LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Grundlagen C und C++

Einheit 03: Grundlagen in C++

Lorenz Schauer
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

Teil 1: Wiederholung C

- Nutzereingaben verarbeiten

Teil 2: Grundlagen in C++

- Erstes Programm
- Variablen
- Aus- und Eingaben
- Kontrollstrukturen
- Arrays, Pointers, Funktionen
- Namensräume

Übungen

- Nutzereingaben in C++
- Personendaten
- Kreisberechnung

Lernziele

- Einstieg in die Programmiersprache C++
- Grundlagen zu C++ kennenlernen
- Erste Übungen und Programme in C++

Programmieraufgabe (letzte Stunde)

Aufgabe: Nutzereingaben verarbeiten

Schreiben Sie ein C-Programm, das folgenden Aufgaben erledigt:

- Der Benutzer wird immer wieder aufgefordert eine Zahl einzugeben, und zwar solange, bis er eine 0 eingibt.
  - Achten Sie darauf, dass der Benutzer mit seinen Eingaben nicht in einen fremden Speicherbereich schreiben kann
- Die eingegebenen Zahlen sollen dann zur Kontrolle auf der Konsole ausgegeben werden
- Daraufhin berechnet das Programm den Durchschnitt, die kleinste und die größte eingegebene Zahl.
- Die Ergebnisse werden dem Benutzer dann auf der Konsole ausgegeben.

Nutzen Sie für die Bearbeitung der Aufgabe die eben vorgestellten C-Grundlagen

Zum ersten Programm: Hello World

Theoretisch können wir unser Hello World auch wieder komplett in C schreiben und es mit dem C++ Compiler übersetzen lassen.

- C Programme lassen sich in der Regel auch mit dem C++ Compiler übersetzen

Wir wollen aber nun C++ Syntax verwenden.

- C++ Programme lassen sich nicht mit dem C-Compiler übersetzen

Für Hello World brauchen wir wieder die Standardausgabe

- In der Header-File `iostream` sind die Funktionen für `input` und `output` vorhanden.
  - Also Einbinden mittels: `#include <iostream>`
- Ausgaben erzeugen wir dann mittels `std:: cout << „Meine Ausgabe!“;`
- Der Grundsätzliche Aufbau eines C++ Programm ist sehr ähnlich zu C

Zum ersten Programm: Hello World

Schreiben Sie ein Hello World Programm in C++ und übersetzen Sie dieses mit dem C++ Compiler

- `g++ meinHelloWorld.cpp –o meinHelloWorld`
- Anmerkung: Dateiendung kann auch `.C` `.c++` `.cxx` oder `.cc` sein

```
// Erstes Programm Hallo Welt

#include <iostream>

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

   std::cout << „Servus Welt! Wie geht’s dir?\n“;

   return 0;
}
```

Arbeiten mit dem C++ Compiler

Korrektes Arbeiten mit dem Compiler wichtig, um Syntax und Laufzeitfehler zu erkennen, oder Performance zu verbessern

- Bietet neben dem grundsätzlichen Aufruf einige Optionen (Flags) an, zur Anzeige von mehr Informationen

```
// Compiler-Flags und ihre Bedeutung:

g++ -Wall HelloWorld.cpp –o HelloWorld
   warning all: Gibt uns alle Warnungen mit aus

g++ -O HelloWorld.cpp –o HelloWorld
   Für schneller Programmausführung. Aber Kompilierung dauert länger

g++ -g HelloWorld.cpp –o HelloWorld
   Hält Debugging-Informationen bereit

g++ -lm HelloWorld.cpp –o HelloWorld
   Linker: Erlaubt das verlinken einer Library. Hier mathematische Routinen m

Mehr Informationen zu Optionen von g++ unter der eingebauten Hilfe:
   g++ --help bzw. g++ --target-help
```

C++ Variablen & Datentypen

In C++ gibt es im Prinzip die gleichen Datentypen, wie in C

- Also: `char`, `short`, `int`, `long` und `long long`, jeweils auch als `unsigned`
- Oder für Fließkommazahlen: `float`, `double`, `long double`

Darüber hinaus gibt es einen Datentyp für Wahrheitswerte: `bool`

- Bsp.: `bool allesKlar = true;`
- Oder: `bool gelernet = false;`

```
// Beispiele für Deklaration:
double meine_zahl;
double meine_zahl1, meine_Zahl2, meine_zahl3;

// Beispiele für Initialisierung:
int zahl = 100;
int zahl2(200);

// Oder Konstanten:
const double Pi(3.14);
const int MAX_VAL = 100;
```

C++ Variablen & Datentypen

Auch Strings sind jetzt möglich mittels header file `string`

- `#include <string>`
- Bietet verschiedene Methoden auf Strings
  - Bsp.: `mein_string.length(); // Gibt länge des Strings zurück`
  - Bsp.: `mein_string.at(number); // Liefert Zeichen an der Stelle number`

```
// Beispiel für Strings in C++

#include <iostream>
#include <string>

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

   std::string city = “Berlin”;
   std::string address;
   address=“Kreuzberg 1”;

   //Ausgabe
   std::cout << “Willkommen in ”<<city;
   std::cout << “\n”
   std::cout << “Ich wohne in ”<<address<<“\n”;
   std::cout << “Laenge des Strings: ”<<city.length();
}
```

Konsolen Ausgaben und Eingaben

Konsolenausgeben mittels `std::cout << „Meine Ausgabe“`

- Erfordert `#include <iostream>`
- Variablen können einfach eingehängt werden:
  - `std::cout << „Der Wert von x ist: “<<x;`

Konsoleneingaben mittels `std::cin >> eingabe;`

- Erfordert `#include <iostream>`

```
// Beispiel für einfache Nutzereingabe:
#include <iostream>

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

   int pin;
   std::cout << „Geben Sie hier Ihre Pin-Nummer ein: “;
   std::cin >> pin;
}
```

Weitere Nutzereingaben


```
// Auch mehrere Nutzereingaben möglich:
   int account_nr, pin;

   std::cout << „Bitte geben Sie ihre Kontonummer und dann Ihre Pin ein\n“;
   std::cout << „Bestaetigen Sie Ihre Eingaben mit Enter.“;
   std::cin >> account_nr >> pin;

// Auch Stringeingaben möglich:

   #include <string>
   //…

   std::string name;
   std::cout << „Geben Sie Ihren Namen ein: “;
   std::getline(std::cin, name);

   //Ausgabe
   std::cout << „Ihr Name lautet: “<<name <<„\n“;
```

Kontrollstrukturen

Kontrollstrukturen werden bzgl. Ihrer Syntax und Funktion wie bei C oder Java verwendet

- Verzweigungsstrukturen:
  - `if-else, switch-case`
- Wiederholungsstrukturen:
  - `while, do-while, for-Schleife`
- Sprunganweisungen:
  - `break, continue, goto`

```
// Beispiele:

if(x+y<z)
   goto Ziel;

Ziel:
   while(a>b){
      a--;
   }
```

Arrays werden analog zu C verwendet

```
// Beispiele:

//Deklaration:
int array1[2];
int array2[3][4];
double array3[] = {3.2, 2.2, 1.1};
char array4[3] ={‘a‘, ‘b‘, ‘c‘};

//Initialisierung:
array1[0] = 3;
array1[1] = 7;
array2[0][0] = 5;

//Ausgabe:

std::cout << „Buchstabe 1: “<< array4[0];
std::cout << „Wert 2: “<< array3[1];
```

Und auch Pointers werden analog zu C verwendet

```
// Beispiele:
int meine_zahl = 3;
double mein_array[] = {1.1, 2.2, 3.3};

//Deklaration von Pointer:
int* mein_zeiger;
double *mein_zeiger_auf_mein_array;

// Zuweisung:
mein_zeiger = &meine_zahl;
mein_zeiger_auf_mein_array = mein_array;

//Zeigeroperation
mein_zeiger_auf_mein_array++

//Ausgaben:
std::cout << „Der Wert, auf den mein Zeiger zeigt: “<< *mein_zeiger;
std::cout << „Die Adresse, auf die mein Zeiger zeigt: “<< mein_zeiger;
std::cout << „Die Adresse meines Zeigers: “<< &mein_zeiger;

std::cout << „2. Wert des Arrays: “<<*mein_zeiger_auf_mein_array;
```

C++ Funktionen

Syntax von C++ Funktionen analog zu C bzw. Java:

Allgemeine Funktionsdefinition:

```
return_type function_name(type param1, type param2){
        //Body
}
```

- Die Deklaration (Kopf) der Funktion sollte dem Compiler bereits am Dateianfang mittgeteilt werden
  - Notwendig, falls Aufruf und Definition in verschiedenen Quelldateien liegen

```
// Beispiel:
int add(int a, int b); // auch möglich: int add(int,int);

int main(){

   int addition = add(3,4);
   std::cout << „Ergebnis: “<<addition;
   return 0;
}

int add(int a, int b){
   return a+b;
}
```

Namensräume (Namespaces)

Namensräume beugen dem Problem vor, dass sich Funktionen, Variablen usw. mit gleichem Namen überschatten können

- Grenzen also den Gültigkeitsbereich ein
- Ermöglichen eine strukturierte Verwendung
  - Bsp.: `std` (Namensraum der C++ Standardbibliothek)
  - `std::cout` (Greift auf die Ausgabefunktion der Standardbibliothek zurück)
  - `::` ist ein Bereichsoperator

Namensräume können mittels `using`-Direktive global bekannt gemacht werden

- Aber Vorsicht! Überschattungen mit globalem Namensraum möglich
- Daher (wenn überhaupt) eher innerhalb von Funktionen verwenden
- Oder nur einen Teilbereich des Namensraums verwenden

```
// Beispiel mit using-Direktive
// std komplett bekannt machen

using namespace std;
int main(){
   cout << „Hallo Welt“;
}
```

```
// Beispiel mit using-Direktive
// Nur cout bekannt machen

using std::cout;
int main(){
   cout << „Hallo Welt“;
}
```

Namensräume definieren

Namensräume können natürlich selbst definiert werden.

- Allgemein: `namespace name{//Deklarationen}`
- Das folgende Beispiel zeigt gleich die Bedeutung von Namensräumen

```
#include <iostream>

using namespace std;

// first name space
namespace first_space{
   void func(){
      cout << "Inside first_space" << endl;
   }
}
// second name space
namespace second_space{
   void func(){
      cout << "Inside second_space" << endl;
   }
}

using namespace first_space;
int main (){

   // This calls function from first name space.
   func();
   return 0;
}
```

Programmieraufgaben

Aufgabe 1:

- Schreiben Sie Ihr C-Programm vom Anfang der Stunde in C++ Code um, so dass die Funktion des Programm beibehalten wird.

Aufgabe 2:

- Schreiben Sie ein Programm, welches zu einer Person folgende Daten von der Tastatur einliest und diese anschließend wieder ausgibt:
  - Name, Konfession, Alter, Größe, Gehalt.
- Verwenden Sie passende Datentypen!
- Jeder Abfrage geht eine Nutzereingabeaufforderung vorraus.

Aufgabe 3:

- Schreiben Sie ein Programm Kreisberechnung, das folgende Aufgaben erledigt:
  - Der Benutzer soll einen Kommawert für den Radius eingeben!
  - Das Programm errechnet mittels passenden Funktionen:
    - Den Durchmesser (2*r)
    - Den Umfang (2*r*PI)
    - Die Fläche (r^2*Pi)
  - Die Berechnungen sollen dem Nutzer auf der Konsole angezeigt werden.
  - Hinweis: Sie können Pi entweder selber als Konstante definieren oder mittels M_PI aus der Header-Datei math.h verwenden