Name: Matrikelnummer:

Aufgabe 1 (Iteratoren in Java)

In dieser Aufgabe sollen Sie Klassen schreiben, die die Schnittstelle `Iterator` implementieren.

a) **(6 Punkte)**

Schreiben Sie eine generische Klasse `Cycle<A>`. Die Klasse soll die Schnittstelle `Iterator<A>` implementieren.

Die Klasse soll im Konstruktor ein Objekt des Typs `java.util.List<A>` erhalten.

Die Methode `next` soll nacheinander die über Elemente der übergebenen Liste iterieren. Wurde über das letzte Listenelement iteriert, so soll wieder von vorne in der Liste begonnen werden. Es soll also potentiell unendlich immer wieder über die Elemente der Liste im Kreis iteriert werden.

```
class Cycle<A> implements Iterator<A>{
  Cycle(List<A> es){
    this.es = es;  it = es.iterator();
  }
  List<A> es;
  @Override public boolean hasNext(){
    return true;
  }
  Iterator<A> it;
  @Override public A next(){
    if(!it.hasNext()) it = es.iterator();
    return it.next();
  }
```

Name: Matrikelnummer:

b) (6 Punkte)

Gegeben sei folgende Record Klasse:

```
record Pair<A,B>(A fst , B snd){}
```

Listing 1: Pair

Schreiben eine Klasse `Zip<A,B>`, die `Iterator<Pair<A,B>>` implementiert.

Sie soll zwei Objekte im Konstruktor übergeben bekommen: ein Objekt `itA` des generischen Typs `Iterator<A>` und `itB` des generischen Typs `Iterator<B>`.

Ein Objekt der Klasse `Zip<A,B>` soll beim Aufruf von `next` jeweils von beiden übergebenen Iteratoren `itA` und `itB` das nächste `A`- bzw. `B`-Objekt erfragen und ein `Pair<A,B>`-Objekt errechnen, welches das Ergebnis des Aufrufs ist.

Die Methode `hasNext` gibt solange `true` zurück, wie beide übergebenen Iteratoren noch weitere Elemente zurückgeben

```
class Zip<A,B> implements Iterator<Pair<A,B>>{
  Iterator<A> itA; Iterator<B> itB;
  Zip(Iterator<A> itA, Iterator<B> itB){
    this.itA = itA; this.itB = itB;
  }
  @Override public boolean hasNext(){
    return itA.hasNext() && itB.hasNext();
  }
  @Override public Pair<A,B> next(){
    return new Pair<>(itA.next(), itB.next());
  }
}
```

Name Matrikelnummer

c) (4 Punkte)

Gegeben sei folgende Schnittstelle

```
interface Zipper<A,B,C>{
  C zip(Pair<A,B> p);
}
```

Listing 2: Pair

Schreiben Sie eine Klasse `Combine<A,B,C>` die `Iterator<C>` implementiert.

Sie soll zwei Objekte im Konstruktor übergeben bekommen: ein Objekt `itP` des generischen Typs `Iterator<Pair<A,B>>` und ein Objekt `f` der Schnittstelle [redacted]`<A,B,C>`

Beim Aufruf von `next` soll aus `itP` das nächste Paarobjekt genommen werden und mit der Funktion `f` das Ergebnis berechnet werden

```
class Combine<A,B,C> implement Iterator<C>{
  Iterator<Pair<A,B>> it;
  Zipper<A,B,C> f;
  Combine(Iterator<Pair<A,B>> it, Zipper<A,B,C> f){
    this.it = it; this.f = f;
  }
  @Override public boolean hasNext(){
    return it.hasNext();
  }
  @Override public C next(){
    return f.zip(it.next());
  }
}
```

Name: Matrikelnummer:

d) (4 Punkte)

Gegeben sei ein Objekt des Typs `Iterator<Pair<Integer,Integer>>`. Erzeugen Sie mit `Combine` ein Iteratorobjekt, das über die Produkte der beiden Zahlen der Paarobjekte iteriert.

```
Iterator<Pair<Integer,Integer>> it= ...
use it = new Combine<>(it,
        (p) -> p.fst() * p.snd()),
```

Name: Matrikelnummer:

Aufgabe 2 (Stream)

a) (4 Punkte)

In der Standard Schnittstelle `LongStream` gibt es folgende statische Methode zum Erzeugen eines potentiell unendlich langen Streams über long Zahlen:

```
LongStream iterate(long seed, LongUnaryOperator f)
```

Dabei ist `LongUnaryOperator` eine funktionale Schnittstelle mit folgender abstrakten Methode:

```
long applyAsLong(long operand)
```

Machen Sie einen Aufruf von `iterate`, um eine Stream aller durch 3 teilbaren positiven Zahlen zu erhalten.

var ls =

LongStream.iterate(3L, x -> x+3L);

Name: Matrikelnummer:

b) (4 Punkte)

Machen sie einen Aufruf auf das in a) erzeugte Stream-Objekt, so dass es gefiltert wird auf einen Stream, der nur noch gerade Zahlen kleiner 100 enthält.

ls.filter(x -> x%2==0
&& x < 100)

c) (4 Punkte)

Gegeben sei ein Objekt des Typs `LongStream`. Machen Sie mit `reduce` einen Aufruf, der die größte Zahl in diesem Stream erzeugt. Ist der Stream leer, so sei das Ergebnis des Aufrufs `Long.MIN_VALUE`

ls.reduce(Long.MinValue,
(r,x) -> r>x? r : x,
(x,y) -> x>y ? x : y)

Name: Matrikelnummer:

Aufgabe 3 (XML)

a) (6 Punkte)

Gegeben sei folgende Schnittstelle:

```
interface Property{
  boolean test(Node n);
}
```

Listing 3: Property

Schreiben Sie mit dem DOM API (Paket `org.w3c.dom`) die Methode `collectNodes`.

Sie soll den ganzen Baum durchlaufen und all die Baumknoten der Liste `java.util.List<Node> result` zufügen, für die das Schnittstellenobjekt `true` ergibt.

```
static void collectNodes(Node node, Property p,List<Node> result){
```

```
if(p.test(node))
    result.add(node);
var nds = node.getChildNodes();
for(var i=0; i<nds.getLength(); i++){
  collectNodes(nds.item(i), p, result);
}
```

Name: Matrikelnummer:

b) (4 Punkte)

Verwenden Sie die in a) entwickelte Methode, um alle Textknoten eines Baumes in die Liste result einzufügen.

```
static void collectTextNodes(Node node,List<Node> result){
```

collectNodes
(node
, n -> n.getNodeType
== Node.TEXT_NODE
, result);

Name Matrikelnummer

Aufgabe 4 (Strings in C)

Gegeben sei die folgende Struktur, die String-Objekte realisiert.

```
#include <stdbool.h>

typedef struct{
  unsigned int length;
  char* data;
} String;
```

Listing 4: String.c

Schreiben Sie folgende Funktionen in C ohne Verwendung der Funktionen aus der Standardbibliothek `string.h`

a) (5 Punkte)

Schreiben Sie die Funktion `numberOfOccurrence(String this,char c)`
Es soll berechnet werden, wie oft das Zeichen c im String enthalten ist

```
int numberOfOccurrence(String this,char c){
int r = 0;
int i;
for (i=0; i<this.length; i++){
  if (this.data[i] == c) r++;
}
return r;
```

Name: Matrikelnummer:

b) (5 Punkte)

Schreiben Sie die Funktion `map(String this,char f(char))`.

Sie soll den String `this` so modifizieren, dass jeder Buchstabe durch die Funktion f verändert wird.

```
void map(String this,char f(char)){
  int i;
  for (i=0; i< this.length; i++){
    this.data[i] = f(this.data[i]);
  }
}
```

c) (5 Punkte)

Schreiben Sie die Funktion `removeFirst(String* this)`.

Der String soll so modifiziert werden, dass das erste Zeichen gelöscht wird und der String somit ein Zeichen weniger enthält.

```
void removeFirst(String* this){
  int i=0;
  for(i=1; i<this->length; i++){
    this->data[i-1]=this->data[i];
  }
  this->length--;
}
```

] (5 Punkte)

Schreiben Sie die Funktion `reverse(String this)`

Die Zeichen des Strings sollen so getauscht werden, dass sie in umgekehrter Reihenfolge erscheinen.

```
void reverse(String this){
int i;
for(i=0; i<this.length/2; i++){
  char c = this.data[i];
  this.data[i] = this.data
                 [this.length-i-1];
  this.data[this.length-i-1] = c;
}
```

Aufgabe 5 Gegeben sei folgende Datenstruktur für einen Binärbaum:

```
typedef struct BT{
  struct BT* left;
  int element;
  struct BT* right;
} BinTree;
```

Listing 3: Expression.h

Die linken und rechten Kinder können auch NULL sein. Wenn beide NULL sind, liegt ein Blatt vor.

Schreiben Sie folgende Funktionen für diese Datenstruktur.

a) (6 Punkte)

Schreiben Sie eine Konstruktorfunktion zum Erzeugen eines Baumes

```
BinTree* newBinTree(BinTree* left, int e,BinTree* right){
```

```
BinTree* this
    =(BinTree*)malloc(sizeof(BinTree));
this->left = left;
this->ekemat = e;
this->right = right;
return this;
```

(6 Punkte)

Schreiben Sie eine Methode, die testet, ob eine bestimmte Zahl i im Baum enthalten ist. (Es handelt sich nicht um einen binären Suchbaum.)

```
bool contains(BinTree* this, int i){
  if( this == NULL) return false;
  if(this->element == i ) return true;
  if(contains( this->left, i)) return true;
  return contains( this->right, i) ;
}
```

(6 Punkte)

Schreiben Sie eine Funktion, die einen Baum mit derselben Struktur erzeugt, dessen Knoten Zahlen haben, die mit der Funktion f aus der ursprünglichen Zahl an entsprechender Stelle berechnet wird.

```
BinTree* map(BinTree* this, int f(int)) {
  if (this == NULL) return NULL ;
  return newBinTree
    ( map( this -> left, f)
    , f( this -> element)
    , map( this -> right, f)
    );
}
```

Name: Matrikelnummer:

Aufgabe 6 Erklären Sie in kurzen Worten

a) Welche drei Phasen durchläuft ein Stream Objekt in Java. Nennen Sie für jede Phase exemplarisch eine Methode.

i) Erzeugung, z.B. aus Collection mit Methode stream().

ii) Modifizierung, z.B. filter, map limit ...

iii) Statt der Iteration durch terminierende Funktion: forEach, collect, reduce

b) In Java gibt es kein direktes Konstrukt, das die Aufgabe von `union` in C zur Bildung von Summentypen übernimmt. Wie kann man stattdessen die Summe zweier Typen realisieren.

Durch eine Schnittstelle, die von zwei Klassen implementiert wird. Die Schnittstelle ist dann der Summentyp.

Name: Matrikelnummer:

c) Was passiert mit einem Iteratorobjekt, nachdem es zum Iterieren verwendet wurde.

Es ist unbrauchbar und kann gar nicht mehr verwendet werden und wird im Regelfall von der garbage collection gelöscht.

d) Wozu dient der Operator & in C. Auf was kann er angewendet werden

i) Binärer Operator für bitweise And. Kann auf zwei Zahlen angewendet werden

ii) Unärer Operator als Adress operator Kann nur auf Variablen angewendet werden.

Name: Matrikelnummer:

e) Was berechnet in C der Operator `sizeof` für einen Parameter, mit dem ein Array übergeben wurde?

Die Länge der Adresse im Speicher
bei einer 64 Bit System also 8.