C++ Crashkurs

Mentoring SoSe 2019

Felix Droop, Julia Mellert & Anton Kriese
Freie Universität Berlin

13. Juni 2019

Inhalt

Vorstellung der Sprache

Variablen

Funktionen

Includes

I/O

Kontrollstrukturen

Schleifen (Loops)

Kompilieren und Ausführen

Über C++

Vorstellung der Sprache:

- 1979 erweitertes C
- Maschinennah -> gute Performance
- ist von der ISO genormt

Über C++

Vorstellung der Sprache:

- 1979 erweitertes C
- Maschinennah -> gute Performance
- ist von der ISO genormt
- Nach 40 Jahren immer noch aktuell
- Objektorientiert
- Imperativ

Über C++

Vorstellung der Sprache:

- 1979 erweitertes C
- Maschinennah -> gute Performance
- ist von der ISO genormt
- Nach 40 Jahren immer noch aktuell
- Objektorientiert
- Imperativ

Anwendung der Sprache:

- Betriebssysteme
- Treiber
- Background Computation

Inhalt

Variablen

- Variable ist Speicherbereich mit Namen

Variablen

- Variable ist Speicherbereich mit Namen
- Dessen Inhalt ist der Wert

Variablen

- Variable ist Speicherbereich mit Namen
- Dessen Inhalt ist der Wert
- Der Wert wird durch den Datentyp interpretiert

Erzeugen von Variablen

- Deklaration (Achtung!)

Erzeugen von Variablen

- Deklaration (Achtung!)

```
1  int val;
```

Erzeugen von Variablen

- Deklaration (Achtung!)

```
1   int val;
```

- Initialisierung

Erzeugen von Variablen

- Deklaration (Achtung!)

```
1  int val;
```

- Initialisierung

```
1  val = 5;
```

Erzeugen von Variablen

- Deklaration (Achtung!)

```
1  int val;
```

- Initialisierung

```
1  val = 5;
```

- Deklaration + Initialisierung

Erzeugen von Variablen

- Deklaration (Achtung!)

```
1  int val;
```

- Initialisierung

```
1  val = 5;
```

- Deklaration + Initialisierung

```
1  char lex = 'a';
2  double dub{}; // == 0.0
```

Nutzen von Variablen

- Imperative Programmierung:

Nutzen von Variablen

- Imperative Programmierung:

```
1  int a = 1;
2  int b = a + 1;
3  b = a + b;
4  a = b + a;
```

Nutzen von Variablen

- Imperative Programmierung:

```
1  int a = 1;
2  int b = a + 1;
3  b = a + b;
4  a = b + a;
```

- Popquiz: Was ist a?

Nutzen von Variablen

- Imperative Programmierung:

```
1  int a = 1;
2  int b = a + 1;
3  b = a + b;
4  a = b + a;
```

- Popquiz: Was ist a?
- Und was passiert hier?

```
1  a + b;
```

Nutzen von Variablen

- Imperative Programmierung:

```
1  int a = 1;
2  int b = a + 1;
3  b = a + b;
4  a = b + a;
```

- Popquiz: Was ist a?
- Und was passiert hier?

```
1  a + b;
```

- Antwort: Wert wird nicht gespeichert, also sinnlose Berechnung.

Nutzen von Variablen

- Imperative Programmierung:

```
1  int a = 1;
2  int b = a + 1;
3  b = a + b;
4  a = b + a;
```

- Popquiz: Was ist a?
- Und was passiert hier?

```
1  a + b;
```

- Antwort: Wert wird nicht gespeichert, also sinnlose Berechnung.
- -, *, / , % usw. gehen auch!

Nutzen von Variablen

- Imperative Programmierung:

```
1  int a = 1;
2  int b = a + 1;
3  b = a + b;
4  a = b + a;
```

- Popquiz: Was ist a?
- Und was passiert hier?

```
1  a + b;
```

- Antwort: Wert wird nicht gespeichert, also sinnlose Berechnung.
- -, *, / , % usw. gehen auch!
- Schönere Syntax:

```
1  a = a * 3;
2  a *= 3;
```

Inhalt

Die main Funktion

- Wird beim Ausführen des Programms aufgerufen

Die main Funktion

- Wird beim Ausführen des Programms aufgerufen
- Jedes C++-Programm muss genau eine haben

Die main Funktion

- Wird beim Ausführen des Programms aufgerufen
- Jedes C++-Programm muss genau eine haben

```
1  int main()
2  {
3      return 0;
4  }
```

Die allgemeine Funktion

- Führt Codeblock aus (function body)

Die allgemeine Funktion

- Führt Codeblock aus (function body)
- Meist in Abhängigkeit von Parametern

Die allgemeine Funktion

- Führt Codeblock aus (function body)
- Meist in Abhängigkeit von Parametern
- Muss Rückgabewert haben (kann void, also nichts sein)

Die allgemeine Funktion

- Führt Codeblock aus (function body)
- Meist in Abhängigkeit von Parametern
- Muss Rückgabewert haben (kann void, also nichts sein)

```
1  return_type my_function(p1_type p1, p2_type p2)
2  {
3    // my process
4    return <something>; // for non-void
5  }
```

Die allgemeine Funktion

- Führt Codeblock aus (function body)
- Meist in Abhängigkeit von Parametern
- Muss Rückgabewert haben (kann void, also nichts sein)

```
1  return_type my_function(p1_type p1, p2_type p2)
2  {
3    // my process
4    return <something>; // for non-void
5  }
```

- Beispiel:

```
1  int add(int x, int y)
2  {
3    return a + b;
4  }
```

Funktionsaufruf

- Funktionen können aufgerufen werden, nachdem sie definiert wurden

- Funktionen können aufgerufen werden, nachdem sie definiert wurden

```
1   int add(int x, int y)
2   {
3       return a + b;
4   }

6   int main()
7   {
8       int sieben = add(3,4);
9       return 0;
10  }
```

Inhalt

Includes

- Kern von C++ ist sehr beschränkt

Includes

- Kern von C++ ist sehr beschränkt
- Weitere Funktionalitäten werden inkludiert

Includes

- Kern von C++ ist sehr beschränkt
- Weitere Funktionalitäten werden inkludiert
- Einige wichtige Includes:

Includes

- Kern von C++ ist sehr beschränkt
- Weitere Funktionalitäten werden inkludiert
- Einige wichtige Includes:

```
1  #include <iostream>   //In und Output auf der Kommandozeile
2  #include <vector>    //Für std::vector (Datencontainer)
3  #include <string>    //liefert spezielle Funktionen für Strings
4  #include <algorithm>  //stellt Sortier- und weitere Algorithmen zur
   Verfügung
5  #include <cmath>     //log, pow, sqrt, ceil, floor, abs usw.
```

Inhalt

- cout und cin sind Workhorses für Userinteraktion über die Kommandozeile

- cout und cin sind Workhorses für Userinteraktion über die Kommandozeile
- Hello World!-Programm:

```
1  #include <iostream>
2
3  int main()
4  {
5     std::cout << "Hello World!" << std::endl;
6     return 0;
7  }
```

- cout und cin sind Workhorses für Userinteraktion über die Kommandozeile
- Hello World!-Programm:

```
1  #include <iostream>
2
3  int main()
4  {
5     std::cout << "Hello World!" << std::endl;
6     return 0;
7  }
```

- Tipp für Übersichtlichkeit am Anfang:

```
1  using namespace std;
```

- Beispiel mit cin:

- Beispiel mit cin:

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main ()
6   {
7      int zahl{};
8      //fordert Benutzer zur Eingabe auf
9      cout << "Gib eine Zahl ein:" << endl;
10     //lässt den Benutzer eine Zahl eingeben
11     cin >> zahl;
12     //gibt dem Benutzer seine eingegebene Zahl aus
13     cout << "Du hast die Zahl " << zahl << " eingegeben." << endl;
14
15     return 0;
16  }
```

Inhalt

Abfragen von Konditionen

if, **else if** und **else** werden benutzt, um *Konditionen* zu überprüfen und abhängig von verschiedenen Zuständen verschiedenen Code auszuführen.

Abfragen von Konditionen

if, **else if** und **else** werden benutzt, um *Konditionen* zu überprüfen und abhängig von verschiedenen Zuständen verschiedenen Code auszuführen.

Logische Operatoren: `||`, `&&`, `!`, `==`, `<`, `>`, `<=`, `>=`

Abfragen von Konditionen

if, **else if** und **else** werden benutzt, um *Konditionen* zu überprüfen und abhängig von verschiedenen Zuständen verschiedenen Code auszuführen.

Logische Operatoren: `||`, `&&`, `!`, `==`, `<`, `>`, `<=`, `>=`

```
1   //Prinzip:
2   if ( Bedingung ) { Anweisung }

4   //Beispiel:
5   bool b1 = 0, b2 = true;
6   if ( b1 && b2 ) {
7      cout << "Beide bools sind true." << endl;
8   }
9   else if ( b1 || b2 ) {
10     cout << "Einer der bools ist true." << endl;
11  }
12  else { //deckt alle anderen Fälle ab, optional
13     cout << "Beide bools sind false." << endl;
14  }
```

Das switch-Statement

switch wird für Fallunterscheidungen benutzt, die viele Fälle beinhalten und sorgt für mehr Übersichtlichkeit.

Das switch-Statement

switch wird für Fallunterscheidungen benutzt, die viele Fälle beinhalten und sorgt für mehr Übersichtlichkeit.

```
1     //Beispiel:
2     int zahl = 3;
3     switch (zahl % 10) {
4        case 0:   //do something
5             break;
6        case 1:   //do something else
7             break;
8        //...
9        default:  //do something otherwise
10    }
```

Inhalt

Was sind Schleifen?

Eine **Schleife (”loop”)** wird verwendet, um einen Abschnitt des Programms mehrfach auszuführen. Loops sind meistens an Konditionen von Variablen gekoppelt wodurch die Wiederholrate kontrolliert werden kann.

Was sind Schleifen?

Eine **Schleife (”loop”)** wird verwendet, um einen Abschnitt des Programms mehrfach auszuführen. Loops sind meistens an Konditionen von Variablen gekoppelt wodurch die Wiederholrate kontrolliert werden kann.

Arten von Schleifen:

1. `for`-Loop: Wiederholungszahl durch Range festgelegt
2. `while`-Loop: Wiederholung bis Kondition sich ändert
3. Noch weitere, die man nie braucht

Was sind Schleifen?

Eine **Schleife (”loop”)** wird verwendet, um einen Abschnitt des Programms mehrfach auszuführen. Loops sind meistens an Konditionen von Variablen gekoppelt wodurch die Wiederholrate kontrolliert werden kann.

Arten von Schleifen:

1. `for`-Loop: Wiederholungszahl durch Range festgelegt
2. `while`-Loop: Wiederholung bis Kondition sich ändert
3. Noch weitere, die man nie braucht

Gut zu wissen!

Prinzipiell können alle Schleifenarten ineinander *umgeformt* werden. `for` bietet sich für abgezählte Anwendungen an, während `while` für ”offene” Schleifen besser geeignet ist.

for-Loop

```
1   // Prinzip klassischer for-Loops
2   for ( Init. d. Zählvar.; Konidition; Aktion auf Zählvar.) { //Anweisungen }

4   // Beispiel 1: Ausgeben der Zahlen 0 bis 9
5   for ( int i = 0; i < 10; ++i ) {
6       cout << i << endl;
7   }

9   // Beispiel 2: Summe der Quadrate der ersten zehn geraden Zahlen
10  int squares = 0;
11  for ( int i = 0; i < 20; i+=2 ) {
12     squares += i*i;
13  }
14  cout << squares << endl;
```

while-Loop

```
1  // Prinzip:
2  while ( Kondition ) { Anweisungen }

4  // Beispiel, addiere vom Nutzer eingegebene Zahlen
5  int summe{};
6  int zahl = 1;
7  while ( zahl != 0 ) {  // verlässt die Schleife, nachdem 0 eingegeben wird
8    cin >> zahl;      // Nutzer gibt zu addierende Zahl ein
9    summe += zahl;
10 }
```

Inhalt

Ausführen eines Programms

Um ein C++-Programm ausführen zu können, muss es zunächst kompiliert werden. Oft verwendete Compiler:

Ausführen eines Programms

Um ein C++-Programm ausführen zu können, muss es zunächst kompiliert werden. Oft verwendete Compiler:

Betriebssystem	Compiler
Linux	GCC (g++)
Windows	Visual C++
MacOS	Clang, GCC, Xcode (Clang)

Ausführen eines Programms

Um ein C++-Programm ausführen zu können, muss es zunächst kompiliert werden. Oft verwendete Compiler:

Betriebssystem	Compiler
Linux	GCC (g++)
Windows	Visual C++
MacOS	Clang, GCC, Xcode (Clang)

GCC

Kompilieren:

- `g++ SourceCode.cpp -o Executable`

Folgende flags sind für das Kompilieren zu empfehlen:

- `-std=c++17 -Wall -Werror -pedantic`

Ausführen:

- `./Executable`

Quellen und nützliche Links

Bildquelle:

- `https://de.wikipedia.org/wiki/Datei:ISO_C%2B%2B_Logo.svg`

Websites, die interessant sein könnten:

- cppreference.com
- cplusplus.com