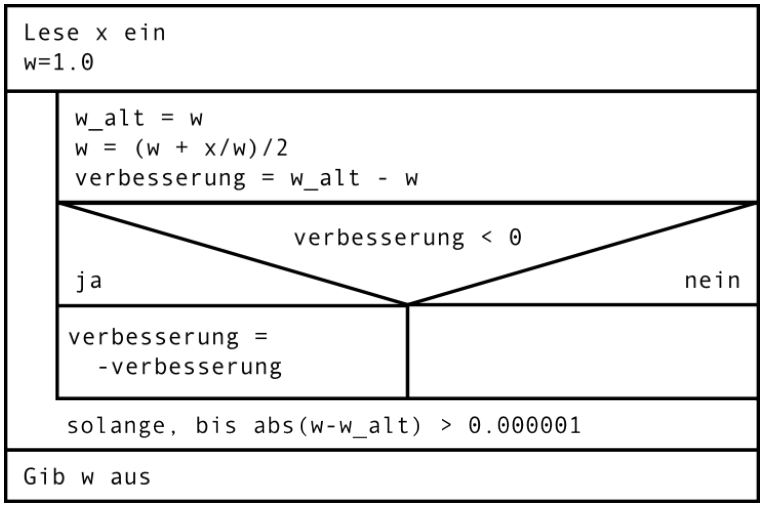
1. C/C++ Grundlagen

Aufgabe 1.1. Implementieren Sie eine Funktion `double round(double x, short s)`, die die Zahl `x` auf `s` Stellen hinter dem Komma rundet, falls $0 \leq$ `s` ≤ 10 gilt. Andernfalls soll x unverändert als Ergebnis zurückgegeben werden. Setzen Sie für Ihre Funktion Typkonvertierungen ein.

Aufgabe 1.2. Die babylonische Methode zur Berechnung der Quadratwurzel einer reellen Zahl x ist durch folgendes Struktogramm gegeben:



Implementieren Sie diesen Algorithmus als Funktion `double wurzel(double x)`.

Aufgabe 1.3. In einem `int`-Array werden die in der Euro-Währung vorhandenen Münzwerte als Cent-Beträge angegeben:

```
int muenzen[] = { 200, 100, 50, 20, 10, 5, 2, 1 };
```

Schreiben Sie ein C++ Programm mit folgender Funktionalität:

- Das Programm liest einen Kaufpreis von der Konsole ein.
- Nach der Eingabe des Kaufpreises werden solange Zahlungen angefordert, bis die Summe der Zahlungen größer oder gleich dem Kaufpreis ist.
- Bei einer Überzahlung wird der Restbetrag in möglichst wenigen, möglichst großen Münzen zurückgegeben.

Aufgabe 1.4. Die aus der Mathematik bekannte Sinusfunktion ist durch folgende Summe berechenbar:

$$\sin(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$$

a) Betrachte die `double` Variablen `k`, `vorzeichen`, `zaehler` und `nenner`. Angenommen, der aktuelle Wert der Variablen ist wie folgt:

Variable	*Wert*
`k`	k
`vorzeichen`	$(-1)^k$
`zaehler`	x^{2k+1}
`nenner`	$(2k+1)!$

Wie muss man die Variablen aktualisieren, um den $(k+1)$-ten Summanden zu berechnen?

b) Implementieren Sie die Funktion

```
double sinus(double x, unsigned int max)
```

zur Berechnung der obigen Summe, wobei maximal `max` Summanden aufaddiert werden. Vergleichen Sie das von der Funktion berechnete Ergebnis mit dem der Sinusfunktion aus der C Standardbibliothek.

Aufgabe 1.5. Ziel dieser Aufgabe ist es, mit Zeigern zu arbeiten.

a) Legen Sie drei `double`-Variablen `x`, `y` und `z` an und weisen Sie ihnen Werte zu.

b) Legen Sie zusätzlich drei Zeiger auf `double`-Werte `px`, `py` und `pz` an. Weisen Sie ihnen die Adressen von `x`, `y` und `z` zu.

c) Geben Sie über die Zeigervariablen sowohl die Inhalte der Variablen aus als auch die Adressen, auf die sie zeigen. Ermitteln Sie den Speicherort der Zeiger-Variablen.

d) Bilden Sie die Summe der Variablen `x`, `y` und `z`

- auf direkte Art und Weise,
- über die entsprechenden Zeiger-Variablen und
- über anonyme Zeiger.

Aufgabe 1.6. Ziel dieser Aufgabe ist die Implementierung einer Funktion für die Suche eines Patterns in einem String. Beispielsweise ist das Pattern `ab` in dem String `Tablet` enthalten. Implementieren Sie die Funktion

```
int search(char* s, int len_s, char* p, int len_p)
```

mit der man überprüfen kann, ob das Pattern `p` mit der Länge `len_p` in dem String `s` der Länge `len_s` enthalten ist. Falls ja, dann soll die Startposition des Pattern zurückgegeben werden. Andernfalls ist der Rückgabewert -1. Erstellen Sie zuerst für Ihren Algorithmus ein Struktogramm und implementieren Sie dann den Algorithmus anhand des Struktogramms.

Aufgabe 1.7. Implementieren Sie eine C++ Funktion `void swap(int& a, int& b)`, die die Werte von `a` und `b` vertauscht. Nach Aufruf dieser Funktion soll `a` den ursprünglichen Wert von `b` und `b` den ursprünglichen Wert von `a` enthalten.

Aufgabe 1.8. Ziel ist die Entwicklung einer C Funktion

```
double* randomArray(unsigned int n),
```

die ein Array mit den Zahlen $1, \ldots, n$ als Inhalt berechnet. Die Funktion soll folgende Anforderungen erfüllen:

- Die Funktion erhält als Eingabe die Länge des Arrays n.
- Jede Zahl in $\{1, \ldots, n\}$ ist genau einmal im Array enthalten.
- Die Reihenfolge der Zahlen im Array wird mittels einem Pseudozufallszahlengenerator festgelegt.
- Das Array wird dynamisch allokiert und mittels einem Zeiger zurückgegeben.

Implementieren Sie die Funktion entsprechend den Vorgaben.

Aufgabe 1.9. Die Fakultät $n!$ einer natürlichen Zahl $n \in \mathbb{N}$ ist definiert als $n! = 1 \cdot 2 \cdot \ldots \cdot n$. Implementieren Sie sowohl eine iterative als auch rekursive C Funktion zur Berechnung der Fakultätsfunktion.

Aufgabe 1.10. Der *größte gemeinsame Teiler* zweier ganzer Zahlen a und b ist definiert als die größte ganze Zahl d, die sowohl ein Teiler von a als auch ein Teiler von b ist, formal:

$$\gcd(a, b) = \max\{d \in \mathbb{N} \mid d \text{ teilt } a \text{ und } d \text{ teilt } b\}.$$

Hierbei muss mindestens eine der Zahlen a und b von Null verschieden sein.

Zur effizienten Berechnung von $\gcd(a, b)$ wird in der Regel der Algorithmus von Euklid eingesetzt. Dieser ist durch folgende Funktion definiert:

$$\mathit{Euklid}(a, b) = \begin{cases} a, & \text{falls } b = 0, \\ \mathit{Euklid}(b, a \bmod b), & \text{sonst.} \end{cases}$$

a) Implementieren Sie den Algorithmus von Euklid als C++ Funktion `int gcd(int a, int b)`.

b) Berechnen Sie den größten gemeinsamen Teiler von $a = 21033$ und $b = 56580$.

Aufgabe 1.11. Bubblesort ist ein sehr einfaches Sortierverfahren. Eine umgangssprachliche Beschreibung des Algorithmus ist:

> Wiederhole das folgende Verfahren solange, bis keine Vertauschung mehr stattgefunden hat:
>
> > Vergleiche die Elemente $A[i]$ und $A[i+1]$ für $i = 0, \ldots, n-2$. Ist $A[i]$ größer als $A[i+1]$, dann vertausche den Inhalt dieser beiden Elemente.

a) Stellen Sie den Bubblesort Algorithmus als Struktogramm dar.

b) Implementieren Sie eine Funktion `void swap(int& a, int& b)`, die den Inhalt der Variablen `a` und `b` vertauscht.

c) Implementieren Sie den Bubblesort Algorithmus als C-Funktion `void bubbleSort(int A[], int n)`.

d) Testen Sie Ihre Implementierung, indem Sie ein Array mit den Zahlen $1, \ldots, n$ in zufälliger Reihenfolge sortieren lassen.

2. Objektorientierte Programmierung

Aufgabe 2.1. Eine Parabel ist eine Funktion $f(x) = ax^2 + bx + c$, wobei $a, b, c \in \mathbb{R}$. Die Terme a, b und c nennt man Parameter. Ziel ist die Implementierung einer Klasse `Parabel` für die Arbeit mit Parabeln. Diese Klasse soll folgende Vorgaben erfüllen:

- Die Parameter einer Parabel werden als Attribute gespeichert werden. Die Attribute sollen vor externen Zugriff geschützt sein.
- Der Zugriff auf die Attribute erfolgt mittels sogenannten Getter und Setter Methoden.
- Mit dem Konstruktor der Klasse kann man bei der Instantiierung eines Parabelobjekts die Werte der Parameter festlegen.
- Mit der Methode `double operator()(double x)` kann man den Funktionswert der Parabel für x berechnen.

- Die Methode `int nullstellen(double& x1, double& x2)` dient zur Berechnung der Nullstellen der Parabel. Der Rückgabewert ist gleich der Anzahl der Nullstellen, die Nullstellen (falls vorhanden), werden über die Referenzvariablen `x1` und `x2` zurückgegeben.

a) Modellieren Sie die Klasse `Parabel` mit einem UML-Diagramm.

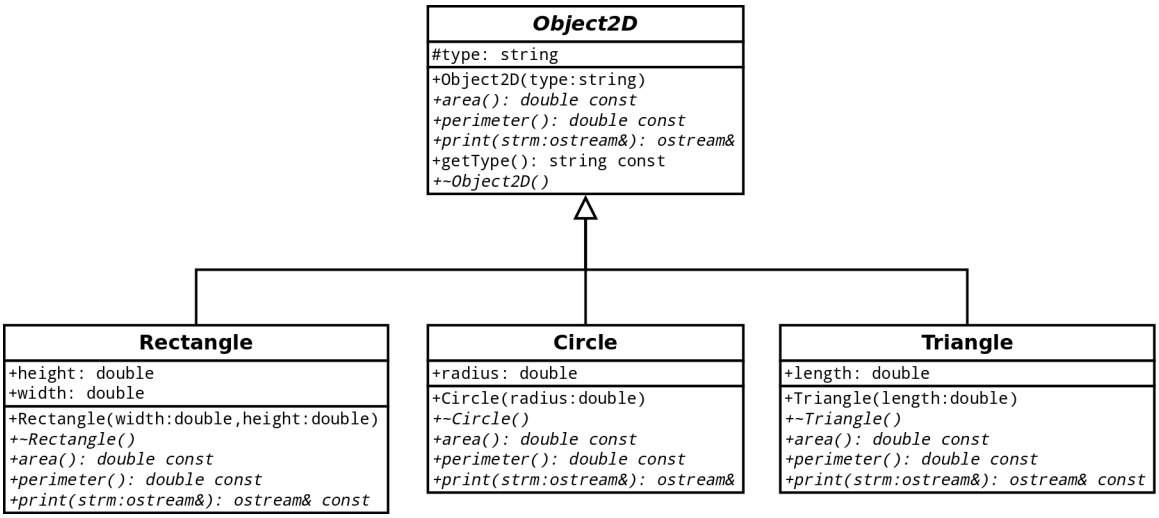
b) Implementieren Sie die Klasse `Parabel` anhand des UML-Diagramms.

c) Testen Sie Ihre Implementierung, indem Sie ein Objekt für die Parabel $f(x) = 2x^2 - 4x + 10$ erzeugen und für verschiedene Werte für x berechnen.

3. Vererbung

Ein wichtiges Konzept in der objektorientierten Softwareentwicklung ist Vererbung. Dieses Konzept wird in diesem Abschnitt anhand einer einfachen Klassenhierarchie behandelt.

Ziel ist es, eine Klassenhierarchie für zweidimensionale Objekte zu implementieren. Jedes dieser Objekte hat einen Typ sowie Methoden zur Berechnung des Flächeninhalts und des Umfangs. Es wird zwischen Rechtecken, gleichseitigen Dreiecken und Kreisen unterschieden. Dies führt zu folgendem UML-Diagramm:



Aufgabe 3.1. Implementieren Sie die (abstrakte) Klasse `Object2D` in folgenden Schritten:

a) Erstellen Sie eine C++ Klasse.

b) Deklarieren Sie die Attribute der Klasse.

c) Implementieren Sie die Methoden.

d) Überladen Sie den globalen Ausgabeoperator `<<` so, dass man Objekte von Typ `Object2D` direkt in einem Stream ausgeben kann.

Aufgabe 3.2. Implementieren Sie die Klasse `Rectangle` in folgenden Schritten:

a) Erstellen Sie eine C++ Klasse.

b) Deklarieren Sie die Attribute der Klasse.

c) Implementieren Sie den Konstruktor der Klasse. Als Typ soll „`Rectangle`“ an die Basisklasse weitergegeben werden.

d) Implementieren Sie die restlichen Methoden der Klasse.

Aufgabe 3.3. Implementieren Sie die Klasse `Circle` und `Triangle` analog zu Aufgabe 3.2.

Aufgabe 3.4. Erzeugen Sie ein Array von Zeigern auf `Object2D`-Objekte mit folgendem Inhalt:

- Kreis mit Radius 500
- Dreieck mit Seitenlänge 400
- Rechteck mit Breite 600 und Höhe 200
- Kreis mit Radius 350
- Rechteck mit Breite 100 und Höhe 500

Geben Sie die Daten der gespeicherten Objekte mittels einer `for`-Schleife aus.

4. Standard Template Library (STL)

Aufgabe 4.1. Gegeben ist die folgende Tabelle mit Telefonnummern:

Name	*Nummer*
Heinz	0170/120377
Martin	0176/55554444
Claudia	0173/6198014
Andreas	0711/52231
Sabine	089/19223211
Anja	0172/3334123

a) Speichern Sie diese Tabelle in einer passenden von der STL bereitgestellten Datenstruktur.

b) Geben Sie die Tabelle mit den Methoden der Datenstruktur aus.

c) Suchen Sie in der Datenstruktur nach den Telefonnummern von Anja und Thomas.

Aufgabe 4.2. Ziel ist die Entwicklung einer FIFO-Warteschlange für Arbeitsaufträge. FIFO steht für den Begriff *First In First Out*. Dies bedeutet, die Aufträge in chronologischer Reihenfolge verarbeitet werden, in der sie in die Warteschlange eingefügt wurden. Ein Auftrag beinhaltet zwei Informationen: den Namen des Kunden und die Art der zu verrichtenden Arbeit.

a) Erstellen Sie eine Klasse zur Speicherung von Aufträgen.

b) Implementieren Sie die Warteschlange mit einer passenden STL-Datenstruktur.

c) Testen Sie Ihre Implementierung, indem Sie mehrere Aufträge in die Warteschlange einfügen und verarbeiten.

Aufgabe 4.3. Gegeben ist folgende Zeichenfolge:

```
ichbineingaaanzerlangerstringohnetieferebedeutungabermitvielen
buchstabendiemangutzaehlenundalsstatistikerfassenkannnadannvielspass
```

Erstellen Sie unter Einsatz einer passenden STL-Datenstruktur eine Statistik über die in obigem String vorkommenden Buchstaben und deren Häufigkeit.

Aufgabe 4.4. Die Polybios Chiffre ist ein klassisches Chiffrierverfahren, mit dem man Nachrichten verschlüsseln kann, die aus kleinen Buchstaben oder Ziffern bestehen. Der Schlüssel wird durch ein sogenanntes Polybios Quadrat festgelegt. Ein solches Quadrat ist eine 6×6 Tabelle, bei der jede Zelle einen Buchstabe oder eine Ziffer enthält. Zur Illustration ein Beispiel:

	0	1	2	3	4	5
0	c	0	k	q	l	s
1	5	m	g	9	n	z
2	w	4	7	6	t	b
3	v	j	1	2	y	o
4	3	e	d	i	f	8
5	x	r	a	u	p	h

Um einen Klartext zu verschlüsseln, wird ein Symbol (Buchstabe oder Ziffer) ersetzt durch die Zeilen- und Spaltennummer der Zelle, die diese Symbol enthält. Beispielsweise führt der Klartest `akad` zum Geheimtext `52025242`. Zur Entschlüsselung eines Geheimtexts wird die Nachricht in Zweierblöcke zerlegt. Die erste Ziffer eines Blocks steht für die Zeile, die

zweite Ziffer für die Spalte der Zelle im Polybios Quadrat, die den Klartextbuchstaben enthält. Beispielsweise liefert die Entschlüsselung des Geheimtexts `052453242412525124` das Ergebnis `stuttgart`.

Ziel dieser Aufgabe ist es, die Polybios Chiffre in Form einer C++ Klasse `PolybiosChiffre` zu implementieren. Zur Implementierung dürfen Klassen der STL verwendet werden.

a) Überlegen Sie sich zunächst, welche Attribute für die Speicherung des Schlüssels erforderlich sind. Reicht ein Attribut aus oder ist es praktischer, den Schlüssel in zwei Attributen zu speichern, wobei eines der Attribute zur Verschlüsselung eingesetzt wird und das andere zur Entschlüsselung.

b) Erstellen Sie die Klasse `PolybiosChiffre` mit den passenden Attributen.

c) Implementieren Sie die Methode `bool setKey(const string& key)`, mit der ein Schlüssel vergelegt wird. Der Schlüssel wird als String übergeben, wobei die einzelnen Zeilen der Tabelle hintereinander geschrieben werden. Beispielsweise wird der Schlüssel in obigem Polybios Quardrat durch den String

   ```
   c0kqls5mg9nzw476tbvj12yo3edif8xrauph
   ```

   festgelegt. Die Methode soll zunächst überprüfen, ob der String `key` einen korrekten Schlüssel darstellt. Falls nein, dann gibt die Methode `false` zurück. Falls ja, den wird der Schlüssel in den dafür vorgesehenen Attributen gespeichert und der Wert `true` zurückgegeben.

d) Implementieren Sie die Methode `void printTable()`, die den aktuellen Schlüssel als Polybios Quadrat ausgibt.

e) Implementieren Sie die Methode `string encrypt(const string& plain_text)` zur Verschlüsselung des im String `plain_text` gespeicherten Klartexts unter Einsatz des aktuell in der Klasse gespeicherten Schlüssels. Enthält `plain_text` unzulässige Symbole, dann soll der leere String als Ergebnis zurückgegeben werden.

f) Implementieren Sie die Methode `string decrypt(const string& cipher_text)` zur Entschlüsselung des im String `cipher_text` gespeicherten Geheimtexts unter Einsatz des aktuell in der Klasse gespeicherten Schlüssels. Enthält `cipher_text` unzulässige Symbole, dann soll der leere String als Ergebnis zurückgegeben werden.

g) Implementieren die Methode `string randomKey()`, mit der man einen Zufallsschlüssel erzeugen kann. Setzen Sie hierzu die Funktion `random()` der C-Bibliothek ein.

h) Testen Sie Ihre Implementierung, indem Sie den Geheimtext

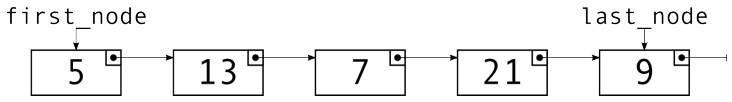
   ```
   554151150443005541141204534100022053140500554355515451351 2
   5152111144531402244335144341512 4
   ```

unter Einsatz des obigen Schlüssels dechiffrieren.

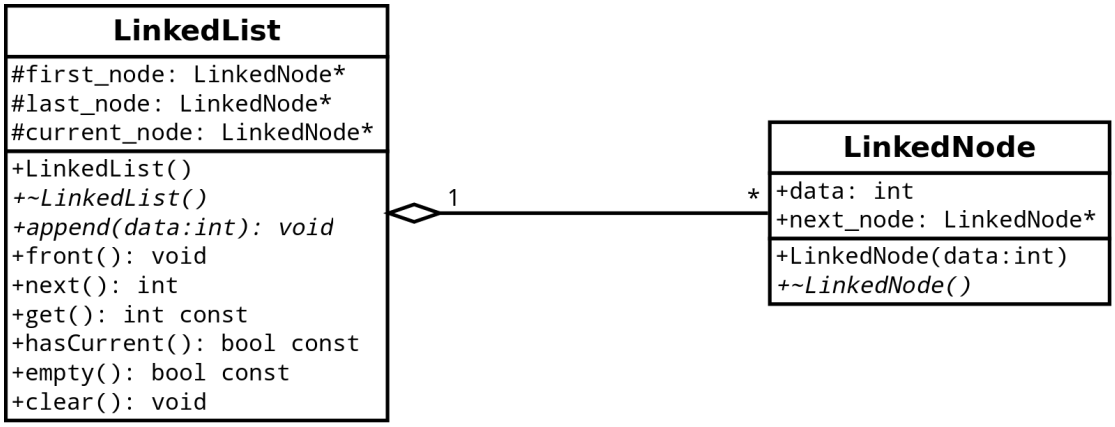
5. Verkettete Listen

Ziel dieses Abschnitts ist die Entwicklung einer einfach verketteten und einer doppelt verketteten Liste zur Speicherung von ganzen Zahlen. Es dient als weiteres Beispiel für das Konzept der Vererbung.

Eine einfach verkettete Liste ist eine Datenstruktur, die mehrere Elemente enthält, die mittels einem Zeiger verknüpft sind. Oder als Grafik:



Die Implementierung einer verketteten Liste besteht aus zwei Klassen. Die Klasse `LinkedNode` repräsentiert ein Glied der Liste und speichert die Nutzdaten sowie den Zeiger. Die Liste an sich wird in der Klasse `LinkedList` implementiert. Diese Klasse wendet das Blackbox-Prinzip an, gemäß dem die internen Datenstrukturen vor dem Zugriff von außen geschützt werden. Stattdessen erfolgt der Zugriff über entsprechende Methoden der Klasse. Das UML-Diagramm der beiden Klassen ist wie folgt:



Aufgabe 5.1. Implementieren Sie die Klasse `LinkedNode`.

Die Klasse `LinkedList` enthält als Attribute drei Zeiger. `first_node` zeigt auf den Anfang, `last_node` auf das Ende der verketteten Liste. Der Zeiger `current_node` wird zum Durchlaufen der Liste verwendet.

Aufgabe 5.2. Implementieren Sie die Klasse `LinkedList` in folgenden Schritten:

a) Implementieren Sie den Konstruktor. Dieser weist allen Zeigern den Wert `NULL` zu.

b) Implementieren Sie die Methode `void append(int data)`. Diese Methode hängt einen neuen Knoten vom Typ `LinkedNode` mit dem Inhalt `data` an das Ende der Liste an.

c) Implementieren Sie die Methode `bool hasCurrent()`. Diese Methode liefert genau dann `true` zurück, wenn der Zeiger `current_node` ungleich `NULL` ist.

d) Implementieren Sie die Methode `void front()`. Diese Methode weist `current_node` den Zeiger auf das erste Element der Liste zu.

e) Implementieren Sie die Methode `int get()`. Diese Methode liefert den Inhalt des Listenelements, auf das `current_node` zeigt.

f) Implementieren Sie die Methode `int next()`. Diese Methode liefert den Inhalt des Listenelements, auf das `current_node` zeigt und weist `current_node` den Zeiger auf das nächste Element zu.

g) Implementieren Sie die Methode `bool empty()`, die genau dann `true` zurückliefert, wenn die Liste leer ist.

h) Implementieren Sie die Methode `void clear()`. Diese Methode löscht die verkettete Liste, indem der entsprechende Speicher frei gegeben wird.

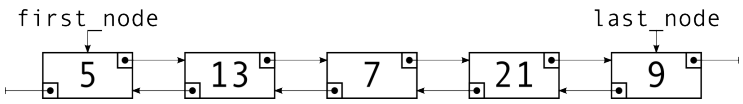
i) Implementieren Sie den Destruktor. Der Destruktor gibt den Speicher der kompletten Liste frei.

Aufgabe 5.3. Testen Sie Ihre Implementierung, in dem Sie eine Folge von 8 Zahlen in der Liste speichern und anschließend den Inhalt der Liste mittels der entsprechenden Methoden wieder ausgeben.

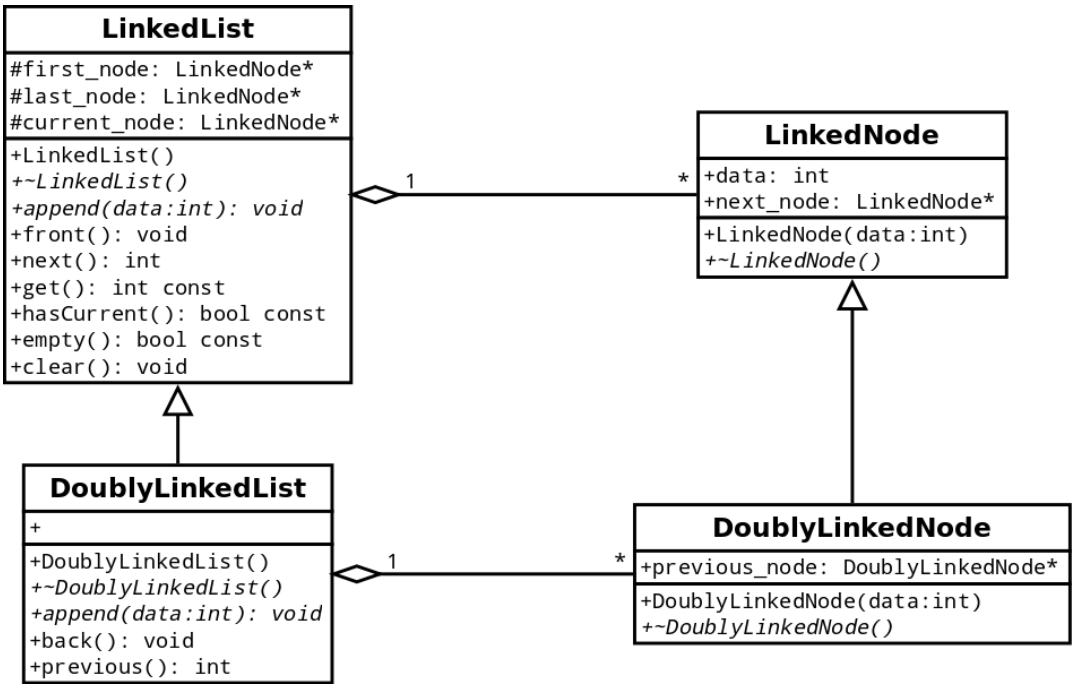
Die Implementierung ist beinhaltet noch einen Fehler. Ist `current_node` gleich `NULL`, dann führt der Aufruf von `get()` oder `next()` zu einem Absturz des Programms. Um diesem Fehler zu korrigieren, bietet sich der Einsatz von Exceptions an.

Aufgabe 5.4. Erstellen Sie eine von `exception` abgeleitete Klasse `ListException` und integrieren Sie diese in die Klasse `LinkedList`. Die Exception

Eine wesentliche Einschränkung einer verketteten Liste ist die Tatsache, dass die Elemente nur in einer Richtung und zwar vom Anfang bis zum Ende durchlaufen werden können. Oft ist es praktisch, die Liste in umgekehrter Reihenfolge zu durchlaufen. Mittels einem zusätzlichen Zeiger kann dies technisch ohne Mühe realisiert werden. Das Ergebnis ist eine doppelt verkettete Liste. Ein Beispiel ist:



Mittels objektorientierter Programmierung und dem Konzept der Vererbung können die Klassen der (einfach) verketteten Liste mit geringem Aufwand zu einer doppelt verketteten Liste erweitert werden. Das nächste UML-Diagramm zeigt die Vererbungshierachie:



Die in der Klasse `DoublyLinkedList` enthaltenen Methoden `void back()` und `int previous()` bieten dabei die Möglichkeit, die Liste vom Ende zum Anfang zu durchlaufen.

Aufgabe 5.5. Implementieren Sie die Klassen `DoublyLinkedNode` und `DoublyLinkedList`. Testen Sie Ihre Implementierung in dem Sie eine Folge von ganzen Zahlen in der Liste speichern und die Liste in umgekehrter Reihenfolge ausgeben.