Aufgabe L02_ProgrammierenInC#_1

Betrachtet die Referenzdokumentation zu den eingebauten Datentypen und die von dort aus in die einzelnen Typen verzweigenden Links.

- Wieviel Speicherplatz in bytes benötigt die Zeichenkette "Hello, World" ?
  ein Zeichen = char. Ein char = 16 bit → 192 bit = 24 byte

- Vergleicht den Umfang der darstellbaren Zahlen zwischen int und short, sowie zwischen float und double. Wie groß ist jeweils der größte und der kleinste Wert? Wie groß ist der kleinste *positive* mit float darstellbare Wert?

  → Int 32 bit: kleinster Wert: – 2.147.483.648; größter Wert: +2.147.483.647
  → Short 16 bit: kleinster Wert: -32.768; größter Wert: +32.767
  → Float 32 bit: kleinster: -3,4*10(hoch38); größter: +3,4*10(hoch38)
  → Double 64 bit: kleinster: -1,7*10(hoch308); größter: +1,7*10(hoch308)

- Was heißt Fließkommazahl und was heißt Festkommazahl? Für welchen Anwendungsbereich ist decimal besonders geeignet? Warum?

  → Fließkommazahl: (float) "einfacher" Genauigkeit - benötigt 32 bit Speicherplatz
  → Festkommazahl: (decimal) mit hoher Genauigkeit - benötigt 128 bit Speicherplatz

Regeln für Identifizierer

- Warum darf der Programmierer **horst** seine Lieblingsvariable nicht nach seiner Freundin **else** benennen? :-)

  → Else ist ein Schlüsselwort

Initialisierung

- Erzeugt in Visual Studio Code ein neues C#-Projekt und fügt oben stehende Deklarationen und Initialisierungen der Variablen i, pi, und salute ein

```
int i = 42;
double pi = 3.1415;
string salute = "Hello, World";
```

- Verändert die Deklarationen so, dass var statt der Typen verwendet wird und überzeugt Euch, dass der Compiler den Code korrekt übersetzt.

```
var i = 42;
var pi = 3.1415;
var salute = "Hello, World";
```

- Was unterscheidet eine int-Konstante von einer double-Konstanten?

  → double ist ein Fließkommadatentyp und speichert Zahlen mit Nachkommastellen.
  → int ist ein integraler Datentyp und speichert ganze Zahlen, ohne Nachkommastellen.

Arrays

- Im o.a. Code-Schnipsel werden drei Array-Variablen deklariert und initialisiert. Wie heißen die Variablen, was ist der jeweilige Grund-Typ und wieviel Speicherplätze sind jeweils reserviert worden?

```
int[] ia = new int[10];
char[] ca = new char[30];
double[] da = new double[12];
```

int = Variable: ia; Grund-Typ: integer; reservierte Speicherplätze: 10
char = Variable: ca; Grund-Typ: character; reservierte Speicherplätze: 30
double = Variable: da; Grund-Typ: double; reservierte Speicherplätze: 12

Zugriff auf einzelne Array-Speicherplätze

- Erzeugt einen Array vom Grund-Typ double, der drei Speicherplätze enthält in denen in der angegebenen Reihenfolge
  die Zahl PI,
  die Eulersche Zahl und
  die Kepler-Konstante
  enthalten sind.

  →

```
int[] ia = {1, 0, 2, 9, 3, 8, 4, 7, 5, 6};
double[] ergebnis = {3.14159265359, 2.71828, 2.97*(10^-19)};
Console.WriteLine(ergebnis[1]);
Console.WriteLine(ergebnis[2]);
Console.WriteLine(ia.Length);
```

Strings

- Fügt den o.a. Beispielcode zu Strings einem C#-Projekt zu und überprüft jeweils die Variableninhalte
  von meinString, c, a_eq_b, a_eq_c und zeichen mit Console.WriteLine oder mit dem Debugger.

→

```
string meinString = "Dies ist ein String";
string a = "Dies ist ";
string b = "ein String";
string c = a + b;
string a = "eins";
string b = "zwei";
string c = "eins";
bool a_eq_b = (a == b);
bool a_eq_c = (a == c);
string meinString = "Dies ist ein String";
char zeichen = meinString[5];
Console.WriteLine(meinString);
//Console.WriteLine(c);
//Console.WriteLine(a_eq_b);
//Console.WriteLine(a_eq_c);
//Console.WriteLine(zeichen);
```

Verzweigungen

- Schreibt ein C#-Programm, das zwei Zahlen von der Konsole einliest. Diese sollen verglichen werden. Ist die erste größer als die zweite, soll der Text "a ist größer als b" ausgegeben werden, ansonsten der Text "b ist größer als a".

→

```
int main (void) {

   int a;
   int b;

   a=0;
   b=0;

   Console.WriteLine(„Gebe die erste Zahl ein);
   scanf(„%d“, &a);
   Console.WriteLine(„Gebe die zweite zahl ein);
   scanf(„%d“, &b);


if(a<b){
   Console.WriteLine(„%d<%d\n,a,b);
}

if(a>b){
   Console.WriteLine(„%d ist größer als %d\n,a,b);
}
```

```
if(a==b){
   Console.WriteLine(„%d ist gleich groß wie %d\n,a,b);
```

- Ändert das Programm aus dem letzten TODO so ab, dass wenn die erste Zahl größer drei und die zweite Zahl gleich 6 sechs ist, der Text "Du hast gewonnen" ausgegeben wird. Ansonsten soll "Leider verloren" ausgegeben werden.

  →

```
if(a>3 && b==6){
   Console.WriteLine(„Du hast gewonnen“);
}
```

Switch/Case

- Erweitert den Code um einen weiteren Switch für eine Zahl Eurer Wahl.
- Ändert den Typ der Variablen i von int nach string und ändert die case-Labels, so dass diese aus Strings bestehen.
- Was passiert, wenn man an einer Stelle das break vergisst? Denkt euch Fälle aus, bei denen das sinnvoll sein kann.
- Versucht, die oben mit der switch / case Anweisung implementierte Funktionalität mit if/ else Anweisungen zu implementieren.

→

```
int i = int.Parse(Console.ReadLine());
switch (i)
{
   case a:
      Console.WriteLine("Du hast a eingegeben");
      break;
   case b:
      Console.WriteLine("b war Deine Wahl");
      break;
   case c:
      Console.WriteLine("Du tipptest eine c");
      break;
   default:
      Console.WriteLine("Die Zahl " + i + " kenne ich nicht");
      break;
}
```

Schleifen

- Erzeugt ein C# Programm, das in einer while-Schleife die Zahlen von 1 bis 10 auf der Konsole ausgibt.

  →

```
int [] zahleneinsbiszehn= {1,2,3,4,5,6,7,8,9,10);
int a=0;
while (a<10){
   Console.WriteLine(zahleneinsbiszehn[x].ToString());
   a++;
}
```

- Wie lauten hier die Teile <INITIALISIERUNG>, <BEDINGUNG> und <INKREMENT>?

  → Initialisierung = int a=0;
  → Bedingung = while(a<10)
  → Inkrement = a++;

- Lasst o.a. Code laufen. Was tut er
  ➔ Der Inhalt des Arrays „someString" wird ausgegeben
- Wandelt die for-Schleife in eine while-Schleife um
- Lasst statt der erste 5 *alle* in someStrings gespeicherten Werte ausgeben, indem die Bedingung statt eines konstanten Wertes die jeweilige Anzahl der Inhalte von someStrings ausgibt (geht mit someStrings.Length).

  →

```
 string[] someStrings =
 {
   "Hier",
   "sehen",
   "wir",
   "einen",
   "Array",
   "von",
   "Strings
 };

   int i =1;
 while (i<someStrings.Length){
   Console.WriteLine(someStrings[i]);
 }
```

Schleifenausstieg/Zeitpunkt der Bedingungsprüfung

- Lasst die obigen Beispiele für do while und break mit dem Array aus dem for-Beispiel laufen und überzeugt Euch, dass die gleiche Ausgabe erzeugt wird.
- Sind die beiden Beispiele wirklich äquivalent zur for-Schleife?
    - Verändert in allen drei Fällen die Abbruchbedingung so, dass *alle* Array-Einträge ausgegeben werden und zwar in Abhängigkeit von der Anzahl der Array-Einträge (someStrings.Length).

```
int i = 0;
  do ( ){
    Console.WriteLine(someStrings[i]);
    i++;
  }
while (i < someStrings[i]);

int j = 0;
while (true){
    Console.WriteLine(someStrings[j]);
    if (j >= someStrings.Length-1)
      break;
    j++;
  }
```

    - Was passiert in den drei Fällen, wenn someStrings *keine* Einträge enthält?
      → Fehlermeldung: Unhandled Exception