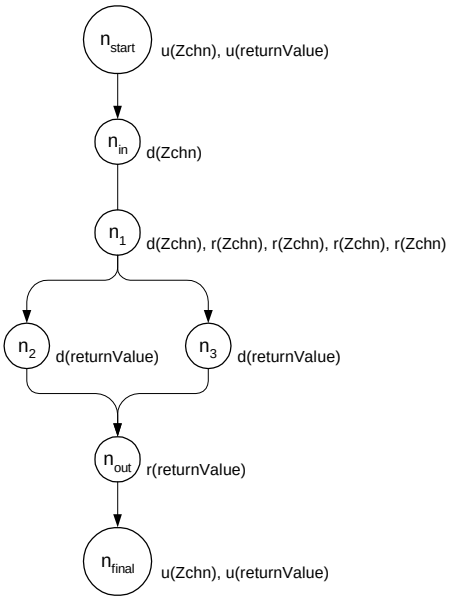
Aufgabe 1

Ein Kontrollflussgraph erleichtert die Durchführung einer Datenflussanalyse erheblich:




```
                typedef enum {FALSE = 0, TRUE = 1} boolean;

n_start         boolean isVocal(char Zchn)
n_in            {
                   boolean returnValue;

n_1                if((Zchn =  'A') || (Zchn == 'E') ||
                      (Zchn == 'I') || (Zchn == 'O') ||
                      (Zchn == 'U'))
                   {
n_2                   returnValue = TRUE;
                   }
                   else
                   {
n_3                   returnValue = FALSE;
                   }

n_out              return returnValue;
n_final         } // end isVocal()
```

a) Die if-Bedingung ist fehlerhaft programmiert worden: ein „=“ wurde vergessen, so dass der Vergleich zu einer Zuweisung mutierte. Korrekt muss Knoten n_1 heißen:

```
if((Zchn == 'A') || (Zchn == 'E') ||
   (Zchn == 'I') || (Zchn == 'O') ||
   (Zchn == 'U')
```

b) C kompiliert den fehlerhaften Code anstandslos. Die Zuweisung `Zchn = 'A'` gibt als Ergebnis `'A'` zurück (über diese Technik sind Ausdrücke wie `X = Y = Z = 'A'` möglich). Dieses ASCII-Zeichen ist ungleich Null und wird daher als TRUE betrachtet. Für den Programmlauf heißt das, dass die Alternative stets n_2 wählt und n_3 nie durchläuft, also alle Zeichen als Vokale erkannt werden.

c) In der Datenflussanalyse entsteht eine *dd*-Anomalie zwischen n_{in} und n_1, die kritischen Knoten sind grau unterlegt:

Variable	n_{start}	n_{in}	n_1	n_2	n_3	n_{out}	n_{final}
`Zchn`	u	d	d,r,r,r,r		d		u
`returnValue`	u			d		r	u

d) Nein, es ist nicht möglich, alle derartigen Fehler zu finden. Es ist z.B. denkbar, dass der Fehler beim zweiten Vergleich passiert, dann scheint der Graph okay zu sein, da C alle Ausdrücke stets von links nach rechts auswertet:

```
if((Zchn == 'A') || (Zchn =  'E') ||
   (Zchn == 'I') || (Zchn == 'O') ||
   (Zchn == 'U')
```

Variable	n_{start}	n_{in}	n_1	n_2	n_3	n_{out}	n_{final}
`Zchn`	u	d	r,d,r,r,r		d		u
`returnValue`	u			d		r	u

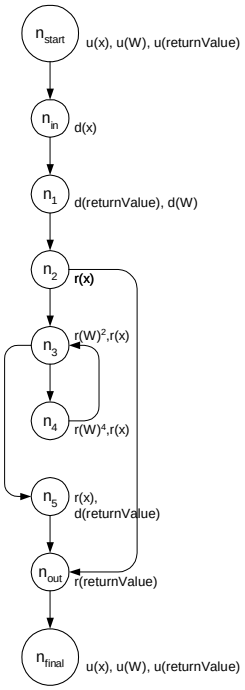
e) Ich gehe davon aus, dass am Markt bereits etliche Tools existieren, die diese Fehler ausfindig machen. Sie könnten kontrollieren, ob in `if`s Zuweisungen auftauchen und diese monieren sie dann.
Der Programmierer kann aber bereits diesen Fehlern vorbeugen, indem er die Seiten des Vergleiches vertauscht und somit schreibt:

```
if(('A' == Zchn) || ('E' == Zchn) ||
   ('I' == Zchn) || ('O' == Zchn) ||
   ('U' == Zchn)
```

Sollte ein Gleichheitszeichen vergessen werden, so bemerkt dies bereits der Compiler. Leider funktioniert diese Vorgehensweise nicht mehr, wenn zwei Variablen miteinander verglichen werden.

Aufgabe 2

Zunächst möchte ich anmerken, dass die Initialisierung von `W` mit `0` zu einem Laufzeitfehler führt (Division durch Null). Daher drucke ich hier bereits den korrigierten Code ab:



n_{start}	`double Sqrt(double X)`
n_{in}	`{`
n_1	`  double returnValue = 0;`
	`  double W = 1;`
n_2	`  if (X > 0.0)`
	`  {`
n_3	`    while (ABS(W*W-X) > 0.01)`
n_4	`      W = W - ((W*W-X) / (2.0 * W));`
n_5	`    returnValue = W;`
	`  } // end if`
	`  return (returnValue);`
n_{out}	`} // end Sqrt()`
n_{final}	

a) Wenn sich die Alternative in Knoten n_2 abweisend verhält, dann entsteht ein *du*-Anomalie:

Variable	n_{start}	n_{in}	n_1	n_2	n_{out}	n_{final}
`X`	u	d		r		u
`W`	u		d			u
`returnValue`	u		d		r	u

Das Durchlaufen der `while`-Schleife birgt eine *dd*-Anomalie:

Variable	n_{start}	n_{in}	n_1	n_2	n_3	n_4	n_3	n_4	n_3	n_5	n_{out}	n_{final}
`X`	u	d		r	r	r	r	r	r			u
`W`	u		d		r	r,d	r	r,d	r	r		u
`returnValue`	u		d							d	r	u

Beide Anomalien haben allerdings keinerlei negative Auswirkungen auf den Programmlauf.

b) *(fehlt)*

c) Eine Optimierung des Programmcodes durch Weglassen redundanter Variablendefinitionen kann dazu führen, dass sogar notwendige Initialisierungen entfernt werden, was i.d.R. zu einem Fehlverhalten des Programms führt. Dieses Risiko ist so groß, dass es besser scheint, die Redundanzen im Code zu belassen und dafür die Datenflussanalyse abzuändern.

d) Generell ist der Code akzeptabel, aber es ist eine Verbesserung möglich: größtmögliche Lokalität von Variablen, `W` wird daher erst später definiert.

```
double Sqrt(double X)
{
   double returnValue = 0;

   if (X > 0.0)
   {
      double W = 1;
      while (ABS(W*W-X) > 0.01)
         W = W - ((W*W-X) / (2.0 * W));

      returnValue = W;
   } // end if

   return (returnValue);
} // end Sqrt()
```

Allein diese Änderung führt dazu, dass die erste Datenflussanomalie verschwindet. Gleichzeitig sollte der Compiler schnelleren Code generieren können.

Aufgabe 3

a) nur Reviews
b) Reviews und Analysetools
c) nur Tests
d) Reviews, Analysetools und Tests
e) Reviews, Analysetools und Tests
f) nur Reviews (eingeschränkt auch Tools, aber fraglich)
g) nur Reviews
h) Reviews und Analysetools

Quelltext

```
////////////////////////////////////////////////
// Softwarebasistechnologien III, WS 2001/2002
// Test-Code für Aufgabenblatt 4
//
// Autor: Stephan Brumme
// letzte Änderung: 20.November 2001


#include <stdio.h>
#include <string.h>


// definiere boolean
typedef enum {FALSE = 0, TRUE = 1} boolean;

// fehlerhaft
boolean isVowel(char Zchn)
{
   boolean returnValue;

   if((Zchn =  'A') || (Zchn == 'E') ||
      (Zchn == 'I') || (Zchn == 'O') ||
      (Zchn == 'U'))
   {
```

```
      returnValue = TRUE;
   }
   else
   {
      returnValue = FALSE;
   }

   return returnValue;
}


// korrigiert
boolean isVowel2(char Zchn)
{
   boolean returnValue;

   if((Zchn == 'A') || (Zchn == 'E') ||
      (Zchn == 'I') || (Zchn == 'O') ||
      (Zchn == 'U'))
   {
      returnValue = TRUE;
   }
   else
   {
      returnValue = FALSE;
   }

   return returnValue;
}


// here we go !
void main(void)
{
    char* arTestSet = "AEIOUBCD";

    // Testdaten auf fehlerhaften Code anwenden
    printf("Fehlerhafter Code:\n");
    for (unsigned i=0; i<strlen(arTestSet); i++)
        if (isVowel(arTestSet[i]))
            printf("'%c' ist ein Vokal\n", arTestSet[i]);
        else
            printf("'%c' ist ein Konsonant\n", arTestSet[i]);


    // Testdaten auf korrekten Code anwenden
    printf("\nKorrekter Code:\n");
    for (i=0; i<strlen(arTestSet); i++)
        if (isVowel2(arTestSet[i]))
            printf("'%c' ist ein Vokal\n", arTestSet[i]);
        else
            printf("'%c' ist ein Konsonant\n", arTestSet[i]);
}
```