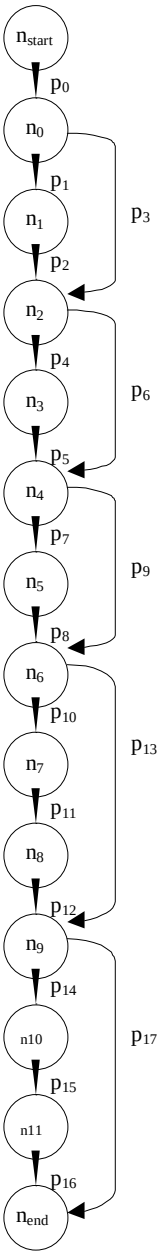
Aufgabe 1



Der Kontrollflussgraph der Operation `order()` wird im Wesentlichen von `if`-Anweisungen bestimmt. Er besitzt keine Schleifen, sondern verläuft sequentiell von n_{start} bis n_{end}.

Ich habe schon sowohl die Anweisungen n_{nummer} als auch die Zweige p_{nummer} in Hinblick auf die folgenden Aufgaben benannt.

```
nstart  private void order()
        {
   n0      if ((a>b)  && (a>c) && (b>c)  ||
               (a>b)  && (a>c) && (b==c) ||
               (a==b) && (a>c) && (b>c))
           {
   n1         flip_ac();
           }

   n2      if ((a>b) && (a<c)  && (b<c)  ||
               (a>b) && (a==c) && (b<c))
           {
   n3         flip_ab();
           }

   n4      if ((a<b) && (a<c)  && (b>c)  ||
               (a<b) && (a==c) && (b>c))
           {
   n5         flip_bc();
           }

   n6      if ((a>b) && (a>c) && (b<c))
           {
   n7         flip_ab();
   n8         flip_bc();
           }

   n9      if ((a<b) && (a>c) && (b>c))
           {
  n10         flip_ac();
  n11         flip_bc();
           }
  nend  }
```

Anmerkung: Diese Lösung wurde erst *nach* der entsprechenden Übung erstellt. Die Testfälle entsprechen deshalb exakt der Vorlage von Christopher Robinson-Mallett.

Aufgabe 2

Für den Anweisungsüberdeckungstest (C_0-Test) ist es notwendig, dass man Testfälle erzeugt, die jede Anweisung mindestens einmal durchlaufen. Für `order()` könnte man z.B. verwenden (ich habe jeweils die durchlaufenen Anweisungen grau unterlegt, auf n_{start} und n_{end} verzichte ich):

Testfall	a	b	c	n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}
1	3	2	1												
2	2	1	3												
3	1	3	2												
4	3	1	2												
5	2	3	1												

Aufgabe 3

Die Zweigüberdeckung (C_1-Test) erfordert, dass jeder Pfad mindestens einmal durchlaufen wird. Rein zufällig sind die Testfälle aus Ausgabe 2 auch dafür geeignet:

Testfall	a	b	c	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{17}
1	3	2	1																		
2	2	1	3																		
3	1	3	2																		
4	3	1	2																		
5	2	3	1																		

Aufgabe 4

Die Überprüfung des Codes

```
if (x==5 && y<0 || z>0)
{
   ...
}
```

auf Bedingungsüberdeckung unterteilt sich in mehrere Stufen:

a) Einfache Bedingungsüberdeckung: Man testet alle atomaren Teilentscheidungen gegen *wahr* und *falsch*.
Muster: *Jede Spalte muss min. 1x wahr und 1x falsch sein.*

vollständig	Testfall	x==5	y<0	z>0
	1	5	-1	-1
	2	0	0	0

links-unvollständig	Testfall	x==5	y<0	z>0
	1	5	-1	-1
	2	0	0	0
	3	5	0	-1

Hinweis: Bei unvollständiger Evaluierung kommt es wahr, dass einzelne Bedingung nicht überprüft werden, da sie das Ergebnis nicht beeinflussen können (da `0&&a=0` und `1||a=1` unabhängig von `a`). Aus diesem Grunde finden sich in den entsprechenden Tabellen mehrere Zellen, die nicht farblich hinterlegt sind, z.B. `z>0` im Testfall 1.

b) Mehrfache Bedingungsüberdeckung: Man testet alle Entscheidungen gegen *wahr* und *falsch*.
Muster: Alle binären Kombinationen sind zu durchlaufen.

vollständig	Testfall	x==5	y<0	z>0
	1	5	-1	1
	2	5	-1	0
	3	5	0	1
	4	5	0	0
	5	0	-1	1
	6	0	-1	0
	7	0	0	1
	8	0	0	0

links-unvollständig	Testfall	x==5	y<0	z>0
	1,2	5	-1	1
	3	5	0	1
	4	5	0	0
	5,7	0	-1	1
	6,8	0	0	0

c) Minimale mehrfache Bedingungsüberdeckung: Man testet alle atomaren und zusammengesetzten Teilentscheidungen gegen *wahr* und *falsch*.
Muster: Für jede zusammengesetzte Bedingung ist eine neue Spalte zu erstellen. Jede Spalte muss min. 1x wahr und 1x falsch sein.

vollständig	Testfall	x==5	y<0	z>0	x==5 && y<0	x==5 && y<0 || z>0
	1	5	-1	1		
	2	0	0	0		

links-unvollständig	Testfall	x==5	y<0	z>0	x==5 && y<0	x==5 && y<0 || z>0
	1	5	-1	1		
	2	0	0	0		
	3	5	0	1		

d) Modifizierter Bedingungs-/Entscheidungsüberdeckungstest: Alle Testfälle sollen demonstrieren, dass jede atomare Teilentscheidung den Wahrheitswert der Gesamtentscheidung unabhängig von den anderen Teilentscheidungen beeinflussen kann.
Muster: Bei Änderung nur einer Teilbedingungen muss sich das Gesamtergebnis ändern.

vollständig	Testfall	x==5	y<0	z>0	x==5 && y<0	x==5 && y<0 || z>0
	1	0	-1	0		
	2	5	-1	0		
	3	5	0	0		
	4	5	0	1		

links-unvollständig	Testfall	x==5	y<0	z>0	x==5 && y<0	x==5 && y<0 || z>0
	1	0	-1	0		
	2	5	-1	0		
	3	5	0	0		
	4	5	0	1		

Aufgabe 5

Zieht man die Anweisung n_0 aus Aufgabe 1 als Bedingung heran, so ergeben sich für die verschiedenen Testmethoden folgende mögliche Testfälle:

a) Einfache Bedingungsüberdeckung

Testfall	a	b	c	a>b	a>c	b>c	a=b	b=c
1	3	2	1					
2	1	1	1					

b) Minimale mehrfache Bedingungsüberdeckung:

```
X = (a>b) && (a>c) && (b>c)
Y = (a>b) && (a>c) && (b==c)
Z = (a==b) && (a>c) && (b>c)
F = X || Y || Z
```

Testfall	a	b	c	a>b	a>c	b>c	a=b	b=c	X	Y	Z	F
1	3	2	1									
2	1	1	1									
3	3	1	1									
4	2	2	1									

c) Modifizierte Bedingungs-/Entscheidungsüberdeckung:

```
X = (a>b) && (a>c) && (b>c)
Y = (a>b) && (a>c) && (b==c)
Z = (a==b) && (a>c) && (b>c)
F = X || Y || Z
```

Testfall	a	b	c	a>b	a>c	b>c	a=b	b=c	X	Y	Z	F
1	3	2	1									
2	2	3	1									
3	3	1	2									
4	3	2	2									
5	2	2	1									

Da die Bedingungen `X`, `Y` und `Z` stark gekoppelt sind (sie schließen sich gegenseitig aus), kann kein Testfallpaar für die atomare Bedingung `a>c` gefunden werden.

d) Kopplung von atomaren Bedingungen:

	a>b	¬(a>b)
a>c		
¬(a>c)		

⇒ nicht gekoppelt

	a>b	¬(a>b)
b>c		
¬(b>c)		

⇒ nicht gekoppelt

Hinweis: Grau bedeutet, dass die Kombination dieser Bedingungen möglich ist. Wenn alle vier Felder grau sind, so existiert keine Kopplung. Ist lediglich ein einzelner Fall ausgeschlossen, z.B. dass beide Bedingungen gleichzeitig wahr sind, so ist es eine schwache Kopplung. Folgt aus dem Wahrheitswert einer Bedingung unmittelbar der Wahrheitswert einer anderen, so spricht man von einer starken Kopplung.

	a>b	¬(a>b)
a=b		
¬(a=b)		

⇒ *schwach* gekoppelt

	b=c	¬(b=c)
a>b		
¬(a>b)		

⇒ nicht gekoppelt

	b>c	¬(b>c)
a>c		
¬(a>c)		

⇒ nicht gekoppelt

	b=c	¬(b=c)
a>c		
¬(a>c)		

⇒ nicht gekoppelt

	b=c	¬(b=c)
a>c		
¬(a>c)		

⇒ nicht gekoppelt

	a=b	¬(a=b)
b>c		
¬(b>c)		

⇒ nicht gekoppelt

	b=c	¬(b=c)
b>c		
¬(b>c)		

⇒ *schwach* gekoppelt

	b=c	¬(b=c)
a=b		
¬(a=b)		

⇒ nicht gekoppelt

e) *(fehlt)*
f) *(fehlt)*