

Funktionen

Zum Strukturieren komplexerer Programme werden funktionale Einheiten ausgelagert.

- **Syntax**

```
returnType FunctionName (ParameterList)
{
    C++ Anweisungen
}
```



falls Rückgabewert
oder Parameter leer

- Funktionen stehen meistens **hinter** dem „Hauptprogramm“ `int main() { }`
Dabei ist die Reihenfolge nicht von Bedeutung.

- Sie werden **vor** dem Hauptprogramm deklariert

```
returnType FunctionName (ParameterList);
```

Die Parameterliste muss nur die Typ Deklaration enthalten

- Die Funktionsdeklarationen werden häufig in Files mit der Endung `.h` ausgelagert. Das Einfügen erfolgt mit

```
#include "myHeader.h"
```

Funktionen – Ein Beispiel

- Beispiel

```
int max ( int N , int M )  
{  
    int result ;  
    if ( N > M )  
        result = N ;  
    else  
        result = M ;  
    return result ;  
}
```

Rückgabe des Maximums 2er Zahlen

Parameter typ deklarieren

Funktionstyp deklarieren

Lokale Variable, nur in der Funktion bekannt

Rückgabewert

```
int max ( int , int );
```

Funktionsdeklaration

- steht vor dem main Programm

- Parameter Liste enthält nur die Typ Deklaration

Funktionen – Ein Beispiel

• Beispiel

```
int max ( int N , int M )  
{  
    int result ;  
    if ( N > M )  
        result = N ;  
    else  
        result = M ;  
    return result ;  
}
```

Rückgabe des Maximums 2er Zahlen

Parametertyp deklarieren

Funktionstyp deklarieren

Lokale Variable, nur in der Funktion bekannt

Rückgabewert

```
int max ( int , int );  
steht vor dem main Programm
```

functionSum.cc

Arbeitsvorschlag:

- Schreiben Sie ein Programm, das in einer Funktion 2 Zahlen summiert.
- Erzeugen Sie eine header Datei mit der Funktionsdeklaration und binden Sie sie ein.
- Lagern Sie die header Datei in ein include dir aus. Mit der Compiler Option `-I dir` koennen Sie das directory einbinden

functionSum_1.h

functionSum_1.cc

Referenzen

Eine Referenz Variable, z.B. `ref`, ist als eine Bindung (alias) an eine bereits existierende Variable `v` definiert. Wird die Referenz Variable im Programmablauf geändert, so ändert sich auch die Variable.

- während des Programmablaufs wird keine Kopie der Variablen angelegt.
- bei Zuweisung an eine Referenz wird nicht die Referenz geändert, sondern die Variable!

• Syntax

```
datatype Name ;  
datatype & ReferenceName = Name ;
```

• Beispiel

.....

```
int v ;  
int& ref = v ;
```



Parametertyp deklarieren
Referenz erzeugen

....

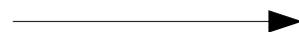
```
v = 99 ;
```



Variable v hat nun 99 zugewiesen bekommen, die Referenz ist auch 99

....

```
ref = 7 ;
```



Referenz wird auf 7 geändert, v ist auch 7

Funktionen - Parameterübergabe

Beim Übergeben der Funktionsargumente ist zu beachten, dass die Speicherbereiche beim Aufruf der Funktion erzeugt werden und beim Verlassen der Funktion nicht mehr existieren!

- Methoden zur Übergabe von Funktionsparametern

- Call by value (default)

```
double function (int A, int B) { }
```

A und B werden beim Funktionsaufruf kopiert und Änderungen innerhalb der Funktion haben **keinen Effekt auf die Werte A und B** im Hauptprogramm

- Call by reference

```
double function (int &A, int &B) { }
```

Die Referenzen von A und B werden beim Funktionsaufruf übergeben und Änderungen der Adressinhalte innerhalb der Funktion **beeinflussen die Werte A und B** im Hauptprogramm (siehe Referenzen)

```
void swap( int &N , int &K ) {  
    int temp;  temp = N;    N = K;    K = temp;  
    return ;  
}
```

[functionSwap_I.cc](#)

[functionSwap_II.cc](#)

Alternativ können auch pointer übergeben werden

Funktionen - Parameterübergabe

Arrays werden häufig an die Funktionen als Zeiger übergeben. Dabei wird die Arraylänge als konstanter Wert ebenfalls übergeben.

```
// test function feature of pointer to array
#include <iostream>
using namespace std;

void sum (const int i , double *a , double *b, double *c) ;

int main()
{
    const int i = 5 ;
    double ValA[i]={2.,3.,4.,5.,8.};
    double ValB[i]={20.,30.,40.,50.,80.};
    double ValC[i];

    sum (i,ValA,ValB,ValC);
    cout << "Sum = " << ValC[2] << endl;

    return 0;
}

// Add 2 double
void sum (const int i , double *a , double *b, double *c) {
    for (int k=0; k < i ; k++)  c[k] = a[k] + b[k] ;
}
```

addArray.cc

Der Inhalt von Array ValC wird in der Funktion `sum` modifiziert. Die Werte werden by reference (pointer) zurückgegeben, daher sind die Änderungen sichtbar.

Funktionen - Parameterübergabe

Arrays werden häufig an die Funktionen als Zeiger übergeben. Dabei wird die Arraylänge als konstanter Wert ebenfalls übergeben.

```
// test function feature of pointer to array
#include <iostream>
using namespace std;

void sum (const int i , double *a , double *b, double *c) ;

int main()
{
    const int i = 5 ;
    double ValA[i]={2.,3.,4.,5.,8.};
    double ValB[i]={20.,30.,40.,50.,80.};
    double ValC[i];

    sum (i,ValA,ValB,ValC);
    cout << "Sum = " << ValC[2] << endl;

    return 0;
}

// Add 2 double
void sum (const int i , double *a
    for (int k=0; k < i ; k++) c[k]
```

addArray.cc

Der Inhalt von Array ValC wird in der Funktion sum modifiziert. Die Werte werden by reference (pointer) zurückgegeben, daher sind die Änderungen sichtbar.

dotProduct.cc

Arbeitsvorschlag:

- Schreiben Sie ausgehend von addArray.cc ein Programm, das das Skalarprodukt von 2 Vektoren in einer Funktion berechnet.

Geben Sie das Ergebnis der Funktion als double an das main() zurück.

Funktionen - Parameterübergabe

Komplexe Datenstrukturen werden aus Funktionen in der Regel als Zeiger zurückgegeben. Im folgenden ist ein Beispiel mit strings implementiert

```
#include <iostream>
#include <cstring>
#include <stdio.h>
#define MAXCHAR 255
```

charpointer.cc

```
using namespace std;
```

```
char *readinput (char *ptr) ;
```

```
int main() {
```

```
    char inA [] = "Vorname" ;
```

```
    char inB [] = "Nachname" ;
```

```
    char *ptr;
```

```
    ptr = readinput(inA);
```

```
    cout << "Hi " << ptr << endl;
```

```
    ptr = readinput(inB);
```

```
    cout << ptr << " ? Ich bleib beim Vornamen! " << endl;
```

```
    return 0 ;
```

```
}
```

```
char *readinput (char *str) {
```

```
    char input[MAXCHAR];
```

```
    cout << "Bitte " << str << " eingeben: " << endl;
```

```
    fgets(input, MAXCHAR, stdin);
```

```
    return strtok(input, "\n");
```

```
}
```

Die Funktion `readinput` gibt eine Zeichenkette zurück bzw die Adresse des Anfangs der Zeichenkette

liest von `stdin` bis newline gefunden

Rückgabewert ist ein char pointer
newline ist entfernt

Funktionen – statische Variablen

Die lokalen Variablen von Funktionen existieren nur für die Lebensdauer der Funktion. **Zugewiesene Werte existieren nach dem Aufruf nicht mehr.**

- **Statische Variable**

```
static Datentyp Name ;
```

Diese Anweisung definiert eine Variable, deren Inhalt nach dem Schließen der Funktion erhalten bleibt und bei erneutem Aufruf zur Verfügung steht.

- **Globale Variable**

```
Datentyp myGlobalVariable ;
```

```
....
```

```
void function() { }
```

```
main() {
```

```
....
```

```
}
```

Variablen, die außerhalb von Funktionen und **vor dem Hauptprogramm definiert** werden, stehen in allen Funktionen zur Verfügung und behalten ebenfalls den Inhalt nach dem Schließen der Funktionen.

Überladen von Funktionen

Wenn **verschiedene Funktionen unter dem gleichen Namen** angesprochen werden können, sprechen wir von **überladenen Funktionen**. Dabei ist nur eine Änderung der Anzahl der Parameter in der Parameterliste oder der Parametertypen zulässig.

- **Beispiel**

```
double multiply (int a , int b) {  
return (double) (a*b) ; }
```

```
double multiply (double a , double b) {  
return a*b ; }
```

```
double multiply (int a , int b , int c) {  
return (double) (a*b*c) ; }
```

```
double multiply (double a , int b , int c) {  
return a * (double) (b*c) ; }
```

Rekursive Funktionen

Der Selbstaufwurf von Funktionen wird als Rekursion bezeichnet. Eine Abbruchbedingung sorgt dabei für eine endliche Anzahl von Aufrufen.

```
unsigned int fakultaet(unsigned int zahl) {  
    // Die Fakultät von 0 und 1 ist als 1 definiert.  
    if (zahl <= 1) return 1;  
    return fakultaet (zahl - 1) * zahl;  
}
```

- Bei einer rekursiven Lösung wird ein Problem immer wieder schrittweise in ein einfacheres Problem verwandelt.
- Rekursive Programme können fast immer auch iterativ implementiert werden. Meist führt das jedoch zu komplizierterem Code. Rekursive Lösungen sind in der Regel rechenzeit- und speicherintensiver.

Inline Funktionen

Der Aufruf von Funktionen wird durch das Schlüsselwort `inline` verändert. Der Programmcode wird durch den Compiler direkt in das Programm eingefügt.

- **Syntax**

```
inline returnType myFunction( ParameterList ) {  
    C++ code ;  
}
```

Diese Anweisung ist für den Compiler nicht bindend. Typischerweise werden nur kurze Funktionen direkt eingefügt.

Funktionen – Eingabe Parameter

Der Funktion `main()` können auch Parameter übergeben werden. Die vollständige Definition ist `int main(int argc , char *argv[])`

Dabei gibt `argc` die Anzahl der Argumente an; im vector `argv[]` stehen Pointer auf die übergebenen Parameter. In `argv[0]` steht dabei immer der Programmname. Hilfsfunktionen zum Entpacken der Parameter stehen über die Standard Bibliotheken zur Verfügung.

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

[readOptions.cc](#)

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

```
{
```

```
    for(int i=0; i < argc; i++)
```

```
    {
```

```
        printf("argv[%d] = %s ", i, argv[i]);
```

```
        printf("\n");
```

```
    }
```

```
    return 0;
```

```
}
```

Umwandeln von `argv[]` in `int / double`:

```
int i = atoi(argv[1]);
```

```
double a = atof(argv[1]);
```

Komplexere Beispiele finden Sie hier

[testOptions.cc](#)

[adderOptions.cc](#)

Funktionen - Bibliotheken

Funktionen können in Bibliotheken als Objektfiles sowohl statisch (.a) als auch dynamisch (.so) gespeichert werden.

- Erzeugen einer statischen Bibliothek

Der Quellcode ist die function myFunc.cc und die header Datei myFunc.h

- compilieren der Quelldateien und erzeugen des Objektfiles

```
gcc -c myFunc.cc -o myFunc.o
```

- verwenden der archiver ar zur Erzeugung der statischen Bibliothek

```
ar rcs libmyFunc.a myFunc.o
```

Der Bibliotheksname muss die ersten 3 Buchstaben lib enthalten und mit .a enden.

Im link Schritt von gcc muss -static hinzugefügt werden wenn nur .a libs verwendet werden sollen.

- Erzeugen einer dynamisch gelinkten Bibliothek

- compilieren der Quelldateien und erzeugen des Objektfiles

```
gcc -fPIC -c myFunc.cc -o myFunc.o
```

Die Option -fPIC erzeugt positionsunabhängigen code

- verwenden von gcc zur Erzeugung der dynamischen Bibliothek

```
gcc -shared -Wl,-soname,libmyFunc.so -o libmyFunc.so myFunc.o
```

- beim link Schritt in gcc zum Einbinden der dynamischen Bibliothek reicht die Endung .so der Bibliothek

```
g++ myProg.cc -L. -lmyFunc
```

Funktionen – Werkzeug Bibliotheken

Es existiert eine C++ Bibliothek, die standardisierte Werkzeugprogramme enthält (C++ Standard Library). Die C Bibliothek ist darin enthalten. Sie ist nicht mit der Standard Template Library (STL) zu verwechseln, die von Hewlett Packard entwickelt wurde.

- **Bibliotheksname**

`/usr/lib64/gcc/x86_64-suse-linux/7/libstdc++.so`

- **Header Files**

<https://de.cppreference.com/w/cpp/header>

- **Standard Funktionen und Klassen aus den Bereichen**

Mathematische Funktionen, Input / Output, Zeichenketten, Zeit / Datum, dynamische Speichermanipulation, Numerische Grenzen, Fehlerbehandlung, Numerik (komplexe Zahlen, Zufallszahlen, Manipulation von Arrays), **Kompatibilität zu C Funktionen**, Algorithmen, Container, Iteratoren.

veraltete C header: z.B.

<code><float.h></code>	→	<code><cfloat></code>
<code><math.h></code>	→	<code><cmath></code>
<code><complex.h></code>	→	<code><complex></code>

Funktionen – Werkzeug Bibliotheken

- Verwendung Beispiele

```
#include <cmath>
...
double t, x;
x = sin(t);
x = log(t);
x = exp(t);
x = sqrt(t);
abs(int); erf ;
// kopiert Vorzeichen
x=copysig(double, double);
// Zahl not a number
bool isnan (double) ;
// Zahl negativ
bool signbit(double) ;
// Zahl unendlich
bool isinf (double) ;
```

Erzeugung von Zufallszahlen:

```
// dynamische Speicherverwaltung,
// Zufallszahlen, Sort algorithmen, ...
#include <cstdlib>
// C style Zeit und Datum Tools
#include <ctime>
...
// initialize with machine time
srand(time(0));
// random integer
int r = rand();
...

```

[poissonVerteilung.cc](#)

Die C++ Standard Library wird im g++ Link Schritt automatisch hinzugefügt.

Funktionen – Werkzeug Bibliotheken (2)

Die C++ Standard Library wird bei der Verwendung des g++ Compilers im Link Schritt automatisch hinzugefügt.

Neben der C++ Standard Library gibt es viele kommerzielle und public domain Bibliotheken mit spezieller Funktionalität. Vor dem Programmieren einer speziellen Softwarelösung sollten existierende Bibliotheken durchgesehen werden.

- Bibliotheken - Beispiele

boost	freie C++ Bibliothek (mathematische Lösungen,
OpenMP	Paralleles Computing
cuda	Programmieren auf Nvidia Graphikkarten

- Verwendung von Bibliotheken

- Hinzufügen des header files im Programmcode

```
#include <boost/random.hpp>
```

- Aufrufen der Funktion im Hauptprogramm

```
gen()
```

- Hinzufügen der Bibliothek im Link Schritt des Compilers

```
g++ myCode.cc -L/usr/lib64/boost/ -lboost_random  
(/usr/lib64/boost/libboost_random.so wird hinzugefügt)
```

Arbeitsvorschlag:

- Schreiben Sie ein Programm, das die 3. Wurzel aus einer Zahl mit der Newton Methode bestimmt.
[KubikWurzelNewton_0.cc](#)
- Wie sieht der Algorithmus aus?
- Der Wert aus dem die Wurzel gezogen werden soll und die Genauigkeit sind im Programm festgelegte Parameter.
[KubikWurzelNewton_I.cc](#)
- Verwenden Sie nun command line Eingaben um den Wert aus dem die Wurzel gezogen werden soll und die Genauigkeit zu übergeben.
[KubikWurzelNewton_II.cc](#)
- Der Wert und die Genauigkeit sollen in beliebiger Reihenfolge eingegeben werden können (siehe Beispiel [testOptions.cc](#))
[KubikWurzelNewton_III.cc](#)
- Benutzen Sie Funktionen und schreiben sie diese in die Bibliothek libmyFunc.a in MeinArbeitsDirectory/lib
[KubikWurzelNewton_III.cc](#)

Rezept zur Verwendung von
Bibliotheken am Beispiel von KubikWurzelNewton

[TestLibrary_KubikWurzelNewton.txt](#)

Funktions-Templates

C++ beinhaltet Möglichkeiten zur generischen Programmierung. Ein Funktions-Template verhält sich wie eine Funktion, die Argumente verschiedener Typ Definitionen bzw Rückgabe Wert Typen akzeptiert. Die Funktionen enthalten dazu Platzhalter Typ Definitionen.

```
// Beispiel einer Template Funktion
```

```
#include <iostream>
using namespace std;
```

Platzhalter Typ Definition



```
template <typename Type>
Type maximum ( Type a , Type b ) {
    return ( a > b ? a : b ) ;
}
```

Angabe des verwendeten Typs



```
int main() {
    int x = 5 , y = 7 ;
    int m = maximum <int> (x,y) ;
    cout << "maximum of " << x << " and " << y << " is " << m << endl;

    char a = 'a' , b = 'b' ;
    char r = maximum <char> (a,b) ;
    cout << "maximum of " << a << " and " << b << " is " << r << endl;

    return 0 ;
}
```

funcTemplate_0.cc

Funktionen – Werkzeug Bibliotheken

Standard C++ library ist eine Programmierbibliothek, die auch die C Bibliotheken enthält.

- **Beispiel Container – vector class**

vector ist eine container class, die dynamische arrays implementiert

```
#include <vector>
```

```
....
```

```
vector <int> myVector;
```

```
vector <int> p;
```

```
vector <double> yourVector(10, 5.0)
```

```
int iElem = 10 ;
```

```
myVector.push_back(iElem);
```

```
myVector.push_back(74);
```

```
myVector.pop_back();
```

```
int myVectorSize = myVector.size();
```

```
for(int i=0;i<yourVector.size();i++)
```

```
    cout << "Elem " <<i<< " is " << yourVector[i]<< endl;
```

```
....
```

Intialisieren von vector

Hinzufügen eines Elements
am Ende von myVector

Löschen des letzten
Elements von myVector

Grösse von myVector

Schleife über alle Elemente von yourVector

Beispiel Vector Class

readFile_0.cc

```
// read text File with 4 columns
```

```
#include <fstream>
```

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    char InputFileName[256] = "myDataFile.txt" ;
```

```
    // open a file in read mode.
```

```
    ifstream inFile ( InputFileName );
```

```
    double colA , colB , colC , colD ;
```

```
    vector<double> vColA , vColB , vColC , vColD ; // instanciate
```

```
    int nRow = 0 ;
```

```
    if (inFile.is_open()) {
```

```
        while ( inFile >> colA >> colB >> colC >> colD ) { // read one row
```

```
            vColA.push_back(colA);
```

```
            vColB.push_back(colB);
```

```
            vColC.push_back(colC);
```

```
            vColD.push_back(colD);
```

```
        }
```

```
        nRow = vColA.size() ; // number of elements in the vector
```

```
        inFile.close();
```

```
    }
```

```
    else {
```

```
        cerr << "Can not open file " << InputFileName << endl;
```

```
    }
```

```
    return 0;
```

```
}
```

Filename des Text Files

Öffnen des Files

Schleife über alle
Einträge des Files

Schliessen des Files

Beispiel Vector Class

Komplexe Datenstrukturen, wie z.B. vector, können an Funktionen übergeben werden. Dies sollte mit Hilfe von Referenzen erfolgen, wie im Beispiel gezeigt

```
#include <vector>
double mean ( vector <double> & );
...
vector <double> myVector;
double x = mean ( myVector );
...
double mean ( vector <double> &v ) {
    double s = 0. ;
    for(int i=0;i<v.size();i++) s+=v[i];
    return s / (double) v.size() ;
}
```

Daten werden als Referenz auf einen vector in der Funktion übernommen

Wenn der vector in der Funktion nicht modifiziert wird, sollte er in der Funktion als `const` deklariert/übergeben werden.

```
double mean ( vector <double> const &v ) {
...
}
```

v ist hier read only!
es wird keine Kopie von v erstellt!

Beispiel Vector Class

```
// read text File with 4 columns
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
    char InputFileName[256] = "myDataFile.txt" ;
    // open a file in read mode.
    ifstream inFile ( InputFileName );

    double colA , colB , colC , colD ;
    vector<double> vColA , vColB , vColC , vColD ; // instanciate
    int nRow = 0 ;
    if (inFile.is_open()) {
        while ( inFile >> colA >> colB >> colC >> colD ) { // read one row
            vColA.push_back(colA);
            vColB.push_back(colB);
            vColC.push_back(colC);
            vColD.push_back(colD);
        }
        nRow = vColA.size() ; // number of e
        inFile.close();
    }
    else {
        cerr << "Can not open file " << InputF
    }

    return 0;
}
```

readFile_0.cc

Filename des Text Files
Öffnen des Files

Schleife über alle
Einträge des Files

Arbeitsvorschlag:

readFile_1.cc

- Schreiben Sie ein Programm, das die Werte aus dem Textfile mit 4 Spalten liest und unter Verwendung von vector die Mittelwerte und Standardabweichungen in einer Funktion bestimmt und die Werte in ein File schreibt.

Strings

Statt Zeichenketten in ein `char array` zu schreiben lässt sich auch die Standard Library Klasse `string` verwenden. Eine explizite Angabe der Länge der Zeichenkette ist hier nicht notwendig.

- **Syntax und einige wichtige Funktionen**

Deklaration der `string` Variablen

Strings lassen sich durch addieren aneinander hängen

```
#include <string>
```

```
.....
```

```
string myStrA = "Warum" , myString ;
```

```
string myStrB = " auch das noch"
```

```
myString = myStrA + myStrB;
```

```
cout << myString << endl;    →  Warum auch das noch
```

```
.....
```

```
int nLength = myString.length(); → nLength = 19
```

```
pos = myStrA.find("um"); → pos = 3
```

```
myStrA.insert(5, " nicht"); → Warum nicht
```

```
myStrB.erase(5, 9); → auch
```

```
myStrA.c_str();
```

Länge eines strings
Finden eines sub strings
Einfügen / löschen eines sub strings
Gibt eine nicht-modifizierbare Standard C Zeichenarray Version der Zeichenfolge zurück

Viele Verbesserungen in C++11

Sortieren von Arrays

Es ist eine beliebige Folge von ganzen Zahlen gegeben, die durch Algorithmen aufsteigend sortiert werden soll. Die Algorithmen sollen dabei möglichst effizient vorgehen, d.h. die Zahl der durchzuführenden Operationen pro Element soll klein sein.

- **Sortieren durch Einfügen**

Gehe wie beim Sortieren von Spielkarten vor. Es gibt 3 Gruppen

- I. links stecken die sortierten Karten
- II. eine einzusortierende Karte
- III. rechts ein unsortierter Bereich mit Karten

Algorithmus:

Nimm die erste Karte und betrachte sie als sortiert (I.)

Nimm die nächste Karte (II.) und suche die Position für die Karte im linken sortierten Teil (I.).

Nimm die nächste Karte im rechten unsortierten Teil (III.)

Beispiel: 111 | 120 | 555 90 100
 0 1 2 3 4

Element 0 ist sortiert

Schleife über Element 1 bis 4

Sortieren von Arrays

Es ist eine beliebige Folge von ganzen Zahlen gegeben, die durch Algorithmen aufsteigend sortiert werden soll. Die Algorithmen sollen dabei möglichst effizient vorgehen, d.h. die Zahl der durchzuführenden Operationen pro Element soll klein sein.

- Sortieren durch Einfügen

Beispiel: 111 | 110 | 555 90 100
 0 1 2 3 4

Element 0 ist sortiert

Schleife über Element K=1 bis 4

K=1: 110 ist kleiner als 111, füge 110 vor 111 ein

110 111 555 90 100
 0 1 2 3 4

K=2: 555 ist grösser als 111 (letzte sortierte Element), 555 bleibt am Platz

110 111 555 90 100
 0 1 2 3 4

K=3: 90 wird an den Anfang geschoben

90 110 111 555 100
 0 1 2 3 4

K=5: 100 rückt an die 2. Stelle und alle anderen rücken 1 Position vor

90 100 110 111 555
 0 1 2 3 4

Sortieren von Arrays

Es ist eine beliebige Folgen von ganzen Zahlen gegeben, die durch Algorithmen aufsteigend sortiert werden soll. Die Algorithmen sollen dabei möglichst effizient vorgehen, d.h. die Zahl der durchzuführenden Operationen pro Element soll klein sein.

- Sortieren durch Einfügen

```
int Array[n];  
int K, J ,EinZuOrdnen;  
for (K=1, K < N, K++) {  
    EinZuOrdnen = Array[K];  
    J = K-1;  
    while ( J >= 0 && Array[J] > EinZuOrdnen) {  
        Array[J+1] = Array[J];  
        J--;  
    }  
    Array[J+1] = EinZuOrdnen;  
}
```

zu sortierendes Array der Grösse n

Schleife vom 2. Element bis Ende

Zeiger auf das letzte sortierte Element

Verschiebe die Elemente von Array[0..K-1] eins weiter nach vorn, wenn sie grösser als das einzuordnende Element sind. Setze J an die einzuordnende Stelle.

Setze einzusortierende Zahl ins Array

Sortieren von Arrays

Es ist eine beliebige Folgen von ganzen Zahlen gegeben, die durch Algorithmen aufsteigend sortiert werden soll. Die Algorithmen sollen dabei möglichst effizient vorgehen, d.h. die Zahl der durchzuführenden Operationen pro Element soll klein sein.

- Sortieren durch Einfügen

```
int Array[n];
int K, J, EinZuOrdnen;
for (K=1, K < N, K++) {
    EinZuOrdnen = Array[K];
    J = K-1;
    while ( J >= 0 && Array[J] > EinZuOrdnen) {
        Array[J+1] = Array[J];
        J--;
    }
    Array[J+1] = EinZuOrdnen;
}
```

zu sortierendes Array der Grösse n

Schleife vom 2. Element bis Ende

Zeiger auf das letzte sortierte Element

wuerfeln_1.cc

Arbeitsvorschlag:

- Schreiben Sie ein Programm:
Würfeln mit j Würfeln, der i Flächen hat und es soll k mal gewürfelt werden. i, j, k werden dem ausführbaren Programm gegeben.
Das Würfelergebnis soll einmal unsortiert und einmal sortiert ausgegeben werden.

Setze einzusortierende Zahl ins Arr

Sortieren von Arrays

- Quicksort Algorithmus

Sortiere beliebige Folge von ganzen Zahlen, $A[0, n-1]$

- I. Wähle ein Element aus der Folge, Pivotelement $A[p]$ so das A in einen linken $A[L]$ und einen rechten $A[R]$ Teil geteilt wird.
- II. Ordne $A[L]$ so das alle Elemente groesser als das Pivotelement sind,
 $A[L] \geq A[p]$
Ordne $A[R]$ so das alle Elemente kleiner als das Pivotelement sind,
 $A[R] \leq A[p]$
Das Ordnen erfolgt durch Vertauschen der linken und rechten Elemente
- III. Wiederhole I. und II. solange fuer die linken und rechten Felder bis jeweils nur 2 Elemente uebrig sind.

- Anschauliche Darstellung verschiedener Sortier-Algorithmen

<http://www.bluffton.edu/~nesterd/java/SortingDemo.html>