Prof. Dr. Uwe Schmidt

ptlwedel
PRIVATE BERUFSFACHSCHULE

6. Februar 2014

Aufgaben zur Klausur **C** im WS 2013/14 (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:

Gegeben seien die folgenden sieben Hash-Funktionen für Strings:

```
typedef unsigned int Hash;
typedef char *String;

Hash hash1(String s) {
    return *s;
}

Hash hash2(String s) {
    Hash res;
    for (res = 0; *s; res += *s, ++s);
    return res;
}

Hash hash3(String s) {
    return (Hash) s;
}

Hash hash4(String s) {
    Hash res;
    for (res = 0; *s; res = 31 * res + *s, ++s);
    return res;
}

Hash hash5(String s) {
    Hash res;
    for (res = 0; *s; res >>= 5, res += *s, ++s);
    return res;
}

Hash hash6(String s) {
    Hash res;
    for (res = 1; *s; res *= *s, ++s);
    return res;
}

Hash hash7(String s) {
    Hash res;
    for (res = 0; *s; res = (res << 5) + *s - res, ++s);
    return res;
}
```

1. Welche dieser Funktionen erfüllen nicht die funktionalen Anforderungen an eine Hash-Funktion?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

2. Welche Funktionen sind aus Effizienzgründen als Hash-Funktion ungeeignet?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

3. Welche dieser Funktionen sind als Hash-Funktionen ungeeignet, da die Hash-Werte sehr ungleichmäßig häufig getroffen werden?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

4. Welche dieser Funktionen sind als Hash-Funktionen ungeeignet, da sie *ähnliche* Werte auf gleiche Hash-Werte abbilden?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

5. Welche Funktionen sind für die effizente Implementierung von Hash-Tabellen geeignet?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

6. Welche dieser Funktionen arbeiten bei der Berechnung mit undefinierten Werten, liefern also zufällige Resultate?

   hash1 ☐ hash2 ☐ hash3 ☐ hash4 ☐ hash5 ☐ hash6 ☐ hash7 ☐

Aufgabe 2:

Die folgende C Header Datei enthält Deklarationen für eine Listenimplementierung mit verketteten Listen.

```
#include <string.h>

typedef char * Element;
typedef struct node * List;
struct node {
   Element info;
   List next;
};

#define isEmpty(l) ((l) == (List)0)

extern int compare(Element e1, Element e2);
extern int invList(List l);
extern List merge(List l1, List l2);
```

Die hier eingeführten Größen sind bei der Lösung der folgenden Aufgaben zu verwenden.

Implementieren Sie als erstes die *compare* Funktion. Diese soll zwei Elemente vergleichen und als Resultat die Werte -1, 0 und $+1$ liefern, -1 wenn $e1 < e2$ gilt, $+1$ wenn $e1 > e2$ gilt, 0 sonst.

Die *compare* Funktion:

```
#include "List.h"

int compare(Element e1, Element e2) {

   ..........

   ..........

   ..........

   ..........

   ..........
}
```

In dieser Aufgabe soll mit sortierten verketteten Listen gearbeitet werden. Die Listen sollen als absteigend sortierte Listen organisiert sein, doppelte Elemente sind nicht erlaubt.

Entwickeln Sie die entsprechende Invariante für diese Bedingungen.

```
#include "List.h"

int invList(List l) {
    ..........
    ..........
    ..........
    ..........
    ..........
    ..........
    ..........
    ..........
    ..........
    ..........
}
```

Es fehlt aus der Header Datei noch die *merge* Funktion. Entwickeln Sie diese Funktion so, dass aus den Knoten der Listen $l1$ und $l2$ eine neue Liste aufgebaut wird, die alle Elemente aus $l1$ und $l2$ enthält und die Invariante wieder gilt. Diese Funktion soll rekursiv arbeiten. Die Funktion soll keine neuen Knoten erzeugen, sondern nur die Knoten aus $l1$ und $l2$ neu verketten. Nicht weiter genutzter Speicher soll freigegeben werden.

```
#include "List.h"

List merge(List l1, List l2) {
   .....
   .....
   .....
   .....
   .....
   .....
   .....
   .....
   .....
   .....
   .....
   .....
   .....
   .....
}
```

Aufgabe 3:

Gegeben seien die folgenden Variablen:

int x;
unsigned int u;
long int s;
float f;
char ∗p1;
long int ∗p2;
void ∗p3, ∗p4;

Bestimmen Sie für die folgenden Ausdrücke den Typ gemäß ANSI-C. Vorsicht: Es kommen fehlerhafte und logisch falsche Ausdrücke von. Kennzeichnen Sie diese mit dem Wort **FEHLER**

p2[*p2] .....

s || x || p1 .....

s | x | p1 .....

p1 ? x : f .....

++f, x-- .....

p1 == p3 .....

p3 == p4 .....

x = p3 == p4 .....

x == p3 == p4 .....

1 + ~p2 .....

p1 && p3 .....

x += f .....

!!! p3 .....

!!! s .....

~~~s .....

0x23 + p1 + 0x23 .....
~~~