Aufgabe 1: Grundlagen (5 Punkte)

Welche der folgenden Behauptungen über die Programmiersprache Go sind wahr, welche falsch?

Behauptung	wahr	falsch
`float` ist ein Datentyp in Go.		**X**
Ein `string` kann mit `append` verlängert werden.		**X**
Funktionsparameter müssen immer verwendet werden.		**X**
Variablen können oft ohne explizite Typangabe deklariert werden.	**X**	
Funktionen haben immer einen Namen		**X**

Anmerkung: Korrekt angekreuzte Zeilen geben einen Punkt, für falsch angekreuzte Zeilen wird ein Punkt abgezogen.

Aufgabe 2: Signaturen (10 Punkte)

Betrachten Sie das folgende Programmfragment:

```
1   Foo1(Foo2(42))
2   x1 := Foo3(true, 23)
3   if len(x1) > 0 {
4     fmt.Println(Foo2(75 - Foo4()))
5   }
6   return x1 == "" && Foo1(len(x1))
```

Welche Signaturen haben die Funktionen `Foo1` bis `Foo4`? Welchen Rückgabetyp liefert das `return`?

Anmerkung: Die *Signatur* einer Funktion ist die erste Zeile, in der die Argument- und Rückgabetypen definiert werden. Hier ist also gefragt, welche Typen die Funktionen erwarten und liefern. Sie können davon ausgehen, dass Funktionen, deren Ergebnis nicht verwendet wird, auch keinen Rückgabetyp haben.

Lösung

- `Foo1`: `func(int) bool`
- `Foo2`: `func(int) int`
- `Foo3`: `func(bool, int) string`
- `Foo4`: `func() int`
- `return`: `func() bool`

Aufgabe 3: Fehlersuche: Compilerfehler (10 Punkte)

Der folgende Code enthält eine Reihe an Fehlern, durch die er nicht compiliert. Markieren Sie alle Zeilen, die einen Fehler enthalten und erläutern Sie kurz, was jeweils falsch ist.

```
1  package fehlersuche1
2
3  import "fmt"
4
5  func Never {
6    fmt.Println("Never")
7  }
8
9  func Count(n int) string {
10   if n = 0 {
11     return ""
12   }
13   return Count(n-1) += fmt.Sprint(n)
14 }
15
16 func Upwards(s int) {
17   Println(s + Count(5))
18 }
```

Hinweis: Es geht hier nur um Syntaxfehler. Für jede falsch markierte Zeile gibt es Punktabzug!

Lösung

Hier ist eine Version, in der die Fehler markiert und korrigiert sind.

```
1  package fehlersuche1
2
3  import "fmt"
4
5  func Never() { // Klammern bei Funktion fehlen.
6    fmt.Println("Never")
7  }
8
9  func Count(n int) string {
10   if n == 0 { // `==` statt `=`
11     return ""
12   }
13   return Count(n-1) + fmt.Sprint(n) // `+=` in return nicht erlaubt
14 }
15
16 func Upwards(s string) { // Typfehler: `s` müsste `string` sein.
17   fmt.Println(s + Count(5)) // `fmt.` fehlt
18 }
```

Aufgabe 4: Fehlersuche: Inhaltliche Fehler (5 Punkte)

Die folgende Funktion ist zwar syntaktisch korrekt, sie erfüllt aber nicht ihre Aufgabe. Erläutern Sie den/die Fehler und machen Sie einen Vorschlag zur Korrektur.

```
1  // AllEqual liefert true, falls alle Zeichen in `s` gleich sind.
2  func AllEqual(s []int) bool {
3    for el := range s {
4      if el != s[0] {
5        return false
6      }
7    }
8
9    return len(s) != 0
10 }
```

Anmerkung: Ihre Korrektur muss nicht syntaktisch korrekt sein. Eine Erklärung in Worten genügt.

Lösung

Eine korrekte Version der Funktion wäre z.B. die Folgende. Hier sind auch die Fehler markiert:

```
1  func AllEqual(s []int) bool {
2    // `el` muss das Element sein, nicht der Zähler
3    for _, el := range s {
4      if el != s[0] {
5        return false
6      }
7    }
8
9    // Nicht die Länge prüfen, bei leerem String ist `true` richtig.
10   return true
11 }
```

Aufgabe 5: Programmverständnis (10 Punkte)

Erläutern Sie die Funktionsweise der folgenden Datenstruktur. Geben Sie eine Erklärung für die Struktur und erläutern Sie kurz den Zweck der gegebenen Methoden.

```
type S struct {
  els []string
}

func (s S) C(el string) bool {
  for _, e := range s.els {
    if e == el {
      return true
    }
  }
  return false
}

func (s *S) A(el string) {
  if !s.C(el) {
    s.els = append(s.els, el)
  }
}

func (s S) U(o S) S {
  r := S{}
  for _, e := range s.els {
    r.A(e)
  }
  for _, e := range o.els {
    r.A(e)
  }
  return r
}

func (s S) I(o S) S {
  r := S{}
  for _, e := range s.els {
    if o.C(e) {
      r.A(e)
    }
  }
  return r
}
```

Lösung

Die S implementiert eine Menge von Strings (im mathematischen Sinn). Die innere Liste ”els” speichert eine Liste von Strings, wobei jeder String nur einmal in der Liste vorkommen darf.

- **C** steht für ”Contains” und prüft, ob ein Element in der Menge enthalten ist.
- **A** steht für ”Add” und fügt ein Element zur Menge hinzu. Dabei wird geprüft, ob das Element bereits in der Menge enthalten ist.
- **U** steht für ”Union” und berechnet die Vereinigungsmenge zweier Mengen.
- **I** steht für ”Intersection” und berechnet die Schnittmenge zweier Mengen.

Aufgabe 6: Rekursion (10 Punkte)

Betrachten Sie die folgende Funktion:

```
1  func Foo(s string, c byte) int {
2    if len(s) == 0 {
3      return 0
4    }
5
6    count := 0
7    if s[0] == c {
8      count = len(s)
9    }
10
11   return count + Foo(s[1:], c)
12 }
```

Beschreiben Sie in Worten, was die Funktion berechnet.
Hinweis: Der Typ `byte` repräsentiert ein einzelnes Zeichen. Die Funktion erwartet also einen String und ein Zeichen als Eingabe. Mögliche Beispielrechnungen: `Foo("a",'a')`, `Foo("ab",'a')`, `Foo("aa",'a')`, `Foo("aba",'a')`.

Lösung

Die Funktion berechnet für den String `s`, wie viele Teilstrings von `s` mit dem Buchstaben `c` beginnen.

- Für einen leeren String ist das Ergebnis 0.
- Für `Foo("a",'a')` ist das Ergebnis 1.
- Für `Foo("ab",'a')` ist das Ergebnis 2, da die Teilstrings ”ab” und ”a” beide mit ’a’ beginnen.
- Für `Foo("aa",'a')` ist das Ergebnis 3, da die Teilstrings ”aa”, ”a” (der erste Buchstabe) und ”a” (der zweite Buchstabe) alle mit ’a’ beginnen.
- Für `Foo("aba",'a')` ist das Ergebnis 4, da die Teilstrings ”aba”, ”a” (der erste Buchstabe), ”a” (der dritte Buchstabe) und ”ab” alle mit ’a’ beginnen.