M118– Analysieren und strukturiert implementieren

1 Übersicht und Einführung .NET

1.1.1 Kompilierende versus interpretierende Programmiersprachen

Interpreter:

übersetzen Programme zeilenweise, d.h., die Befehle werden Zeile für Zeile in Maschinensprache übersetzt und vom Prozessor ausgeführt. Aus diesem Grunde laufen die Programme *langsamer* als kompilierte Programme ab, und es können weniger Optimierungen des Programmcodes vorgenommen werden.

Compiler

übersetzen Programme vollständig in Maschinensprache und speichern das Ergebnis in einer ausführbaren Datei (z.B. exe oder dll). Während der Kompilierung optimiert der Compiler die Programmgrösse und –geschwindigkeit. Dadurch laufen diese Programme *schneller* ab als interpretierte Programme.

1.1.2 Die Idee hinter .NET

- Plattformübergreifend
- sprachübergreifend

Wichtigstes Merkmal dieser Umgebung ist, dass die Quelltexte der verschiedenen Programmiersprachen zunächst mit Hilfe entsprechender Compiler in eine einheitliche Zwischensprache (Common Language oder Intermediate Language IL) übersetzt werden. Dieser IL-Code ist sprach- und plattformunabhängig. Alle NET-Programmiersprachen müssen bestimmten Richtlinien (CLS – Common Language Specification) entsprechen. Dazu gehört auch ein einheitliches Datentyp System.
Die Laufzeitumgebung übernimmt nicht nur das Übersetzen und Optimieren des Programmcodes, sondern auch Sicherheits- und Versionsprüfung, Fehler- und Speicherverwaltung. Deshalb wird der auf diese Weise behandelte Programmcode auch „Managed Code“ (verwalteter Code) genannt. Er lässt sich sicherer und stabiler ausführen als herkömmlicher „Unmanaged Code“.

1.2 Die Entwicklung und Darstellung eines Algorithmus

EVA-Prinzip
Eingabe(Daten) → Verarbeitung → Information (Ausgabe)

Algorithmus:

Ein Algorithmus ist eine präzise und eindeutige Beschreibung eines allgemeinen Schemas, das unter Verwendung von endlich vielen Schritten zur Lösung gleichartiger Aufgaben schrittweise abgearbeitet werden kann.

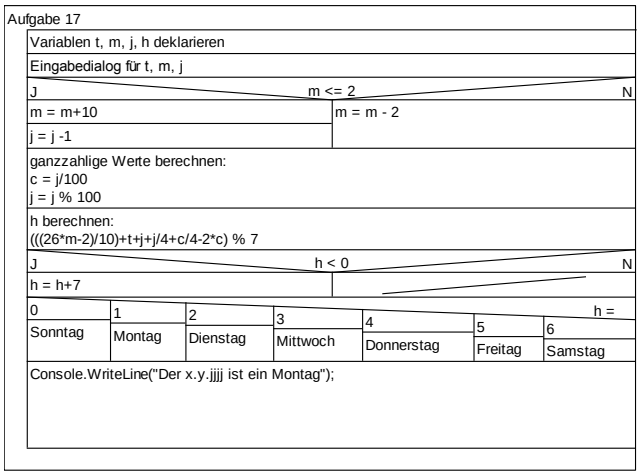
kurz:
Algorithmus = exakte Arbeitsanweisung für einen PC

1.2.1 Programmablaufplan (PAP oder Flussdiagramm)

Programmablaufpläne sind grafische Darstellungen mit Hilfe genormter Symbole. Sie sind weit verbreitete Hilfsmittel bei der Programmentwicklung. Die Programmstruktur lässt sich bildhaft als Ablauf darstellen. Die Pfeile zeigen dabei die Richtung der Verarbeitung an.

[Bild B1: Seite 8 → PAP-Symbole]

1.2.2 Struktogramme (Nassi-Schneiderman)



1.2.3 Pseudocode

Bei dieser Darstellungsart wird die Programmlogik unter Verwendung von „normalen“ Sätzen beschrieben.

2 Prozeduren & Funktionen

2.1 Unterprogrammaufrufe (Prozeduren & Funktionen)

Unterprogramme können von übergeordneten Unterprogrammen aufgerufen werden. Der Aufruf eines Unterprogramms erfolgt über den Unterprogrammnamen. Nach Abarbeitung der letzten Anweisung innerhalb eines Unterprogramms

wird die Programmausführung mit der nächsten Anweisung des aufrufenden Unterprogramms fortgesetzt.

Warum Unterprogramme?
Wenn Sie mehrmals bestimmte Aufgaben innerhalb des Programms ausführen müssen, kann man dafür Unterprogramme erstellen. Diese Arbeit ist nur einmal notwendig, das Unterprogramm kann jedoch beliebig oft aufgerufen werden.
Unterprogramme können in selbständigen Bibliotheken bzw. Assemblies (dll) abgelegt und damit auch anderen Anwendungen zugänglich gemacht werden.

mit Parameterübergabe:
Call by Value (Wertübergabe)
Wertübergabe heisst, das Unterprogramm arbeitet mit einer Kopie der per Parameter übergebenen Variable weiter. Bei dieser Form der Parameterübergabe bleibt der Wert ausserhalb der Methode unverändert erhalten, auch wenn innerhalb der Methode der Wert verändert wird. Er ist somit geschützt.

Globale Variablen werden immer unmittelbar nach dem Programmstart erstellt (Speicherplatz reserviert). Dies geschieht auch dann, wenn die Variablen gar nicht gebraucht werden.
Die gleiche Variable könnte von verschiedenen Orten gleichzeitig verwendet werden.

➔ Gefahr, dass der Inhalt ungewollt überschrieben/verändert wird.

Parameterübergabe per Referenz:
Call by Reference

Prozeduren mit Rückgabewerten
Prozeduren können auch einen Rückgabewert eines bestimmten Datentyps haben. Man spricht dann von Funktionen.
→ return…

3 Erweiterte Datentypen

3.1 Enumeratoren

Mittels Enumeratoren kann man neue, eigene Datentypen definieren. Den Variablen dieses neuen Datentyps können die im Enumerator definierten sprechenden Bezeichnungen, anstelle der Werte, zugewiesen werden.

3.2 Datenfelder (Arrays)

In einem Array können mehrere Variablen desselben Datentyps erstellt werden. Die so entstandenen Variablen liegen lückenlos aneinander im Arbeitsspeicher. Dadurch ist ein sehr schneller Zugriff möglich. Mit einem Index kann auf die einzelnen Elemente zugegriffen werden.

4 Testen

4.1 Warum Testen?

Fehler haben schwerwiegende Folgen:

- Kosten für Reparatur oder Ersatz des fehlerhaften Produkts
- Ertragseinbusse durch Minderung
- Fehlerfreiheit bezahlt
- Schadenersatzforderungen
- Konventionalstrafe

Imageschaden
Strafrechtlich

- immer personenbezogen

Schäden an Mensch und Natur sind mit Geld überhaupt nicht reparierbar…

Fehler sind oftmals so versteckt, dass sie nur durch systematisches Testen gefunden werden können. Der Entwickler ist dabei in seiner Sicht oftmals so eingeschränkt, dass eine andere Person das Testen übernehmen sollte.

4.2 Grundlagen

Testen ist der Prozess, ein Softwareprodukt durch manuelle oder automatisierte Hilfsmittel zu bewerten, um damit die Erfüllung der spezifizierten Anforderungen nachzuweisen.

Durch den Test wird das Ziel verfolgt, Fehler aufzudecken.
Ein Fehler ist jede Abweichung von der geforderten Qualität (z.B. Funktionalität, Stabilität).

<u>Testprinzipien</u>

- Fehler möglichst frühzeitig aufdecken
- Aus Fehlern lernen und sie zukünftig vermeiden.
- Bei komplexen Testobjekten unabhängige Tests durchführen
- Testziele erreichbar und messbar formulieren
- Testfälle professionell handhaben
- Testaktivitäten planen

- Testaktivitäten dokumentieren Testpläne, Testspezifikationen, Testfallbeschreibungen, Test- und Fehlerprotokolle

4.4 Testfälle ermitteln

Blackbox-Testing (Testen anhand von Daten)
Bei diesen Methoden werden die Testfälle aus der vorliegenden Spezifikation oder aber aus der Oberflächenstruktur (Erfassungsmasken) hergeleitet. Die interne Struktur des Testobjektes wird bei der Zusammenstellung der Testfälle nicht berücksichtigt. Von daher sind BlackBox-Methoden insbesondere auch für Anwender geeignet.

- Äquivalenzklassenanalyse
- Grenzwertanalyse
- Schnittstellenanalyse
- Fehlererwartung

Whitebox-Testing (Testen der Logik)
Bei diesen Methoden werden die Testfälle mit Kenntnis der internen Struktur des Testobjektes (Sourcecode-Listing) entwickelt, d.h, diese Gruppe von Methoden kann nur von Entwicklern eingesetzt werden.

- Anweisungsüberdeckung
- strukturierte Pfadüberdeckung
- Bedingungsüberdeckung
- Kombination der Methoden

Dokumententest
Review
Ein Review ist ein geplanter und strukturierter Prozess, in dem Arbeitsergebnisse (Dokumente) einem Team von Gutachtern mit dem Ziel präsentiert werden, Fehler aufzudecken.

5 Such- und Sortieralgorithmen

5.2 Suchverfahren

5.2.1 Sequentielle Suche

→ von vorne nach hinten

5.2.2 Binäre Suche

→ braucht eine sortierte Reihenfolge

5.3 Sortierverfahren

5.3.2 Sortieren durch Einfügen (InsertionSort)

Algorithmus der typisch menschlichen Vorgehensweise, etwa beim Sortieren eines Stapels von Karten.

1. Starte mit der ersten Karte einen neuen Stapel
2. Nimm jeweils die nächste Karte des Originalstapels und füge diese an der richtigen Stelle in den neuen Stapel ein.

5.3.3 Sortieren durch Selektion (SelectionSort)

Die Idee ist hierbei, jeweils das grösste Element auszuwählen und an das Ende der Folge zu setzen. Dies wird in jedem Schritt mit einem jeweils um eins verkleinerten Bereich der Folge ausgeführt, so dass sich am Ende der Folge die bereits sortierten Elemente sammeln.

5.3.4 Sortieren durch Vertauschen (BubbleSort)

Das Grundprinzip besteht darin, die Folge immer wieder zu durchlaufen und dabei benachbarte Elemente, die nicht der gewünschten Sortierreihenfolge entsprechen, zu vertauschen. Elemente, die grösser sind als ihre Nachfolger, überholen diese daher und steigen zum Ender der Folge auf. „Blasen“-Prinzip.

5.3.5 Sortieren durch Mischen (MergeSort)

Teile und herrsche
→ Sortieren durch Quicksort

6 Dateien und Streams

6.1 Operationen mit Verzeichnissen und Dateien

Die Klassen befinden sich im System.IO-Namensraum → using System.IO;

➔ siehe Code

7 Kombination der einzelnen Blöcke

Grafikprogrammierung, siehe Block 7 in den Unterlagen…

8 Theorie-Fragen

Beschreiben Sie mit eigenen Worten die Funktionweise des Bubblesort-Algorithmus:

Bubblesort arbeitet nach dem Blasen-Prinzip. Grosse Blasen überholen die kleinen Blasen.
Die Folge wird immer wieder durchlaufen. Beim Bubblesort werden immer zwei benachbarte Elemente miteinander verglichen. Ist das erste Element grösser als das zweite Element, wird getauscht. Das Flag wird gesetzt. Die Folge wird solange durchlaufen, bis keine Vertauschung mehr auftritt (d.h. das Flag nicht mehr gesetzt ist).

Unterschied zwischen White- & Blackbox-Testing:

Blackbox-Testing: Testen anhand von Daten
Whitebox-Testing: Testen der Logik

Der Unterschied zwischen Black- & Whitebox-Testing besteht darin, dass beim Blackbox-Testing die Software mithilfe von Daten, die eingegeben werden, getestet wird. Diese Arbeit können auch normale User übernehmen, da keine Programmierkenntnisse verlangt sind. Diese wiederum braucht man beim Whitebox-Testing, da dort die Logik des Quellcodes getestet wird.

9 Code

Was liefern die folgenden Programme als Resultat?

- Übungen
- Prüfungen

Allgemeiner Prüfungsquellcode…

10 Sortierverfahren

10.1 Insertion Sort

```
private void InsertionSort()
        {
            //Hier müssen Sie den Programmcode ergänzen
            for(int i=1;i<array.Length;i++)
            {
                int j=i;
                int m = array[i];
                //für alle Elemente links vom Marker Feld
                while(j>0 && array[j-1]>m)
                {
                    //verschiebe alle grösseren elemente nach hinten
                    array[j] = array[j-1];
                    j--;
                }
                //setze m auf das freie Feld
                array[j]=m;
            }
        }
```

10.2 Selection Sort

```
private void SelectionSort()
        {
            //Hier müssen Sie den Programmcode ergänzen
            int marker = array.Length - 1;
            while (marker >= 0)
            {
                //bestimme grösstes Element
                int max = 0;
                for (int i = 1; i <= marker; i++)
                {
                    if (array[i] > array[max])
                        max = i;
                }
                //tausche array[marker] mit diesem element
                swap(array, marker, max);
                marker--;
            }

        }
```

10.2.1 Swap

```
private void swap(int[] array, int marker, int max)
        {
            int temp = array[marker];
            array[marker] = array[max];
            array[max] = temp;
        }
```

10.3 Bubblesort

```
private void BubbleSort()
        {
            //Hier müssen Sie den Programmcode ergänzen
            bool swapped;
            do
            {
                swapped = false;
                for (int i = 0; i <= array.Length - 2; i++) //-2, weil sonst der Index überschritten wird
                {
                    if (array[i] > array[i + 1])
                    {
                        //Elemente vertauschen
                        swap(array, i, i + 1);
```

```
                    swapped = true;
                }
            }
        } while (swapped); //solange Vertauschung auftritt
```

10.4 Sequentielle Suche

```
private int SequentielleSuche()
        {
            //Hier müssen Sie den Programmcode ergänzen
            int key = Convert.ToInt32(txtZuSuchendeZahl.Text);
            for (int i = 0; i < array.Length; i++)
            {
                if (array[i] == key)
                    return i;
            }
            return -1;
           }
```

10.5 Binäre Suche

```
private int BinaereSuche()
        {
            //Hier müssen Sie den Programmcode ergänzen
            int key = Convert.ToInt32(txtZuSuchendeZahl.Text);
            int u = 0, o = array.Length - 1;
            while (u <= o)
            {
                int m = (u + o) / 2;
                if (array[m] == key)
                    return m;
                else if (array[m] > key)
                    o = m;
                else
                    u = m;
            }
            return -1;
        }
```

10.6 Mergesort

```
private void msort(int[] array,int le, int ri)
        {
            //Hilfe zu Mergesort
            int i, j, k;
            int[] b = new int[array.Length];
            if(ri > le)
            {
                //zu sortierendes Feld teilen
                int mid = (ri+le)/2;
                //Teilfelder sortieren
                msort(array,le,mid);
                msort(array,mid+1,ri);
                //Hilfsfeld aufbauen
                for(k=le;k<=mid;k++)
                    b[k]=array[k];
                for(k=mid;k<ri;k++)
                    b[ri+mid-k]=array[k+1];
                //Ergebnisse mischen über Hilfsfeld d
                i = le;
                j=ri;
                for(k=le;k<=ri;k++)
                {
                    if(b[i] < b[j])
                        array[k] = b[i++];
                    else
                        array[k] = b[j--];
                }
            }
        }

        private void MergeSort(int[] array)
        {
            //Hier müssen Sie den Programmcode ergänzen
            msort(array,0,array.Length-1);
        }
```

10.7 Quicksort

```
private void qsort(int[] array,int le, int ri)
        {
            //Hilfe zu Quicksort
            int lo = le, hi = ri;
            if (hi > lo)
            {
                //Pivot Element bestimmen
                int mid = array[(lo + hi) / 2];
                while (lo <= hi)
                {
                    /*Erstes Element suchen das grösser oder gleich dem
                     Pivot Element ist, beginnend vom linken Index*/
                    while (lo < ri && array[lo] < mid)
                        ++lo;
                    /*Element suchen das kleiner oder gleich dem
                     Pivot Element ist, beginnend vom rechten Index*/
                    while (hi > le && array[hi] > mid)
                        --hi;
                    //Wenn Indexe nicht gekreuzt -> Inhalt vertauschen
                    if (lo < hi)
                    {
                        swap(array, lo, hi);
                        ++lo;
                        --hi;
                    }
                    //linke Partition sortieren
                    if (le < hi)
                        qsort(array, le, hi);
                    //rechte Partition sortieren
                    if (lo < ri)
                        qsort(array, lo, ri);
                }
            }

        }

        private void QuickSort(int[] array)
        {
            //Hier müssen Sie den Programmcode ergänzen
            msort(array, 0, array.Length - 1);

        }
```

10.8 Strukturen

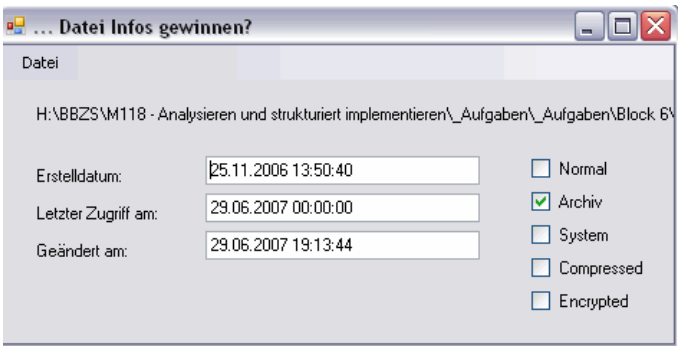
```
public struct Person
        {
            public string vorName, nachName;
            public DateTime geburt;
            public bool student;
        }

Person p1;
P1.vorName = „Flavio“;
```

10.9 Enumeratoren

Public enum eTage {Montag=1, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, Sonntag};

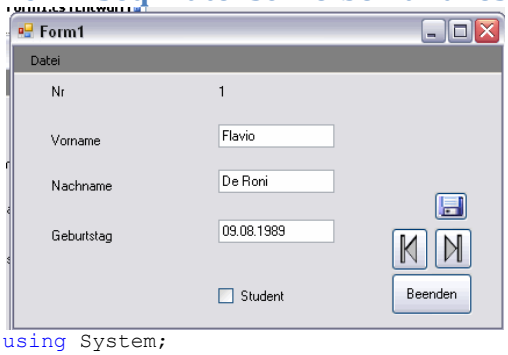
eTage d1 = eTage.Montag;

10.10 Datei-Infos lesen




```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace Datei_Infos_Lesen
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }


        private void öffnenToolStripMenuItem_Click(object sender, EventArgs e)
        {
            string datName = "";
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
                datName = openFileDialog1.FileName;
            label1.Text = datName;
            textBox1.Text = File.GetCreationTime(datName).ToString();
            textBox2.Text = File.GetLastAccessTime(datName).ToString();
            textBox3.Text = File.GetLastWriteTime(datName).ToString();

            FileAttributes attbs = File.GetAttributes(datName);

            if (attbs == (attbs | FileAttributes.Normal))
                checkBox1.Checked = true;
            else checkBox1.Checked = false;

            if (attbs == (attbs | FileAttributes.Archive))
                checkBox2.Checked = true;
            else checkBox2.Checked = false;

            if (attbs == (attbs | FileAttributes.System))
                checkBox3.Checked = true;
            else checkBox3.Checked = false;

            if (attbs == (attbs | FileAttributes.Compressed))
                checkBox4.Checked = true;
            else checkBox4.Checked = false;

            if (attbs == (attbs | FileAttributes.Encrypted))
                checkBox5.Checked = true;
            else checkBox5.Checked = false;

        }

        private void beendenToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
```

10.11 Seq. Datei schreiben und lesen

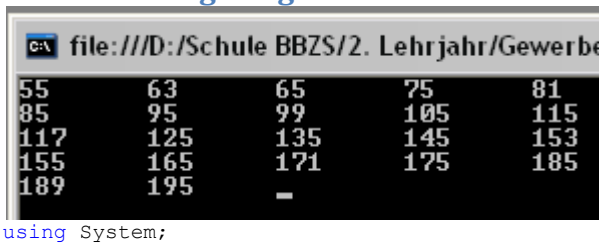



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace Übung_4___Seq.Dat_rW
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        public struct Person
        {
            public string vorName, nachName;
            public DateTime geburt;
            public bool student;
        }

        private int pmax = 20;
        private int pos = 0;
        private Person[] pListe;
        private string path = "file.bin";

        private void readFile()
        {
            FileStream rStream = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Read);
            BinaryReader binReader = new BinaryReader(rStream);
            if (rStream.Length > 0) //nicht beu neu erzeugter Datei
            {
                for (int i = 0; i < pmax; i++)
                {
                    pListe[i].vorName = binReader.ReadString();
                    pListe[i].nachName = binReader.ReadString();
                    pListe[i].geburt = Convert.ToDateTime(binReader.ReadString());
                    pListe[i].student = binReader.ReadBoolean();
                }
            }
            binReader.Close();
            rStream.Close();
        }
        private void speichern()
        {
            pListe[pos].nachName = txt_nachname.Text;
            pListe[pos].vorName = txt_vorname.Text;
            pListe[pos].geburt = DateTime.Parse(txt_gebtag.Text);
            pListe[pos].student = checkBox1.Checked;
        }

        private void writeFile()
        {
            FileStream wStream = new FileStream(path, FileMode.OpenOrCreate, FileAccess.Write);
            BinaryWriter binWriter = new BinaryWriter(wStream);
            for (int i = 0; i < pmax; i++)
            {
                binWriter.Write(pListe[i].vorName);
                binWriter.Write(pListe[i].nachName);
                binWriter.Write(pListe[i].geburt.ToShortDateString());
                binWriter.Write(pListe[i].student);
            }
            binWriter.Flush(); //Puffer => Disk
            binWriter.Close();
            wStream.Close();
```

```
}

private void anzeigen()
{
    lblNr.Text = Convert.ToString(pos+1);
    txt_vorname.Text = pListe[pos].vorName;
    txt_nachname.Text = pListe[pos].nachName;
    txt_gebtag.Text = pListe[pos].geburt.ToShortDateString();
    checkBox1.Checked = pListe[pos].student;
}



private void Form1_Load(object sender, EventArgs e)
{
    pListe = new Person[pmax];
    //Datensätze im Array initialisieren
    for (int i = 0; i < pmax; i++)
    {
        pListe[i].vorName = "";
        pListe[i].nachName = "";
        pListe[i].geburt = Convert.ToDateTime("1.1.2000");
        pListe[i].student = false;
    }
    readFile();//File lesen
    anzeigen();//anzeigen
 }

private void btn_end_Click(object sender, EventArgs e)
{
    this.Close();
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    speichern();
    writeFile();
}

private void btn_save_Click(object sender, EventArgs e)
{
    speichern();
}

private void btn_next_Click(object sender, EventArgs e)
{
    if (pos < pListe.Length - 1)
    {
        speichern();
        pos++;
        anzeigen();
    }
}

private void btn_last_Click(object sender, EventArgs e)
{
    if(pos > 0)
    {
        speichern();
        pos--;
        anzeigen();
    }

}

private void beendenToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}

private void zurückToolStripMenuItem_Click(object sender, EventArgs e)
{
    speichern();
    pos--;
    anzeigen();
}

private void weiterToolStripMenuItem_Click(object sender, EventArgs e)
{
    speichern();
    pos++;
    anzeigen();
```

```
        }

        private void speichernToolStripMenuItem_Click(object sender, EventArgs e)
        {
            speichern();
        }

    }
}
```

10.12 Prüfungsaufgabe 5 von alter Prüfung




```
using System;
using System.Collections.Generic;
using System.Text;

namespace Prüfungsaufgabe_5
{
    class Program
    {
        static void Main(string[] args)
        {
            int Startwert = 50;
            int Endwert = 200;
            int zaehler=1;

            const int ZAHLEN_PRO_ZEILE = 5;
            for (int i = Startwert; i <= Endwert; i++)
            {
                if(i%5==0||i%9==0)
                {
                    if (i % 2 == 1)
                    {
                        Console.Write(i + "\t");
                        zaehler++;
                    }
                }
                if (zaehler > ZAHLEN_PRO_ZEILE)
                {
                    Console.WriteLine();
                    zaehler = 1;
                }
            }
            Console.ReadLine();
        }
    }
}
```