Fachhochschule Wedel
Prof. Dr. Uwe Schmidt

10. August 2001

Aufgaben zur Klausur **C** im SS 2001 (PI h742)

Zeit: 60 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.

Viel Erfolg !

Diese Klausur besteht einschließlich dieses Deckblattes aus 7 Seiten

---

Aufgabe 1:

Gegeben seien die folgenden Typdefinitionen und Variablendeklarationen:

```
typedef struct X * Tree;

typedef Tree (*Tf)(Tree);

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

struct D {
  enum E {A, B, C} t;
  union U {
    int i;
    Tf f;
    char * s;
  } v;
};

Tree root;
```

Bestimmen Sie für die folgenden Ausdrücke den Typ gemäß ANSI-C. Vorsicht: Es kommen Ausdrücke vor, die zur Übersetzungszeit Fehlermeldungen erzeugen. Kennzeichnen Sie diese mit dem Wort `FEHLER`

root→k .....

root→k[1] .....

root→k[2] .....

∗((∗root).k) .....

(∗(root→cs))→a→t .....

∗(root→cs + 1) .....

root→a→v.f .....

(root→a→v.f)(root) .....

root→a→v.s++ .....

root ? root→cs : 0 .....

root ? root→cs[0] : root .....

sizeof ∗root .....

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

Des weiteren seien folgende Funktionen definiert:

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

Tree mkTree(char * k,
            struct D * a,
            Tree t1,
            Tree t2) {
  Tree res = malloc(sizeof *res);
  if (! res) exit(1);
  res->k = k;
  res->a = a;
  res->cs[0] = t1;
  res->cs[1] = t2;
  return res;
}

struct D * mkDi(int i) {
  struct D * res = malloc(sizeof *res);
  if (! res) exit(1);
  res->t = A;
  res->v.i = i;
  return res;
}

struct D * mkDf(Tf f) {
  struct D * res = malloc(sizeof *res);
  if (! res) exit(1);
  res->t = B;
  res->v.f = f;
  return res;
}

Tree left(Tree t) {
  assert(t != 0);
  return t->cs[0];
}

Tree right(Tree t) {
  assert(t != 0);
  return t->cs[1];
}
```

Welche Ausgaben erzeugt das folgende Programm? Es kommen in diesem Programm Ausdrücke vor, die nicht definiert sind. Kennzeichnen Sie diese mit UNDEF in der Lösungszeile. Geben Sie alle Ausgaben an, auch wenn vorher undefinierte Auswertungen gemacht wurden.

```
#include <stdio.h>

int main(void) {
  Tree t1, t2;

  t1 = mkTree("t1", mkDi(43), 0, 0);
  t2 = mkTree("t2", mkDi(41), t1, 0);
  root = mkTree("root", mkDf(left), t1, t2);

  printf("1) %s\n", root→k+1);
  printf("2) %d\n", (int)(root→a→t));
  printf("3) %d\n", root→a→v.i);
  printf("4) %s\n", (root→a→v.f)(root)→k);
  printf("5) %s\n", root→a→v.s);
  printf("6) %d\n", root→cs[1]→cs[0]→a→v.i+1);
  printf("7) %d\n", (*(root→cs))→cs[0]→a→v.i);

  return 0;
}
```

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

Die Funktionen “left” und “right” sind so kurz, dass eine Realisierung als Makro sinnvoll erscheint.

Das folgende Makro, bei dem auf die Überprüfung der Zusicherung verzichtet wurde, hat grobe Mängel:

```
#define left(t) t→cs[0]
```

Korrigieren Sie dieses Makro:

. . . . . .

Warum ist ein Makro für die “left” Funktion mit Zusicherung nicht sinnvoll?

. . . . . .

. . . . . .

Aufgabe 2:

Gegeben sei das folgende Programm

```
#include <stdio.h>

int main(int argc, char *argv[]) {

  int i, sum=0;

  for(i=0; i<10; ++i) {
    switch (i) {
    case 2:
    case 3:
      continue;
    case 1:
    case 4:
    case 7:
    case 0:
      sum += 3*i+1;
    default:
      continue;
    case 6:
      break;
    }
    break;
  }

  printf("%d\n",sum);

  return 0;
}
```

Welche Ausgabe erzeugt dieses Programm?

.....

Physikalisch–Technische Lehranstalt Wedel
Prof. Dr. Uwe Schmidt

10. August 2001

Aufgaben zur Klausur **C** im SS 2001 (IA 302)

Zeit: 60 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.

Viel Erfolg !

Diese Klausur besteht einschließlich dieses Deckblattes aus 7 Seiten

---

Aufgabe 1:

Gegeben seien die folgenden Typdefinitionen und Variablendeklarationen:

```
typedef struct X * Tree;

typedef Tree (*Tf)(Tree);

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

struct D {
  enum E {A, B, C} t;
  union U {
    int i;
    Tf f;
    char * s;
  } v;
};

Tree root;
```

Bestimmen Sie für die folgenden Ausdrücke den Typ gemäß ANSI-C. Vorsicht: Es kommen Ausdrücke vor, die zur Übersetzungszeit Fehlermeldungen erzeugen. Kennzeichnen Sie diese mit dem Wort `FEHLER`

root→k .....

root→k[1] .....

root→k[2] .....

∗((∗root).k) .....

(∗(root→cs))→a→t .....

∗(root→cs + 1) .....

root→a→v.f .....

(root→a→v.f)(root) .....

root→a→v.s++ .....

root ? root→cs : 0 .....

root ? root→cs[0] : root .....

sizeof ∗root .....

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

Des weiteren seien folgende Funktionen definiert:

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

Tree mkTree(char * k,
            struct D * a,
            Tree t1,
            Tree t2) {
  Tree res = malloc(sizeof *res);
  if (! res) exit(1);
  res->k = k;
  res->a = a;
  res->cs[0] = t1;
  res->cs[1] = t2;
  return res;
}

struct D * mkDi(int i) {
  struct D * res = malloc(sizeof *res);
  if (! res) exit(1);
  res->t = A;
  res->v.i = i;
  return res;
}

struct D * mkDf(Tf f) {
  struct D * res = malloc(sizeof *res);
  if (! res) exit(1);
  res->t = B;
  res->v.f = f;
  return res;
}

Tree left(Tree t) {
  assert(t != 0);
  return t->cs[0];
}

Tree right(Tree t) {
  assert(t != 0);
  return t->cs[1];
}
```

Welche Ausgaben erzeugt das folgende Programm? Es kommen in diesem Programm Ausdrücke vor, die nicht definiert sind. Kennzeichnen Sie diese mit UNDEF in der Lösungszeile. Geben Sie alle Ausgaben an, auch wenn vorher undefinierte Auswertungen gemacht wurden.

```
#include <stdio.h>

int main(void) {
  Tree t1, t2;

  t1 = mkTree("t1", mkDi(43), 0, 0);
  t2 = mkTree("t2", mkDi(41), t1, 0);
  root = mkTree("root", mkDf(left), t1, t2);

  printf("1) %s\n", root→k+1);
  printf("2) %d\n", (int)(root→a→t));
  printf("3) %d\n", root→a→v.i);
  printf("4) %s\n", (root→a→v.f)(root)→k);
  printf("5) %s\n", root→a→v.s);
  printf("6) %d\n", root→cs[1]→cs[0]→a→v.i+1);
  printf("7) %d\n", (*(root→cs))→cs[0]→a→v.i);

  return 0;
}
```

1) .....

2) .....

3) .....

4) .....

5) .....

6) .....

7) .....

Die Funktionen “left” und “right” sind so kurz, dass eine Realisierung als Makro sinnvoll erscheint.

Das folgende Makro, bei dem auf die Überprüfung der Zusicherung verzichtet wurde, hat grobe Mängel:

```
#define left(t) t→cs[0]
```

Korrigieren Sie dieses Makro:

. . . . . .

Warum ist ein Makro für die “left” Funktion mit Zusicherung nicht sinnvoll?

. . . . . .

. . . . . .

Aufgabe 2:

Gegeben sei das folgende Programm

```
#include <stdio.h>

int main(int argc, char *argv[]) {

  int i, sum=0;

  for(i=0; i<10; ++i) {
    switch (i) {
    case 2:
    case 3:
      continue;
    case 1:
    case 4:
    case 7:
    case 0:
      sum += 3*i+1;
    default:
      continue;
    case 6:
      break;
    }
    break;
  }

  printf("%d\n",sum);

  return 0;
}
```

Welche Ausgabe erzeugt dieses Programm?

.....
