Lösung der Klausur 2

Datum der Klausur: 19. Juni 2020, 16:45 bis 18:15

Aufgabe 1 (5 Pkt.)

Gegeben ist das folgende fehlerhafte C Programm.

```
1  #include <stdlib.h>
2  #include <stdbool.h>
3  #include <math.h>
4  #include <time.h>
5
6  int main(void){
7      srand(time(NULL)); //Initialisierung des Zufallszahlengenerators
8      bool isPrime = false;
9      while(!isPrime)
10         int N = 1+rand()%99; //Weist N eine Zufallszahl in [1,99] zu
11         isPrime=true;
12         for(int i=1; i<=sqrt(N); ++i){
13             if(N%i=0){
14                 isPrime = false;
15                 break;
16             }
17         }
18     printf("Found a prime number: %d\n", N);
19     return 0;
20 }
```

Das Programm soll eigentlich zufällig eine ganze Zahl zwischen 1 und 99 auswählen, bis eine Primzahl gefunden wurde. Finde und nenne die fünf im Programm befindlichen Fehler mit einer kurzen Erklärung, warum es sich um einen Fehler handelt. Gib eine entsprechende korrigierte Version des Programms an.
Hinweis: Zeile 7 und 10 sind korrekt.

Lösung zu Aufgabe 1

The hint in the exercise was referring to the random number functionality but written in that way was indeed confusing. On line 10 there was the declaration of the variable `N` which should have been done outside the `while` loop. A bonus point has been given to everybody.

1. The header `<stdio.h>` is missing.
2. Add `<stdio.h>` before line 1.
3. The variable `N` is used undeclared on line 18.
4. Declare the variable `N` outside the `while` loop.
5. The body of the `while` loop is made by one line only.
6. Add curly braces to delimit lines 10 to 17.
7. If the index of the `for` loop starts at 1 the `if` clause will always be entered.
8. The `for` loop index should start at 2.
9. The equal sign in the `if` clause is wrong.
10. Change `=` to `==` on line 13.

Aufgabe 2 (8 Pkt.)

Schreibe ein `C` Programm, das die im gleichen Verzeichnis liegende Datei `measurements.dat` öffnet und daraus double-Zahlen im Dezimalformat einliest, bis das Dateiende erreicht ist. Gleichzeitig soll das Programm eine Datei `measurements_exp.dat` im gleichen Verzeichnis anlegen und die eingelesenen Zahlen in gleicher Reihenfolge in Exponentialdarstellung mit Vorzeichen und insgesamt 4 Ziffern in der Mantisse in diese Datei ausgeben. Treten Fehler beim Einlesen oder Ausgeben aus, soll das Programm entsprechende Fehlermeldungen ausgeben.

Beispiel:

```
measurements.dat          measurements_exp.dat
    4.0                     +4.000e+00
   30.99                    +3.099e+01
        -4.90               -4.900e+00
```

Lösung zu Aufgabe 2

A possible solution might read as it follows.

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

int main(){
    double number;
    int read;
    FILE *file1, *file2;

    if((file1 = fopen("measurements.dat", "r")) == NULL){
        printf("Error opening file to read.\n");
        exit(0);
    }

    if((file2 = fopen("measurements_exp.dat", "w")) == NULL){
        printf("Error opening file to write.\n");
        exit(0);
    }

    while(1) {
        read = fscanf(file1, "%lf", &number);
        if(read == EOF)
            break;
        if(read != 1){
            printf("Error in fscanf.\n");
            exit(0);
        }
        fprintf(file2, "%+.3e\n", number);
    }

    fclose(file1);
    fclose(file2);
}
```

Aufgabe 3 (10 Pkt.)

(i) Beschreibe den in der Vorlesung besprochenen Mergesort-Algorithmus in Worten.

(ii) Implementiere den in der Vorlesung besprochenen Mergesort-Algorithmus in einer `C` Funktion

```
void mergesort(int num, double v[])
```

`v` soll dabei ein `double`-Array mit `num` Elementen sein, die die Funktion in aufsteigender Reihenfolge sortiert.

(iii) Nimm an, dass Dein Computer 64 Mikrosekunden benötigt um ein 16-elementiges Array `v` zu sortieren. Schätze mathematisch ab, wie lange er für ein 32-elementiges Array benötigt, und erläutere in diesem Zusammenhang den Begriff des Laufzeitverhaltens.

Lösung zu Aufgabe 3

(i) Conceptually, a merge sort works dividing the unsorted list into n sublists, each containing one element, which is considered trivially sorted. Sublists are repeatedly merged to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list.

(ii) The code discussed in the lecture follows.

```
void mergesort(int num, double v[])
{
    if(num < 0){
        printf("Fehler bei mergesort\n"); exit(0);
    }
    if(num == 0 || num == 1)
        return; // Liste ist bereits sortiert

    //Linke Teilliste sortieren.
    double *vl = v;  int numl = num/2;
    mergesort(numl, vl);
    //Rechte Teilliste sortieren.
    double *vr = v+numl;  int numr = num-numl;
    mergesort(numr, vr);
    //Sortierte Teilliste zu sortierter Liste verschmelzen
    double *tmp;
    if((tmp = (double *)malloc(num*sizeof(double))) == NULL) {
        printf("Fehler bei mergesort\n");  exit(0);
    }
    int i, il=0, ir=0;
    for(i=0; i<num; i++){
        if(ir == numr){
            tmp[i] = vl[il];  il++;
        } else if (il == numl){
            tmp[i] = vr[ir];  ir++;
        } else {
            if(vl[il] < vr[ir]){ tmp[i] = vl[il];  il++; }
            else                 { tmp[i] = vr[ir];  ir++; }
        }
    }
    for(i = 0; i < num; i++)
        v[i] = tmp[i];

    free(tmp);
}
```

(iii) The computational cost of the `mergesort` algorithm is $\mathcal{O}(N \log_2(N))$, where N is the size of the array to be sorted. Assuming to have two arrays of sizes N_1 and N_2 to sort, the ratio of the sorting times can be approximatly estimated to be

$$\frac{t_1}{t_2} = \frac{N_1 \log_2(N_1)}{N_2 \log(N_2)} \quad .$$

If $N_2 = 16$, $t_2 = 64\mu s$ and $N_1 = 32$, it follows $t_1 = \frac{5}{2}t_2$, namely $t_1 = 160\mu s$.

Aufgabe 4 (5 Pkt.)

Schreibe eine `C` Funktion, die als Parameter eine Integer-Zahl erhält. Falls diese nicht negativ ist, soll das Produkt der Ziffern der Zahl berechnet und das Ergebnis zurückgegeben werden. Ansonsten soll eine Fehlermeldung ausgegeben und das Programm beendet werden.

Lösung zu Aufgabe 4

The solution could look like the following.

```
unsigned int ProductOfDigits(int inputNumber)
{
     if (inputNumber<0)
          exit(1);
     else if (inputNumber==0)
          return 0;
     else
     {
          unsigned int product=1;
          while(inputNumber!=0)
          {
               product *= inputNumber%10;
               inputNumber/=10;
          }
          return product;
     }
}
```

Aufgabe 5 (5 Pkt.)

Gegeben ist das folgende `C` Programm:

```
1  #include <stdio.h>
2
3  unsigned int MysteryCounter(unsigned int inputNumber){
4      unsigned int counter = 0;
5      while(inputNumber!=1){
6          if(inputNumber%2==0){
7              inputNumber/=2;
8          } else {
9              inputNumber=3*inputNumber+1;
10         }
11         counter++;
12     }
13     return counter;
14 }
15
```

```c
int main(void){
    int trial;
    printf("Enter an integer number: ");
    scanf("%d", &trial);
    if(trial<=0)
        return 0;
    else
        printf("Length cycle: %u\n", MysteryCounter(trial));
}
```

(i) Beschreibe in Worten, was in der Funktion `MysteryCounter` passiert.

(ii) Welche Ausgabe erhält man, wenn man das Programm ausführt und

(a) 0
(b) 1
(c) 2
(d) 3

eingibt?

(iii) Wird die Variable `trial` von der Funktion `MysteryCounter` verändert? Begründe Deine Antwort.

Lösung zu Aufgabe 5

(i) The `MysteryCounter` function takes a natural number and if it is even it halfs it, otherwise it multiply it by 3 and adds 1 to it. This operation is repeated until[1] the result is equal to 1. The function return the number of times such a operation has been done.

(ii) When the input to the program is 0, the `return` statement in the `main` function is executed and no output is printed. In the other cases, the output reads:

- Input 1 → `Length cycle: 0`
- Input 2 → `Length cycle: 1`
- Input 3 → `Length cycle: 7`

(iii) Although the `MysteryCounter` function changes the `inputNumber` parameter, the variable `trial` is copied to the function and therefore, it will remain unchanged in the `main`.

Aufgabe 6 (7 Pkt.)

Gegeben ist die folgende `C` Struktur, die reelle 2×2 Matrizen repräsentiert:

```c
struct matrix2x2
{
    double A[2][2];
};

typedef struct matrix2x2 Matrix2x2;
```

(i) Schreibe eine Funktion, die als Parameter eine 2×2 Matrix in Form einer Variable vom Typ `Matrix2x2` erhält und die zugehörige Adjunkte berechnet und ebenfalls als `Matrix2x2` zurückgibt. **Hinweis:** Die Adjunkte einer 2×2 Matrix

$$A = \begin{pmatrix} +a & +b \\ +c & +d \end{pmatrix}$$

ist

$$\mathrm{adj}(A) = \begin{pmatrix} +d & -b \\ -c & +a \end{pmatrix} \quad .$$

[1] It is a mathematical open conjecture that this will eventually happen for any natural number.

(ii) Schreibe eine Funktion, die als Parameter eine 2×2 Matrix in Form einer Variable vom Typ `Matrix2x2` erhält und die Determinante berechnet und zurückgibt.

(iii) Schreibe eine Funktion, die als Parameter eine 2×2 Matrix in Form einer Variable vom Typ `Matrix2x2` erhält und die zugehörige Inverse berechnet und ebenfalls als `Matrix2x2` zurückgibt. Falls A nicht invertierbar ist, soll eine geeignete Fehlermeldung auf stderr ausgegeben und das Programm beendet werden.
Hinweis: Die Inverse kann leicht gemäß

$$A^{-1} = \frac{\text{adj}(A)}{\det(A)}$$

berechnet werden. Es ist damit zweckmäßig, Deine in Teilaufgaben (i) und (ii) implementierten Funktionen zu verwenden.

Lösung zu Aufgabe 6

(i) A possible implementation might be:

```
Matrix2x2 adjoint(Matrix2x2 M)
{
  Matrix2x2 M_adj = { M.A[1][1], -M.A[0][1],
                      -M.A[1][0],  M.A[0][0]};
  return M_adj;
}
```

(ii) A possible implementation might be:

```
double determinant(Matrix2x2 M)
{
    return M.A[0][0]*M.A[1][1] - M.A[1][0]*M.A[0][1];
}
```

(iii) A possible implementation might be:

```
Matrix2x2 inverse(Matrix2x2 M)
{
    double det = determinant(M);
    if(fabs(det) < 1.e-12){
        fprintf(stderr, "Matrix not invertible\n");
        exit(1);
    }
    Matrix2x2 M_inv=adjoint(M);
    M_inv.A[0][0]/=det;
    M_inv.A[0][1]/=det;
    M_inv.A[1][0]/=det;
    M_inv.A[1][1]/=det;
    return M_inv;
}
```