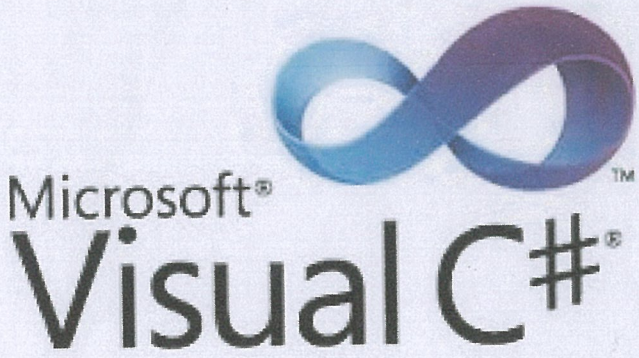
MODUL PRAKTIKUM

PEMROGRAMAN BERORIENTASI OBJEK I



FAKULTAS ILMU KOMPUTER

UNIVERSITAS BANTEN JAYA

2017

BAB I
PENGENALAN BAHASA PEMROGRAMAN

Tujuan :

1. Pengenalan Framework. NET
2. Mampu menggunakan variable, pernyataan, operator dan tipe data
3. Mampu membuat aplikasi sederhana menggunakan C#

1.1 Pengenalan Framework .NET

Framework .NET adalah suatu komponen windows terintegrasi yang dibuat dengan tujuan untuk mendukung pengembangan berbagai macam jenis aplikasi serta untuk dapat menjalankan berbagai macam aplikasi generasi mendatang termasuk pengembangan aplikasi Web Services XML.

Framework .NET didisain untuk dapat memenuhi beberapa tujuan berikut ini :

- Untuk menyediakan lingkungan kerja yang konsisten bagi bahasa pemrograman yang berorientasi objek (*object-oriented programming - OOP*)
- Untuk menyediakan lingkungan kerja di dalam mengeksekusi kode yang dapat meminimaliasi proses *software deployment* dan menghindari konflik penggunaan versi software yang dibuat.
- Untuk menyediakan lingkungan kerja yang aman dalam hal pengeksekusian kode, termasuk kode yang dibuat oleh pihak ketiga (*third party).*
- Untuk menyediakan lingkungan kerja yang dapat mengurangi masalah pada persoalan performa dari kode atau dari lingkungan *interpreter* nya.

Sekilas Pemrograman C#

Pada tahun 2000 Microsoft meluncurkan bahasa pemrograman baru yang diberi nama *C# Programming Language*. C# dikembangkan oleh Microsoft oleh tim yang dipimpin oleh Anders Hejlsberg dan Scott Wiltamuth. C# memiliki kesamaan bahasa dengan C, C++, dan Java, sehingga memudahkan developer yang sudah terbiasa dengan bahasa C untuk menggunakannya, C# mengambil fitur-fitur terbaik dari ketiga bahasa tersebut dan juga menambahkan fitur-fitur baru. C# adalah bahasa pemrograman *Object Oriented* dan memiliki *class library* yang

sangat lengkap yang berisi prebuilt component sehingga memudahkan programer untuk men-develop program lebih cepat. C# juga distandarkan oleh Ecma International pada bulan desember 2002.

Dengan C# dapat dibuat bermacam aplikasi seperti aplikasi console, aplikasi windows form, aplikasi Web, aplikasi Web services, dan aplikasi untuk *mobile device*. Jadi cukup belajar satu bahasa saja tapi sudah dapat digunakan untuk mengembangkan berbagai macam aplikasi.

Ada dua komponen utama dalam .NET Framework yaitu CLR (Common Language Runtume) dan FCL (.NET Framework Class Library).

Common Language Runtime (CLR) adalah pondasi utama dari Framework .NET. CLR merupakan komponen yang bertanggung jawab terhadap beberapa tugas, seperti mengatur managemen memory, melakukan eksekusi kode, melakukan verifikasi terhadap keamanan kode, menentukan hak akses dari kode, melakukan kompilasi kode, dan berbagai tugas lainnya. Dengan adanya komponen CLR ini, maka aplikasi berbasis .NET biasa juga disebut dengan *managed code*, sedangkan aplikasi di luar .NET disebut dengan *un-managed code*.

Pada .NET ada dua tahap kompilasi yang dilakukan, pertama CLR akan melakukan kompilasi kode-kode aplikasi kita menjadi bahasa assembly MSIL (*Microsoft Intermediate Language*), kedua ketika aplikasi dieksekusi compiler yang lain yang bernama JIT (Just-in-time compiler) yang juga salah satu kompenen dalam CLR untuk menterjemahkan MSIL kedalam bahasa mesin yang disesuaikan dengan platformnya.

Editor C#

- **Notepad**

  Kita dapat membuat aplikasi C# dalam notepad . File-file C# disimpan dengan ekstension .cs, jika kita tidak hati-hati pada saat menyimpan file C# di Notepad, misal kita bermaksud menyimpan file dengan nama test.cs maka tidak tertutup kemungkinan file tersebut akan menjadi test.cs.txt kecuali kita telah mensetting terlebih dahulu box drop down list pada fungsi Save As menjadi "All Files".

- **Visual Studio 6**

  Jika anda telah terbiasa menggunakan Visual Studio 6, maka tools tersebut bisa digunakan untuk membuat aplikasi dengan C#, khususnya dengan menggunakan editor Microsoft Visual C++.

Salah satu keuntungan menggunakan editor khusus buat pemrograman (seperti Microsoft Visual C++) adalah adanya syntax highlighting , yang memudahkan kita pada saat membaca dan menganalisa kode-kode program kita. Namun, karena Visual Studio 6 (khususnya Visual C++) ini di buat sebelum adanya bahasa C#, maka perlu sedikit "kreatifitas" kita untuk memodifikasi setting editor tersebut agar dapat menampilkan syntax highlighting C#.

- **Visual Studio .NET**

  Visual Studio .NET merupakan editor yang paling ideal untuk membuat aplikasi yang berbasis Framework .NET, termasuk aplikasi dengan bahasa C#. Editor ini tidak hanya menyediakan berbagai macam tools dan wizard untuk membuat aplikasi C#, tapi juga termasuk fitur-fitur produktif seperti IntelliSense dan bantuan yang dinamis.

Menulis Aplikasi dengan Notepad

1. Buka file baru pada aplikasi Notepad. Lalu simpanlah dengan nama HaloDunia.cs
2. Ketiklah kode berikut ini

```
using System;
class HaloCsharp
{
// Bagian utama program C#
public static void Main( string [] args)
   {
System.Console.WriteLine("Ini adalah program pertamaku dengan C#");
   }
}
```

3. Untuk proses kompilasi, kamu perlu buka "Visual Studio 2008 Command Prompt". Kemudian atur path sesuai dengan tempat penyimpanan HaloDunia.cs. Perintah yang digunakan untuk kompilasi:

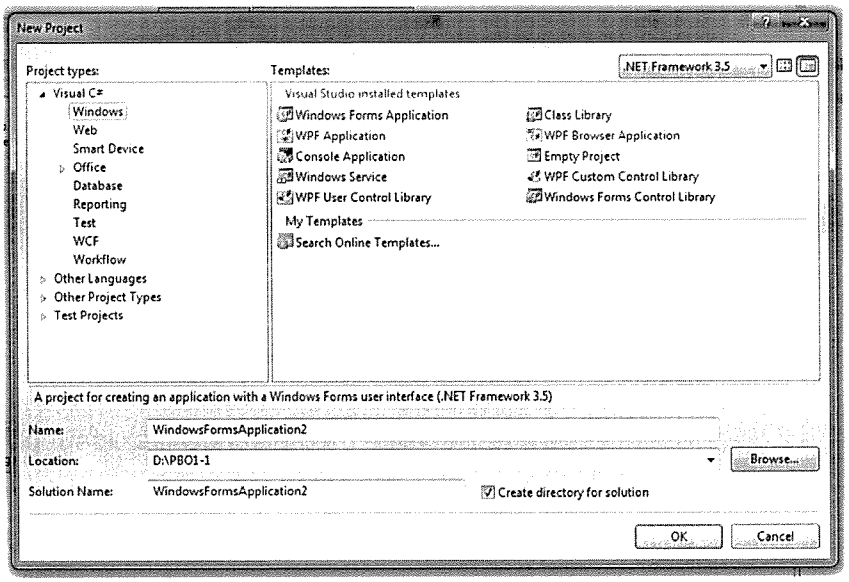
   csc HaaloCsharp.cs

   Perintah tersebut akan mengkompilasi program HaloDunia.cs menjadi HaloCsharp.exe.

4. Setelah dikompilasi, jalankan/eksekusi program tersebut dengan perintah : HaloCsharp atau HaloCsharp.exe

Buat aplikasi Console baru, pilih

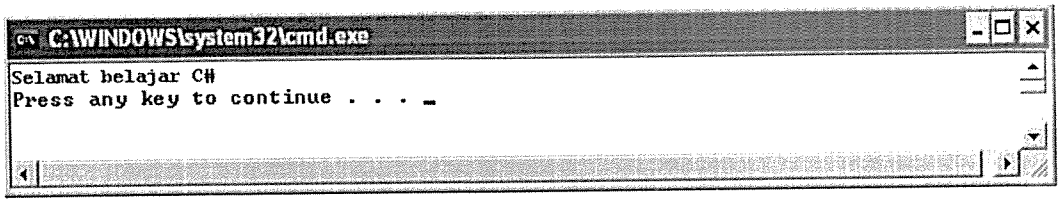
1. **File > New Project > Console Application** kemudian beri nama **Baru** kemudian klik tombol **OK**



2. Kemudian tuliskan source code sebagai berikut :

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Latihan1 {
class Program {
static void Main( string [] args) {
Console .WriteLine( "Selamat belajar C#" );
                }
    }
}
```

3. Kemudian jalankan dengan menekan tombol Ctrl + F5 (untuk Run tanpa debug) maka akan ditampilkan hasilnya sebagai berikut.



Penjelasan perbagian dari Latihan1.1 dan 1.2 di atas adalah sebagai berikut:

1. T*ag* // digunakan sebagai komentar, pada C# digunakan t*ag* // (jika hanya satu baris saja) atau diapit t*ag* /* */ (jika komentarnya lebih dari satu baris). Kode atau keterangan di dalam tag komentar tidak akan dieksekusi oleh compiler.
2. Keyword **using** digunakan untuk memberitahu compiler class yang digunakan pada aplikasi. Salah satu keunggulan dari C# adalah tersedianya predefined class yang dapat langsung digunakan dalam aplikasi.
3. Keyword **namespace** digunakan untuk mendeklarasikan ruang lingkup dari class yang kamu buat. Class di dalam namespace dapat dipanggil dengan mencantumkan nama namespace-nya terlebih dahulu.
4. Keyword **using System;** pada baris pertama mempunyai arti kamu dapat menggunakan class-class yang ada pada namespace System.
5. Keyword **class** digunakan untuk mendeklarasikan class dengan nama Program.
6. Deklarasi **static void Main(string[] args)** adalah method utama yang dijalankan pada **class Program**.
7. Sintaks **Console.WriteLine** digunakan untuk menampilkan output ke layar console.

1.2 Tipe Data

C# mempunyai dua jenis *built-in* tipe data yaitu *value types* dan *reference types*. Referenced type didefinisikan dengan menggunakan class.

C# mempunyai 13 tipe data dasar yang ditunjukan pada table dibawah ini.

Type	Keterangan	Type	Keterangan
Bool	Merepresentasikan nilai true/false	Sbyte	8-bit signed integer
Byte	8-bit unsigned integer	Short	Short integer
Char	Character	Uint	Unsigned integer
Decimal	Numeric type untuk financial calculations	Ulong	Unsigned long integer
Double	Double-precision floating point	ushort	Unsigned short integer
Float	Single-precision floating point		
Int	Integer		
Long	Long Integer		

Tipe-tipe data yang ada diatas disebut dengan simple types

Tipe Data Integer

C# mempunyai sembilan tipe integer: **char, byte, sbyte, short, ushort, int, uint, long,** dan **ulong**. Tetapi char type digunakan untuk mendefinisikan tipe data karakter, delapan sisanya digunakan untuk kalkulasi numerik.

Latihan1.3 :

```
using System;
class Latihan1_3 {
static void Main( string [] args) {
ushort hari, bulan, tahun;
Console .WriteLine( "Hitung Usia" );
Console .WriteLine( "----------" );
try {
Console .Write( "Tanggal Lahir\t : " );
hari = Convert .ToUInt16( Console .ReadLine());
Console .Write( "Bulan Lahir\t : " );
bulan = Convert .ToUInt16( Console .ReadLine());
Console .Write( "Tahun Lahir\t : " );
tahun = Convert .ToUInt16( Console .ReadLine());
DateTime ultah = new DateTime (tahun, bulan, hari);
DateTime hariIni = DateTime .Now;
TimeSpan usia = hariIni.Subtract(ultah);
Console .WriteLine( "------------------------------------------------------------------------" );
Console .WriteLine( "Umur kamu adalah {0} hari, {1} jam " + "{2} menit.", usia.Days,
usia.Hours, usia.Minutes);
}
catch ( FormatException e) {
Console .WriteLine( "Data tanggal, bulan, tahun harus berupa angka." );
}
catch ( Exception e) {
Console .WriteLine( "Terjadi Kesalahan : " ,e.Message);
                }
```

```
    }
}
```

1.3 Tugas Praktikum

1. Buatlah program untuk menginputkan biodata terdiri dari nama lengkap, nama panggilan, npm, umur, tempat lahir,telepon, dan alamat.

   Output :

   Assalamu'alaikum.Let me introduce my self. My name is (nama lengkap), but you can call me (nama panggilan). My NPM is (npm). I was born in (tempat lahir) and I am (umur) years old. I am very glad if you want to invite my house in (alamat). So, don't forget to call me before with the number (telepon).Thank you.
2. Buatlah program konversi waktu dari detik ke jam, menit, detik!
3. Buatlah program konversi suhu dari Celcius (C) ke Fahrenheit (F), Reamur (R), dan Kelvin (K). Suhu Celsius di masukkan melalui keyboard saat program dieksekusi !

   F = C * 9/5 +32

   K =C+273,15

   R = 4/5*C

BAB II
STRUKTUR RUNTUTAN & PERCABANGAN

Tujuan :

1. Memahami struktur runtutan
2. Memahami struktur pemilihan if-else dan switch-case
3. Mampu membuat program sederhana dengan menggunakan struktur percabangan

2.1 Pendahuluan

Struktur kontrol yang ada pada bahasa C# adalah struktur runtutan, struktur pemilihan dan struktur pengulangan. Struktur kontrol pemilihan dan pengulangan memanfaatkan operator dalam menentukan suatu kondisi. Operator-operator yang terkait dengan proses seleksi kondisi adalah Operator Logika dan Relasional (hubungan).

Relational dan Logical Operators

Berikut daftar relational operator pada C# :

Operator	Meaning
>	Lebih besar
>=	Lebih besar sama dengan
<	Lebih kecil
<=	Lebih kecil sama dengan
==	Sama dengan
!=	Tidak sama dengan

Relational operator pada C# biasanya digunakan dalam statement If untuk pengecekan kondisi. Dan berikut adalah **logical operator** pada C#:

Operator	Meaning
&	AND
I	OR
^	XOR (Exclusive OR)
&&	Short-circuit AND
II	Short-circuit OR
!	Not

2.2 Struktur Runtutan

Secara umum, cara compiler bekerja adalah membaca perintah mulai dari baris atas ke bawah secara berurutan . Setiap baris dibaca mulai dari kiri ke kanan. Namun tidak menutup kemungkinan dalam Pemrograman ada struktuk lain seperti struktur percabangan, strukur pengulangan atau lainnya, dibahas pada sub bab selanjutnya.

Latihan 2.1

```
//program menjumlahkan dua buah bilangan
using System;
class Tambah{
static void Main(string[] args){
        int number1; //variabel unt menampung nilai pertama
        int number2; //variabel unt menampung nilai kedua
        int sum; //variabel yg menampung hasil penjumlahan
        Console.Write("Masukan nilai pertama :");
        //konversi dari tipe string ke integer
        number1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("Masukan nilai kedua :");
        number2 = Convert.ToInt32(Console.ReadLine());
        sum = number1 + number2;
        Console.WriteLine("Jadi hasil penjumlahannya : "+sum);
        }
}
```

2.3 Struktur Pemilihan

Bentuk umum If

if (kondisi yang diseleksi){
Pernyataan 1;
:
:
:
Pernyataan *n* ;
}

Jika kondisi yang diseleksi benar maka pernyataan 1 sampai pernyataan n akan dikerjakan, sedangkan jika kondisi tersebut tidak terpenuhi maka program akan keluar dari struktur if. Jika lebih dari satu pernyataan , maka harus menggunakan tanda '{' dan '}'.

Bentuk umum if – else

```
if (kondisi yang diseleksi)
   {
   Pernyataan 1;
   }
Else
   {
   Pernyataan 2;
   }
```

Jika kondisi yang diseleksi bernilai benar atau terpenuhi maka pernyataan pertama yang dilaksanakan dan jika kondisi yang diseleksi bernilai salah maka pernyataan yang kedua yang dilaksanakan. Setiap pernyataan diakhiri tanda titik koma (;) walaupun sebelum else.

Bentuk umum Nested if

```
if (kondisi yang diseleksi){
     Pernyataan 1;
     }
Else
     if (kondisi yang diseleksi){
     Pernyataan 2;
     }
Else
     if (kondisi yang diseleksi){
     Pernyataan 3;
     }
Else
{
     Pernyataan 4;
}
```

Bentuk Umum Switch

```
Switch (ekspresi)
{
   case konst-1 :
      Pernyataan -1;
      break;
   case konst-2 :
      Pernyataan -2;
      break;
      :
      :
   case konst-n :
      Pernyataan -n;
      break;
   default :
      Pernyataan -df;
      break;
}
```

Struktur switch–case–default digunakan untuk penyeleksian kondisi dengan kemungkinan yang terjadi cukup banyak. "ekspresi" dapat berupa konstanta integer atau karakter.

- Case digunakan sebagai label yang menandai awal eksekusi deret instruksinya hingga ditemukan pernyataan break
- Default adalah label yang jika label-label Case diatasnya tidak ada yang memenuhi maka label ini yang akan dieksekusi
- Break adalah perintah pengontrol alur program yang berfungsi untuk keluar dari suatu blok kondisi ataupun iterasi.

Latihan2.2 :

```
using System;
class Latihan2_2 {
      static void Main( string [] args) {
         Console .Write( "Masukan IPK : " );
```

```
        float ipk = Convert .ToSingle( Console .ReadLine());
        string predikat;
        if (ipk > 3.5)
            predikat = "Cum Laude" ;
        else if (ipk > 3)
                predikat = "Sangat Memuaskan" ;
        else if (ipk >= 2.75)
                predikat = "Memuaskan" ;
        else if (ipk >= 2)
                predikat = "Cukup" ;
        else
                predikat = "Memuakkan" ;
        Console .WriteLine( "Predikat Anda adalah : {0}" , predikat);
    }
}
```

Latihan2.3 :

```
using System;
class Pileg {
static void Main() {
int intHari;
Console.Write("Masukan Pilihan Hari (1-7) :");
intHari = Int32.Parse(Console.ReadLine());
switch (intHari){
        case 1:
                Console.WriteLine("Hari Minggu");
                break;
        case 2:
                Console.WriteLine("Hari Senin");
                break;
        case 3:
                Console.WriteLine("Hari Selasa");
                break;
        case 4:
```

```
            Console.WriteLine("Hari Rabu");
            break;
        case 5:
            Console.WriteLine("Hari Kamis");
            break;
        case 6:
            Console.WriteLine("Hari Jumat");
            break;
        case 7:
            Console.WriteLine("Hari Sabtu");
            break;
        default:
            Console.WriteLine("Anda salah memasukan input (1-7)");
            break;
            }
     }
}
```

2.4 Tugas Praktikum

1. Buatlah Program untuk mengetahui karakter yang diinputkan, apakah huruf Besar, huruf kecil, spasi, digit, atau yang lainnya !

   Input :

   Masukkan Karakter : A

   Output: Karakter yang diinputkan adalah huruf besar

2. Buatlah program untuk mempermudah pembayaran di suatu bioskop

Jenis	No	Judul	Harga
Horor	1	Jelangkung	Rp 75.000
	2	Pengabdi Setan	Rp 50.000
	3	Hantu Jeruk Purut	Rp 45.000
Romantik	1	Ayat-ayat cinta II	Rp 75.000
	2	Kesucian Cinta	Rp 50.000
	3	Asmara	Rp 50.000

   Output :

   a. Buatlah Tampilan Menu

b. Baru pilih horror atau romantic

Horor : 2

Romantic : 3

c. Hitung total harga

Total Harga Rp. 8500,-

BAB III
STRUKTUR PENGULANGAN

Tujuan :

1. Memahami struktur pengulangan for, while, do-while,foreach
2. Memahami struktur lompat break, continue dan goto
3. Mampu membuat program sederhana dengan menggunakan struktur pengulangan

3.1 Pendahuluan

Sering kali dalam membuat program, kamu menuliskan beberapa perintah baris yang sama. Penulisan perintah tersebut dapat disingkat dengan menggunakan struktur pengulangan, seperti for, while, dan do-while. Beberapa hal penting pada struktur pengulangan adalah adanya inisialisasi, kondisi, dan iterasi. Ada empat struktur pengulangan yang dapat digunakan pada bahasa C#, yaitu:

a. Struktur for
b. Struktur while
c. Struktur do...while
d. Struktur foreach

Berkaitan dengan proses pengulangan, pemrograman C# juga menyediakan pernyataan *break (untuk mengakhiri pengulangan)* , *goto* , dan *continue(untuk melakukan pengulangan selanjutnya).*

3.2 Struktur for

Struktur pengulangan for biasa digunakan untuk mengulang suatu proses yang telah diketahui jumlah pengulangannya.

Bentuk Umum :

```
for (inisialisasi;kondisi;iterasi)
{
   Pernyataan ;
}
```

Keterangan:

- **Inisialisasi** : pernyataan untuk menyatakan keadaan awal dari variable kontrol.
- **Kondisi :** ekspresi relasi yang menyatakan kondisi untuk keluar dari pengulangan.
- **Iterasi :** pengatur perubahan nilai variabel kontrol.

Latihan3.1 :

```
static void Main(string[] args)
{
    //mencetak bilangan dari 1-10, increment
    for (int i = 1; i <= 10; i++)
    Console.WriteLine(i);
    Console.WriteLine();
    //mencetak bilangan dari 10 ke 1, decrement
    for (int i = 10; i >= 1; i--)
    Console.WriteLine(i);
}
```

3.3 Struktur While

Pengulangan ini banyak digunakan bila jumlah pengulangannya belum diketahui. Proses pengulangan akan terus berlanjut selama kondisinya bernilai benar (true) dan akan berhenti bila kondisinya bernilai salah.

Bentuk Umum :

```
Inisialisasi; <optional>
while (kondisi){
    pernyataan;
    iterasi; <optional)
}
```

Latihan3.2 :

```
static void Main(string[] args){
        int bil;
        int i = 1;
        int banyak;
        int besar = 0;
        Console.Write("Masukan banyak bilangan yang akan dicek :");
        banyak = Convert.ToInt32(Console.ReadLine());
        while (i <= banyak){
            Console.Write("Masukan Bilangan ke-"+i+": ");
            bil = Convert.ToInt32(Console.ReadLine());
            if (bil >= besar){
            besar = bil;
        }
            i++;
}
Console.WriteLine("Jadi bilangan yang terbesar adalah :"+besar);
}
```

3.4 Struktur do..while

Pengulangan ini digunakan bila jumlah pengulangan do..while sama saja dengan struktur while, hanya saja pasa proses seleksi kondisi letaknya berada di bawah batas pengulangan. Jadi dengan menggunakan struktur do..while sekurang-kurangnya akan terjadi satu kali pengulangan.

Bentuk Umum :

```
Inisialisasi <optional>
do {
   Pernyataan ;
   iterasi ; <optional>
} while (kondisi);
```

Latihan3.3 :

```
static void Main(string[] args){
        int num;
        int nextdigit;
        Console.Write("Masukan bilangan :");
        num = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Bilangan sebelum dibalik "+num);
        Console.Write("Bilangan setelah dibalik :");
        Do
        {
                nextdigit = num % 10;
                Console.Write(nextdigit);
                num = num / 10;
        } while (num > 0);
}
```

3.5 Struktur foreach

Pada struktur for, kamu menggunakan counter untuk iterasi. Dengan struktur foreach, kamu tidak perlu membuat counter, karena proses iterasi dilakukan secara internal array. Kelebihannya adalah cepat mendapatkan nilai array secara keseluruhan. Kekurangannya adalah tidak bias mengakses nilai dari suatu indeks tertentu.

Sintaks foreach

foreach *(tipe_data nama_var* ***in*** *nama_array)*

statement;

tipe_data : menyatakan tipe data dari variable

nama_var : menyatakan tempat penampungan nilai elemen dari array

nama_array : menyatakan array yg digunakan

statement : menyatakan badan program yang akan diiterasi

Latihan 3.4

```
//Progam Demo foreach
using System;
public class ForeachDemo {
        static void Main() {
```

```
        string [] strings = { "Matematik" , "Bahasa" };
    foreach ( string item in strings){
        Console .WriteLine( "{0}" , item);
    }
}
}
```

3.6 Pernyataan Break

Break statement digunakan untuk keluar dari kalang bila kondisi tertentu yang tambahkan dipenuhi. Ketika break statement dieksekusi maka kalang / loop akan secara otomatis berhenti.

3.7 Pernyataan Continue

Pernyataan *continue* digunakan untuk mengarahkan eksekusi ke proses berikutnya pada *loop* yang sama. Pada *do-while* dan *while* , pernyataan *continue* menyebabkan eksekusi menuju ke pengulangan kembali.

```
do {
continue;

}while(kondisi)
    while(kondisi){

    continue;
    }
```

3.8 Pernyataan Goto

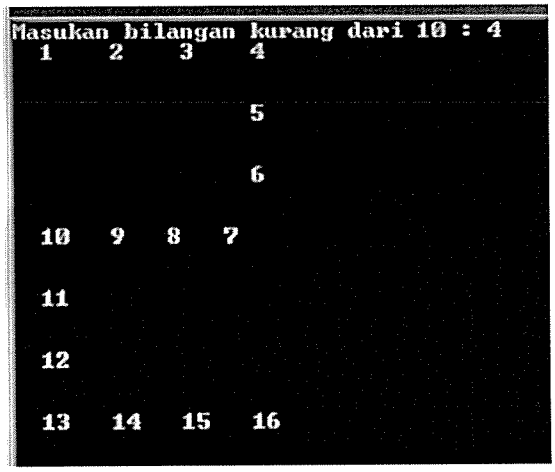
Pernyataan goto ini merupakan perintah yang digunakan un ke pernyataan yang di awali dengan suatu label. Label merupakan suatu pengenal yang diikuti dengan tanda titik koma(;).

Tugas Praktikum

1. Buatlah program menggunakan nested loop!
   a. Input : UNBAJA
      Output : AJABNU
   b. Masukan Bilangan : 7

1	1
2	3
3	6
4	10
5	15
6	21
7	28

2. Buatlah program dengan menggunakan conditional looping!



3. Masukan tiap program di atas kedalam suatu menu pilihan!

BAB IV

ARRAY DAN COLLECTION

Tujuan :

1. Mengetahui sekilas mengenai cara penulisan array dan collection dalam pemrograman bahasa C#
2. Mengetahui penerapan array dan collection dalam pemrograman bahasa C
3. Mampu membuat program sederhana dengan menggunakan array dan collection

4.1 Array (Larik)

Pemrograman membutuhkan variabel untuk menyatakan suatu kegiatan proses tertentu. Dalam kondisi tertentu, terkadang membutuhkan kumpulan data yang sama dalam tipe. Di dalam pemrograman dikenal dengan nama array. Array mempunyai keterbatasan, yaitu harus memdefinisikan banyaknya data yang dibutuhkan. Dengan demikian, array disebut kumpulan variabel berindeks terhingga yang mempunyai tipe data yang sama.

1. ***Array satu dimensi***

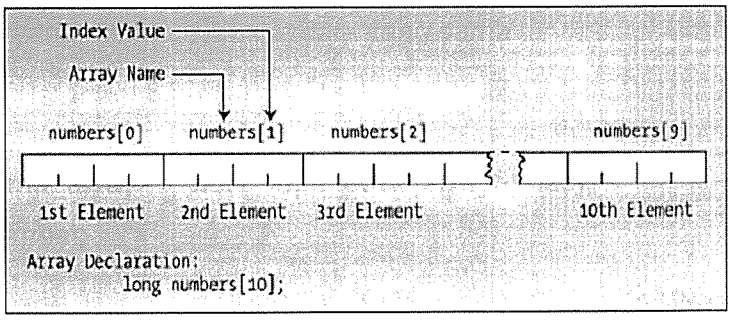
   Deklarasi umum :

   <tipe_data_array>[] <nama_array> = new <tipe_data_array>[size]; < nama_array > [indeks] = nilai;

   Ket :

   - <type_data_array> = semua jenis tipe data
   - [] = penanda array
   - <nama_array> = nama suatu array sebagai identifikasi
   - [indeks] = elemen ke-berapa dari <nama_array>
   - [size] = jumlah terhingga yang dapat dimiliki <nama_array>

   Misal : int numbers[3];



Latihan program 4.1 :

```
static void Main(string[] args) {
        int[] bil = new int[10];
        //Console.Write("Masukkan bilangan : ");
        Console.WriteLine("{0}{1,8}", "indeks", "nilai");
        for (int i = 0; i < bil.Length; i++){
            bil[i] = i + i;}
        for (int i = 0; i < bil.Length; i++){
            Console.WriteLine("{0}{1,8}", i, bil[i]);
        }
        Console.ReadLine();}
```

2. *Array Multi Dimensi*

a. Array dari Array

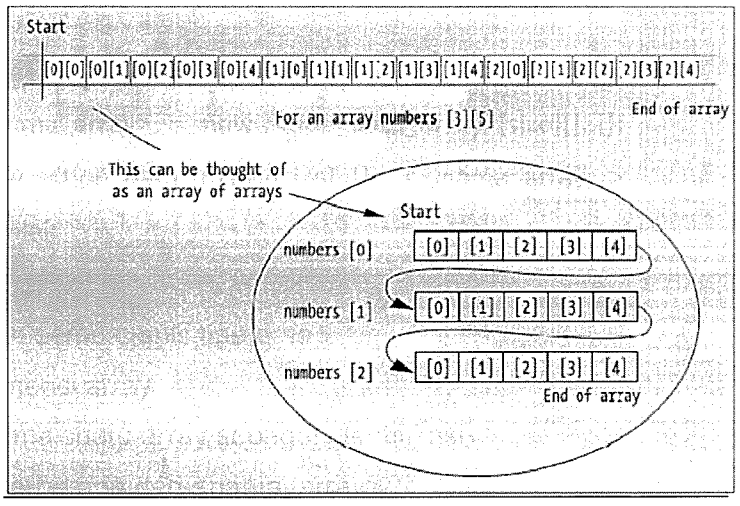
<tipe_data_array>[][]..[] <nama_array> = new <tipe_data_array>[size][]..[];

<nama_array>[indeks]= new <tipe_data_array>[size]..[]; < nama_array > [indeks][indeks]..[indeks] = nilai;

Ket :

- <type_data_array> = semua jenis tipe data
- [][]..[] = penanda dimensi array
- <nama_array> = nama suatu array sebagai identifikasi
- [indeks] = elemen ke-berapa dari <nama_array>
- [size] = jumlah terhingga yang dapat dimiliki <nama_array>

Misal : int numbers[3][5]



Latihan program 4.2 :

```
using System;
namespace Bab4 {
class Latihan42 {
        // Array-of-arrays (jagged):
        public static void Main( string [] args) {
        int [][] matriks= new int [3][];
        Console .WriteLine( "Masukkan nilai matriks" );
        for ( int i = 0; i < matriks.Length; i++) {
        matriks[i]= new int [3];
        for ( int j = 0; j < matriks[i].Length; j++) {
        Console .Write( "matriks[{0}][{1}]= " , i, j);
        matriks[i][j] = int .Parse( Console .ReadLine());
    }
}
      Console .WriteLine();
      Console .WriteLine( "Menampilkan nilai matriks" );
      for ( int i = 0; i < matriks.Length; i++) {
         for ( int j=0;j<matriks[i].Length;j++) {
         Console .Write( "{0} " , matriks[i][j]);
         }
         Console .WriteLine();
      }
      Console .ReadLine();
   }
}
}
```

b. Multidimensi Array

<tipe_data_array>[, , … ,] <nama_array> = new <tipe_data_array>[size, size, … , size]; < nama_array > [indeks, indeks, … , indeks] = nilai;

Ket :

- <type_data_array> = semua jenis tipe data
- [, , … ,] = penanda dimensi array

- <nama_array> = nama suatu array sebagai identifikasi
- [indeks] = elemen ke-berapa dari <nama_array>
- [size] = jumlah terhingga yang dapat dimiliki <nama_array>

Latihan program 4.3 :

```
using System;
namespace Bab4 {
        class Latihan43 {
        // Multidimensional arrays:
        public static void Main( string [] args){
        int [,] matriks = new int [3,3];
        Console .WriteLine( "Masukkan nilai matriks" );
        for ( int i = 0; i < 3; i++) {
        for ( int j = 0; j < 3; j++) {
        Console .Write( "matriks[{0}, {1}]= " , i, j);
        matriks[i,j] = int .Parse( Console .ReadLine());
    }
}
        Console .WriteLine();
        Console .WriteLine( "Menampilkan nilai matriks" );
        for ( int i = 0; i < 3; i++) {
        for ( int j = 0; j < 3; j++) {
        Console .Write( "{0} " , matriks[i,j]);
    }
        Console .WriteLine();
    }
        Console .ReadLine();
        }
}
}
```

4.2 Collections

.NET Framework mempunyai paket (kumpulan kelas) untuk menangani kebutuhan data yang tidak bisa ditentukan jumlah banyaknya data suatu array yang dibutuhkan. Untuk menangani permasalahan di atas, .NET Framework menyediakan paket yang bernama Collection yang terdapat di bawah paket "System.Collections".

Dalam paket tersebut mempunya kelas-kelas, seperti stack, queue, list, hash dan sebagainya.

1. Kelas-kelas Collection

Kelas-kelas Collection mempunyai tipe data yang sama. Semua data akan diubah tipe data-nya ke dalam tipe data object. Data yang dapat diterima menjadi anggota suatu kelas Collection tidak terbatas jumlahnya. Untuk membangkitkan berapa banyak-nya data yang ada di dalam kelas Collection dapat meminta bantuan dari kelas IEnumerator disebabkan keterbatasan ukuran maksimum dari nilai integer. IEnumerator adalah `interface` yang mendukung iterasi dari suatu koleksi baik dari kelas-kelas Collection atau array.

Contoh di bawah ini adalah satu kasus penggunaan ArrayList.

Latihan program 4.4 :

```
using System;
using System.Collections;
using System.Collections.Generic;
namespace Bab4 {
        class Latihan44 {
        // Penggunaan ArrayList
        public static void Main( string [] args) {
        ArrayList arrayList = new ArrayList ();
        Console .WriteLine( "Masukkan nilai array" );
        while ( true ) {
        Console .Write( "Sembarang kata= " );
        arrayList.Add( Console .ReadLine());
        Console .Write( "Selesai (Y/T)= " );
        string jawab = Console .ReadLine();
        if (jawab.ToLower().Equals( "y" )) {
```

```
                break ;
            }
        }
                Console .WriteLine();
                Console .WriteLine( "Tampilkan semua kata" );
                IEnumerator iterator = arrayList.GetEnumerator();
                while (iterator.MoveNext()) {
                Console .WriteLine(iterator.Current);
            }
            Console .ReadLine();
            }
        }
}
```

2. Generic Collection Class

.NET Framework mendukung penggunaan array yang telah diidentifikasikan tipe datanya. Dukungan ini dimulai dari .NET Framework 2.0. Generic di sini adalah tipe data dari suatu kelas koleksi telah didefinisikan. Dibutuhkan framework dari "System.Collections.Generic"

<collection_class><T> = new <collection_class><T>();

Ket :

- <collection_class> = kelas-kelas yang mendukung generic tipe data. Misal: LinkedList<string>
- <T> = tipe data generic yang digunakan

Latihan program 4.5 :

```
using System;
using System.Collections.Generic;
namespace Bab4 {
class Latihan45 {
            // Penggunaan Array Generic
            public static void Main( string [] args) {
            LinkedList < string > listString = new LinkedList < string >();
            Console .WriteLine( "Masukkan nilai array" );
```

```
            while ( true ) {
            Console .Write( "Sembarang kata= " );
            listString.AddLast( Console .ReadLine());
            Console .Write( "Selesai (Y/T)= " );
            string jawab = Console .ReadLine();
            if (jawab.ToLower().Equals( "y" )) {
        break ; }
    }
            Console .WriteLine();
            Console .WriteLine( "Tampilkan semua kata" );
            IEnumerator < string > iterator = listString.GetEnumerator();
            while (iterator.MoveNext()) {
            string str = iterator.Current;
            Console .WriteLine(str); }
            Console .ReadLine();
        }
    }
}
```

BAB V
SORTING DAN SEARCHING

Tujuan :

1. Mengetahui konsep dasar pengurutan data dalam bahasa C#
2. Mengetahui penerapan searching(pencarian) data pada bahasa C#

SORTING

Sorting merupakan proses pengelompokan suatu informasi yang berhubungan secara berurutan baik ascending(dari kecil ke besar) atau descending(dari besar ke kecil). .NET Framework juga menyediakan dalam kelas koleksi untuk mengurutkan data. Tetapi dalam pembelajaran kali ini kita dituntut untuk menguasai proses pengurutan data secara manual, dengan alasan sebagai berikut :

1. Pada umumnya fungsi misalnya qshort() tidak dapat diaplikasikan pada semua situasi
2. Karena qshort() adalah suatu parameter yang khusus untuk untuk pengoperasian data dalam jumlah besar, sehingga prosesnya berjalan sangat pelan dibanding proses pengurutan data dengan cara yang lain.
3. Alogaritma quickshort (yang biasanya diterapkan pada qshort()),meskipun baik digunakan pada proses yang umum,tetapi bukanlah proses shorting yang terbaik.

Ada 3 metode yang dapat digunakan pada shorting array :

1. Exchange
   a. Bubble Sort
   b. Shaker Sort
2. Selection
3. Insertion

5.1 BUBLE SORT

```
static void BubbleSort( int [] array) {
int count = array.Length;
            int temp;
```

```
                for ( int i = (count - 1); i >= 0; i--) {
for ( int j = 1; j <= i; j++) {
                    if (array[j - 1] > array[j]) {
                    temp = array[j - 1];
                    array[j - 1] = array[j];
                    array[j] = temp;
                }
            }
}
    PrintArray(array);
}
```

5.2 SHAKER SORT

```
static void ShakerSort( int [] array) {
int count = array.Length;
int b, c, d;
int temp;
                    c = 1;
                    b = count - 1;
                    d = count - 1;
                    do {
                    for ( int i = d; i >= c; --i)
{
                    if (array[i - 1] > array[i])
{
                    temp = array[i - 1];
                    array[i - 1] = array[i];
                    array[i] = temp;
                    b = i;
}
                }
                    c = b + 1;
                    for ( int i = c; i < d + 1; ++i){
                    if (array[i - 1] > array[i]) {
```

```
                temp = array[i - 1];
                array[i - 1] = array[i];
                array[i] = temp;
                b = i;
            }
        }
            d = b - 1;
        } while (c <= d);
            PrintArray(array);
        }
```

5.3 SELECTION SORT

```
static void SelectionSort( int [] array) {
int count= array.Length;
int temp;
int indeks;
        for ( int i=0;i<count-1;++i){
        indeks=i;
        temp=array[i];
        for ( int j=i+1;j<count;++j){
        if (array[j]<temp){
        indeks=j;
        temp=array[j]; }
}
        array[indeks]=array[i];
        array[i]=temp; }
        PrintArray(array);
}
```

5.4 INSERTION SORT

```
static void InsertionSort( int [] array) {
int count = array.Length;
int temp;
for ( int i = 1; i < count; ++i) {
```

```
temp = array[i];
int j = i - 1;
while (j >= 0 && temp < array[j]) {
array[j + 1] = array[j];
j--;
}
array[j + 1] = temp;
}
PrintArray(array);
}
using System;
using System.Collections.Generic;
namespace Bab5 {
class Latihan51 {
public static void Main( string [] args) {
int [] numbers = new int [] { 1488, 1578, 1973, 3665, 4426, 4548, 5435,
5446,6333,6385,6455, 6504, 6937, 6965, 7104, 7230, 8340, 8958, 9208,
9364,9550, 9645, 9686 };
Console .WriteLine( "Bubble Sort" );
BubbleSort(numbers);
Console .WriteLine( "Shaker Sort" );
ShakerSort(numbers);
Console .WriteLine( "Selection Sort" );
SelectionSort(numbers);
Console .WriteLine( "Insertion Sort" );
SelectionSort(numbers);
Console .ReadLine();
}
static void BubbleSort( int [] array) {
int count = array.Length;
int temp;
for ( int i = (count - 1); i >= 0; i--) {
for ( int j = 1; j <= i; j++) {
if (array[j - 1] > array[j]) {
```

```
temp = array[j - 1];
array[j - 1] = array[j];
array[j] = temp;
}
}
}
PrintArray(array);
}
static void ShakerSort( int [] array) {
int count = array.Length;
int b, c, d;
int temp;
c = 1;
b = count - 1;
d = count - 1;
do {
for ( int i = d; i >= c; --i) {
if (array[i - 1] > array[i]) {
temp = array[i - 1];
array[i - 1] = array[i];
array[i] = temp;
b = i;
}
}
c = b + 1;
for ( int i = c; i < d + 1; ++i) {
if (array[i - 1] > array[i]) {
temp = array[i - 1];
array[i - 1] = array[i];
array[i] = temp;
b = i;
}
```

```
}
d = b - 1;
} while (c <= d);
PrintArray(array);
}
static void SelectionSort( int [] array) {
int count= array.Length;
int temp;
int indeks;
for ( int i=0;i<count-1;++i){
indeks=i;
temp=array[i];
for ( int j=i+1;j<count;++j){
if (array[j]<temp){
indeks=j;
temp=array[j];
}
}
array[indeks]=array[i];
array[i]=temp;
}
PrintArray(array);
}
static void InsertionSort( int [] array) {
int count = array.Length;
int temp;
for ( int i = 1; i < count; ++i) {
temp = array[i];
int j = i - 1;
while (j >= 0 && temp < array[j]) {
        array[j + 1] = array[j];
        j--;
        }
        array[j + 1] = temp;
```

```
}
PrintArray(array);
}
static void PrintArray( int [] a) {
for ( int i = 0; i < a.Length; i++) {
Console .WriteLine(a[i]);
}
}
}
}
```

SEARCHING

Pencarian (Searching) merupakan proses yang fundamental dalam pemrograman, guna menemukan data (nilai) tertentu di dalam sekumpulan data yang bertipe sama. Fungsi pencarian itu sendiri adalah untuk memvalidasi (mencocokkan) data. Metode pencarian dibagi menjadi 2, yaitu:

1. Metode Pencarian Beruntun (Sequential Search)
2. Metode Pencarian Bagi Dua (Binary Search)

5.5 SEQUENTIAL SEARCH

```
static int SequentialSearch( int [] a, int v) {
int count = 0;
for ( int i = 0; i <= a.Length; i++) {
Console .WriteLine( "Checking ... {0}" , ++count);
if (v == a[i])
return i;
}
return -1; }
```

5.6 BINARY SEARCH

```
static int BinarySearch( int [] a, int v) {
int l = 0;
int r = a.Length;
int count = 0;
```

```
while (r >= l){
Console .WriteLine( "Checking ... {0}" , ++count);
int m = (l + r) / 2;
if (v == a[m])
return m;
if (v < a[m])
r = m - 1;
else
l = m + 1;
}
return -1; }

using System;
using System.Collections.Generic;
namespace Bab5 {
class Latihan52 {
public static void Main( string [] args) {
int [] numbers = new int [] { 1488,1578,1973,3665,4426,4548,5435,5446,
6333,6385,6455,6504,6937,6965,7104,7230, 8340, 8958, 9208, 9364,9550,
9645, 9686};
Console .WriteLine( "Indeks ke : {0}" , SequentialSearch(numbers, 5435));
Console .WriteLine( "Indeks ke : {0}" , BinarySearch(numbers, 5435));
Console .ReadLine();
}
static int SequentialSearch( int [] a, int v) {
int count = 0;
for ( int i = 0; i <= a.Length; i++) {
Console .WriteLine( "Checking ... {0}" , ++count);
if (v == a[i])
return i;
}
return -1;
}
static int BinarySearch( int [] a, int v) {
```

```
int l = 0;
int r = a.Length;
int count = 0;
while (r >= l){
Console .WriteLine( "Checking ... {0}" , ++count);
int m = (l + r) / 2;
if (v == a[m])
return m;
if (v < a[m])
r = m - 1;
else
l = m + 1;
}
return -1;
}
}
}
```

BAB VI
CLASS

Tujuan :

1. Mengetahui sekilas mengenai cara penulisan class, variabel dan fungsi
2. Mengetahui penerapan object dari class C#

Class, Variabel, Fungsi

Class adalah cetak biru dari suatu objek riil di dunia. Penerapan objek riil direpresentasikan dalam bentuk class. Misalnya objek mahasiswa. Mahasiswa merupakan objek dari dunia nyata dan dapat dipresentasikan ke dalam suatu class yaitu class Mahasiswa.

Objek mahasiswa mempunyai ciri-ciri khusus yang dimiliki oleh mahasiswa. Misalnya nomor induk mahasiswa (nim), nama, mail dan sebagainya. Ciri-ciri khusus mahasiswa tersebut harus menjadi identitas utama sebagai objek mahasiswa dan bukan objek dosen. Di dalam mempresentasikan ciri-ciri khusus di dalam class Mahasiswa disebut atribut atau variabel.

Objek mahasiswa memiliki beberapa fungsi khusus untuk mendeskripsikan mahasiswa. Misalnya mahasiswa menjawab soal, bertanya, mendapatkan nilai suatu ujian mata kuliah dan sebagainya. Fungsi adalah urutan instruksi terurut yang harus menghasilkan suatu nilai (*return value*). Misalnya, mahasiswa menjawab soal. Menjawab soal adalah fungsi karena ketika dosen menerima jawaban dari mahasiswa maka dosen akan memberi hasil nilai seberapa tepat jawaban mahasiswa.

Objek mahasiswa memiliki beberapa prosedur khusus untuk mendeskripsikan mahasiswa. Misalnya mahasiswa mencetak kartu ujian. Prosedur adalah urutan intruksi terurut. Prosedur dikatakan fungsi yang tidak menghasilkan suatu nilai. Di dalam programming OOP di representasikan sebagai fungsi dengan keyword "void". Struktur class:

<access_modifier> class <nama_class>

Struktur variable:

<access_modifier> <tipe_data> <nama_variabel>;

Struktur fungsi:

<access_modifier> <return_type> <nama_fungsi>(<tipe_data> <nama_parameter>...);

Ket :

- <access_modifier> = cara mengakses class, variabel, property dan fungsi (sementara diterima dulu)
- <nama_class> = nama suatu class sebagai identifikasi ⎫ <tipe_data> = jenis-jenis tipe data
- <nama_variabel> = nama suatu variabel sebagai identifikasi
- <return_type> = nilai yang dihasilkan dari suatu fungsi
- <nama_fungsi>() = nama suatu fungsi sebagai identifikasi
- <tipe_data> = jenis-jenis tipe data
- <nama_parameter> = nama suatu parameter sebagai identifikasi

Latihan program 6.1 :

```
using System;
namespace Bab6 {
public class Mahasiswa {
private int nim;
private string nama;
private string mail;
public int Jawab( string soal) {
int nilai = 0;
// instruksi-intruksi lainnya
return nilai; }
public void CetakKartuUjian() {
Console .WriteLine( "Nomor Induk Mahasiswa: " + nim);
Console .WriteLine( "Nama Mata Kuliah: Matematika Dasar" );
}
}
}
```

a. Construktor

Construktor/Konstruktor adalah fungsi khusus yang sama dengan nama class dan tidak mempunyai nilai balik atau return value. Kegunaan utama adalah untuk mendefinisikan suatu objek dari class. Secara default, construktor tidak ber-parameter. Class Mahasiswa dapat terdiri dari beberapa construktor

antara lain contruktor tidak ber-parameter, ber-parameter nim, ber-parameter nim dan nama dan/atau beberapa kombinasi yang lain dari paramater.

<access_modifier> class <nama_class> {
<access_modifier> <nama_class>() {
}
<access_modifier> <nama_class>(<tipe_data> <nama_parameter>...) {
}
}

Ket :

- <access_modifier> = cara mengakses class, variabel, property dan fungsi (sementara diterima dulu)
- <nama_class> = nama suatu class sebagai identifikasi
- <nama_class>() = kontruktor
- <tipe_data> = jenis-jenis tipe data
- <nama_parameter> = nama suatu parameter sebagai identifikasi

Latihan program 6.2 :

```
using System;
namespace Bab6 {
public class Mahasiswa {
private int nim;
private string nama;
private string mail;
public Mahasiswa()
{ }
public Mahasiswa( int nim, string nama) {
this .nim = nim;
this .nama = nama;
}
public Mahasiswa( string nama, int nim) {
this .nama = nama;
this .nim = nim;
```

```
}
}
}
```

Syarat-syarat dalam membuat konstruktor:

1. Sama dengan nama kelas
2. Konstruktor ada tidak ber-parameter (default) dan ada yang ber-parameter
3. Parameter berdasarkan tipe data dan bukan berdasarkan nama paramater-nya.

Misal: Mahasiswa (string nama, string mail) itu artinya sama dengan Mahasiswa (string mail, string nama). Yang membedakan adalah fungsi dari parameter-nya yang satu sebenarnya untuk nama dan satu lagi untuk mail.

Parameter tidak terbatas banyak-nya.

Fungsi parameter adalah mendefinisikan suatu objek dari kelas.

Latihan program 6.3 :

```
using System;
namespace Bab6 {
public class Mahasiswa {
private int nim;
private string nama;
private string mail;
public Mahasiswa() {
nim = 3;
nama = "Amir" ;
}
public Mahasiswa( int nim, string nama) {
this .nim = nim;
this .nama = nama;
}
public Mahasiswa( string nama, int nim) {
this .nama = nama;
this .nim = nim;
```

```
}
public void CetakMahasiswa() {
Console .WriteLine( "NIM: {0}" , nim);
Console .WriteLine( "Nama: {0}" , nama);
}
public static void Main( string [] args) {
Mahasiswa aku = new Mahasiswa ();
aku.CetakMahasiswa();
aku = new Mahasiswa (4, "Budi" );
aku.CetakMahasiswa();
aku = new Mahasiswa ( "Wati" , 5);
aku.CetakMahasiswa();
}
}
}
```

b. Access Modifier

Access Modifier ada 4 jenis yang umum:

1. Private, akses terbatas pada class itu sendiri
2. Protected, akses terbatas pada class itu sendiri atau dapat diturunkan ke class turunannya dan berganti akses ke private.
3. Public, akses tidak terbatas pada dan dari luar bebas mengakses ke class tersebut.

```
using System;
namespace Bab6 {
public class Mahasiswa2 {
private int nim;
protected string name;
public string mail;
public Mahasiswa2() {
Console .WriteLine( "NIM: {0}" , nim);
Console .WriteLine( "Nama: {0}" , name);
Console .WriteLine( "Mail: {0}" , mail);
```

```
}
public static void Main( string [] args) {
Mahasiswa2 mahasiswa = new Mahasiswa2 ();
}
}
public class Mahasiswa3 : Mahasiswa2 {
public Mahasiswa3()
{
//Console.WriteLine("NIM: {0}", nim);
Console .WriteLine( "Nama: {0}" , name);
Console .WriteLine( "Mail: {0}" , mail);
}
public static void Main( string [] args) {
Mahasiswa3 mahasiswa = new Mahasiswa3 ();
}
}
public class Mahasiswa4 : Mahasiswa3 {
public Mahasiswa4()
{
//Console.WriteLine("NIM: {0}", nim);
//Console.WriteLine("Nama: {0}", name);
Console .WriteLine( "Mail: {0}" , mail);
}
public static void Main( string [] args) {
Mahasiswa4 mahasiswa = new Mahasiswa4 (); }
}
}
```

Catatan: keyword ":" yang terdapat pada Mahasiswa3 : Mahasiswa2 adalah extends. Di sini diminta untuk diterima dulu.

c. Property

Salah satu sifat dari Pemrograman Berbasis Objek adalah bagaimana menyembunyikan ciri karakteristik (variabel) yang dimiliki oleh class

(enkapsulasi). Salah satu penyembunyian karakteristik adalah dengan menggunakan "Property" di dalam .NET Framework. Sehingga user dibatasi untuk mengakses ciri khas class melalui "Property". Property .NET Framework dimaksudkan untuk menyediakan metode get atau set terhadap variabel tertentu. User akan mengakses variabel tersebut melalui "Property".

Latihan program 6.4 :

```
using System;
namespace Bab6
public class Mahasiswa65 {
private int nim;
public int NomorIndukMahasiswa {
get { return nim; }
set { nim = value ; }
}
}
public class PropertyMahasiswa {
public static void Main( string [] args) {
Mahasiswa65 siswa = new Mahasiswa65 ();
siswa.NomorIndukMahasiswa = "1999" ;
Console .WriteLine( "Nomor Induk Mahasiswa= {0}" ,
siswa.NomorIndukMahasiswa);
// Ganti Nomor Induk Mahasiswa
siswa.NomorIndukMahasiswa = "2009" ;
Console .WriteLine( "Nomor Induk Mahasiswa= {0}" ,
siswa.NomorIndukMahasiswa);
}
}
}
```

BAB VII
CLASS LANJUTAN

Tujuan :

1. Mengetahui sekilas mengenai pewarisan, abtract class, interface
2. Mengetahui penerapan pewarisan, abtract class, interface

7.1 Pewarisan

Pewarisan (inheritance) adalah kemampuan class untuk menurunkan sifat-sifat ke class turunannya. Class turunan dapat diturunkan hanya satu class kecuali interface. Struktur pewarisan:

<access_modifier> class <nama_class_turunan> : <nama_class_induk>

Latihan program 7.1 :

```
using System;
namespace Bab7 {
public class Induk {
protected int nim;
public Induk() {
nim = 1999;
}
}
public class Turunan : Induk {
public Turunan() {
Console .WriteLine( "NIM: {0}" , nim);
}
public static void Main( string [] args) {
Turunan turunan = new Turunan ();
}
}
}
```

7.2 Abstract Class

Abstract class adalah class yang belum lengkap strukturnya. Class ini belum dapat menciptakan objek, dalam artian abstract class baru class setengah jadi. Yang belum jadi adalah fungsi-fungsi. Sehingga harus di override di kelas turunan-nya. Override adalah penulisan ulang fungsi di kelas turunannya. Turunan class hanya dapat diturunkan dari satu class induk.

Struktur pewarisan:

<access_modifier> abstract class <nama_class_turunan> : <nama_class_induk>

Latihan program 7.2 :

```
using System;
namespace Bab7 {
public abstract class AbstractClass {
protected int nim;
public abstract void setNIM( int nim); {
}
public class AbstractTurunan : AbstractClass {
public override void setNIM( int nim) {
this .nim = nim;
}
public int getNIM() {
return nim;
}
public static void Main( string [] args) {
AbstractTurunan at = new AbstractTurunan ();
at.setNIM(1999);
Console .WriteLine( "NIM: {0}" , at.getNIM());
}
}
}
```

7.3 Interface

Interdace adalah class yang belum lengkap strukturnya dan berisi definisi fungsi-fungsi belum implementasi. Interface ini belum dapat menciptakan objek, dalam artian interface baru class setengah jadi. Yang belum jadi adalah fungsi-fungsi. Sehingga harus di override di kelas turunan-nya. Override adalah penulisan ulang fungsi di kelas turunannya. Turunan class hanya dapat diturunkan dari beberapa interface. Struktur pewarisan:

<access_modifier> interface <nama_class_turunan> : <nama_interface_induk1>, <nama_interface_induk2>, ..., <nama_interface_indukN>

Latihan program 7.2 :

```
using System;
namespace Bab7 {
public interface Shape {
void Draw();
}
public class Rectangle : Shape {
public void Draw() {
Console .WriteLine( "Draw Rectangle" ); }
}
public class Cirle : Shape {
public void Draw() {
Console .WriteLine( "Draw Cirle" ); }
}
public class Implementasi {
public static void Main( string [] args) {
// Shape untuk rectangle
Shape shape = new Rectangle ();
shape.Draw();
// Shape untuk cirle
shape = new Cirle ();
shape.Draw();
}
```

}
}

7.4 Tugas Praktikum

1. Buatlah program pembelian barang dengan menggunakan kelas-kelas !

	DVD	Mainboard
Asus	**18500**	**650000**
GigaByte	**20000**	**700000**

2. Buatlah program untuk menghitung gaji harian seorang pegawai disuatu proyek Ved Corporation, dengan ketentuan :

   Input :

   Nama, NIP, Jumlah jam kerja. Ketentuan: Jumlah jam kerja = 8 jam atau kurang. Jika jumlah jam kerja > 8 maka Lembur = jjk – 8 x 3/2 x honor/jam Selain itu lembur=0 dan honor/jam=Rp. 6000 Jika jumlah jam lembur >=3.5, diberi uang makan = Rp. 7000 selain itu tidak mendapat uang makan. Honor = jjk x honor/jam Total = honor + lembur + uang makan

   Gunakan fungsi dan struktur dalam array

Output :

No	Nama	NIP	JK	Honor/jam	Honor Lembur	Uang makan	Total
1	.....	.....	....	.....	.....	.....	.....
2	.....	.....	....	.....	.....	.....	.....
3	.....	.....	....	.....	.....	.....	.....