Aufgabe 1 (Iteratoren in Java)

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

a) (7 Punkte)

Gegen sei folgende Klasse für arraybasierte Listen:

```
1  import java.util.*;
2  import java.util.stream.*;
3  import java.util.function.*;
4  class Li<A>{
5    int size=0;
6    A[] data = (A[]) new Object[10];
7    void add(A a){
8      if (size>=data.length){
9        A[] newData = (A[])new Object[data.length+10];
10       for (int i = 0; i<size;i++) newData[i] = data [i];
11       data = newData;
12     }
13     data[size++] = a;
14   }
15 //}  wird fortgesetzt...
```

Listing 1: Li

Schreiben Sie eine Iteratorklasse `Literator<A>`. Die Klasse soll die Schnittstelle `Iterator<A>` implementieren.

Die Klasse soll im Konstruktor ein Objekt des Typs `Li<A>` erhalten. Der Iterator soll dann über alle Elemente der übergebenen Liste iterieren.

```
class Literator<A> implements Iterator<A>{
    int i=0;
    Li<A> li;
    Literator(Li<A> l){li=l;}
    @Override public boolean hasNext(){
        return i<li.size();
    }
    @Override public A next(){
        return li.data[i++];
    }
```

b) (7 Punkte)

Gegeben sei folgende Schnittstelle:

```
1 interface Fun<A,B>{
2   B apply(A a);
3 }
```

Listing 2: Fun

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

Sie soll zwei Objekte im Konstruktor übergeben bekommen: ein Objekt `fun` des Interfaces `Fun<A,B>` und ein Objekt `itA` des Typs `Iterator<A>`.

Wenn der übergebene Iterator `itA` über die Folge $(a_1, a_2, \ldots, a_n)$ iteriert, dann iteriere der Iterator `new MapIterator<>(fun,itA)` über die Folge:

$$(fun(a_1), fun(a_2), \ldots, fun(a_n))$$

```
class MapIterator<A,B> implements Iterator<B>{
  Fun<A,B> fun; Iterator<A> itA;
  MapIterator(Fun<A,B> F, Iterator<A> it){
    fun = F; itA = it;
  }
  @Override public boolean hasNext(){
    return itA.hasNext();
  }
  @Override public B next(){
    return fun.apply(itA.next());
  }
}
```

c) (4 Punkte)

Gegeben sei die folgende Klasse:

```
1 import java.util.*;
2 class Count implements Iterator<Integer>{
3   int i=0;
4   public int next(){return i++;}
5   public boolean hasNext(){return i<=Integer.MAX_VALUE && i>=0;}
6 }
```

Listing 3: Count

Machen Sie einen Aufruf, des Konstruktors der Klasse `MapIterator`, so dass ein Iteratorobjekt über Integer-Objekte entsteht, das absteigend über die nicht-negativen int Zahlen iteriert.

new MapIterator<>(x -> Integer.MAX_VALUE - x, new Count())

Aufgabe 2 (Spliteratoren)

Gegeben sei die Klasse für eine Array-basierte Liste aus Aufgabe 1.

Die Klasse enthalte jetzt zusätzlich folgende Spliterator-Implementierung:

```
1  public Stream<A> stream(){
2    return StreamSupport.stream(new LiSplit(this,0,size),true);
3  }
4  private class LiSplit implements Spliterator<A>{
5    Li<A> list;
6    int from;
7    int to;
8    LiSplit(Li<A> l,int f,int t){list = l;from=f;to=t;}
9
10   public boolean tryAdvance(Consumer<? super A> c){
11     if (to<=from) return false;
12
13     c.accept(list.data[from++]);
14     return true;
15   }
16
17   public int characteristics(){return 0;}
18   public long estimateSize(){return Long.MAX_VALUE;}
19
20   public Spliterator<A> trySplit(){
21     return null;
22   }
23 }
24 }
```

Listing 4: Li.java

a) (8 Punkte)

Schreiben Sie die Methode `trySplit` der inneren Klasse `LiSplit` neu, so dass tatsächlich nicht `null` sondern ein neuer Spliterator zurück gegeben wird, der über die Hälfte der Liste iteriert und der Originalspliterator über die andere Hälfte.

```
public Spliterator<A> trySplit(){ if(to-from<4) return null;
  var m=(from+to)/2;
  var r = new LiSplit(list, from, m);
  from = m;
  return r;
}
```

Aufgabe 3 (XML)

a) (8 Punkte)

Schreiben Sie mit dem DOM API (Paket `org.w3c.dom`) folgende Methode:

```
public static void collectText(Node node, StringBuffer result){
```

/* Sie soll den ganzen Baum durchlaufen und den Inhalt aller Textknoten mit `append` dem StringBuffer `result` zufügen. */

```
if (node.getNodeType() == Node.TEXT_NODE)
    result.append (node.getNodeValue());

var cs = node.getChildNodes();
for(int i =0; i<cs.getLength(); i++){
    collectText(cs.item(i), result);
}
}
```

b) (8 Punkte)

Gegeben sei die folgende Schnittstelle:

```
1 interface Pred<A>{
2   boolean test(A a);
3 }
```

Listing 5: Pred

Schreiben Sie mit dem DOM API folgende Methode:

```
static void collectWithProperty(Node n,Pred<Node> p,Set<Node> r){
```

/* Sie soll den ganzen Baum durchlaufen und alle Knoten mit der übergebenen Eigenschaft `p` in die Ergebnismenge `r` einfügen. */

```
if(p.test(n)) r.add(n);
var cs = n.getChildNodes();
for (int i=0; i<cs.getLength(); i++) {
    collectWithProperty(cs.item(i), p, r);
}
}
```

Aufgabe 4 (Strings in C)

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

```
1 typedef struct{
2   unsigned int length;
3   char* data;
4 } String;
```

Listing 6: String.c

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

a) (5 Punkte)

```
String mkStringForLength(unsigned int n){
```

/* Ein neues String-Objekt soll erzeugt werden. Dieses soll die übergebene Länge haben und bereits NULL-terminiert sein. */

```
String r;
r.length = length;
r.data = (char *) malloc( (length+1) * sizeof(char));
r.data[length] = '\0';
return r;
}
```

b) (5 Punkte)

```
String everySecond(String this){
```

/* Es soll ein neuer Teilstring erzeugt werden, der nur aus jedem zweiten Zeichen des übergebenen Strings besteht. Aus "abcdef" wird zum Beispiel "ace". */

```
String r = mkString ForLength ((this.length+1)/2);
for (int i = 0; i<r.length; i++){
    r.data[i] = this.data[2·i];
}
return r;
}
```

c) (5 Punkte)

```
bool contains(String this, bool p(char)){
```

/* Es soll geprüft werden, ob der String ein Zeichen enthält, für das die übergebene Funktion true ergibt. */

```
for(int i = 0; i<this.length; i++) {
    if (p(this.data[i])) return true;
}
return false;
}
```

d) (3 Punkte)

Machen Sie einen Aufruf von `contains`, bei dem geprüft wird, ob der String einen Großbuchstaben enthält.

```
bool isCapital(char c){
    return c >= 'A' && c <= 'Z';
}
.
.
.
String s = ...;

    contains(s, isCapital);
```

Aufgabe 5 Gegeben sei folgende Datenstruktur für einfache arithmetische Ausdrücke, die nur aus Zahlenliteralen und Substraktionsausdrücken bestehen.

```
1  typedef enum {
2    SUB_OP, LIT
3  } ExprType;
4
5  struct Expr;
6  typedef struct Expr Expression;
7
8  typedef struct {
9    long int value;
10 } LiteralExpr;
11
12 typedef struct{
13   Expression* left;
14   Expression* right;
15 } SubExpr;
16
17 typedef union{
18   SubExpr subExpr;
19   LiteralExpr literalExpr;
20 }  ExprUnion;
21
22 struct Expr{
23   ExprType type;
24   ExprUnion expr;
25 };
```

Listing 7: Expression.h

Schreiben Sie folgende Funktionen für diese Datenstruktur.

a) (3 Punkte)

Zum Erzeugen eines Literal-Ausdruckes:

```
Expression* newLiteralExpr(long int value){
    Expression* r = malloc(sizeof(Expression));
    r->type = LIT;
    r->expr.literalExpr.value = value;
    return r;


}
```

b) (3 Punkte)

Zum Erzeugen eines Subtraktions-Ausdruckes:

```
Expression* newSubExpr(Expression* left, Expression* right){
    Expression* r = malloc(sizeof(Expression));
    r->type = SUB_OP;
    r->expr.subExpr.left = left;
    r->expr.subExpr.right = right;
    return r;
}
```

c) (3 Punkte)

Zum Auswerten eines Literal-Ausdruckes:

```
long int evalLiteralExpr(LiteralExpr expr){
    return expr.value;
}
```

d) (3 Punkte)

Zum Auswerten eines Subtraktions-Ausdruckes:

```
long int evalSubOp(SubExpr expr){
    return eval(expr.left) - eval(expr.right);
}
```

e) (3 Punkte)

Zum Auswerten eines beliebigen Ausdruckes:

```
long int eval(Expression* this){
    if (this->type == LIT)
        return evalLiteralExpr(this->expr.literalExpr);
    return evalSubOp(this->expr.subExpr);
}
```

f) (3 Punkte)

Zum Löschen eines Subtraktions-Ausdruckes:

```
void deleteSubExpr(SubExpr binExpr){
    deleteExpr(binExpr.left);
    deleteExpr(binExpr.right);
    Free
}
```

g) (2 Punkte)

Zum Löschen eines beliebigen Ausdruckes:

```
void deleteExpr(Expression* this){
    if (this->type == SUB_OP)
        deleteSubExpr(this->expr.subExpr);
    Free this;
}
```

Aufgabe 6 Erklären Sie in kurzen Worten.

a) Warum kompiliert folgender Code nicht. Was wurde nicht beachtet?

```
1 void plusplus(int* x){
2   *x++;
3 }
4 int main(){
5   plusplus(&(40+1));
6   return 0;
7 }
```

Listing 8: plusplus.c

Der Adressoperator & darf nur auf Variablen angewendet werden und nicht auf temporäre Ergebnisse von Berechnungen. Zeile 5 gibt somit einen Fehler bei der Kompilierung.

b) Welche unterschiedlichen Bedeutungen hat das Symbol * in C?

Es hat 3 Bedeutungen:

i) binärer Infixoperator der Multiplikation.
ii) unärer Präfixoperator der Dereferenzierung.
iii) Teil des Typs, der angibt, dass es sich um einen Zeigertyp handelt.

c) Welches sind die zwei wichtigsten Speicherbereiche beim Ausführen eines Programms in Bezug auf die Daten der Anwendung. Was unterscheidet Java und C in Hinblick auf diese Speicherbereiche?

Die Speicherbereiche sind der Heap und der Stack. Auf dem Stack werden Parameter und lokale Variablen für einen Funktionsaufruf gespeichert. Auf dem Heap liegen Daten, die über einen Funktionsaufruf hinaus existieren und über ihre Adresse referenziert werden. In Java werden nicht mehr benötigte Heap Objekte automatisch gelöscht, in C muss dieses explizit im Programm durch free geschehen.

d) Vergleichen Sie die APIs SAX und DOM? Was sind die jeweiligen Vor- und Nachteile?

e) Was ist die Ausgabe folgenden Programms. Erklären Sie, wie es dazu kommt.

```
1 #include <stdio.h>
2 int main(){
3   int xs [] = {1,2,3};
4   int* ys = xs;
5   printf("%ld\n",sizeof(xs));
6   printf("%ld\n",sizeof(ys));
7   return 0;
8 }
```

Listing 9: array.c

In Zeile 5 wird 12 ausgegeben, weil bekannt ist, dass dort 3 int Zahlen im Speicher stehen.

In Zeile 6 wird 8 ausgegeben, weil die Größe der Adressen eines 64 Bit Systems gefragt wird.