1. Klausur

zur Vorlesung

„Einführung in C++“
(Kapitel 1 bis 4 [elementares C++])

1. **Iterationen [4 Punkte]**
   Formen Sie die folgenden `for`-Anweisungen in semantisch äquivalente C++ -Anweisungsfolgen unter Verwendung von `while`- und `do`- Anweisungen (ohne `for`) um!

```
for (int i=0; i<MAX; i++) s+=i*i;

for (p=liste.first; p; p=p->next) if (p->key==n) break;
```

2. **Horner-Schema [4 Punkte]**
   Implementieren Sie eine C++ -Funktion

```
double horner (double x, int c[], int n); // n+1 Koeff.
```

   die den Wert des Polynoms $p(x) = \sum_{i=0}^{n} c_i x^i$ an der Stelle `x` nach dem Horner-Schema [$p(x) = ...((c_n*x)+c_{n-1})*x .... +c_0$] berechnet

3. **Palindrom-Test [6 Punkte]**
   Schreiben Sie ein C++ -Programm, welches (aufgerufen mit einem Parameter) entscheidet, ob die als Parameter übergebene Zeichenkette ein Palindrom ist oder nicht:

```
% p otto
ja
% p willi
nein
% p einnegermitgazellezagtimregennie
ja
```

4. **Speicher kopieren [6 Punkte]**
   Implementieren Sie in C++ eine Funktion

```
void bytecopy (void* destination, void* source, int bytes);
```

   die `bytes` Bytes im Speicher von der Adresse `source` an die Adresse `destination` kopiert. **Hinweis**: Die Speicherbereiche können auch überlappend sein !

5. **Türme von Hanoi [8 Punkte]**
   Schreiben Sie ein C++ Programm, welches die optimale Zugfolge des Problems „Türme von Hanoi“ (n Scheiben unterschiedlichen Durchmessers sind von A nach C unter Verwendung von B zu bewegen, es darf jeweils nur eine Scheibe bewegt werden, nie darf eine grö-

ßere Scheibe auf einer kleineren liegen) berechnet und ausgibt, `n` wird dem Programm beim Aufruf als Parameter übergeben:

```
% hanoi 1
A-->C
% hanoi 2
A-->B
A-->C
B-->C
```

Hinweise: 1. Versuchen Sie, das Problem rekursiv zu lösen. 2. Die Funktion `atoi` [ascii to int] (aus `stdlib.h`) kann die Stringrepräsentation einer Zahl in ihren numerischen Wert umwandeln: `int atoi (const char*);` z.B. atoi("1234") ----> 1234

6. **Fünfte Wurzel [8 Punkte]**
   Implementieren Sie in C++ eine Funktion

   ```
   double root5 (double x); // x > 0.0
   ```

   die (näherungsweise, z.B auf 4 Stellen genau) die fünfte Wurzel aus `x` berechnet.
   Hinweise: 1. Intervallschachtelung mit abnehmender Intervall-Länge. 2. Man unterscheide x > 1.0 und x < 1.0 .

7. **Mengentyp [10 Punkte]**
   Implementieren Sie auf der Basis der Built-In C++ -Typen und der elementaren Typstrukturierungsmittel (keine Klassen!) einen Datentyp zur Darstellung von Mengen mit maximal 256 Elementen (`int`'s im Bereich 0...255) mit den folgenden Operationen

   ```
   void init (Set& s); // s = ∅
   void incl (Set& s, int e); // s = s ∪ {e}
   void excl (Set& s, int e); // s = s ∩ ¬{e}
   int in (const Set& s, int e); // e ∈ s  ?
   void print (const Set &s); // gibt s in lesbarer Form aus
       // z.B. [] - leere Menge [1, 2, 3] - Elemente 1,2,3
   Set intersect (const Set& s1, const Set& s2);
       // returns s1 ∩ s2
   Set unite (const Set& s1, const Set& s2)
       // returns s1 ∪ s2
   ```

   Hinweise: 1. Man bemühe sich um eine möglichst effiziente Implementation (zur Darstellung, ob ein Element in der Menge enthalten ist, reicht ein einzelnes Bit aus!) 2. Felder sind als Rückgabetypen von Funktionen nicht erlaubt!

8. **Quicksort [10 Punkte]**
   Die C-Standardbibliothek enthält eine Funktion `qsort`, die den „Quicksort"-Algorithmus auf beliebigen Feldern realisiert. Man schreibe ein C++ -Programm welches zunächst (max. 100) Datensätze des Typs `struct Record { int key; char value[20]; };` interaktiv in ein Feld `Record data [100];` einliest, und dieses dann unter Verwendung von `qsort` nach dem Schlüssel `key` sortiert. **Hinweise**: 1. Man definiere einen geeigneten C++ -Prototyp für `qsort`. 2. der folgende Auszug aus der *manual page* zu `qsort` beschreibt die (Kernighan/Ritchie) C-Schnittstelle und ein Beispiel.

NAME

qsort - quicker sort

SYNOPSIS

```
qsort(base, nel, width, compar)
char *base;
int (*compar)();
```

DESCRIPTION

qsort() is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

base points to the element at the base of the table. nel is the number of elements in the table. width is the size, in bytes, of each element in the table. compar is the name of the comparison function, which is called with two arguments that point to the elements being compared. As the function must return an integer less than, equal to, or greater than zero, so must the first argument to be considered be less than, equal to, or greater than the second.

NOTES

The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

EXAMPLE

The following program sorts a simple array:

```
static    int intcompare(i,j)
int *i, *j;
{
     return(*i - *j);
}

main()
{
    int a[10];
    int i;
    a[0] = 9;
    a[1] = 8;
    a[2] = 7;
    a[3] = 6;
    a[4] = 5;
    a[5] = 4;
    a[6] = 3;
    a[7] = 2;
    a[8] = 1;
    a[9] = 0;

    qsort(a,10,sizeof(int),intcompare)
    for (i=0; i<10; i++) printf(" %d",a[i]);
    printf("\n");
}
```