Kapitel C++ als PDF exportieren

C++

C+ + ist eine von der ISO genormte Programmiersprache. Sie wurde ab 1979 von Bjarne Stroustrup bei AT&T als Erweiterung der Programmiersprache C entwickelt. C+ + ermöglicht sowohl die effiziente und maschinennahe Programmierung als auch eine Programmierung auf hohem Abstraktionsniveau. Der Standard definiert auch eine Standardbibliothek, zu der verschiedene Implementierungen existieren.

Lernplattform SoloLearn

- C++ spielerisch lernen

Nachschlagewerke

- Einstieg in C++
- WikiBook zur C++ Programmierung
- Kurzanleitung für GNU C/C++ - Compiler
- C++ unter Linux

Behandelte Inhalte

- Funktionen
- Parameterübergabe
- Zeiger
- Arrays & Sortieralgorithmen

Beispiele vom Unterricht

- Beispiel zur Parameterübergabe bei Funktionen
- Beispiel zu Zeiger und Funktionen
- Beispiel zu Arrays, Schleifen und Funktionen
- Beispiel zu Arrays, Funktionen, Schleifen und Zufallszahlen
- Beispiel zu Arrays, Sortieralgorithmen und Funktionen

Funktionen in C++

Ein wichtiges Sprachelement von C+ + kam bisher noch überhaupt nicht vor: die Funktion. Die Möglichkeit, Funktionen zu bilden, ist ein herausragendes Merkmal einer Programmiersprache. Ganz allgemein versteht man unter einer Funktion einen in sich geschlossenen Programmteil, der eine bestimmte Aufgabe erfüllt. Der Vorteil einer Funktion ist die Wiederverwendbarkeit.

Betrachten wir folgendes Beispiel:

```
int add( int x, int y)
{
  int z = x+y;

  return z;
}
```

An diesem Codeausschnitt erkennen Sie alle Bestandteile einer Funktion:

- Typ des Rückgabewerts: int
- Funktionsname: add
- Argumentliste: (int x, int y)
- Funktionskörper: Anweisung innerhalb des Blocks der Funktion, begrenzt durch die geschweiften Klammern { und }
- Return-Anweisung: return z; (bei Rückgabetyp void nicht nötig)

Zu all diesen Teilen ist natürlich noch Einiges zu sagen. Vorher noch ein typografischer Hinweis: Es hat sich eingebürgert, in Büchern zwei Klammern hinter Bezeichner zu setzen, die sich auf eine Funktion beziehen, zum Beispiel in Sätzen wie: Mit add() erreichen Sie die Addition zweier Ganzzahlen. Das sagt noch nichts über Art und Umfang der Argumentliste aus, sondern soll Sie lediglich daran erinnern, dass es dabei um eine Funktion und nicht um eine Variable geht. Ich will mich auch in diesem Buch daran halten.

Rückgabewert

In C+ + muss jede Funktion einen Typ für den Wert angeben, den sie zurückliefert. Manchmal ist es aber auch gar nicht nötig oder sinnvoll, dass eine Funktion überhaupt einen Rückgabewert hat. In diesem Fall geben Sie als Typ void an.

Was macht man nun mit einem solchen Wert? Der Programmteil, der die Funktion aufruft, kann diese an allen Stellen einsetzen, wo er sonst eine Variable oder Konstante angeben würde (in obiger Form allerdings nur dort, wo lediglich der Wert benötigt wird), also etwa:

```
int main()
{
  int a = 5;
  int b = 12;
  int c = add(a,b);
```

```
  cout << ``a = `` << a << ``, c = `` << c << ``, a+c = `` << add(a,c) <<
endl;

  return 0;
}
```

Dieses Programm hat dann die Ausgabe: a = 5, c = 17, a+c = 22

Übrigens: Selbst wenn eine Funktion einen Wert zurückgibt, müssen Sie ihn nicht beachten. Sie dürfen auch schreiben:

```
int a = 5;
int b = 12;
add(a,b);
```

auch wenn das hier keinen Sinn machen würde. Bei Funktionen mit Rückgabetyp void ist das hingegen die übliche Form des Aufrufs. Allgemein kommt es aber häufiger vor, dass Rückgabewerte ignoriert werden. Beispielsweise geben viele Funktionen Statusinformationen darüber zurück, wie gut (oder schlecht) sie ihre Aufgabe erfüllen konnten. Viele Anwender solcher Funktionen interessieren sich nicht für den Status und übergehen ihn. Das kann manchmal aber auch gefährlich werden, wenn etwa aufgetretene Fehler aus diesem Grund zunächst unentdeckt bleiben.

Funktionsname

Wie alle anderen Bezeichner in C++ dürfen Sie auch Funktionsnamen nur aus Buchstaben, Ziffern sowie dem Unterstrich _ bilden. Außerdem ist es nicht erlaubt, Funktionsnamen zu verwenden, die Schlüsselwörten gleichen (etwa for).

Argumentliste

Eine Funktion kann immer nur auf den Daten arbeiten, die ihr lokal vorliegen. Außer global (das heißt außerhalb aller Funktionen) definierten Variablen sind das nur die Parameter, die das Hauptprogramm an die Funktion übergibt. Von diesen Parametern (auch Argumente genannt) können Sie keinen, einen oder mehrere angeben, die Sie dann durch Kommas trennen.

Wenn Sie eine Funktion ohne Argumente schreiben wollen, lassen Sie den Bereich zwischen den beiden runden Klammern einfach leer – denn die Klammern müssen Sie stets schreiben! – oder Sie setzen ein Argument vom Typ void ein.

Ansonsten geben Sie für jeden Parameter seinen Datentyp und einen Namen an, unter dem er in der Funktion bekannt sein soll. Dieser Name kann vollkommen anders sein, als der im Hauptprogramm beim Aufruf verwendete. Auch in obigem Beispiel heißen die Summanden in der Funktion x und y, im Hauptprogramm aber a und b.

Funktionskörper

Hier stehen die Anweisungen, die bei einem Aufruf der Funktion ausgeführt werden. Man kann darüber streiten, wie lang Funktionen sein sollten. Es gibt Experten, die fordern, dass eine Funktion aus nicht mehr als 50 Zeilen bestehen dürfe, sonst werde sie unleserlich. Es gibt jedoch in der Praxis immer wieder Fälle, in denen längere Funktionen sinnvoll sind. Bei der objektorientierten Programmierung werden Sie allerdings ohnehin wesentlich mehr Funktionen (beziehungsweise Methoden) verwenden, die im Durchschnitt wesentlich kürzer sind als bei der prozeduralen Programmierung.

Innerhalb des Funktionskörpers können Sie die Funktionsparameter wie normale Variablen verwenden; zusätzlich können Sie natürlich auch noch lokale Variablen definieren. Außerdem ist es selbstverständlich erlaubt, aus einer Funktion wieder andere Funktionen aufzurufen. (Sie dürfen sogar die Funktion selbst wieder aufrufen; man spricht dann von einer rekursiven Funktion – aber das ist ein eigenes Thema.)

Return-Anweisung

Die Anweisungen im Funktionskörper werden so lange abgearbeitet, bis das Programm auf das Ende der Funktion oder eine return-Anweisung trifft. Diese erfüllt einen doppelten Zweck:

Sie legen fest, welchen Wert die Funktion an das Hauptprogramm zurückliefern soll. Das kann eine Variable sein oder ein Ausdruck, eine Konstante oder der Rückgabewert einer anderen Funktion (wobei Letzteres als schlechter Stil gilt). Hat Ihre Funktion den Rückgabetyp void, geben Sie an dieser Stelle überhaupt nichts an. Der Typ des angegebenen Werts muss jedoch in jedem Fall mit dem deklarierten Rückgabetyp übereinstimmen. Sie beenden die Funktion und kehren zum Hauptprogramm zurück. Jede return-Anweisung – sei sie nun am Ende oder irgendwo inmitten des Funktionskörpers – markiert das Ende der Abarbeitung der Funktion und den Rücksprung an die Stelle, von der aus die Funktion aufgerufen wurde. Sie können also die Funktion schon beenden, bevor alle Anweisungen ausgeführt sind, zum Beispiel wenn eine bestimmte Bedingung erfüllt ist. Ist der Rückgabetyp void, muss am Ende der Funktion keine return-Anweisung stehen (auch nicht das Schlüsselwort return), wie in folgendem Beispiel:

```
void ausgabe( int z)
{
  cout << ``Das Ergebnis ist: `` << z << endl;
}
```

Wenn Sie allerdings bei Funktionen mit irgendeinem anderen Rückgabetyp die return-Anweisung vergessen, meldet der Compiler einen Fehler.

Der Prototyp

Bevor Sie eine Funktion verwenden können, müssen Sie dem Compiler zunächst mitteilen, dass es eine Funktion dieses Namens gibt, wie viele und welche Parameter sie hat und welchen Typ sie zurückliefert. Dies geschieht mit einem so genannten Prototyp der Funktion. Der Prototyp sieht

genauso aus wie die Funktion selbst bis auf den Funktionskörper; dieser fehlt und wird durch ein einfaches Semikolon ; ersetzt. (Es ist sogar erlaubt, die Namen der Argumente wegzulassen und nur ihre Typen anzugeben. Analog zu den Klassen (siehe Seite [*]) ist also der Prototyp die Deklaration und die Funktion mit Körper die Definition.

Aufruf

Der Aufruf einer Funktion erfolgt durch Nennung des Funktionsnamens. An den Funktionsnamen schließt sich immer ein Klammerpaar an, das gegebenenfalls auch Parameter enthalten kann. Dieses Klammerpaar ist zwingend erforderlich. Die Parameter des Aufrufs müssen zu den Parametern der Funktion passen. Besitzt die Funktion einen Rückgabewert, kann der Funktionsaufruf als Ausdruck verwendet werden. Er kann also beispielsweise auf der rechten Seite einer Zuweisung stehen.

Übergabe der Argumente

C+ + kennt mehrere Varianten, wie einer Funktion die Argumente übergeben werden können: call-by-value, call-by-reference und call-by-pointer.

call-by-value (Wertübergabe)

Bei call-by-value (Wertübergabe) wird der Wert des Arguments in einen Speicherbereich kopiert, auf den die Funktion mittels Parametername zugreifen kann. Ein Werteparameter verhält sich wie eine lokale Variable, die „automatisch“ mit dem richtigen Wert initialisiert wird. Der Kopiervorgang kann bei Klassen (Thema eines späteren Kapitels) einen erheblichen Zeit- und Speicheraufwand bedeuten!

Beispiel

```
#include <iostream>
#include <conio.h>

using namespace std;

//Prototyp, Deklaration
int sumW(int x,int y);   //Parameterübergabe per Wert (Value)

int main(int argc, char** argv) {
    cout << "Hello World" << endl;

    int a=5,b=6,erg=0;
    cout << "Parameteruebergabe per Wert" << endl;
    erg=sumW(a, b);
    cout << "a: " << a <<endl;
    cout << "b: " << b <<endl;

    getch();

    return 0;
}

//Funktionendefinition - Parameterübergabe per Wert
int sumW(int x,int y){
    x+=1; //x=x+1; x++; ++x;
    y+=1;

    cout << "x: " << x <<endl;
    cout << "y: " << y <<endl;

    return x+y;
```

```
}
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Variable	Adresse	Wert
a	0x0001	5
b	0x0005	6
...	.	.
...	.	.
x	0x00a1	5 → 6
y	0x00a5	6 → 7

Die Ausgabe des Programms ist:

```
Hello World
Parameteruebergabe per Wert
x: 6
y: 7
a: 5
b: 6
```

call-by-reference (Übergabe per Referenz)

Sollen die von einer Funktion vorgenommen Änderungen auch für das Hauptprogramm sichtbar sein, müssen in C sogenannte Zeiger verwendet werden. C+ + stellt ebenfalls Zeiger zur Verfügung. C+ + gibt Ihnen aber auch die Möglichkeit, diese Zeiger mittels Referenzen zu umgehen, was im alten C nicht möglich war. Beide sind jedoch noch Thema eines späteren Kapitels.

Im Gegensatz zu call-by-value wird bei call-by-reference die Speicheradresse des Arguments übergeben, also der Wert nicht kopiert. Änderungen der (Referenz-)Variable betreffen zwangsläufig auch die übergebene Variable selbst und bleiben nach dem Funktionsaufruf erhalten. Um call-by-reference anzuzeigen, wird der Operator & verwendet, wie Sie gleich im Beispiel sehen werden. Wird keine Änderung des Inhalts gewünscht, sollten Sie den Referenzparameter als const deklarieren, um so den Speicherbereich vor Änderungen zu schützen. Fehler, die sich aus der ungewollten Änderung des Inhaltes einer übergebenen Referenz ergeben, sind in der Regel schwer zu finden.

```
#include <iostream>
#include <conio.h>

using namespace std;

//Prototyp, Deklaration
int sumR(int &x,int &y);   //Parameterübergabe per Referenz (Reference)


int main(int argc, char** argv) {
    cout << "Hello World" << endl;
```

```
    cout << "Parameteruebergabe per Referenz" << endl;
    a=5,b=6,erg=0;
    erg=sumR(a, b);
    cout << "a: " << a <<endl;
    cout << "b: " << b <<endl;


    getch();

    return 0;
}

//Funktionendefinition - Parameterübergabe per Referenz
int sumR(int &x,int &y){
    x+=1; //x=x+1; x++; ++x;
    y+=1;

    cout << "x: " << x <<endl;
    cout << "y: " << y <<endl;

    return x+y;
}
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Variable	Adresse	Wert
x,a	0x0001	5 → 6
y,b	0x0005	6 → 7
...	.	
...	.	
...	.	
...	.	

Die Ausgabe des Programms ist:

```
Hello World
Parameteruebergabe per Referenz
x: 6
y: 7
a: 6
b: 7
```

call-by-pointer (Übergabe per Zeiger)

Wenn Sie einen Zeiger als Parameter an eine Funktion übergeben, können Sie den Wert an der übergebenen Adresse ändern.

```
#include <iostream>
#include <conio.h>

using namespace std;

//Prototyp, Deklaration
int sumZ(int *x,int *y);   //Parameterübergabe per Zeiger (Pointer)

int main(int argc, char** argv) {
    cout << "Hello World" << endl;
    cout << "Parameteruebergabe per Zeiger" << endl;
    a=5,b=6,erg=0;
    erg=sumZ(&a, &b);
    cout << "a: " << a <<endl;
    cout << "b: " << b <<endl;

    getch();

    return 0;
}


//Funktionendefinition - Parameterübergabe per Zeiger
int sumZ(int *x,int *y){
    *x+=1; //x=x+1; x++; ++x;
    *y+=1;

    cout << "x: " << *x <<endl;
    cout << "y: " << *y <<endl;

    return *x+*y;
}
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Variable	Adresse	Wert
a	0x0001	5 → 6
b	0x0005	6 → 7
...	.	
...	.	
*x	0x00a1	0x0001
*y	0x00a5	0x0005

Die Ausgabe des Programms ist:

```
Hello World
Parameteruebergabe per Zeiger
x: 6
y: 7
```

```
a: 6
b: 7
```

Funktionen - Beispiel

Angabe

Dieses Programm zeigt die verschiedenen Arten der Parameterübergabe bei Funktionen auf. Es werden jeweils dieselben Variablen auf 3 verschiedenen Arten (per Wert, per Referenz, per Zeiger) übergeben.

Lösung

```
/* Beispiel: Funktionen - Parameterübergabe
   Filename:Parameteruebergabe.cpp
   Author:Lahmer
   Title:Parameterübergabe
   Description: Anhand dieses Beispiels werden die verschiedenen Arten der
Parameterübergabe erklärt
   Last Change:16.01.2018
*/

//Hedader-Dateien bzw. Bibliotheken
#include <iostream>
#include <conio.h>

//Namespace (Namensraum)
using namespace std;

//Prototyp, Deklaration
int sumW(int x,int y);   //Parameterübergabe per Wert (Value)
int sumR(int &x,int &y);   //Parameterübergabe per Referenz (Reference)
int sumZ(int *x,int *y);   //Parameterübergabe per Zeiger (Pointer)


//Hauptprogramm
int main(int argc, char** argv) {
    //Ausgabe
    cout << "Hello World" << endl;

    //Lokale Variablen mit Datentyp int => ganzzahlige Zahlen
    int a=5,b=6,erg=0;

    //Parameterübergabe per Wert
    cout << "Parameteruebergabe per Wert" << endl;
    //Funktionsaufruf per Wert = Kopie der Variable
    erg=sumW(a, b);
    //Ausgabe der Variablen a bzw. b
    cout << "a: " << a <<endl;
```

```
    cout << "b: " << b <<endl;

    //Parameterübergabe per Referenz
    cout << "Parameteruebergabe per Referenz" << endl;
    a=5,b=6,erg=0;
    //Funktionsaufruf per Referenz = Verknüpfung der Variablen a bzw. b
    erg=sumR(a, b);
    //Ausgabe der Variablen a bzw. b
    cout << "a: " << a <<endl;
    cout << "b: " << b <<endl;

    //Parameterübergabe per Zeiger
    cout << "Parameteruebergabe per Zeiger" << endl;
    a=5,b=6,erg=0;
    //Funktionsaufruf per Referenz = Übergabe der Adressen der Variablen a
bzw. b
    erg=sumZ(&a, &b);
        //Ausgabe der Variablen a bzw. b
    cout << "a: " << a <<endl;
    cout << "b: " << b <<endl;

    //Auf Tastendruck warten
    getch();

    //Rückgabewert = 0
    return 0;
}

//Funktionendefinition - Parameterübergabe per Wert
int sumW(int x,int y){
    x+=1; //x=x+1; x++; ++x;
    y+=1;

    cout << "x: " << x <<endl;
    cout << "y: " << y <<endl;

    //Rückgabewert Summe von x+y
    return x+y;
}

//Funktionendefinition - Parameterübergabe per Referenz
int sumR(int &x,int &y){
    x+=1; //x=x+1; x++; ++x;
    y+=1;

    cout << "x: " << x <<endl;
    cout << "y: " << y <<endl;

    //Rückgabewert Summe von x+y
    return x+y;
}
```

```
//Funktionendefinition - Parameterübergabe per Zeiger
int sumZ(int *x,int *y){
    *x+=1; //x=x+1; x++; ++x;
    *y+=1;

    cout << "x: " << *x <<endl;
    cout << "y: " << *y <<endl;

    //Rückgabewert Summe von x+y
    return *x+*y;
}
```

Ausgabe

```
Hello World
Parameteruebergabe per Wert
x: 6
y: 7
a: 5
b: 6
Parameteruebergabe per Referenz
x: 6
y: 7
a: 6
b: 7
Parameteruebergabe per Zeiger
x: 6
y: 7
a: 6
b: 7
```

Zeiger (Pointer)

Zeiger (engl. pointers) sind Variablen, die als Wert die Speicheradresse einer anderen Variable enthalten.

Jede Variable wird in C++ an einer bestimmten Position im Hauptspeicher abgelegt. Diese Position nennt man Speicheradresse (engl. memory address). C++ bietet die Möglichkeit, die Adresse jeder Variable zu ermitteln. Solange eine Variable gültig ist, bleibt sie an ein und derselben Stelle im Speicher.

Am einfachsten vergegenwärtigt man sich dieses Konzept anhand der globalen Variablen. Diese werden außerhalb aller Funktionen und Klassen deklariert und sind überall gültig. Auf sie kann man von jeder Klasse und jeder Funktion aus zugreifen. Über globale Variablen ist bereits zur Kompilierzeit bekannt, wo sie sich innerhalb des Speichers befinden (also kennt das Programm ihre Adresse).

Zeiger sind nichts anderes als normale Variablen. Sie werden deklariert (und definiert), besitzen einen Gültigkeitsbereich, eine Adresse und einen Wert. Dieser Wert, der Inhalt der Zeigervariable, ist aber nicht wie in unseren bisherigen Beispielen eine Zahl, sondern die Adresse einer anderen Variable oder eines Speicherbereichs. Bei der Deklaration einer Zeigervariable wird der Typ der Variable festgelegt, auf den sie verweisen soll.

```
#include <iostream>

int main() {
     int   Wert;      // eine int-Variable
     int *pWert;      // eine Zeigervariable, zeigt auf einen int
     int *pZahl;      // ein weiterer "Zeiger auf int"

     Wert = 10;       // Zuweisung eines Wertes an eine int-Variable

     pWert = &Wert;   // Adressoperator '&' liefert die Adresse einer
Variable
     pZahl = pWert;   // pZahl und pWert zeigen jetzt auf dieselbe Variable
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Variable	Adresse	Wert
Wert	0x0001	10
*pWert	0x0005	0x0001
*pZahl	0x0009	0x0005
...	.	.
...	.	.

Der Adressoperator & kann auf jede Variable angewandt werden und liefert deren Adresse, die man einer (dem Variablentyp entsprechenden) Zeigervariablen zuweisen kann. Wie im Beispiel gezeigt, können Zeiger gleichen Typs einander zugewiesen werden. Zeiger verschiedenen Typs bedürfen einer Typumwandlung. Die Zeigervariablen pWert und pZahl sind an verschiedenen Stellen im Speicher abgelegt, nur die Inhalte sind gleich.

Wollen Sie auf den Wert zugreifen, der sich hinter der im Zeiger gespeicherten Adresse verbirgt, so verwenden Sie den Dereferenzierungsoperator *.

```
  *pWert += 5;
  *pZahl += 8;

   std::cout << "Wert = " << Wert << std::endl;
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Variable	Adresse	Wert
Wert	0x0001	10 –> 15 –> 23
*pWert	0x0005	0x0001
*pZahl	0x0009	0x0005
...	.	.
...	.	.

Ausgabe:

```
Wert = 23
```

Man nennt das den Zeiger dereferenzieren. Im Beispiel erhalten Sie die Ausgabe Wert = 23, denn pWert und pZahl verweisen ja beide auf die Variable Wert.

Um es noch einmal hervorzuheben: Zeiger auf Integer (int) sind selbst keine Integer. Den Versuch, einer Zeigervariablen eine Zahl zuzuweisen, beantwortet der Compiler mit einer Fehlermeldung oder mindestens einer Warnung. Hier gibt es nur eine Ausnahme: die Zahl 0 darf jedem beliebigen Zeiger zugewiesen werden. Ein solcher Nullzeiger zeigt nirgendwohin. Der Versuch, ihn zu dereferenzieren, führt zu einem Laufzeitfehler.

Zeiger- Beispiel

Angabe / Beschreibung

Dieses Programm demonstriert die Anwendung von Zeigern. Weiters werden auch wieder Variablen per Wert, Referenz und Zeiger an Funktionen übergeben.

Lösung

```
/* Beispiel: Zeiger
   Filename:main.cpp
   Author:Lahmer
   Title:Verdeutlichung von Zeiger
   Description: Anhand dieses Beispiels werden Zeiger näher erläutert
   Last Change:16.01.2018
*/


//Hedader-Dateien bzw. Bibliotheken
#include <iostream>
#include <conio.h>

//Namespace
using namespace std;


//Funktionsprototyp
int diff(int a, int b); //Parameterübergabe per Wert => Rückgabewert int, 2 int-Übergabeparameter
void erhoehezahl(int &a); //Parameterübergabe per Referenz => Rückgabewert void (nichts), Variable i wird als Referenz übergeben
void verringerezahl(int *b); //Parameterübergabe per Zeiger => Rückgabewert void (nichts), Variable b wird als Zeiger übergeben


//Hauptprogramm
int main(int argc, char** argv) {

    //Lokale Variablendeklaration
    int z1=99,z2=23;
    int erg=0;
    int *ptr;
    int *ptr2;    //ptr2,ptr soll auf z2 zeigen: ptr2, ptr --> z2;
    int *ptr3;    //ptrx soll auf z1 zeigen: ptrx --> z1
```

```
    //Ausgabe der Speicheradressen der angelegten Variablen
    cout << "Adresse der Variable z1: " << &z1 << endl;
    cout << "Adresse der Variable z2: " << &z2 << endl;
    cout << "Adresse der Variable erg: " << &erg << endl;
    cout << "Adresse der Variable ptr: " << &ptr << endl;
    cout << "Adresse der Variable ptr2: " << &ptr2 << endl;
    cout << "Adresse der Variable ptr3: " << &ptr3 << endl;
    cout << endl << endl;


    //Die Adresse von der Variable z2 wird in die Variable ptr gespeichert
    ptr=&z2;

    //Der Inhalt von ptr (also die Adresse von z2) wird in die Variable ptr2
gespeichert
    ptr2=ptr;

    //Die Adresse von z1 wird in die Variable ptr3 gespeichert
    ptr3=&z1;


    cout << "Pointer1: " << endl;
    cout << &ptr << endl;    //Ausgabe der Adresse der Variable ptr
    cout << ptr << endl;    //Ausgabe des Inhalts von ptr (also die Adresse
von z2)
    cout << *ptr << endl;    //Ausgabe des Inhalts von der in ptr
gespeicherten Adresse (also der Inhalt von z2)

    cout << "Pointer2: " << endl;
    cout << &ptr2 << endl;    //Ausgabe der Adresse der Variable ptr2 (also
die Adresse von ptr2)
    cout << ptr2 << endl;    //Ausgabe der Variable ptr2 (also die Adresse
von z2)
    cout << *ptr2 << endl << endl;    //Ausgabe des Inhalts an der
gespeicherten Adresse in ptr2 (also der Inhalt von z2)

    cout << "Pointer3: " << endl;
    cout << &ptr3 << endl;    //Ausgabe der Adresse der Variable ptr3
    cout << ptr3 << endl;    //Ausgabe der Variable ptr3 (also die Adresse
von z1)
    cout << *ptr3 << endl << endl;     //Ausgabe des Inhalts an der
gespeicherten Adresse in ptr3 (also der Inhalt von z1)


    cout << "########################" << endl;


    //Funktionsaufrufe
    //Die Variablen z1 und z2 werden per Wert übergeben (=Kopie im Speicher)
    erg=diff(z1,z2);
```

```
    cout << erg << endl;

    //Die Variable z1 wird per Referenz übergeben, d.h. eine neue zweite
Variable in der Funktion erhoehezahl() zeigt auf denselben Speicherplatz von
z1
    erhoehezahl(z1);
    cout << z1 << endl;

    //Die Variable z2 wird per Zeiger übergeben, d.h. die Adresse von z2
wird an die Funktion verringerezahl() übergeben, die Variable b in der
Funktion enthält dann die Adresse von z2
    verringerezahl(&z2);
    cout << z2 << endl;

    *ptr = *ptr*10;  //Der Inhalt an der gespeicherten Adresse in ptr (also
z2) wird mit 10 multipliziert und an den Inhalt der in ptr gespeicherten
Adresse (also z2) geschrieben
    *ptr3 = *ptr2;   //Der Inhalt an der gespeicherten Adresse in ptr2 (also
z2) wird über den Inhalt an der gespeicherten Adresse in ptr3 geschrieben

    cout << endl << endl;
    cout << "Inhalt der Variable z1: " << z1 << endl;
    cout << "Inhalt der Variable z2: " << z2 << endl;


    //Rückgabewert = 0
    return 0;
}

//Funktionsdefinition
int diff(int a, int b)
{
    //Rückgabe der Differenz a-b
    return a-b;
}


void erhoehezahl(int &a)
{
    //Die Variable a wird um 1 erhöht, Aufgrund der Übergabe der Referenz
wird auch gleichzeitig die Variable z1 im Hauptprogramm erhöht
    a++;
}

void verringerezahl(int *b)
{
    //Der Inhalt der in der Variable b gespeicherten Adresse wird um 1
verringert (d.h. z2 wird verringert)
    --*b;
    //Der Inhalt der in der Variable b gespeicherten Adresse wird um 1
verringert (d.h. z2 wird verringert)
```

```
    (*b)--;
    //Der Inhalt der in der Variable b gespeicherten Adresse wird um 1
verringert (d.h. z2 wird verringert)
    *b-=1;
}
```

Beispielhafter Auszug des Speichers

Variable	Adresse	Inhalt
z1	0x9ffe3c	99 → 100 → 200
z2	0x9ffe38	23 → 22 → 21 → 20 → 200
erg	0x9ffe34	0 → 76
ptr	0x9ffe28	0x9ffe38
ptr2	0x9ffe20	0x9ffe38
ptr3	0x9ffe18	0x9ffe3c

Ausgabe des Programms

```
Adresse der Variable z1: 0x9ffe3c
Adresse der Variable z2: 0x9ffe38
Adresse der Variable erg: 0x9ffe34
Adresse der Variable ptr: 0x9ffe28
Adresse der Variable ptr2: 0x9ffe20
Adresse der Variable ptr3: 0x9ffe18


Pointer1:
0x9ffe28
0x9ffe38
23
Pointer2:
0x9ffe20
0x9ffe38
23

Pointer3:
0x9ffe18
0x9ffe3c
99

######################
76
100
20


Inhalt der Variable z1: 200
Inhalt der Variable z2: 200
```

```
--------------------------------
Process exited after 0.07103 seconds with return value 0
Drücken Sie eine beliebige Taste . . .
```

Arrays

Eindimensionale Arrays

Arrays und Funktionen

Einfache Sortieralgorithmen

Sortieralgorithmen

Mehrdimensionale Arrays

Mehrdimensionale Arrays

Eindimensionale Arrays und Funktionen

Beispiel: Ermittlung von n Quicktipps im Lotto 6 aus 45 Wir beginnen mit einer Version ohne und Funktionen und bauen diese dann entsprechend um!

```
void main()
{
  int n;  //Anzahl der Tipps
  int tipp[6];
  srand(time(0));
  cout<<"Wie viele Tipps moechten Sie erhalten?";
  cin>>n;
  for(int i=1;i<=n;i++){     //n Tipps ermitteln
    cout<<"\n"<<i<<". Tipp: ";

    for(int j=0; j<6;j++){
        tipp[j]=rand()%45+1;
        cout<<tipp[j]<<" ";
    }
  }
  getch();
}
```

Bei der Verwendung von Arrays in Funktionen sind einige Besonderheiten zu beachten:

- Eine Funktion kann keinen Array-Typ besitzen.
- Werden Arrays als Parameter verwendet, so sind dies in C++ automatisch Ein-Ausgabe-Parameter. Dh. es wird im Unterprogramm keine Kopie der Array-Variablen angelegt, sondern das Unterprogramm greift direkt auf die Array-Variable der aufrufenden Funktion zu.

Wir erweitern nun das obige Beispiel um den Einsatz von Funktionen:

```
void erzeugeQuicktipp(int a[6]);

void main()
{
  int n;  //Anzahl der Tipps
  int tipp[6];
  srand(time(0));
  cout<<"Wie viele Tipps moechten Sie erhalten?";
  cin>>n;
  for(int i=1;i<=n;i++){     //n Tipps ermitteln
    cout<<"\n"<<i<<". Tipp: ";
    erzeugeQuicktipp(tipp);
  }
  getch();
}

void erzeugeQuicktipp(int a[6]){
  for (int i=0; i<6; i++) {
```

```
      a[i]=rand()%45+1;
      cout<<a[i]<<" ";
  }
}
```

Erweitere das Beispiel nun um eine eingene Funktion für die Ausgabe. Außerdem soll verhindert werden, dass innerhalb eines Tipps Zahlen doppelt vorkommen.

```
    // Programm: lotto.cpp
    // Beschreibung: Erzeugen von Lotto-Tips
    #include <iostream>      // Zusatzbibliothek für Ein-und Ausgaben wird
eingebunden
    #include <conio.h>       // Zusatzbibliothek für Konsole wird
eingebunden
    #include <stdlib.h>      // Wird für Zufallszahl-Generator benötigt
    #include <time.h>        // Wird für Initialisierung des Zufallszahl-
Generator benötigt
    using namespace std;     // Standard-Namensraum wird eingestellt

    void erzeugeQuicktipp(int a[6]);
    void ausgabe(int a[6]);

    int main(){
      int n;  //Anzahl der Tipps
      int tipp[6];
      srand(time(0));
      cout<<"Wie viele Tipps moechten Sie erhalten?"; cin>>n;
      for(int i=1;i<=n;i++){     //n Tipps ermitteln
        cout<<"\n"<<i<<". Tipp: ";
        erzeugeQuicktipp(tipp);
        ausgabe(tipp);
      }
      getch();
      return 0;
    }

    void erzeugeQuicktipp(int a[6]){
      for (int i=0; i<6; i++) {
        a[i]=rand()%45+1;
        //Überprüfe, ob Zahl bereits vorhanden
        for(int j=0;j<i;j++){   //Durchlaufe die vorhandenen Einträge
           if(a[i]==a[j]){  //und überprüfe sie auf Gleichheit mit dem
aktuellen Eintrag
            i--;        //ist die Zahl bereits vorhanden, so setze
                    //Zähler i zurück, damit diese Zufallszahl
                        //automatisch nochmals erzeugt wird!
           }
        }
      }
    }
```

```
void ausgabe(int a[6]){
  for (int i=0; i<6; i++) {
     cout<<a[i]<<" ";
  }
}
```

Aufgaben

- Aufgaben

Arrays, Schleifen und Funktionen - Beispiel

Angabe / Beschreibung

Der Benutzer soll eine beliebige ganzzahlige Zahl eingeben und das Programm prüft, ob diese Zahl eine Primzahl ist. Wenn ja, wird die Primzahl in einem Feld der Größe 10 gespeichert.

Lösung

```
/* Beispiel: Arrays und Schleifen
   Filename: main.cpp
   Author: Lahmer
   Title: Primzahlenüberprüfung
   Description: Der Benutzer soll eine beliebige ganzzahlige Zahl eingeben
und das Programm prüft, ob diese Zahl eine Primzahl ist. Wenn ja, wird die
Primzahl in einem Feld der Größe 10 gespeichert.
   Last Change:16.01.2018
*/
//Header-Dateien
#include <iostream>
#include <conio.h>

//Namespace
using namespace std;

//Funktionsprototyp
bool checkPrimzahl(int z);  //Rückgabewert bool, Übergabe per Wert ->
Variable z
void ausgabe(int array[10],int anzahl);   //Rückgabewert void (nichts),
Übergabe per Zeiger (array, Arrays werden immer per Zeiger übergeben) und
Übergabe per Wert (anzahl)


//Hauptprogramm
int main(int argc, char** argv) {

    //Lokale Variablendeklaration
    //Integer Array
    int feld[10];
    bool erg;
    int zahl;
    char nochmal=' ';
    int k=0;

    //Beginn der Do-While Schleife
    do {
```

```
        cout << "Geben Sie bitte eine ganzzahlige Zahl ein: ";
        cin >> zahl; //Eingabe der Zahl

        //Aufruf der Funktion checkPrimzahl
        erg=checkPrimzahl(zahl);
        cout << endl << endl;

        //Bedinung, Prüfen ob erg == true ist, wenn ja => Zahl ist Primzahl,
wenn nein => Zahl ist keine Primzahl
        if(erg==true)
        {
            cout << "Die eingegebene Zahl ist EINE Primzahl!" << endl;
            feld[k]=zahl;     //Zahl wird in das Array namens feld an die
Stelle k gespeichert
            k++;    //k wird um 1 erhöht
        }
        else
        {
            cout << "Die eingegebene Zahl ist KEINE Primzahl" << endl;
        }
        cout << "Willst du noch eine ganzzahlige Zahl eingeben? (j/n)" <<
endl;
        //Einlesen der Entscheidung, ob der Benutzer nochmals eine Zahl
eingeben will
        cin >> nochmal;
        cout << endl << "################################################"
<< endl;

    } while (nochmal=='j');  //Bedingung der do-while Schleife => Solange
der Benutzer j eingibt, wird das Programm wiederholt

    //Alle Primzahlen gesammelt ausgeben
    ausgabe(feld, k);    //Übergabe der Adresse von feld und der Anzahl der
Primzahlen (k)

    return 0;
}

//Funktionsdefinition
bool checkPrimzahl(int z)
{
    //Lokale Variablendeklaration bool = kann nur true oder false beinhalten
    bool primZ=true;

    //Prüfe alle Teiler von 2 bis z-1
    for(int i=2; i<z; i++)
    {
        //Testen ob z/i ohne Rest teilbar
        if(z%i==0)
        {
            primZ=false;
```

```
        }
    }

    //Rückgabe des booleans ob die Zahl ein Primzahl ist (true oder false)
    return primZ;
}


void ausgabe(int array[10], int anzahl)
{
    //Ausgabe aller Primzahlen
    for(int i=0; i<anzahl; i++)
    {
        cout << array[i] << endl;
    }
}
```

Ausgabe

```
Geben Sie bitte eine ganzzahlige Zahl ein: 16


Die eingegebene Zahl ist KEINE Primzahl
Willst du noch eine ganzzahlige Zahl eingeben? (j/n)
j

####################################################
Geben Sie bitte eine ganzzahlige Zahl ein: 13


Die eingegebene Zahl ist EINE Primzahl!
Willst du noch eine ganzzahlige Zahl eingeben? (j/n)
j

####################################################
Geben Sie bitte eine ganzzahlige Zahl ein: 317


Die eingegebene Zahl ist EINE Primzahl!
Willst du noch eine ganzzahlige Zahl eingeben? (j/n)
n

####################################################
13
317

--------------------------------
Process exited after 19.88 seconds with return value 0
```

```
Drücken Sie eine beliebige Taste . . .
```

Arrays, Funktionen, Schleifen und Zufallszahlen - Beispiel

Angabe / Beschreibung

Der Benutzer gibt die Anzahl der zu erstellenden Zufallszahlen ein und das Programm generiert die Zufallszahlen (zw. 1991 und 2018) und speichert diese in ein Array. Danach wird der Mittelwert und die Summe der zufällig generierten Zahlen berechnet.

Lösung

```
/* Beispiel: Arrays, Funktionen, Schleifen und Zufallszahlen
   Filename: main.cpp
   Author: Lahmer
   Title: Berechnung des Mittelwerts und der Summe von Zufallszahlen
   Description: Der Benutzer gibt die Anzahl der zu erstellenden
Zufallszahlen ein und das Programm generiert die Zufallszahlen (zw. 1991 und
2018) und speichert diese in ein Array. Danach wird der Mittelwert und die
Summe der zufällig generierten Zahlen berechnet.
   Last Change:16.01.2018
*/
//Header-Dateien
#include <iostream>
#include <stdlib.h>
#include <time.h>

//Namensraum
using namespace std;

//Funktionsdeklaration
//Funktion, die das Array mit Zufallswerten befüllt
void erstelleArray(int *feld, int anzahl);    //äquivalent zu void
erstelleArray(int feld[]); -> Rückgabewert void (nichts), Parameterübergabe
per Pointer (feld) und per Wert (anzahl)
void ausgabe(int feld[], int anzahl);  //Rückgabewert void (nichts),
Parameterübergabe per Pointer (feld) und per Wert (anzahl)
int summe(int feld[], int anzahl); //Rückgabewert int (ganzzahlige Zahl),
Parameterübergabe per Pointer (feld) und per Wert (anzahl)
float mittelwert(int summe, int anzahl); //Rückgabewert float
(Gleitkommazahl), Parameterübergabe per Wert (summe, anzahl)


//Hauptprogramm
int main(int argc, char** argv) {
```

```
    //Lokale Variablendeklaration
    int anz=0;

    cout << "Geben Sie bitte die Anzahl der Werte des Arrays an: ";
    //Einlesen einer ganzzahligen Zahl
    cin >> anz;

    //Lokale Variablendeklaration
    int array[anz];        //Deklaration eines Arrays namens array mit der
Größe anz
    int sum=0;
    float mittel=0.0;    //Deklaration der Variable mittel und
Initialisierung mit 0.0

    //Funktionsaufruf
    erstelleArray(&array[0],anz);   //äquivalent zu
erstelleArray(feld,anzahl);
    ausgabe(&array[0],anz);
    sum=summe(&array[0], anz);
    mittel=mittelwert(sum, anz);

    //Ausgabe des Mittelwerts
    cout << endl << "Mittelwert: " << mittel;

    return 0;
}

//Funktionsdefinition
//Erzeugt Zufallszahlen im Bereich von 1991-2018, 10-90
void erstelleArray(int *feld, int anzahl)
{
    srand(time(0)); //time(0) liefert die Sekunden seit 1.1.1970 --> dadurch
wird der Zufallsgenerator initialisiert

    for(int i=0; i<anzahl; i++)
    {
        feld[i]=rand() % 28 +1991;        //*(feld+i)=rand(0);
    }

}

//Ausgabe des Arrays
void ausgabe(int feld[], int anzahl)
{
    for(int j=0; j<anzahl; j++)
    {
        cout << feld[j] << endl;   //äquivalent zu *(feld+j);
    }
}
//Berechnet die Summe aller Zahlen im Array und gibt diese zurück
int summe(int feld[], int anzahl) {
```

```
    int sum=0;

    //Berechne die Summe aller Zahlen des Arrays
    for (int i=0; i<anzahl; i++)
    {
        sum=sum+feld[i];
    }

    //Rückgabe des Inhalts von sum an das Hauptprogramm
    return sum;

}
//Berechnet den Mittelwert aller Zahlen und gibt diesen als Gleitkommazahl
zurück
float mittelwert(int summe, int anzahl)
{
    float mittel=0.0;
    mittel=summe/anzahl;

    //Rückgabe des Inhalts von mittel an das Hauptprogramm
    return mittel;
}
```

Ausgabe

```
Geben Sie bitte die Anzahl der Werte des Arrays an: 13
2017
2002
1996
1991
2000
2015
2005
1992
2001
1993
2006
1999
2003

Mittelwert: 2001
--------------------------------
Process exited after 3.698 seconds with return value 0
Drücken Sie eine beliebige Taste . . .
```

Sortieralgorithmen

Sortierproblem

Das Sortierproblem besteht darin, Datensätze eines gegebenen Datenbestands sortiert anzuordnen. Im Folgenden geht es darum, das Sortierproblem und seine Relevanz in der Informatik genauer zu betrachten.

Lösung des Sortierproblems

Zur Lösung des Sortierproblems sind eine Vielzahl an Verfahren entwickelt worden. Wir werden einige dieser Verfahren hier vorstellen und zur Verdeutlichung der Komplexitätsbetrachtungen in den folgenden Abschnitten nutzen. Um die Ideen und Komplexitätsbetrachtungen möglichst einfach zu gestalten, sollen nur Zahlen anstelle komplexer Datensätze betrachtet werden.

- Grundlagen Sortieralgorithmen
- Beschreibung verschiedener Sortieralgorithmen
- Sortieralgorithmen animiert
- http://www.sorting-algorithms.com/

Angabe

Sortiere ein vorgegebenes Feld (10,6,3,5,1) per SelectionSort in einer Funktion.

Lösung

```
/* Beispiel: Funktionen, Arrays und Sortieralgorithmen
   Filename: main.cpp
   Author: Lahmer
   Title: SelectionSort
   Description: Ein vorgegebenes Feld wird per SelectionSort (Suche nach
Minimum) sortiert
   Last Change:19.01.2018
*/
//Header-Dateien
#include <iostream>

//Namensraum
using namespace std;


//Funktionsdeklaration
void Sortieren(int feld[], int size);  //kein Rückgabewert, Übergabe: Feld,
Größe des Feldes
void ausgabe(int feld[], int size);        //kein Rückgabewert, Übergabe:
Feld, Größe des Feldes

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

    int feld[5] = {10, 6, 3, 5, 1};

    //Funktionsaufruf
    Sortieren(feld, 5);                 //Aufruf der Funktion Sortieren
    ausgabe(feld, 5);                 //Aufruf der Funktion ausgabe

    return 0;
}


//Funktionsdefinition
void Sortieren(int feld[], int size){
    int position=0, hzahl;        //position=Position des Minimums,
hzahl=Hilfszahl für Zahlentausch
    for(int j=0; j<size; j++)      //Gehe von 0 bis 4 durch
    {
```

```
        position=j;    //1. Position des sortierten Teiles des Arrays (1. Durchlauf 0, 2.Durchlauf = 1; 3. Durchlauf =2; ... =4)
        for(int i=j+1; i<size; i++)   //Suche Minimum - beginne immer bei j+1 ; da ein Vergleich von feld[i]<feld[position] keinen Sinn macht, da sie im ersten Schritt auf dieselbe Zahl zeigen
        {
            if(feld[i]<feld[position])    //Schaue ob Feld[i] kleiner als das aktuelle Minimum ist
            {
                position=i;                //Wenn ja, speichere die aktuelle Position i in position
            }
        }
        hzahl=feld[j];                //speichere die Zahl an Position j in die Hilfsvariable hzahl
        feld[j] = feld[position];    //speichere das Minimum an der Position position an die Position j
        feld[position] = hzahl;        //speichere hzahl an die Position position
    }
}

//Funktionsdefinition
void ausgabe(int feld[], int size)   //Ausgabe des Feldes
{

    for(int i=0; i<size;i++)
    {
        cout << feld[i] << endl;
    }

}
```

Ausgabe

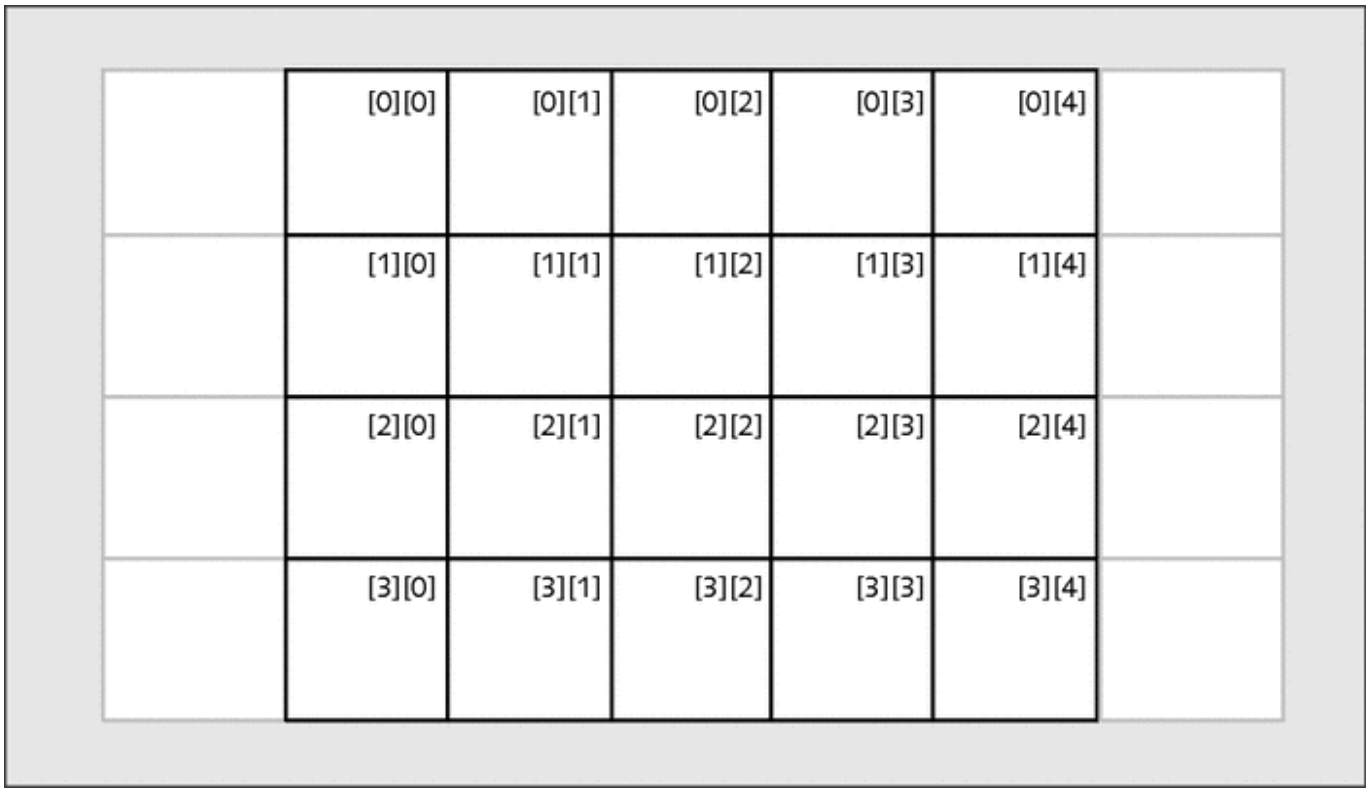
```
1
3
5
6
10

--------------------------------
Process exited after 0.08686 seconds with return value 0
Drücken Sie eine beliebige Taste . . .
```

Mehrdimensionale Arrays

Arrays, wie sie bisher besprochen wurden, können Sie sich als einen Strang von hintereinander aufgereihten Zahlen vorstellen. Man spricht dann von eindimensionalen Arrays oder Feldern. Es ist aber auch möglich, Arrays mit mehr als nur einer Dimension zu verwenden:

```
int Matrix[4][5];    /* Zweidimensional - 4 Zeilen x 5 Spalten */
```

Hier wurde z. B. ein zweidimensionales Array mit dem Namen Matrix definiert. Dies entspricht im Prinzip einem Array, dessen Elemente wieder Arrays sind. Sie können sich dieses Feld wie bei einer Tabellenkalkulation vorstellen.



```
/* Beispiel: Mehrdimensionale Arrays
   Filename: main.cpp
   Author: Lahmer
   Title: Schiffe versenken
   Description: Es soll eine vereinfachte Variation von Schiffe versenken
erstellt werden.
   Last Change: 19.01.2018
*/

//Header-Dateien
#include <iostream>

//Namespace
using namespace std;

//Globale Variablen
#define SPALTEN 5
#define ZEILEN 5
#define SCHIFFLAENGE 3


//Funktionsprototyp
void initializeArray(char feld[ZEILEN][SPALTEN]);  //Rückgabewert void,
Übergabe per Zeiger
void printArray(char feld[ZEILEN][SPALTEN]);   //Rückgabewert void, Übergabe
per Zeiger
int check(char feld[ZEILEN][SPALTEN], char schiff[ZEILEN][SPALTEN], int
zeile, int spalte);  //Rückgabewert int, Übergabe per Zeiger, per Zeiger,
per Wert & per Wert


//Hauptprogramm
int main(int argc, char** argv) {

    //Lokale Variablendeklaration
    //Mehrdimensionales Character-Feld für die Anzeige
    char feld[ZEILEN][SPALTEN];
    //Mehrdimensionales Character-Feld für das Schiff
    char schiff[ZEILEN][SPALTEN];
    int zeile=0, spalte=0, treffer=0;


    //Feld für das Spielfeld mit Werten initialisieren
    initializeArray(feld);

    //Feld für das Schiff  mit Werten initialisieren
    initializeArray(schiff);

    //Schiff verstecken
    schiff[1][2]='X';
    schiff[1][3]='X';
```

```
    schiff[1][4]='X';

    printArray(feld);

    //Do-While Schleife solange nicht das ganze Schiff versenkt ist
    do {

        cout << endl << "Geben Sie die Zeilennummer ein:";
        cin >> zeile;
        cout << endl << "Geben Sie die Spaltennummer ein:";
        cin >> spalte;

        //Die Treffer mitzählen, um zu wissen wann das Schiff versenkt ist
        treffer=treffer+check(feld,schiff,zeile,spalte);

        //Spielfeld ausgeben
        printArray(feld);

    }while(treffer<SCHIFFLAENGE);

    cout << endl << endl << "!!!!!!!!!!! HERZLICHEN GLUECKWUNSCH SIE HABEN
DAS SCHIFF VERSENKT !!!!!!!!!!!!!!!";

    return 0;
}

//Alle Feldelemente mit # initialisieren
void initializeArray(char feld[ZEILEN][SPALTEN]) {

    for (int i=0; i<ZEILEN;i++)
    {
        for (int j=0; j<SPALTEN;j++)
        {
            feld[i][j]='#';
        }
    }
}

//Spielfeld ausgeben
void printArray(char feld[ZEILEN][SPALTEN]) {

    cout << endl << endl << "-----------------------------------------------
--------------------";

    cout << endl << "Spielfeld: " << endl;
    cout << "   ";
    for (int i=0; i<ZEILEN;i++)
    {
        cout << i << " ";
    }
```

```
    cout << endl;
    for (int i=0; i<ZEILEN;i++)
    {
        cout << i << " ";
        for (int j=0; j<SPALTEN;j++)
        {
            cout << feld[i][j] << " ";
        }
        cout << endl;
    }
    cout  << "-----------------------------------------------------------
-----" << endl << endl;

}


//Prüfen, ob ein Schiff getroffen wurde
int check(char feld[ZEILEN][SPALTEN], char schiff[ZEILEN][SPALTEN], int
zeile, int spalte) {

    if(schiff[zeile][spalte]=='X')
    {
        cout << endl << "!!!! TREFFER !!!!!" << endl;
        feld[zeile][spalte]='X';

        return 1;
    }
    else {
        cout << endl << "!!!! KEIN TREFFER !!!!!" << endl;
        feld[zeile][spalte]='-';

        return 0;
    }

    return 0;

}
```