yang dibuat dengan C#.

Tujuan .NET Framework

- Menyediakan lingkungan pemrograman berorientasi objek atau Object Oriented Programming (OOP) yang konsisten meskipun kode objek disimpan dan dijalankan secara lokal tetapi dapat disebarkan melalui internet dan di jalankan secara remote (di jalankan dari suatu tempat).
- Menyediakan lingkungan untuk menjalankan suatu kode yang dapat mengeliminasi masalah performa dari lingkungan scripted (halaman).
- Menyediakan lingkungan untuk menjalankan suatu kode yang menjamin keamanan saat kode di jalankan, termasuk kode yang di buat oleh pihak yang tidak di ketahui/ pihak ketiga yang setengah dipercaya.

.NET Framework terdiri dari dua buah bagian utama, yaitu Common Language Runtime (CLR) dan Base Class Library (BCL).

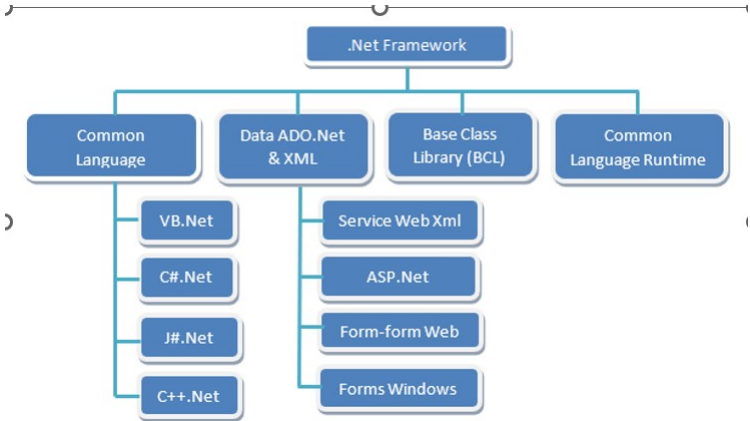
1. Common Language Runtime (CLR)

Adalah pondasi utama (bahasa umum) dari .NET Framework ,yang menjalankan aplikasi .NET Framework menyediakan sejumlah layanan pada berbagai hal seperti :

- Pengaturan Memori
- Mengelola kode (melakukan eksekusi kode)
- Melakukan verifikasi terhadap keamanan kode
- Menentukan hak akses dari kode, melakukan kompilasi kode, dan berbagai layanan system lainnya.

2. Base Case Library (BCL)

Bersifat berorientasi terhadap objek yang akan menyediakan jenis dari fungsi-fungsi pengaturan kode. Seperti telah disebutkan sebelumnya bahwa .Net Framework memilki beberapa bahasa pemrograman di dalamnya, termasuk VB.Net, maka struktur .Net Framewok paling utama dibangun oleh bahasa pemrograman .Net. Dapat rekan-rekan lihat di bawah ini untuk struktur dasar .Net Framework;



Sejarah Singkat C#

Pada tahun 1999, Anders Hejlsberg membentuk sebuah tim di Microsoft untuk membuat bahasa pemrograman baru yang diberi nama **Cool**.

Cool merupakan singkatan dari “*C-Like Object Oriented Language*”. Kalau dalam bahasa indonesia artinya: **Bahasa OOP yang mirip C** Microsoft ingin memepertahankan nama *Cool*, tapi ini tidak bisa dilakukan karena bisa melanggar *treadmark* dari produk lain.

Akhirnya pada tahun 2000 pada acara Preoessional Developer Conference (PDC), nama **Cool** diubah menjadi **C#**. Nama **C#** sendiri diambil dari notasi musik yakni C#. Jika kamu paham notasi musik pasti paham arti tanda # setelah C

Inilah asal usul dari **C#**. Kalau kita lihat dari sisi sintaks, bahasa **C#** banyak terinspirasi dari bahasa **C**, **C++**, dan **Java**. Ini membuat **C#**, mendapat banyak kritik..

Meskipun banyak yang kurang suka dengan **C#**, namun **C#** masih banyak digunakan hingga saat ini.

Contohnya:

Bahasa **C#** lebih direkomendasikan untuk membuat Game dibandingkan bahasa **Java**. Bahkan tidak hanya di Game saja, **C#** juga banyak dipakai untuk membuat aplikasi desktop dan Web.

Versi dan Perkembangan C#

Walau di tahun 2000 sudah diumumkan nama C#, tapi pada tahun tersebut C# belum resmi dirilis. Versi pertama C# resmi dirilis pertama kali pada tahun 2002.

Berikut ini versi C# dan tahun rilisnya:

- C# 1.0 (January 2002) – .NET Framework 1.0
- C# 1.1 (April 2003) – .NET Framework 1.1
- C# 1.2 (April 2003) – .NET Framework 1.1
- C# 2.0 (November 2005) – .NET Framework 2.0, .NET Framework 3.0
- C# 3.0 (Agustus 2007) – .NET Framework 2.0 (Except LINQ), .NET Framework 3.0 (Except LINQ), .NET Framework 3.5
- C# 4.0 (April 2010) – .NET Framework 4
- C# 5.0 (Agustus 2012) – .NET Framework 4.5
- C# 6.0 (Juli 2015) – .NET Framework 4.6, .NET Core 1.0, .NET Core 1.1
- C# 7.0 (Maret 2017) – .NET Framework 4.7
- C# 7.1 (Agustus 2017) – .NET Core 2.0
- C# 7.2 (November 2017) – .NET Core 2.0
- C# 7.3 (Mei 2018) – .NET Core 2.1, .NET Core 2.2, .NET Framework 4.8
- C# 8.0 (September 2019) – .NET Core 3.0
- C# 9 (November 2020) - .NET 5
- C# 10 (November 2021) - .NET 6
- C# 11 (November 2022) - .NET 7

Versi .NET yang digunakan tiap-tiap versi berbeda. Jangan gunakan versi .NET lama pada versi C# terbaru.

Misalnya:

Kita buat program menggunakan C# 8.0. Lalu di komputer kita install .Net Framework 1.0. Jelas ini tidak akan bisa.

Memahami Sintaks Dasar Bahasa Pemrograman C#

1. Struktur Dasar Program C#

Struktur program C# yang paling dasar, terdiri dari tiga bagian:

1. Bagian deklarasi pustaka
2. Bagian Class
3. Bagian Fungsi atau Method

Contoh:

```
// 1. Deklarasi pustaka
using System;

// 2. Bagian Class
class ProgramHello
{
    // 3. Bagian Fungsi
    static void Main(String[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

Penjelasannya:

1. Bagian Deklarasi Pustaka

Ini adalah bagian paling atas dari program C#. Pada bagian ini, kita menuliskan pustaka (library) yang dibutuhkan dalam program.

Apa itu pustaka (library) ?

Pustaka berisi sekumpulan fungsi, method, class, objek, konstanta, dan variabel yang bisa kita gunakan ulang di dalam program.

Sebagai contoh:

Pada contoh di atas, kita menggunakan pustaka `System.`

```
using System;
```

Pustaka ini kita butuhkan untuk menggunakan class Console dan method `WriteLine()`;

2. Bagian Class

Bahasa pemrograman C# adalah bahasa pemrograman yang menggunakan paradigma OOP (Object Oriented Programming) atau pemrograman berorientasikan objek.

Setiap kita membuat program C#, kita harus membungkus fungsi dan variabel di dalam class.

“Class adalah sebuah rancangan atau blue print dari objek.”

Pengertian ini mungkin membingungkan bagi pemula. Apalagi yang belum pernah belajar OOP.

Karena itu...

Cukup pahami class itu sebagai nama program.

Contoh:

```
using System;

class ProgramHello
{
    static void Main(String[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

Berarti nama program di atas adalah `ProgramHello` .

Nah, di dalam class...

...kita bisa isi dengan fungsi dan variabel.

3. Bagian Fungsi

Pada bagian ini, kita bisa menuliskan fungsi-fungsi dari program.

Fungsi yang harus ada di dalam setiap program adalah fungsi `Main()`.

Kalau tidak ada fungsi ini, program tidak akan bisa dijalankan. Karena fungsi `Main()` merupakan fungsi utama yang akan dieksekusi pertama kali.

Oleh sebab itu, kita biasanya akan banyak menulis kode program di dalam fungsi `Main()`.

```
using System;

class ProgramHello
{
    static void Main(String[] args)
    {
        // kode program di tulis di sini
        Console.WriteLine("Hello World!");
    }
}
```

2. Penulisan Statement dan Ekspresi

Penulisan statemen dan ekspresi dalam C# harus diakhiri dengan titik koma `;`.

Kalau tidak...

...maka nanti akan error.

Contoh:

```
using System;

class ProgramHello
{
    static void Main(String[] args)
    {
        // stetmen dan ekspresi bisa ditulis di sini
        Console.WriteLine("Hello World!");
        Console.WriteLine("Hello Indonesia!");
        Console.WriteLine("Hello Lombok!");
        Console.WriteLine("Saya belajar C#!");
    }
}
```

3. Penulisan Blok Kode

Blok kode di dalam C# dibungkus menggunakan kurung kurawal `{ ... }`.

Contoh:

```
if(password == "petanikode")
{
    // ini blok kode IF
    Console.WriteLine("Password Benar!");
    Console.WriteLine("Selamat Datang");
}
```

Biasanya blok kode digunakan untuk membungkus lebih dari satu statement. Jika statement hanya ada satu baris, maka bisa kita tidak bungkus dengan tanda kurung kurawal.

Contoh:

```
if(password == "petanikode")
    Console.WriteLine("Selamat datang");
```

4. Penulisan Komentar

Komentar adalah teks yang tidak akan dieksekusi. Biasanya digunakan untuk menuliskan keterangan dan menon-aktifkan sebuah kode.

Komentar dalam C# ditulis dengan garis miring ganda dan garis miring bintang.

Contoh:

```
// ini komentar

// ini juga komentar

/*
    Komentar yang lebih dari
    satu baris, biasanya
    ditulis menggunakan garis miring
    dan bintang seperti ini
*/
```

5. Gaya Case yang digunakan C#

Bahasa pemrograman C# menggunakan **Pascal Case dan Camel Case**.

Untuk penulisan nama Class dan Fungsi atau Method, selalu diawali dengan huruf kapital.

Contoh:

```
// penulisan nama class
class ProgramCoba
{
    // penulisan nama fungsi atau method
    void NamaFungsi()
    {
        Console.WriteLine("Lakukan sesuatu");
    }
}
```

Apabila nama fungsi dan class terdiri dari dua suku kata, maka ditulis bergabung seperti di atas.

Apa boleh menggunakan huruf kecil di depan?

Ya... boleh-boleh saja.

Tapi, itu akan melanggar *garis pandu (guidelines)* yang sudah ditetapkan Microsoft.

Kemudian untuk nama variabel, ditulis dengan huruf kecil di depan.

Contoh:

```
var nama = "Abdul Somad";
var namaVariabel = "Welcome";
```

Jangan seperti ini:

```
var Nama = "Abdul Somad";
var nama_variabel = "Welcome";
```

6. Penulisan String dan Angka

String atau teks, biasanya ditulis dengan diapit tanda petik. Sedangkan angka dan tipe data boolean, itu tidak ditulis dengan tanda petik.

Contoh:

```
"Ini adalah sebuah string"  
121 // ini angka  
13.3 // ini juga angka  
true // ini boolean
```

7. Reserved Keywords

Reserved Keywords adalah kata kunci yang tidak boleh digunakan sebagai nama variabel.

Pada C#, terdapat beberapa reserved keywords:

```
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      int     interface  internal  
is      lock   long    namespace  
new     null   object  operator  
out     override      params  private  
protected      public  readonly      ref  
return  sbyte   sealed  short  
sizeof  stackalloc      static  string  
struct  switch  this    throw  
true    try     typeof  uint  
ulong   unchecked      unsafe  ushort  
using   using static  virtual      void  
volatile      while
```


Mengenal Fungsi Otput dan Input pada C#

Fungsi Output pada C#

Pertama kita bahas dulu tentang fungsi output, karena lebih mudah dipahami.

Baiklah...

Ini adalah fungsi untuk menampilkan output pada C#:

```
Console.Write("Teks yang akan ditampilkan ke layar");
```

Perhatikan cara penulisannya.. Huruf C dan W menggunakan huruf besar. Kemudian teks yang akan ditampilkan dilayar harus diapit dengan tanda petik.

Memangnya kenapa harus pakai tanda petik?

Kalau tidak diapit dengan tanda petik, dia akan dianggap variabel.

Contoh yang salah:

```
console.write(saya belajar c#);
```

Jika kamu menulisnya seperti ini, dijamin program akan error 😊.

Oke sekarang.. Mari kita coba dalam program.

Buatlah program dengan kode seperti ini: ProgramOutput.cs

```
using System;

public class ProgramOutput
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Selamat Datang di Pemrograman C#");
        Console.WriteLine("Ini adalah tutorial belajar C# dari dasar");
        Console.WriteLine("Terimakasih sudah menggunakan program ini");
    }
}
```

Setelah itu, kompilasi program tersebut dan jalankan.

Tanda Petik Tunggal Vs Tanda Petik Ganda

Kita wajib mengapit teks yang ingin kita tampilkan ke layar dengan tanda petik.

Nah ada dua tanda petik yang bisa digunakan, yakni tanda petik tunggal ('...') dan tanda petik ganda ("...").

Apa bedanya?

Tanda petik tunggal biasanya digunakan untuk menampilkan satu huruf atau karakter saja.

Contoh:

```
Console.WriteLine('D');
```

Sementara tanda petik ganda digunakan untuk menampilkan lebih dari satu huruf, atau bisa kita sebut dengan teks.

Contoh:

```
Console.WriteLine("Bismillah menjadi web developer yang amanah.Aamiin");
```

Jika kamu mencoba menggunakan tanda petik tunggal untuk menampilkan teks yang panjang, maka program akan error dengan pesan error:

```
Too many characters in character literal
```

Artinya terlalu banyak karakter di dalam tanda petik tunggal.

Karena itu...

Pastikan kamu menggunakan tanda petik ganda untuk teks dan tunggal untuk huruf atau karakter.

Perbedaan WriteLine() dengan Write()

WriteLine() dan Write() merupakan dua fungsi yang bisa kita gunakan untuk menampilkan teks atau output ke layar.

Lalu apa bedanya?

...dan kapan saya harus menggunakan WriteLine() dan Write() ?

Fungsi `WriteLine()` akan menampilkan teks dalam satu baris atau baris baru, sedangkan `Write()` tidak akan membuat baris baru.

Untuk lebih jelasnya...

Mari kita coba membuat program dengan kedua fungsi tersebut.

Silahkan buat program baru bernama Biodata.cs, kemudian isi dengan kode berikut:

```
using System;

class Biodata
{
    static void Main(String[] args)
    {
        Console.WriteLine("=== BIODATA SAYA ===");
        Console.Write("Nama: ");
        Console.Write("Dian");
        Console.WriteLine();
        Console.WriteLine("Alamat: Jakarta");
    }
}
```

Jadi kapan kita harus menggunakan `Write()` dan `WriteLine()`?

Gunakanlah fungsi `Write()` saat kita tidak ingin membuat baris baru, dan `WriteLine()` saat kita ingin membuat baris baru.

Oke sekarang kita lanjut ke fungsi input...

Fungsi Input pada C#

Sekarang untuk input ada fungsi `Read()`, `ReadKey()` dan `ReadLine()`.

Fungsi `ReadLine()` akan membaca teks yang kita ketik dalam satu baris (teks).

Sedangkan Fungsi `Read()` dan `ReadKey()` akan membaca satu huruf saja dari teks yang kita ketik. `Read()` akan menghasilkan tipe data `int` sedangkan `ReadKey()` akan menghasilkan data dengan tipe `character`.

Bingung?

Tenang...

Nanti kita akan bahas lebih dalam tentang variabel dan tipe data pada artikel yang berbeda.

Untuk saat ini, pastikan kamu memahami konsep input dan output pada program.

Sekarang mari kita langsung praktek!

Buatlah program baru bernama ProgramInput.cs, kemudian isi dengan kode berikut:

```
using System;

class ProgramInput
{
    static void Main(String[] args)
    {
        Console.WriteLine("=== PROGRAM KEREN ===");
        Console.Write("Tuliskan nama kamu: ");
        string nama = Console.ReadLine();
        Console.WriteLine("Hi, {0} selamat datang!", nama);
    }
}
```

Perhatikan kode program di atas...

Di sana kita menggunakan variabel nama untuk menyimpan teks yang diinputkan.

```
string nama = Console.ReadLine();
```

Kemudian menampilkannya dengan cara seperti ini:

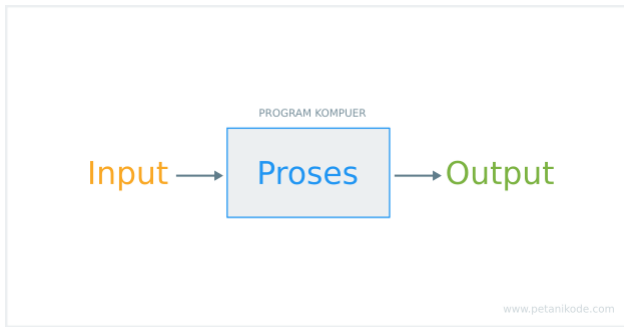
```
Console.WriteLine("Hi, {0} selamat datang!", nama);
```

Mengapa sih kita harus menggunakan variabel?

...dan jelaskan maksud dari tanda `{0}` pada fungsi `WriteLine()`?

Ingat kembali tentang konsep dasar program yang sudah dijelaskan di awal.

Di sana ada input, proses, dan output.



Variabel di sini berperan sebagai penampung data yang akan kita proses dalam program.

Pada contoh program di atas, kita memang belum melakukan proses apapun. Tapi tetap, akan membutuhkan variabel karena kita akan menampilkannya ke layar.

Lalu untuk simbol `{0}` ini adalah sebuah placeholder yang nantinya akan digantikan dengan isi dari variabel.

Sebenarnya kita bisa menggunakan simbol `+` juga seperti ini:

```
Console.WriteLine("Hi, " + nama + " selamat datang!");
```

Namun, saya kira menggunakan placeholder `{}` akan lebih mudah dibaca.

Selain penulisan dengan cara di atas, kita juga bisa menulisnya dengan simbol `$` seperti ini:

```
Console.WriteLine($"Hi, {nama} selamat datang!");
```

Oh iya, teknik ini disebut **string interpolation**.

Akhir Kata... Kita sudah belajar bagaimana konsep dasar program komputer. Ada input, proses, dan output.

Nah untuk fungsi input dan output pada C#, saya harap sudah paham.

Selanjutnya untuk bagian proses, kita harus memahami vairabel, operator, percabangan, perulangan, dll.

Mengenal Variabel, Konstanta, dan Tipe Data

Apa itu Variabel dan Tipe Data?

Saat kita menggunakan fungsi input, kita membutuhkan variabel untuk menyimpan data yang dimasukan ke program.

Itulah fungsi utama variabel.

Jadi:

Variabel adalah sebuah **wadah penyimpanan** data pada program yang akan digunakan selama program itu berjalan.

Data atau nilai yang kita simpan dalam variabel, akan disimpan dalam memori (RAM). Semakin banyak variabel yang dibuat, semakin banyak pula ruang memori yang dibutuhkan.

Secara teknis, variabel bisa kita sebut sebagai nama tempat untuk menggantikan alamat memori.

Sementara itu...

Tipe data adalah **jenis-jenis data** yang akan disimpan di dalam variabel. Seperti data teks, angka, huruf, dll.

Pada C#, ada beberapa jenis tipe data yang umum digunakan.

Apa saja itu?

Ini dia..

Macam-macam Tipe Data pada C#

Tipe data pada C# itu banyak, tapi tidak semuanya harus digunakan.

Karena itu, kamu tidak perlu menghafal semuanya.

Berikut ini beberapa jenis tipe data yang sering digunakan:

- `string` adalah tipe data yang berupa teks, contoh: `"Saya hebat!"`;
- `int` adalah tipe data yang berupa angka, Contoh: `3, 9, 0`;
- `float` adalah tipe data yang berupa angka pecahan, contoh `1.2f, 2.4f, 5.5f` (huruf f artinya float);
- `bool` adalah tipe data boolean yang hanya berisi `true` dan `false`.

Saran saya untuk yang baru belajar pemrograman, cukup ingat yang empat itu saja dulu. Untuk tipe data yang lainnya bisa dipelajari nanti.

Tapi jika kamu memang sudah sering belajar pemrograman, tentu tidak akan asing dengan semua tipe data ini:

Type	Represents	Range	Default Value
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	$(-7.9 \times 10^{28} \text{ to } 7.9 \times 10^{28}) / 10^0 \text{ to } 28$	0.0M
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324} \text{ to } (+/-)1.7 \times 10^{308}$	0.0D
float	32-bit single-precision floating point type	$-3.4 \times 10^{38} \text{ to } +3.4 \times 10^{38}$	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0
long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

Tabel di atas adalah semua tipe data primitif yang ada di C#. Lengkap dengan range dan nilai default-nya.

Sebenarnya masih ada tipe data yang lain lagi, yakni tipe data reference seperti objek, pointer, array, struct, enum, dll.

Oh iya, beda tipe data, beda juga ukurannya.

Karena itu, pastikan menggunakan tipe data yang tepat, agar hemat memori.

Lalu bagaimana sih cara kita menggunakan tipe data?

Mari kita bahas dalam:

Cara Membuat Variabel pada C#

Ada dua cara membuat variabel di C#:

Pertama (eksplisit), dengan menuliskan tipe data lalu diikuti nama variabel.

```
[tipe data] namaVariabel;
```

Contoh:

```
// membuat variabel kosong
string alamat;
int umur;
float beratBadan;
bool isMenikah;

// membuat variabel dan langsung mengisinya
string nama = "Petani Kode";
int umur = 18;
float beratBadan = 59.34;
```

Kedua (implisit), apabila kita tidak tahu tipe data yang akan digunakan, maka membuat variabel bisa menggunakan kata kunci var.

```
var namaVariabel = "isi variabel";
```

Pembuatan variabel dengan var harus kita isi nilainya, karena kalau tidak.. akan terjadi error seperti ini:

```
An implicitly typed local variable declarator must include an initializer
```

Contoh:

```
// membuat variabel dan langsung mengisinya
var namaWeb = "GitHub";
var alamatWeb = "https://github.com/Sawaluddin-JR";
var bahasaPemrograman = "C#";
var umur = 20;
var isMenikah = false;
```

Cara manakah yang kamu suka? Kalau saya sih suka kedua-duanya. 😊

Nah sekarang mari kita membuat program dengan memanfaatkan variabel dan tipe data.

Buatlah program dengan nama `PendaftaranPenduduk.cs`, kemudian isi dengan kode berikut:

```
using System;

class PendaftaranPenduduk
{
```



```
static void Main(String[] args)
{
    // membuat variabel kosong
    string nama;
    int umur;

    Console.WriteLine("=== PROGRAM PENDAFTARAN PENDUDUK ===");
    Console.Write("Masukan nama: ");
    nama = Console.ReadLine();
    Console.Write("Masukan alamat: ");
    var alamat = Console.ReadLine();
    Console.Write("Masukan umur: ");
    umur = int.Parse(Console.ReadLine());

    Console.WriteLine();
    Console.WriteLine("Terimakasih!");
    Console.WriteLine("Data Berikut");
    Console.WriteLine($"Nama: {nama}");
    Console.WriteLine($"Alamat: {alamat}");
    Console.WriteLine($"Umur: {umur} tahun");
    Console.WriteLine("SUDAH DISIMPAN!");
}
}
```

Perhatikanlah kode program di atas...

Pada program tersebut, kita membuat variabel `nama`, `umur`, dan `alamat` dengan tipe data yang berbeda-beda.

Ketiga variabel ini akan diisi dengan nilai input yang kita berikan melalui keyboard.

Sekarang coba perhatikan dibagian pengisian variabel umur.

```
umur = int.Parse(Console.ReadLine());
```

Mengapa kita menggunakan `int.Parse()` di sini?

Ini karena fungsi `Console.ReadLine()` akan menghasilkan data dengan tipe string. Oleh sebab itu, kita harus mengubahnya menjadi **integer** dengan fungsi `int.Parse()` agar dapat disimpan di dalam variabel umur.

Kalau tidak diubah, maka akan terjadi *error*:

```
Cannot implicitly convert type `string' to `int'
```

Nah, konversi tipe data ini, kita akan bahas nanti ya.

Yang penting sekarang sudah paham cara membuat variabel di C#.

Oh iya, dalam membuat nama variabel...








...ada beberapa aturan yang harus diikuti:

Aturan Membuat Nama Variabel pada C# Jujur saja... membuat nama variabel kadang memakan waktu. Karena banyaknya aturan yang harus diikuti.

Kalau tidak diikuti, memangnya kenapa?

Bisa jadi program akan error dan kode program akan sulit dibaca dan dipahami.

Karena itu, silahkan ikuti aturan berikut:

-  Jangan menggunakan angka di awal nama variabel. Contoh: 7eleven, 1cak, 99design, dll. Tapi menggunakan angka di tengah dan diakhir dibolehkan kok.
-  Jangan menggunakan simbol. Contoh: ~sender, !apa, %level, sec()re. Tapi menggunakan underscore dibolehkan kok, contoh nama_lengkap, _jenisKelamin.
-  Jangan menggunakan nama dengan kata kunci yang sudah ada di C#, contoh: for, if, double, class, dll. Untuk daftar semua kata kunci yang dilarang dapat kamu lihat di <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/>
-  Dianjurkan menggunakan cameCase style, contoh: namaLengkap, jenisKelamin. Contoh yang bukan camelCase: nama_lengkap, jenis_kelamin. Tujuannya agar konsisten dan mudah dibaca.
-  Dianjurkan menggunakan bahasa inggris. Pakai bahasa indonesia juga boleh kok, tapi nanti kalau sudah bekerja dengan tim sebaiknya pakai bahasa inggris.
-  Dianjurkan menggunakan nama variabel yang mewakili dari isi variabel.
-  Nama variabel bersifat case sensitive. Artinya huruf besar dan kecil dibedakan, Contoh: namaLengkap dan namalengkap adalah dua variabel yang berbeda.

Nah, itulah beberapa aturan penulisan nama variabel...

Jika ingin program tidak error, maka ikuti aturan yang ke 1,2, 3, dan 7. Dan jika ingin kode programmu mudah dibaca, maka ikutilah aturan 4, 5, dan 6.

Bagaimana, mudah bukan?

Konversi Tipe Data pada C#

Konversi atau mengubah tipe data kadang sering kita lakukan, karena variabel dengan tipe tertentu tidak akan bisa menyimpan data dengan tipe yang berbeda dengannya.

Ada beberapa cara:

- Implicit Casting (secara otomatis) - mengonversi tipe yang lebih kecil ke ukuran tipe yang lebih besar char-> int-> long-> float->double
- Explicit Casting (secara manual) - mengonversi tipe yang lebih besar ke tipe ukuran yang lebih kecil double-> float-> long-> int->char

```
// implicit
int myInt = 9;
double myDouble = myInt;           // Automatic casting: int to double

Console.WriteLine(myInt);          // Outputs 9
Console.WriteLine(myDouble);       // Outputs 9
```

```
// eksplisit
double myDouble = 9.78;
int myInt = (int) myDouble;        // Manual casting: double to int

Console.WriteLine(myDouble);       // Outputs 9.78
Console.WriteLine(myInt);          // Outputs 9
```

Selain menggunakan tanda kurung (tipe data), kita juga bisa menggunakan fungsi helper `Parse()`.

Contoh:

```
int score = int.Parse("10");
```

Type Conversion Methods Dimungkinkan juga untuk mengonversi tipe data secara eksplisit dengan menggunakan metode bawaan, seperti `Convert.ToBoolean`, `Convert.ToDouble`, `Convert.ToString`, `Convert.ToInt32(int)` dan `Convert.ToInt64(long)`:

```
// contohnya
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;
```

```
Console.WriteLine(Convert.ToString(myInt));    // convert int to string
Console.WriteLine(Convert.ToDouble(myInt));    // convert int to double
Console.WriteLine(Convert.ToInt32(myDouble));  // convert double to int
Console.WriteLine(Convert.ToString(myBool));   // convert bool to string
```

Satu Hal Lagi.. Konstanta itu apa?

Konstanta hampir sama seperti variabel, bedanya konstanta bersifat immutable artinya nilainya tidak bisa diisi ulang alias konstan.2

Ya namanya juga konstanta...

Lalu cara membuat konstanta di C# bagaimana?

Gampang! hanya menggunakan kata kunci `const` , lalu diikuti dengan tipe data dan nilainya.

Contoh:

```
const string NamaKonstanta = "isi atau nilai";
```

Oh iya, untuk nama konstanta sendiri.. kita dianjurkan menggunakan huruf kapital di awal namanya.

Ini untuk membedakannya dengan variabel.

Jika di variabel kita menggunakan huruf kecil di awal, maka di konstanta disarankan menggunakan huruf besar.

Paham?

Bagus 👍

...dan jangan pernah untuk mencoba mengisi ulang nilai konstanta, karena akan error.

```
// membuat konstanta
const double Phi = 3.14;

// mengisi ulang (jangan pernah lakukan ini)
Phi = 5.32
```

Contoh program menggunakan konstanta: `LuasLingkaran.cs`

```
using System;
```

```
class LuasLingkaran
{
    static void Main(String[] args)
    {
        const float Phi = 3.14f;

        Console.WriteLine("== PROGRAM LUAS LINGKARAN ==");
        Console.Write("Input jari-jari: ");
        int r = int.Parse(Console.ReadLine());

        var luas = Phi * r * r;

        Console.WriteLine($"Luas Lingkaran = {luas}");
    }
}
```

Mengenal Operator Dasar pada C#

Apa itu Operator? Operator adalah sebuah simbol...

Simbol yang digunakan untuk melakukan operasi tertentu.

Misalnya:

Kita ingin menjumlahkan nilai dari variabel `x` dan `y`, maka kita bisa menggunakan operator penjumlahan `(+)`.

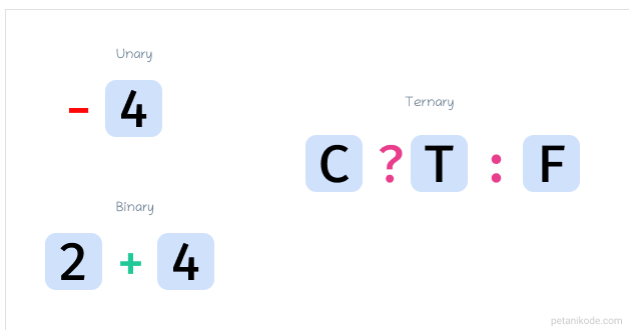
```
x + y
```

`x` dan `y` disebut operand, sedangkan `+` disebut operator.

Paham kan?

Nah, berdasarkan banyaknya operand.. operator dikelompokkan menjadi tiga macam.

Yakni: Unary, Binary, dan Ternary.



Lalu berdasarkan operasi yang dilakukan, operator dibagi lagi menjadi beberapa bagian.

Ada yang namanya operator aritmatika, logika, perbandingan, pengisian, dan bitwise.

Jenis-Jenis Operator di C#

C# memiliki banyak jenis operator, namun yang akan kita bahas di sini adalah operator dasar yang sering digunakan.

Diantaranya:

- Operator Aritmatika
- Operator Penugasan (Assignment)
- Operator Perbandingan (Comparison)

- Operator Logika (Boolean Logical)
- Operator Kondisional (Ternary)

Mari kita bahas satu-persatu..

1. Opeartor Aritmatika

Operator aritmatika adalah operator untuk melakukan operasi aritmatika seperti kali, bagi, tambah, kurang.

Operator ini terdiri dari:

Nama Operator	Simbol
Penjumlahan	+
Pengurangan	-
Perkalian	*
Pembagian	/
Sisa Bagi	%
Increment	++
Decrement	--

Biar lebih jelas, mari kita coba dalam contoh.

Pertama kita akan coba operator penjumlahan.

Silahkan buat program baru bernama `AritmatikaPenjumlahan.cs` , dengan kode sebagai berikut:

```
using System;

class AritmatikaPenjumlahan
{
    public static void Main (string[] args)
    {
        int mangga, apel, hasil = 0;

        Console.Write("mangga = ");
        mangga = int.Parse(Console.ReadLine());
        Console.Write("apel = ");
        apel = int.Parse(Console.ReadLine());

        // operasi penjumlahan dengan operator +
```

```
    hasil = mangga + apel;  
  
    Console.WriteLine($"Hasil mangga + apel = {hasil}");  
}  
}
```

Pada contoh ini, kita memberikan input untuk variabel mangga adalah 5 dan apel adalah 7, maka 5+7 hasilnya akan 12.

Gampang kan?

Operasi penjumlahan dapat kita lakukan untuk tipe data numerik atau angka. Jika kita melakukan penjumlahan pada tipe data string, maka yang terjadi adalah penggabungan.. bukan penjumlahan.

Contoh:

```
string nama = "Petani" + "Kode"
```

Maka variabel nama akan berisi PetaniKode.

Paham kan..?

Nah, berikutnya.. coba juga untuk menggunakan operator yang lainnya.

Mari kita coba untuk pengurangan.

Buatlah program baru dengan nama AritmatikaPengurangan.cs, kemudian isi dengan kode berikut:

```
using System;  
  
class AritmatikaPengurangan  
{  
    public static void Main (string[] args)  
    {  
        int mangga, apel, hasil = 0;  
  
        Console.Write("mangga = ");  
        mangga = int.Parse(Console.ReadLine());  
        Console.Write("appel = ");  
        apel = int.Parse(Console.ReadLine());  
  
        hasil = mangga - apel;
```



```
        Console.WriteLine ("Hasil mangga - apel = {hasil}");  
    }  
}
```

Wohoo.. gampang!

Oh iya, selain pengurangan.. tanda `-` juga berfungsi sebagai tanda minus.

Cara penggunaannya sama seperti pada matematika.

Contoh:

```
int a = -10;  
int b = -(-6);
```

Oke..

Sekarang lanjut ke operator aritmatika berikutnya, yakni perkalian.

Buatlah program baru dengan nama `AritmatikaPerkalian.cs`, kemudian isi dengan kode berikut.

```
using System;  
  
class AritmatikaPerkalian  
{  
    public static void Main (string[] args)  
    {  
        int mangga, apel, hasil = 0;  
  
        Console.Write("mangga = ");  
        mangga = int.Parse(Console.ReadLine());  
        Console.Write("apel = ");  
        apel = int.Parse(Console.ReadLine());  
  
        hasil = mangga * apel;  
  
        Console.WriteLine ("Hasil mangga * apel = {hasil}");  
    }  
}
```

Nah.. untuk perkalian di pemrograman C#, kita pakai simbol `*` (asterik atau bintang).

Mengapa tidak pakai `x` ?

x sendiri merupakan sebuah huruf, yang akan dianggap tipe data karakter.. bukan operator kali. Jadi di perkalian di bahasa pemrograman pakainya * bukan x .

Oke, berikutnya pembagian..

Buatlah program baru dengan nama AritmatikaPembagian.cs dengan kode sebagai berikut.

```
using System;

class AritmatikaPembagian
{
    public static void Main (string[] args)
    {
        int mangga, orang, hasil = 0;

        Console.WriteLine("jumlah mangga = ");
        mangga = int.Parse(Console.ReadLine());
        Console.WriteLine("jumlah orang = ");
        orang = int.Parse(Console.ReadLine());

        hasil = mangga / orang;

        Console.WriteLine ($"Hasil mangga / orang = {hasil}");
    }
}
```

Sama seperti perkalian, untuk pembagian.. kita menggunakan simbol / (slah atau garis miring). Bukan simbol bagi ÷ atau : seperti pada kalkulator.

Operator ini akan menghasilkan sisa dari hasil pembagian, misalnya 10 bagi 3 maka sisanya 1 .

Biar lebih jelas, langsung saja kita coba dalam program..

Buatlah program baru dengan nama AritmatikaSisabagi.cs dengan kode sebagai berikut:

```
using System;

class AritmatikaPembagian
{
    public static void Main (string[] args)
    {
        int mangga, orang, hasil = 0;
```

```
Console.Write("jumlah mangga = ");
mangga = int.Parse(Console.ReadLine());
Console.Write("jumlah orang = ");
orang = int.Parse(Console.ReadLine());

hasil = mangga % orang;

Console.WriteLine($"Hasil mangga % orang = {hasil}");
}
}
```

Nah, terakhir kita akan coba operator **Increment dan Decrement**.

- *Increment* adalah penambahan nilai **+1** sedangkan
- *Decrement* adalah pengurangan nilai **-1**.

Biasanya, kita melakukannya seperti ini:

```
int mangga = 2;
mangga = mangga + 1; // increment
```

Nah, dengan operator Increment/Decrement.. kita bisa buat lebih singkat.

```
int mangga = 2;
mangga++; // increment
```

Mari kita coba contohnya..

Buatlah program baru dengan nama **IncrementDecrement.cs**, kemudian isi dengan kode berikut.

```
using System;

class IncrementDecrement
{
    public static void Main (string[] args)
    {
        int mangga = 3;
        int apel = 4;

        Console.WriteLine($"mangga = {mangga}");
    }
}
```

```
Console.WriteLine($"apel = {apel}");

// increment
mangga++;
++apel;

Console.WriteLine($"mangga+1 = {mangga}");
Console.WriteLine($"apel+1 = {apel}");

// decrement
mangga--;
--apel;

Console.WriteLine($"mangga-1 = {mangga}");
Console.WriteLine($"apel-1 = {apel}");
}
}
```

contoh increment decrement Operator increment dan decrement dapat ditulis di depan maupun belakang operan/variabel.

Penulisan di depan disebut pre-increment atau pra-increment, sedangkan penulisan di belakang disebut post-increment atau pasca-increment.

```
mangga++; // post-increment
++apel; // pre-increment
```

Nantinya kita akan banyak menggunakan operator increment dan decrement pada perulangan.

2. Operator Penugasan

Operator penugasan (Assignment Operator) merupakan operator untuk memberikan tugas pada variabel. Biasanya digunakan untuk mengisi nilai.

Operator Penugasan terdiri dari:

Nama Operator	Simbol
Pengisian Nilai	=
Pengisian dan Penambahan	+=
Pengisian dan Pengurangan	-=
Pengisian dan Perkalian	*=

Nama Operator	Simbol
Pengisian dan Pembagian	/=
Pengisian dan Sisa bagi	%=

Biar lebih jelas, mari kita coba dalam contoh program.

Buatlah program baru dengan nama `ProgramPenugasan.cs` dengan kode sebagai berikut:

```
using System;

class ProgramPenugasan
{
    public static void Main(String[] args)
    {
        // menggunakan operator = untuk mengisi nilai
        int mangga = 10;
        int apel = 8;

        // mengisi ulang nilai variabel mangga
        mangga = 15;

        Console.WriteLine($"mangga = {mangga}");

        // menggunakan += untuk mengisi dan menjumlahkan
        apel += 6;

        Console.WriteLine($"apel = {apel}");
    }
}
```

contoh operator penugasan Coba perhatikan!

Pada contoh tersebut, kita mengisi ulang nilai variabel mangga menjadi 15 dengan operator pengisian `(=)`.

Lalu untuk variabel apel kita menambahkannya dengan 6.

```
apel += 6;
Sebenarnya ini bentuk sederhana dari operasi aritmatika:
```

```
apel = apel + 6;
```

Mirip seperti increment dan decrement ya..

Nah, untuk operator penugasan yang lainnya.. kamu bisa coba-coba sendiri.

Misalnya untuk pengurangan.. ubah saja `+=` menjadi `-=`.

```
apel -= 6;
```

Tapi harap hati-hati..

✗ Jangan sampai kebalik seperti ini:

```
apel -= 6;
```

Karena ini akan membuat variabel apel diisi ulang dengan nilai `-6`, bukan dikurangi `6`.

Paham kan?

Pada dasarnya, operator penugasan sama seperti aritmatika. Hanya saja lebih singkat dan fungsinya untuk mengisi nilai.

Berikutnya, kita akan pelajari oprator yang lebih seru.. yakni perbandingan.

3. Opeartor Perbandingan

Operator pembanding adalah operator untuk memabndingkan dua buah nilai. Operator ini juga dikenal dengan operator relasi.

Operator pembanding terdiri dari:

Nama Operator	Simbol
Lebih Besar	>
Lebih Kecil	<
Sama Dengan	==
Tidak Sama dengan	!=
Lebih Besar Sama dengan	>=
Lebih Kecil Sama dengan	<=

Nilai yang dihasilkan dari operasi perbandingan akan berupa boolean **True** dan **False**.

Biar lebih jelas, mari kita coba dalam program.

Buatlah program baru dengan nama **ProgramPerbandingan.cs** dengan kode sebagai berikut:

```
using System;

class ProgramPerbandingan
{
    public static void Main(String[] args)
    {
        int mangga, apel = 0;

        Console.Write("jumlah mangga = ");
        mangga = int.Parse(Console.ReadLine());
        Console.Write("jumlah apel = ");
        apel = int.Parse(Console.ReadLine());

        Console.WriteLine("Hasil perbandingan: ");
        Console.WriteLine($"mangga > apel : {mangga > apel}");
        Console.WriteLine($"mangga >= apel : {mangga >= apel}");
        Console.WriteLine($"mangga < apel : {mangga < apel}");
        Console.WriteLine($"mangga <= apel : {mangga <= apel}");
        Console.WriteLine($"mangga == apel : {mangga == apel}");
        Console.WriteLine($"mangga != apel : {mangga != apel}");
    }
}
```

contoh operator perbandingan Operator perbandingan nanti akan banyak kita pakai untuk membuat logika program di materi percabangan.

Contohnya seperti ini:

```
if ( mangga > apel )
{
    Console.WriteLine("Mangga lebih banyak");
}
```

Kita bisa bandingkan tipe data apa saja dengan operator perbandingan.

Contoh:

Membandingkan tipe data string.

```
string password = "admin";  
Console.WriteLine(password == "admin"); // True
```

Membandingkan tipe data integer dengan float.

```
Console.WriteLine(0.1f == 1); // False
```

Membandingkan tipe data boolean.

```
Console.WriteLine(True != False); // True
```

Paham kan?

Kalau begitu lanjut ke:

4. Operator Logika

Kalau kamu pernah belajar logika matematika, pasti tidak akan asing dengan operator ini.

Nama Operator	Simbol
Logika AND	&&
Logika OR	//
Negasi/kebalikan	!

Hasil operasi dari operator logika sama seperti operator perbandingan, yakni boolean **True** dan **False**.

Namun perlu diingat:

Operan harus bertipe boolean.

Karena jika tidak, maka akan terjadi error.

Operator Logika digunakan untuk membuat operasi logika.

Misalnya seperti ini:

- Pernyataan 1: Mangga rasanya manis
- Pernyataan 2: Jeruk rasanya manis

Jika kita coba buktikan dengan tabel kebenaran logika, maka bisa kita buat seperti ini.

Mangga dan Jeruk:

Pernyataan 1	Pernyataan 2	Hasil Logika AND
true	true	true
true	false	false
false	true	false
false	false	false

Mangga atau Jeruk:

Pernyataan 1	Pernyataan 2	Hasil Logika OR
true	true	true
true	false	true
false	true	true
false	false	false

Bukan Mangga:

Pernyataan 1	Hasil Logika NOT
true	false
false	true

Sampai di sini paham?

Jika belum paham, coba pelajari lagi tentang logika matematika hehe.. 😊

Namun, yang terpenting adalah.. bagaimana cara kita menggunakannya di dalam program.

Logika AND, OR, dan NOT nantinya akan banyak kita pakai dalam membuat logika di program.

Biar lebih jelas, mari kita coba dalam program.

Buatlah program baru dengan nama `ProgramLogika.cs`, kemudian isi dengan kode berikut.

```
using System;

class ProgramLogika
{
    public static void Main(String[] args)
```

```
{

    Console.Write("Enter your age: ");
    int age = int.Parse(Console.ReadLine());
    Console.Write("Password: ");
    string password = Console.ReadLine();

    bool isAdult = age > 18 ; // pernyataan 1
    bool isValidPassword = password == "admin"; // pernyataan 2

    // menggunakan logika AND
    if(isAdult && isValidPassword)
    {
        Console.WriteLine("WELCOME TO THE CLUB!");
    }
    else
    {
        Console.WriteLine("Sorry, try again!");
    }
}
```

Pada contoh program ini, kita menggunakan logika **AND** untuk mengecek apakah user sudah dewasa dan password yang diinputkan benar.

Untuk saat ini, cukup pahami bagaimana bentuk logika AND dan OR. Karena bagian **if / else** akan kita pelajari nanti.

6. Conditional Operator (Ternary)

Conditional operator adalah operator yang membentuk logika **jika/maka** atau **if/else**.

Conditional Operator disebut juga operator ternary karena memiliki tiga operan.

conditional operator Operan pertama adalah kondisi yang akan dicek. Pada bagian ini, kita bisa membuat ekspresi dengan operator perbandingan dan logika.

Lalu operan berikutnya adalah ekspresi jika kondisi benar, dan sisanya ekspresi yang akan dipakai jika kondisi salah.

Biar lebih jelas, mari kita buat contoh programnya.

Buatlah program baru dengan nama **ProgramConditional.cs**, kemudian isi dengan kode berikut.

```
using System;

class ProgramConditional
{
    static void Main(string[] args)
    {
        Console.Write("Berapa mumurmu: ");
        int age = int.Parse(Console.ReadLine());

        // menggunakan conditional operator
        string message = age < 18 ? "Belum cukup umur" : "sudah cukup umur";

        Console.WriteLine(message);
    }
}
```

Sekian pelatihan tentang operator di C#.