Softwarekonstruktion – Übung 5

5 Blackbox-Testen, Whitebox-Testen

5.1 Grundlagen Äquivalenzklassen und Grenzwert

Gegeben ist die Methode berechneIBAN mit folgender Signatur:

```
1 public String berechneIBAN(long Kontonummer){
2     // Der Code der Methode ist unbekannt
3 }
```

Die Methode berechneIBAN wird innerhalb einer Bank in Deutschland eingesetzt und berechnet die IBAN zu einer beliebigen Kontonummer der Bank; sie wird im Rückgabewert (String) zurückgegeben.

Der ermittelte String soll 22 Zeichen lang sein. Die ersten beiden Zeichen stehen für das Länderkennzeichen, in diesem Fall „DE“ für Deutschland. Anschließend folgt eine zweistellige Prüfziffer, die achtstellige Bankleitzahl und eine zehnstellige Kontonummer.

Hat die eingegebene Kontonummer weniger als zehn Stellen, so wird sie linksbündig mit Nullen auf zehn Stellen aufgefüllt.

Sollte es bei der Berechnung zu einem Fehler kommen, gibt die Methode berechneIBAN den String „Error“ zurück.

5.1.1 Welche Äquivalenzklassen von Eingabewerten sind sinnvollerweise zu testen?

Kontonummer:

- A: Kontonummer < 0
- B: weniger als 10 Stellen (Kontonummer < 1.000.000.000)
- C: 10 stellig (1.000.000.000 <= Kontonummer < 10.000.000.000)
- D: mehr als 10 Stellen (Kontonummer >= 10.000.000.000)

5.1.2 Welche Sonder- und/oder Randfälle sind sinnvollerweise ebenfalls zu testen?

- E: Kontonummer = 0 (Randfall zu A)
- F: Kontonummer = -1 (Randfall zu A)
- G: Kontonummer = 999.999.999 (Randfall zu B bzw. C)
- H: Kontonummer = 1.000.000.000 (Randfall zu B bzw. C)
- I: Kontonummer = 9.999.999.999 (Randfall zu C)
- J: Kontonummer = 10.000.000.000 (Randfall zu C bzw. D)

5.1.3 Geben Sie für jede Äquivalenzklasse, sowie für jeden Sonder- und/oder Randfall einen möglichen Eingabewert und das Soll-Ergebnis an. Bitte verwenden Sie hierzu keine realen Kontonummern. Benutzen sie stellvertretend für die Prüfsumme und Bankleitzahl die Strings „pp“ und „bbbbbbbb“

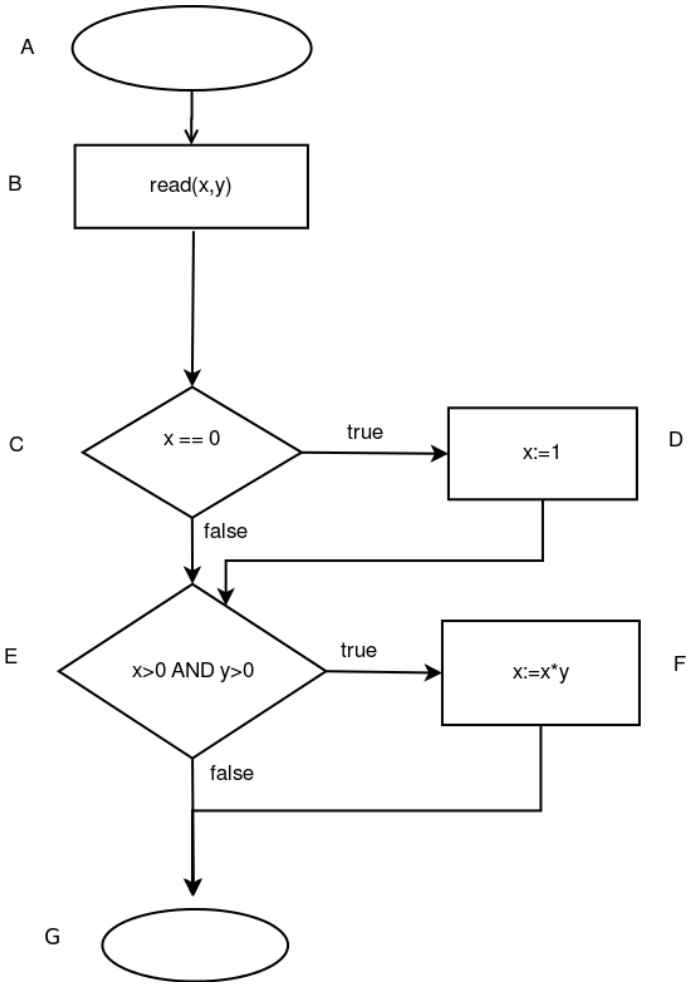
- A: Eingabe: -1000; Soll-Ergebnis: „Error“
- B: Eingabe: 5000; Soll-Ergebnis: „DEppbbbbbbbbb000005000“
- C: Eingabe: 1234567890; Soll-Ergebnis: „DEppbbbbbbbbb1234567890“
- D: Eingabe: 1111111111111; Soll-Ergebnis: „Error“
- E: Eingabe: 0; Soll-Ergebnis: „DEppbbbbbbbbb0000000000“
- F: Eingabe: -1; Soll-Ergebnis: „Error“
- G: Eingabe: 999.999.999; Soll-Ergebnis: „DEppbbbbbbbbb0999999999“
- H: Eingabe: 1.000.000.000; Soll-Ergebnis: „DEppbbbbbbbbb1000000000“
- I: Eingabe: 9.999.999.999; Soll-Ergebnis: „DEppbbbbbbbbb9999999999“
- J: Eingabe: 10.000.000.000; Soll-Ergebnis: „Error“

5.2 Kontrollflussbezogenes Testen – Testerfüllung

Das gegebene Programmsegment soll getestet werden:

```
1  read(x,y)
2  if ( x==0 ) { x:= 1 }
3  if ( (x>0) AND (y>0) ) { x:=x*y }
```

5.2.1 Skizzieren Sie das Flussdiagramm.



5.2.2 Geben Sie alle Entscheidungswege an.

{A, B, C};{C, D, E};{C, E};{E, G};{E, F, G}

Nehmen Sie für die folgenden Aufgaben an, es würde ein Test mit den Testdaten (x=0, y=1) und (x=-1, y=5) durchgeführt.

5.2.3 Erfüllt diese Testmenge die Anweisungsüberdeckung? Begründen Sie Ihre Antwort.

Dies Testmenge erfüllt die Anweisungsüberdeckung. Mit dem Test (x=0, y=1) werden alle Anweisungen durchlaufen.

5.2.4 Erfüllt diese Testmenge die Zweigüberdeckung? Begründen Sie Ihre Antwort.

Die Testmenge erfüllt die Zweigüberdeckung. Der erste Test durchläuft die jeweiligen 'true'-Zweige, der zweite Test durchläeuft die entsprechenden 'false'-Zweige, so dass alle Zweige durchlaufen werden.

5.2.5 Erfüllt diese Testmenge die Entscheidungsüberdeckung? Begründen Sie Ihre Antwort.

Die Testmenge erfüllt die Entscheidungsüberdeckung. Der erste Test durchläuft die jeweiligen 'true'-Zweige, der zweite Test durchläeuft die entsprechenden 'false'-Zweige, so dass alle Entscheidungen einmal 'true' und einmal 'false' sind.

5.2.6 Erläutern Sie den Unterschied zwischen einer Entscheidungs- und einer Zweigüberdeckung. Was ändert sich, wenn man den Abbruch von Tests mit einbezieht?

Bei Betrachung vollständiger Testpfade kommen beide zu demselben Ergebnis. Betrachtet man bei unvollständigen Testpfaden den Überdeckungsgrad, so haben beide eine unterschiedliche Metrik.

5.2.7 Erfüllt diese Testmenge die einfache Bedingungsüberdeckung? Begründen Sie Ihre Antwort.

Die Testmenge erfüllt die einfache Bedingungsüberdeckung nicht, da das atomare Prädikat $y > 0$ nie zu false ausgewertet wird.

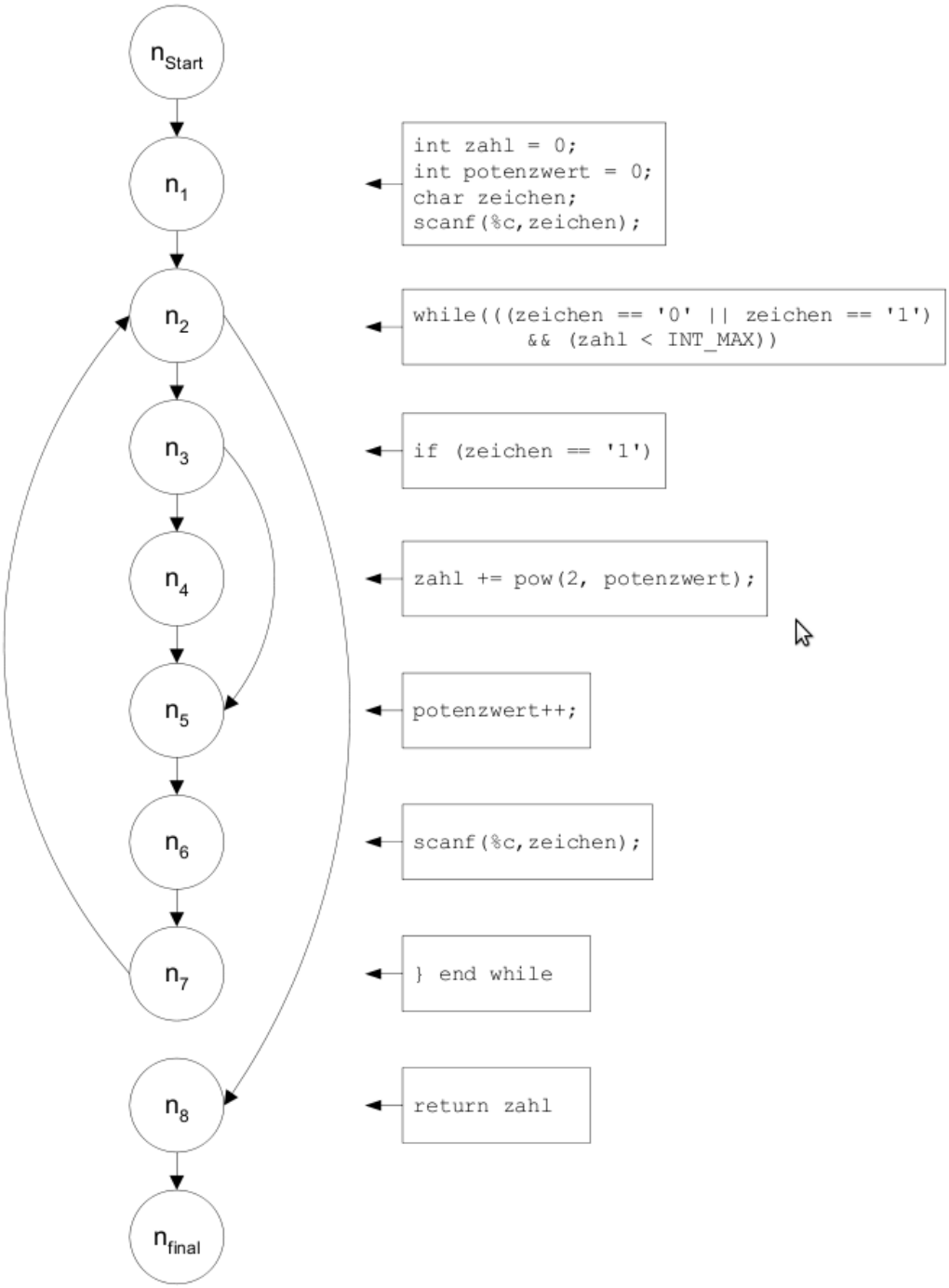
5.3 Kontrollflussbezogenes Testen – Testfälle

Gegeben ist das folgende Programm. Es wandelt eine Binärzahl in eine Dezimalzahl um. Die Stellen der Binärzahl werden invers eingelesen, d. h. die letzte Ziffer zuerst, danach die vorletzte usw.

```
1 #define INT_MAX 200
```

```
2  int wandleDezimalZahl(){
3          int zahl = 0;
4          int potenzwert = 0;
5          char zeichen;
6          scanf(%c,zeichen);
7          while ( ( (zeichen == "0") || (zeichen == "1") )
8                                      && (zahl < INT_MAX) )     {
9                  if (zeichen == "1") {
10                         zahl = zahl + pow(2,potenzwert);
11                 }
12                 potenzwert++;
13                 scanf(%c,zeichen);
14         }
15         return zahl;
16 }
```

5.3.1 Zeichnen Sie den zum Programm gehörenden Kontrollflussgraphen.



Hinweis: Die Zeilen 12-14 können auch zusammengefasst werden (Block).

5.3.2 Erstellen Sie Testfälle für einen minimalen, aber vollständigen Anweisungsüberdeckungstest (C0). Geben Sie den durchlaufenen Pfad an.

zeichen = {"1", "2"}
durchlaufener Pfad: nstart, n1, n2, n3, n4, n5, n6, n7, n2, n8, nfinal

5.3.3 Erstellen Sie Testfälle für einen minimalen, aber vollständigen Zweigüberdeckungstest (C1). Geben Sie den durchlaufenen Pfad an.

zeichen = {"1", "0", "2"}
durchlaufender Pfad: nstart, n1, n2, n3, n4, n5, n6, n7, n2, n3, n5, n6, n7, n2, n8, nfinal