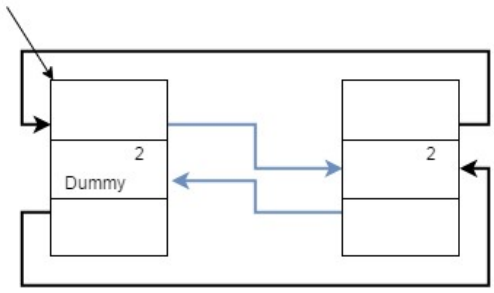
Aufgabe 2.1

1. Ausführungsmodelle:
    a. **Prozedurales (Imperatives) Modell:** ein Programm spezifiziert die sequenziell auszuführenden Befehle, die durch Speicherbelegung oder Speichertransformationen den Zustand des Programms verändern.
    b. **Simulationsmodell (Objektorientiertes Modell):** ein Program simuliert ein reales System durch Beschreibung von den Bestandteilen und dem Zusammenwirken des Systems.
    c. **Funktionales Modell:** die Ausführung des Programms ist (häufig rekursive) Auswertung von verschachtelten Funktionen, die Eingabe in Ausgabe transformieren.
    d. **Data Flow Modell:** ein Programm hat keinen Kontrollfluss, sondern besteht aus einem Graphen von den Operationen (als Knoten) und Datenabhängigkeiten. Parallelität ist hier eine inhärente Eigenschaft.
    e. **Logik-Modell:** anstatt eines Programmes, das beschreibt, „wie“ das Problem entschieden werden muss, wird das Problem („was“ zu lösen ist) als eine logische Formel formuliert, und die Lösung aus einem Wissensbasis hergeleitet.
2. Fünf wünschenswerte Eigenschaften der Programmiersprachen:
    a. Abstraktion
    b. Einfachheit
    c. Orthogonalität
    d. Flexibilität und Erweiterbarkeit
    e. Maschienenunabhängigkeit
3. Fünf wünschenswerte Eigenschaften des Programms (für einen Softwareentwickler):
    a. Angepasste Ausdrucksform
    b. Lesbarkeit
    c. Einfache Modifizierung und Erweiterung
    d. Verlässlichkeit
    e. Effizienz

Aufgabe 2.4

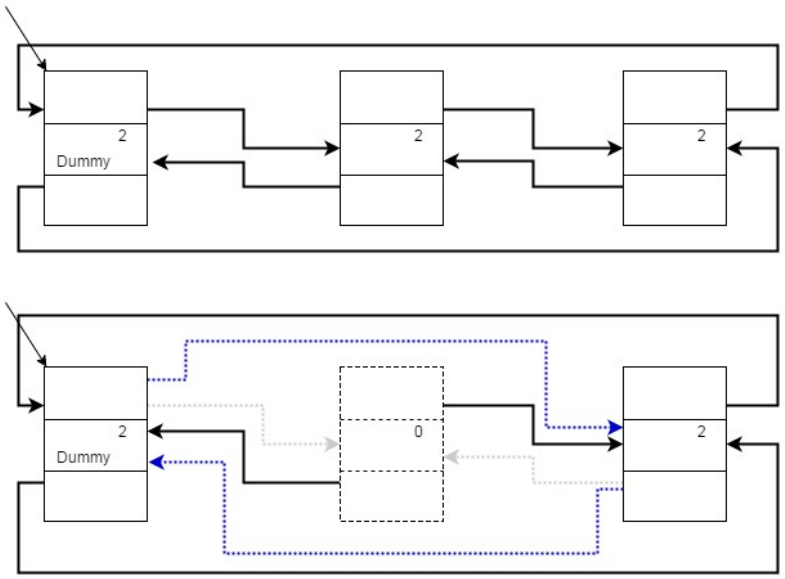
Fall 1: kein Knoten zwischen From und To.

Zwei Referenzen werden gelöscht und wieder erzeugt (blau), also keine Änderung.



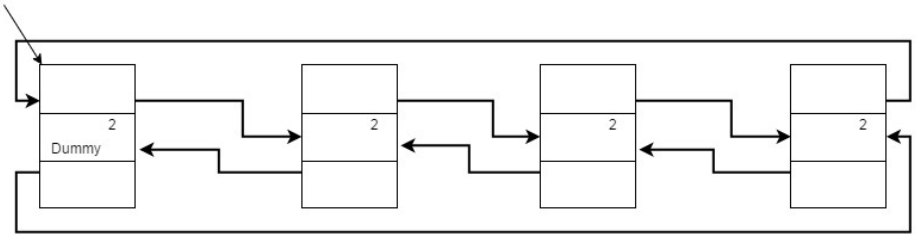
Fall 2: ein Knoten zwischen From (links) und To (rechts).

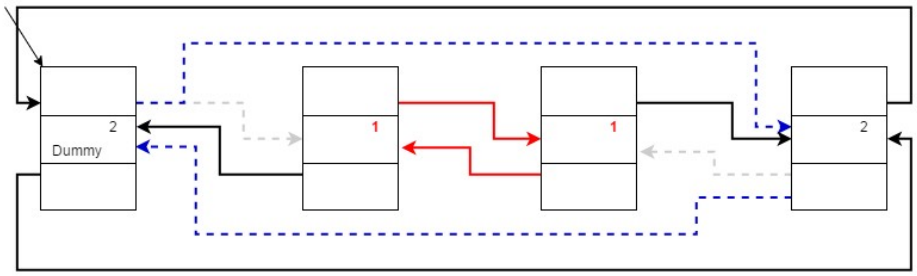
Am Anfang gibt es zwei Referenzen auf das mittlere Element. Beide Referenzen werden gelöscht (grau), und das Element darf auch gelöscht werden, da kein anderes Objekt Referenzen darauf hat.



Fall 3: mehrere Knoten zwischen From (links) und To (rechts).

Obwohl die Referenzen von From/To auf die "gelöschten" Elemente nicht mehr existieren, haben diese Elemente selbst Referenzen aufeinander, und deshalb dürfen sie nicht deallokiert werden. Solche „tote“ Ketten von Referenzen bleiben im Speicher, werden aber nicht mehr benutzt (eine *memory leak*).





Um das zu vermeiden, kann man die Referenzen zwischen den entfernten knoten löschen:

```
procedure Remove_Between (From : in DLL; To : in DLL) is
   Temp, TempNext : DLL;
begin
   Temp := From;
   Loop_of_Destruction :
     while Temp.Forward /= To loop
     TempNext := Temp.Forward;
     Temp.Forward := Null;
     Temp := TempNext;
   end loop Loop_of_Destruction;
   From.Forward := To;
   To.Backward := From;
end Remove_Between;
```

Aufgabe 2.5

1. Call-by-Value

Ausgabe:

X=3 Z=3

A=(2 1 1 4 5)

Prozedur F wird mit Parameter (1,5) aufgerufen.

F setzt X=A(2)=1 und Y=A(1)=2, und ruft die Prozedur G mit Parameter (3,2)

G verändert Z und Y, sowie das 2. Element der Liste: Y=5, Z=3, A(2) = A(A(2)) = A(3)

F verändert A(X): A(X) = A(Z), also A(3) = A(2) = 1

2. Call-by-Value/Result

Ausgabe:

X=2 Z=3

A=(2 1 5 4 1)

Prozedur F wird mit Parameter (X, Y) aufgerufen, setzt X=3, Y=1, und ruft G mit Parameter(X,Z) auf.

G setzt Z = 3, Y=5, A ist jetzt (2, 1, 1, 4, 5).

F setzt X=3, Y=1, Z=5, und verändert die entsprechende Elemente von A (siehe Ausgabe oben).