

Aufgaben zur Klausur C im WS 2014/15 (IA 302)

Zeit: 90 Minuten

erlaubte Hilfsmittel: keine

Bitte tragen Sie Ihre Antworten und fertigen Lösungen ausschließlich an den gekennzeichneten Stellen in das Aufgabenblatt ein. Ist ihre Lösung wesentlich umfangreicher, so überprüfen Sie bitte nochmals Ihren Lösungsweg.

Nutzen Sie die Rückseiten der Klausur zur Entwicklung der Lösungen und übertragen die fertigen Lösungen in das Aufgabenblatt.

Sollten Unklarheiten oder Mehrdeutigkeiten bei der Aufgabenstellung auftreten, so notieren Sie bitte, wie Sie die Aufgabe interpretiert haben.

Tipp: Bei der Entwicklung der Lösung können kleine Skizzen manchmal hilfreich sein.

Viel Erfolg!

Diese Klausur besteht einschließlich dieses Deckblattes aus 8 Seiten.

---

## Aufgabe 1:

In der Vorlesung sind mehrere Datenstrukturen zur Laufzeit- und Speicherplatz-effizienten Implementierung von Mengen von ganzen Zahlen entwickelt worden. In dieser Aufgabe soll eine weitere Möglichkeit teilweise entwickelt werden. Eine Menge von ganzen Zahlen kann in zusammenhängende Bereiche (Intervalle) aufgeteilt werden. Diese Intervalle können in einer sortierten, einfach verketteten Liste gespeichert werden. So kann die Menge  $m_1 = \{1, 2, 3, 5, 7, 8, 9\}$  durch eine dreielementige Liste von Bereichen  $l_1 = [1..3, 5..5, 7..9]$  repräsentiert werden.

Für eine effiziente Implementierung ist zu beachten, dass immer mit der minimalen Anzahl von Intervallen gearbeitet wird. So sollte nach dem Einfügen des Wertes 4 in die Menge  $m_1$  eine neue Liste  $l_2$  mit nur noch 2 Intervallen entstehen. Wird dann anschließend noch die 6 eingefügt, genügt ein Intervall zur Darstellung des Resultats.

Die folgende C Header Datei enthält Deklarationen für eine solche Implementierung.

```
#include <stdlib.h>
#include <assert.h>
typedef struct {
    int lb;
    int ub;
} Range;
typedef struct node * IntSet;
struct node {
    Range info;
    IntSet next;
};
#define isEmpty(s) ((s) == (IntSet)0)
#define mkEmpty() ((IntSet)0)
extern int min(int i, int j);
extern int max(int i, int j);

extern int isEmptyRange(Range r);
extern int less(Range r1, Range r2);
extern int overlap(Range r1, Range r2);
extern Range mkRange(int lb, int ub);
extern Range unionRange(Range r1, Range r2);

extern unsigned int length(IntSet s);
extern unsigned int card(IntSet s);

extern int invIntSet(IntSet s);
extern int isIn(int e, IntSet s);

extern IntSet mk1Set(Range r);
extern IntSet insertRange(Range r, IntSet s);
extern IntSet insertInt(int e, IntSet s);
extern IntSet unionIntSet(IntSet s1, IntSet s2);
```

Die im Headerfile eingeführten Größen sind bei der Entwicklung der folgenden Routinen zu verwenden. Einige einfache Funktionen sind wie folgt implementiert:

```
#include "IntSet.h"

int max(int i, int j) {
    return (i > j) ? i : j;
}
int min(int i, int j) {
    return (i < j) ? i : j;
}
Range mkRange(int lb, int ub) {
    Range res = {lb, ub};
    return res;
}
int isEmptyRange(Range r) {
    return r.lb > r.ub;
}
int less(Range r1, Range r2) {
    return r1.ub < r2.lb;
}
unsigned int length(IntSet s) {
    return (isEmpty(s)) ? 0 : 1 + length(s->next);
}
unsigned int card(IntSet s) {
    return (isEmpty(s)) ? 0 :
        (s->info.ub - s->info.lb + 1) + card(s->next);
}
```

Vorsicht: Aufrufe von *mkRange* in der Form *mkRange(5,4)* sind möglich, solche Intervalle sind aber in *IntSet*-Werten nicht sinnvoll.

Welche dieser Funktionen könnte zur Steigerung der Laufzeiteffizienz durch Makros ersetzt werden:

- 1) .....
- 2) .....
- 3) .....
- 4) .....
- 5) .....

Implementieren Sie als erstes eine Hilfsfunktion *overlap*. In dieser soll getestet werden, ob sich zwei Intervalle überlappen oder aneinander stoßen.

Die *overlap* Funktion:

```
#include "IntSet.h"

int overlap(Range r1, Range r2) {
    assert(! isEmptyRange(r1));
    assert(! isEmptyRange(r2));

    .....

    .....

    .....

    .....

    .....

    .....
}
```

Eine weitere nützliche Hilfsfunktion ist *unionRange* zum Verschmelzen zweier Intervalle:

```
#include "IntSet.h"
#include <assert.h>

Range unionRange(Range r1, Range r2) {
    assert(! isEmptyRange(r1));
    assert(! isEmptyRange(r2));
    assert(overlap(r1, r2));

    .....

    .....

    .....

    .....
}
```







