LANDESBERUFSSCHULE 4 SALZBURG

Informatik

Streams, Namespace, Zeiger und Referenzen

LBS 4

Dieses Skript dient als zusätzliche Lernunterlage für Informatik

Inhalt

Lernziele

Die Schülerinnen und Schüler können/kennen

- Namensräume und deren Verwendung
- Ein- und Ausgaben formatieren
- den Unterschied zwischen einer Referenz und Zeiger
- die Begriffe Stack und Heap erläutern sowie deren Verwendung definieren
- statische und dynamische Objekte erstellen

Was ist C++

C++ ist die logische Weiterentwicklung von C und implementiert das Konzept der objektorientierten Programmentwicklung. Objektorientiertes Programmieren bedeutet die Trennung von Daten und Konstrukt sowie mittels Kapselung den Zugriff auf Daten zu kontrollieren. Weites wurden Namensräume, Polymorphismus und Vererbung eingeführt.

Die vier Grundelemente der Objektorientierung lauten

- Kapselung
- Vererbung
- Polymorphismus
- Abstraktion

Alle bekannten Sprachelemente von C können auch in C++ verwendet werden.

Namensraum (namespace)

Der Namespace wird in objektorientierten Sprachen verwendet und bietet ein besseres Handling für den Gültigkeitsbereich von Variablen, Methoden, Klassen, ... Namenskonflikte von gleichen Klassen, Methoden und Variablen können dadurch vermieden werden.

Normalerweise ist die Sichtbarkeit von Namen durch Blöcke begrenzt. Wie bekannt lassen sich die Blöcke verschachteln. Mit Namespaces kann ein zusätzlicher Gültigkeitsbereich erstellt werden.

Der Standard Namespace heißt **std**.

Syntax Namespace C++

```
Namespace NAME {

Code

}
```

Beispiel C++

```
namespace klasse{

        void calc();

}
```

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

// namespace eins
namespace eins {
   void func(){
      cout << "Im ersten Namespace" << endl;
   }
}
// namespace zwei
namespace zwei{
   void func(){
      cout << "Im zweiten Namespace " << endl;
   }
}
int main() {

eins::func(); //im ersten Namespace
zwei::func(); //im zweiten Namespace

return 0;
}
```

Übung

Schreiben Sie ein erstes Programm und definieren Sie zwei verschiedene Namensräume mit derselben Methode/Funktion. Testen Sie das Programm in der main-Methode

Streams

Alle Ein-und Ausgaben sind in der Klasse <iostream.h> definiert. Für die Ausgabe am Bildschirm wird die Funktion cout() für die Eingabe wird cin() verwendet. Alle Ein- und Ausgabestreams werden gepuffert, damit eine kontinuierliche Ausgabe erfolgen kann. Daten werden mit dem **<<** Operator an cout übergeben. Im Gegensatz zu C

müssen keine Datentypen angegeben werden. Die Ausgabe **endl** schreibt den gesamten Ausgabepuffer und beinhaltet einen Zeilenvorschub. Die Methode **flush()** löscht den Ausgabepuffer.

COUT

cout wird für die Ausgabe am Bildschirm verwendet.

Beispiel cout:

```
cout << „LBS“ << 4 << endl;
```

Formatierung von Ganzzahlen: (oct, hex, dec)

Ganzzahlen können als Oktal-, Hexadezimal- oder Dezimalzahl ausgegeben werden. Die Manipulatoren heißen oct, hex, und dec.

Beispiel: Eine Zahl soll im hexadezimalen Format ausgegeben werden.

```
int zahl = 11;

cout << "Die Zahl " << zahl << " in hexadezimaler Darstellung: "<< hex << zahl;// b

cout << "Die Zahl " << zahl << " in hexadezimaler Darstellung: " << uppercase << hex << zahl; // Ausgabe B
```

Die hexadezimalen Zahlen werden klein ausgegeben. Mit dem Manipulator *uppercase* können diese als Großbuchstaben ausgegeben werden. Mit *nouppercase* wird der Standard wiederhergestellt.

Übung:

Schreiben Sie ein kleines Programm welches eine beliebige Zahl in der Konsole als Hexadezimalzahl, Dezimalzahl und Oktalzahl zeigt

Vorzeichen anzeigen

Der Manipulator *showpos* zeigt bei positiven Zahlen ein Vorzeichen. Der Schalter *noshowpos* löscht den Flag.

```
int a = 1;
int b = 0;
int c = -1;
std::cout << std::showpos << a << '\t' << b << '\t' << c << '\n';
//Ausgabe:    +1      +0      -1
```

Gleitkommazahlen

Die Anzahl an Stellen von Gleitkommazahlen können fix vorgegeben werden. Je nach Datentyp float (Genauigkeit 7 Stellen) oder double (Genauigkeit 15 Stellen) werden die Zahlen mehr oder weniger genau angezeigt. Die Dezimalstellen werden von links nach rechts bewertet.

Genauigkeit ändern (Anzahl der Stellen)

Der Manipulator ***setprecision*** gibt die Genauigkeit der Stellen an. Die Stellen werden von links nach rechts gezählt.

```
double zahl = 11.12;
cout << setprecision(3) << zahl; //Ausgabe 11.1
//Analog
double zahl = 11.12;
cout.precision(3);
cout << zahl; //Ausgabe 11.1
```

Der Manipulator *showpoint* zeigt den Dezimalpunkt an. Es werden so viele Ziffern angezeigt, wie es der eingestellten Genauigkeit (Datentyp) entspricht.

```
double zahl = 11;
cout << showpoint << setprecision(5) << zahl; //Ausgabe 11.000
```

Der Schalter *fixed* gibt die Stellen nach dem Dezimalpunkt an.

```
double zahl = 11;
cout << showpoint << fixed  <<  setprecision(2) << zahl; //Ausgabe 11.00
Der Schalter scientific zeigt die Zahl in exponentieller Notation an.
double zahl = 11;
cout << showpoint << scientific  <<  setprecision(2) << zahl; //Ausgabe 1.10e+001
```

Ausgabe in Felder

Für die formatierte Ausgabe in der Konsole kann man Escape-Sequencen oder den Manipulator *setw()* verwenden. Der Manipulator gibt an, welche Feldbreite die Ausgabe reserviert. Als Standard ist das Füllzeichen ein Blank oder Space. Mit der Methode *setfill()* kann man beliebige Füllzeichen setzen.

Die Feldbreite wird von rechts nach links aufgefüllt. Eine andere Ausrichtung kann man mit den Schaltern *left* und *internal* bewirken.

Beispiel: left

```
double zahl = -123;
cout.width(6); // alternative zu cout << setw(6)
cout.fill('*'); // alternative zu cout << setfill('*')
cout << zahl ; //Ausgabe -123**
```

Internal zeigt das Vorzeichen links an und füllt den Platz von rechts.

Beispiel: Internal

```
double zahl = -123;
cout.width(6); // alternative zu cout << setw(6)
cout.fill('0'); // alternative zu cout << setfill('0')
cout << internal << zahl ; //Ausgabe -00123
```

Übung:

Erweitern Sie ihr Programm und testen Sie die Manipulatoren für Gleitkommazahlen. Implementieren Sie die Methoden setw() und setfill().

Eingabe mit CIN

Das Objekt ***cin*** ist für die Eingabe verantwortlich. Der **>>** Operator übergibt die Daten an die Variable. Die Eingabe wird automatisch in den Datentyp der Variable konvertiert und gespeichert.

Beispiel:

```
#include <iostream>
  int main(){
  int myInt;
  float myFloat;
        cin >> myInt;
        cin >>myFloat;
return 0;
        }
```

Methoden von cin

cin besitzt viele Methoden zum Einlesen von Daten.

- cin.get() // ein einzelnes Zeichen einlesen
- cin.getline() // Zeilenweise einlesen
- cin.ignore(23,‘ ‘) // ignoriert Eingabezeichen
- ……

Genauso wie für die Ausgabe kann man den Schalter *setw()* für die Breite der Eingabefelder verwenden.

Achtung: Das letzte Zeichen wird für das String-Ende-Zeichen verwendet.

Beispiel: setw()

```
cin >> setw(12); // ließt 11 Zeichen von der Tastatur ein. 12.Zeichen String Ende '\0'
```

„cin“ ist zeilenweise gepuffert und wird mit “enter“ (New Line) abgeschlossen. Die Konvertierung der Daten hängt von den verwendeten Variablentypen ab. Mit *cin* werden alle nichtdruckbaren Zeichen abgeschnitten.

Beispiel Eingabe eines einzelnen Zeichens:

```
char ch;
cin >> ch; //bei Eingabe von <tab> x <shift> <enter> →wird x gelesen.
int zahl;
cin >> zahl; //bei Eingabe von 3B4 → Zahl bekommt den Wert 3;Rest bleibt im Buffer
```

Den Buffer und die Flags (Fehlerflags) kann man natürlich löschen. Es gibt zwei Befehle.

```
cin.sync() // Flags löschen
cin.clear() // Buffer leeren
```

Übung:

Erweitern Sie ihr Programm, lesen Sie Zahlen von der Konsole ein und stellen Sie diese mit verschiedenen Formatierungen dar.

Referenzen und Zeiger (Call by Reference)

C++ bietet eine einfachere Möglichkeit zum Verwenden von Zeigern. Diese werden Referenzen genannt. Referenzen sind **konstante** Zeiger die keinen Dereferenzierungsoperator benötigen. Bei der Deklaration einer Referenz wird der Adressoperator **&** verwendet.

Zeiger oder Referenzen werden verwendet, wenn

- keine Kopie des Objektes erstellt werden kann
- das Objekt in einer Methode verändert werden soll
- das Kopieren des Objektes zu viel Zeit oder Speicherplatz benötigt
- das Objekt nur einmal existieren darf

Eine Referenz muss immer auf eine existierende Objektinstanz zeigen und bleibt dauerhaft mit dieser verbunden! Eine Referenz muss nicht de referenziert werden.

Syntax Referenz C++

```
Datentyp  &Name;

int  &rName;
```

Beispiel: C++	Beispiel: C
int age =12;	int age = 12;
int &rAge = age;	int *pAge = &age
rAge zeigt auf age	pAge zeigt auf age

Beispiel C++

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

void swap(int &wert1, int &wert2) { //Referenz bei der Übergabe
    int tmp;
    tmp   = wert1;
    wert1 = wert2;
    wert2 = tmp;
}
void swapP(int *wert1, int *wert2){ //Zeiger bei der Übergabe

   int temp = *wert1;
   *wert1 = *wert2;
   *wert2 = temp;
}
```

```
int main() {
    int a = 1, b = 10;

        cout << "a: " << a << ", b: " << b << "\n";
                swapR(a, b);
        cout << "Swap mit Referenz a: " << a << ", b: " << b << endl;
   a = 1, b = 10;
                swapP(&a,&b);
         cout << "Swap mit Pointer a: " << a << ", b: " << b << endl;
}
```

Referenzen können auch als Rückgabewert in Methoden verwendet werden.

Beachten Sie:

Es muss sichergestellt sein, dass keine **Referenz einer lokalen Variable** der Funktion/Methode zurückgegeben wird.

Beispiel:

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

// gibt die Referenz der Variable s zurück
int &zahl(int &b) {
        int s = b +10;
        cout << s << endl;
        return s;

int main() {
        int a = 10;
        zahl(a);
        cout << a; // Ausgabe:?
        return 0;
        }
```

Statisches erzeugen von Objekten

Ein statisches Objekt wird erzeugt, wenn eine Objektvariable deklariert wird.

```
Rechteck r; //statische Erzeugung
```

Statisch bedeutet hier, dass C++ das Objekt auf dem Stack speichert. Am Stack werden alle lokalen Variablen einer Funktion oder Methode gespeichert. Am Ende jeder Methode oder Funktion werden die Variablen vernichtet und der Stack wird bereinigt.

Statisches Erzeugen von Objekten ist eine Besonderheit von C und C++. Andere Sprachen C#, VB(.NET) oder Java kennen keine statische Erzeugung.

Das Problem bei statischen Objekten ist, dass Sie schon bei der Entwicklung wissen müssen, wie viele Objekte gespeichert werden sollen, weil Sie ja die entsprechende Anzahl an Variablen deklarieren müssen. Für manche Probleme reicht das aber nicht aus. Wollen Sie z. B. ein Programm erzeugen, das einen Stammbaum darstellen soll, können Sie bei der Entwicklung noch gar nicht wissen, wie viele Personen-Objekte der Anwender erzeugen und speichern will.

Wenn ihre Objekte sehr viel Speicher benötigen, so kann die Maximalgröße des Stack (per Voreinstellung 1 MB) überschritten werden kann. (Pufferüberlauf)

Deshalb ist es in C++ auch möglich dynamische Objekte zu erzeugen.

Dynamisches erzeugen von Objekten

Für die dynamische Erzeugung von Objekten deklarieren Sie die Objektvariable als Zeiger:

```
Rechteck *r;
```

Dann erzeugen Sie das Objekt über den *new*-Operator. Dieser liefert einen Zeiger auf den reservierten Speicher zurück.

```
r = new Rechteck;
```

Dynamisch erzeugte Objekte werden auf dem Heap angelegt. Der Heap ist ein Speicherbereich, der für globale Daten reserviert ist. Auf dem Heap werden alle Funktionen und Klassen einer Anwendung gespeichert. Die Größe ist durch den Systemspeicher begrenzt.

Objekte die dynamisch erzeugt wurden, können über die Dereferenzierung des Zeigers verwendet werden.

```
(*r).laenge = 20;
```

Als Kurzschreibweise hat sich der Pfeiloperator durchgesetzt.

```
r->laenge = 20;
```

Dynamisch erzeugte Objekte müssen nach der Verwendung wieder zerstört werden.

Für diesen Zweck gibt es den Befehl delete.

```
delete r;
```

Damit wird der reservierte Speicher wieder frei gegeben.