Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

5. August 1997

Aufgaben zur Klausur **C** im SS 97 (II21)

Zeit: 150 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.

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

Bei allen C Routinen sollen die Parameter und Funktionsresultate mit ANSI–C Prototypen angegeben werden.

Viel Erfolg !

Diese Klausur besteht einschließlich dieses Deckblattes aus 12 Seiten

---

Aufgabe 1:

Gegeben sei das folgende C–Programm. Hinweis: der $\ll$ Operator hat geringere Priorität als + und -.

```
#include <stdio.h>
#include <math.h>

#define twotimes(x) x + x
#define sqr(x) x * x
#define min(x,y) x < y ? x : y
#define squareroot(x) (++cnt, sqrt(x))

int cnt = 0;

int main(void)
{
    int i = 4, j = 3, r;
    double f = 1.25;

    r = twotimes(i << 2);
    printf(".1) r = %d\n",r);

    r = sqr(i + j);
    printf(".2) r = %d\n",r);

    r = 2 * twotimes(i);
    printf(".3) r = %d\n",r);

    r = sqr(twotimes(i));
    printf(".4) r = %d\n",r);

    r = twotimes(2*f);
    printf(".5) r = %d\n",r);

    r = min(twotimes(i+1),sqr(i));
    printf(".6) r = %d\n",r);

    r = min(squareroot(i+1),squareroot(j+1));
    printf(".7) r = %d, cnt = %d\n",r,cnt);

    return 0;
}
```

Welche 7 Ausgabezeilen erzeugt dieses Programm

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Ändern Sie das Makro *sqr* so ab, daß es für `.2)` den gewünschten Effekt hat.

. . . . . .

Ändern Sie das Makro *twotimes* so ab, daß es sich immer wie eine Funktion verhält

. . . . . .

Warum kann das Makro *min* nicht so verändert werden, das es sich immer, bei jedem Gebrauch, wie eine gleichwertige Funktion verhält?

. . . . . .

Warum kann der Gebrauch von Makros in der Form von *min* die Programmausführung verlangsamen (im Vergleich zu Funktionen).

. . . . . .

Aufgabe 2:

Gegeben sei das folgende C–Programm:

```
#include <stdio.h>

int main(void)
{
    int i = 5;

    printf("%d,%d\n",i++,i++);

    return 0;
}
```

Dieses Programm gibt eine Zeile aus. Welche Ausgaben sind für dieses Programm vom ANSI-C Standard zugelassen:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 3:

Die folgende C–Funktion testet, ob die Zeichenreihe $s1$ Anfangsstück der Zeichenreihe $s2$ ist.

```
int isPrefix(char * s1, char * s2)
{
    return
        *s1 == 0
        || ( *s1 == *s2
             &&
             isPrefix(s1+1, s2+1) );
}
```

Formen sie diese Funktion um in eine gleichwertige, die keine Funktionsaufrufe mehr enthält.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Aufgabe 4:

Welche der folgenden Ausdrücke sind equivalent zu dem Ausdruck $a[j][k]$

$*(a[j] + k)$

ja ☐ nein ☐

$**(a[j + k])$

ja ☐ nein ☐

$(*(a + j))[k]$

ja ☐ nein ☐

$(*(a + k))[j]$

ja ☐ nein ☐

$*((*(a + j)) + k)$

ja ☐ nein ☐

$**(a + j) + k$

ja ☐ nein ☐

$*(\&a[0][0] + j + k)$

ja ☐ nein ☐

Aufgabe 5:

Gegeben sei das folgende Modul zur Verarbeitung von Graphen:

```
#include <stdlib.h>
#include <stdio.h>

typedef int Value;

typedef struct node * Tree;
struct node {
    Value v;
    int mark;
    Tree l;
    Tree r;
};

#define empty ((Tree)0)

/* eine Konstruktor-Funktion */

Tree mk(Value v, Tree l, Tree r) {
    Tree t = malloc(sizeof(*t));
    if (t == 0)
        exit(1);

    t->v = v;
    t->mark = 0;
    t->l = l;
    t->r = r;

    return t;
}

typedef void (*Proc)(Tree);

/* eine eigene Kontroll-Prozedur fuer einen Baumdurchlauf */

void forall(Tree t, Proc p) {
    if (t != empty) {
        forall(t->l,p);
        (*p)(t);
        forall(t->r,p);
    }
}
```

```
/* die zu entwickelnden Routinen */

extern void forall_once(Tree t, Proc p);
extern void forall_postorder(Tree t, Proc p);
extern void print1_Tree(Tree t);

/* einige Routinen zum Verarbeiten eines Knotens */

void print(Tree t) { printf("%d,", t→v); }
void incr(Tree t) { ++(t→v); }
void reset(Tree t) { t→mark = 0; }

/* Ausgabe eines Baums */

void print_Tree(Tree t) {
    forall(t,print);
    printf("\n");
}
```

und ein kleines Testprogramm

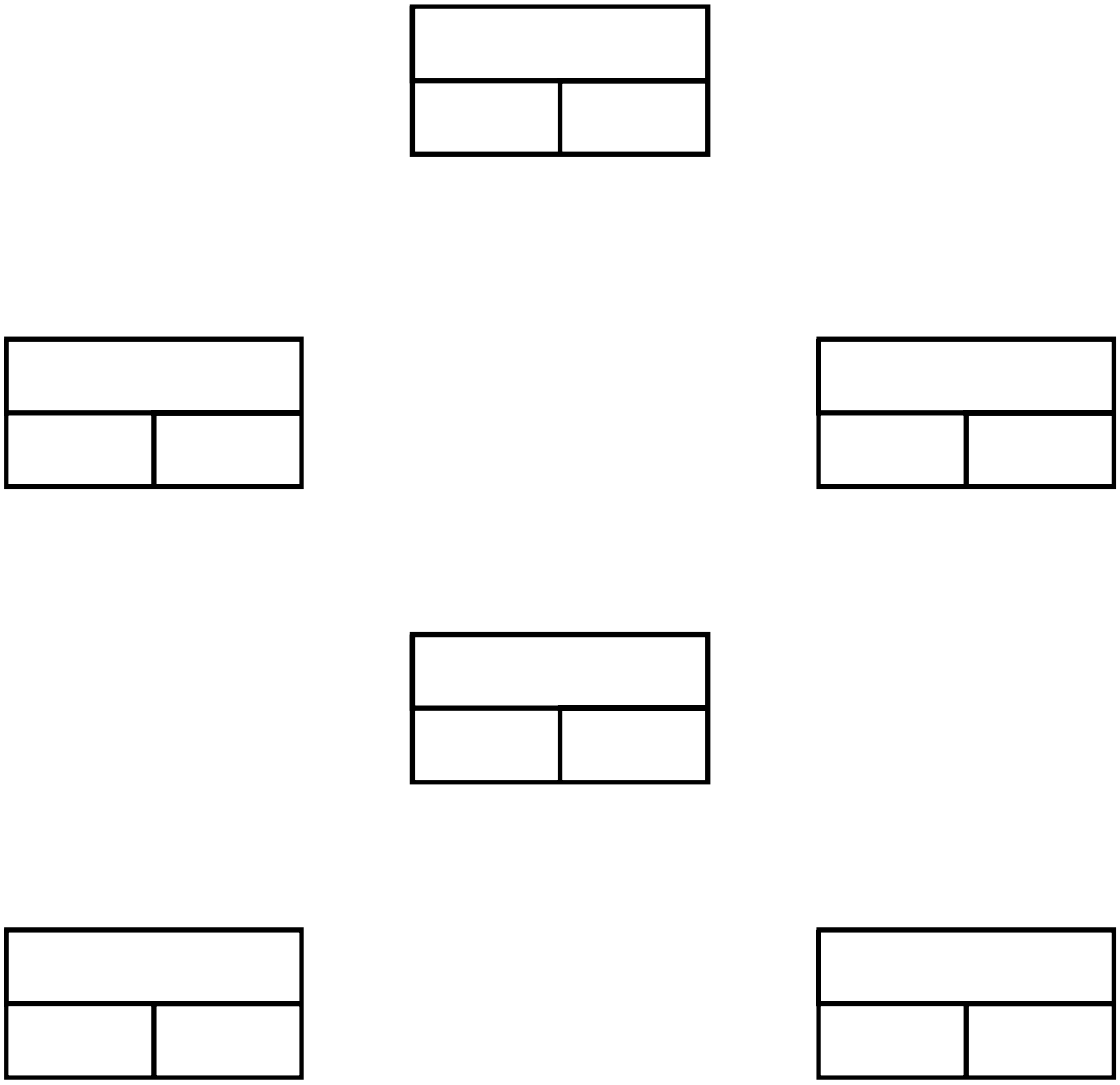
```
/* ein Testprogramm */

int main(void) {
    Tree t1 = mk(13,empty,empty);
    Tree t2 = mk(3,t1,mk(17,empty,empty));
    Tree t3 = mk(5,
                 mk(7,t1,t2),
                 mk(11,t2,empty));

    print_Tree(t3);
    forall(t3,incr);
    print_Tree(t3);

    return 0;
}
```

Die in dem Testprogramm aufgebaute dynamische Datenstruktur kann man mit der folgenden (unvollständigen) Skizze veranschaulichen



Welche beiden Ausgabezeilen erzeugt dieses Programm:

1) .....

2) .....

Wie muß man die *forall*-Funktion zu einer Funktion *forall_postorder* modifizieren, so daß die Wurzelknoten als letztes verarbeitet werden?

die modifizierte *forall_postorder*-Funktion:

.....

.....

.....

.....

.....

.....

.....

.....

Wie muß man die *forall*-Funktion zu einer *forall_once*-Funktion modifizieren, so daß alle Knoten nur einmal verarbeitet werden? Hinweis: nutzen Sie das *mark* Feld.

die modifizierte *forall_once*-Funktion:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Wie muß man die *print_Tree*-Funktion zu einer *print1_Tree*-Funktion modifizieren, damit alle Elemente nur einmal ausgegeben werden, der Graph aber nach einem Aufruf genau den gleichen Zustand hat wie vor dem Aufruf?

die modifizierte *print1_Tree*-Funktion:

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

Welche drei Ausgabezeilen würde das folgende Testprogramm mit den neuen Routinen liefern?

```
int main(void) {
    Tree t1 = mk(13,empty,empty);
    Tree t2 = mk(3,t1,mk(17,empty,empty));
    Tree t3 = mk(5,
                 mk(7,t1,t2),
                 mk(11,t2,empty));

    forall_postorder(t3,print); printf("\n");

    print1_Tree(t3);

    forall_once(t3,incr);
    forall(t3,reset);

    print1_Tree(t3);

    return 0;
}
```

Die Ausgabe:

1) .....

2) .....

3) .....
