Prof. Dr. Uwe Schmidt

fhwedel
UNIVERSITY OF APPLIED SCIENCES

30. Januar 2017

Aufgaben zur Klausur **Systemnahe Programmierung** im WS 2016/17 (B_Inf, B_TInf, B_MInf, B_CGT, B_ITE)

Zeit: 75 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 Typdefinitionen und Variablendeklarationen:

```
#include <stdlib.h>

typedef struct X * Tree;

typedef Tree (*Tf)(Tree);

struct X {
  char * k;
  struct D * a;
  Tree cs[2];
};

struct D {
  Tf f;
  char * n;
  unsigned int i;
};

Tree root;

long int li = 2;
```

Bestimmen Sie für die folgenden Ausdrücke den Typ gemäß ANSI-C. Vorsicht: Sollten Ausdrücke vorkommen, die zur Übersetzungszeit Fehlermeldungen erzeugen, so kennzeichnen Sie diese mit dem Wort `FEHLER`

root→k .....

root→k[1] .....

∗((∗root).k) .....

(∗(root→cs))→a→f .....

root→a→i + li .....

(root→a→f)(root) .....

root→a→i++ .....

∗(root→cs + 1) .....

root→cs[li] .....

root ? root→cs : 0 .....

root ? root→cs[0] : root .....

sizeof ∗root .....

sizeof (**struct** X) .....

Aufgabe 2:

Gegeben sei das folgende C–Programm zur Verarbeitung von Mengen als Bitstrings.

```
#include <stdio.h>

typedef unsigned char Menge;
#define MengeMax 8

void printMenge(Menge s) {
    unsigned int i = MengeMax;
    while ( i-- != 0 ) {
      printf("%1u", (unsigned int)((s >> i) & 1));
      if (i == 4)
        printf("  ");
    }
}
static unsigned int linecnt = 0;
#define PRINT(s) { printf("%2u)  ", ++linecnt); printMenge(s); printf("\n"); }

#define leer ((Menge)0)
#define voll ((Menge)-1)
#define einElement(i) ( (Menge)(1 << (i)) )
#define einStueck(n,m) (dieErsten(m+1) ^ dieErsten(n))
#define dieErsten(n) (einElement(n) - 1)

int main(void) {
    Menge s1;
    PRINT( einElement(0) );
    PRINT( einElement(MengeMax - 1) );
    PRINT( einElement(MengeMax) );

    PRINT( (Menge)1 );
    PRINT( voll );
    PRINT( einStueck(5,5) );
    PRINT( einStueck(1,4) );
    PRINT( einStueck(0,MengeMax-1) );

    PRINT( leer ^ einStueck(2,6) );
    PRINT( ~einStueck(2,6) );

    s1 = einStueck(1,4) | ~einStueck(2,6); PRINT(s1);
    s1 = einStueck(1,4) & einStueck(2,6); PRINT(s1);

    s1 = 4 + 16 + 32;
    s1 = s1 ^ (s1 & (~s1 + 1)); PRINT(s1);
    return 0;
}
```

Die Mengen sind in diesem Beispiel 8 Bits lang, können also die Elemente $0, 1, \ldots, 7$ enthalten. *printSet* gibt eine Menge im Binärformat aus. Die Menge, die nur die 1 enthält würde als $0000\ 0010$ ausgegeben werden. Das *PRINT* Makro gibt jeweils eine Menge pro Zeile aus und numeriert die Zeilen durch.

Welche 13 Ausgabezeilen erzeugt dieses Programm?

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

8) .....

9) .....

10) .....

11) .....

12) .....

13) .....

Aufgabe 3:

Gegeben sei das folgende C–Programm:

```
#include <stdio.h>

#define maxm(x,y) ((x) > (y) ? (x) : (y))

long maxf(long x, long y) { return x > y ? x : y; }

unsigned cnt = 0;

#define eval(x) ( ++cnt, (x) )


int main(void) {
    long f[] = { 1, +3, −5, +7, −9 };
    long res;

    double g[] = { 4.1, 0.5, 5.9 };
    double r;

    cnt = 0;
    res = maxm( maxm( eval(f[1]), eval(f[2]) ),
                eval(f[3]) );

    printf(" res = %ld, cnt = %u\n", res, cnt);

    cnt = 0;
    res = maxf( maxf( eval(f[1]), eval(f[2]) ),
                eval(f[3]) );

    printf(" res = %ld, cnt = %u\n", res, cnt);

    cnt = 0;
    r = maxm( eval(g[1]), eval(g[2]) );

    printf(" r = %3.1f, cnt = %u\n", r, cnt);

    cnt = 0;
    r = maxf( eval(g[1]), eval(g[2]) );

    printf(" r = %3.1f, cnt = %u\n", r, cnt);

    return 0;
}
```

Welche vier Ausgabezeilen erzeugt dieses Programm:

1) .....

2) .....

3) .....

4) .....

Aufgabe 4:

Gegeben sei das folgende Programm:

```
#include <stdio.h>

char * tab [] = {"Haschisch", "Unterleib", "Geschenk", "Blei", "Laster"};

char * * ptab [] = { tab + 4, tab + 3, tab + 2, tab + 1, tab };

char * * * ppp = ptab;

int main ( int argc , char * argv [] )
{
   printf( "%s\n" , * (* (ppp + 3) - 1) + 6 );
   printf( "%s\n" , ppp [3] [0] + 6 );
   printf( "%s\n" , * (* (ppp + 2)) + 5 );
   printf( "%s\n" , * (* (1 + ppp)) + 1 );
   printf( "%s\n" , * (* (ppp + 0)) + 2 );

   return 0;
}
```

Welche Ausgabezeilen liefert dieses Programm:

1) .....

2) .....

3) .....

4) .....

5) .....

Wie viel Speicher wird von den Variablen tab, ptab und ppp und den in den Initialisierungen vorkommenden Konstanten benötigt? Geben Sie hierfür einen Ausdruck mit dem **sizeof**-Operator an. (z.B. 2 * sizeof(char) + 5 * sizeof(char*) + ...)

.....

.....