Praxisorientierte Einführung in C++
Lektion: "Kontrollstrukturen"

Christof Elbrechter

Neuroinformatics Group, CITEC

April 24, 2014

Table of Contents

Übersicht

- Kontrollstrukturen erlauben ...

bedingte Ausführung

- `if-else`
- `switch-case`

Schleifen

- `for`
- `while`
- `do-while`

Sprünge

- `goto` und Spungmarken

if-else

Syntax if-else

```
if (Bedingunung)
  Anweisung1
else
  Anweisung2
```

- `Bedingung` ist ein Ausdruck, der nach `bool` konvertierbar ist
- `Anweisung1` und `Anweisung2` können Ausdrücke, Blöcke oder andere Kontrollstrukturen sein

Beispiel

```
bool x = true;
if (x == true){ // aequivalent: if (x)
   std::cout << "x is true" << std::endl;
}else{
   std::cout << "x is false" << std::endl;
}
```

oder alternativ hier auch ohne Blöcke:

```
bool x = true;
if (x == true)
   std::cout << "x is true" << std::endl;
else
   std::cout << "x is false" << std::endl;
```

(führt allerdings oft zu Fehlern ...)

Negativ-Beispiel für das Weglassen von Blöcken

```
bool x = true;
bool y = true;

if (x == true)
   if (y == false)
      std::cout << "x true and y false" << std::endl;
else
   std::cout << "x false" << std::endl;
```

- Programm gibt "x false" aus
- Zu welchem if gehört das else?
- **Besser:** Explizite Blöcke benutzen!

Gleiches Beispiel mit Blöcken

Explizit geklammert:

```
bool x = true;
bool y = true;

if (x == true){
   if (y == false){
      std::cout << "x true and y false" << std::endl;
   }
}else{
   std::cout << "x false" << std::endl;
}
```

- Nun passt die Einrückung zu der Programm-Syntax
- Programm gibt nichts aus

while-Schleifen

- Anweisung wird wiederholt ausgeführt, solange Bedingung zutrifft (also zu true ausgewertet wird)

Syntax: while

```
while(Bedingung) Anweisung
```

Beispiel für while

```
int x = 0;
while (x < 100) {
   std::cout << x++ << std::endl;
}
```

do-while-Schleifen

- Anweisung wird wiederholt ausgeführt, solange `Bedingung` `true` ist
- Anders als bei `while` wird hier Anweisung mindestens einmal ausgeführt

Syntax: do-while

```
do Anweisung while(Bedingung);
```

Beispiel für do-while

```
int x = 0;
do {
   std::cout << x++ << std::endl;
} while (x < 100);
```

for-Schleifen

- *Großer Bruder* der `while`-Schleife

Syntax: for

```
for (Initialisierung; Bedingung; Schritt)
    Anweisung
```

- `Initialisierung`, `Bedingung`, `Schritt` sind Ausdrücke
- `Bedingung` muss nach `bool` konvertierbar sein

Beispiele für for-Schleifen

```
char text[] = "hallo wie geht's";
for(int i=0;i<strlen(text);++i){
   if(text[i] == ' ') text[i] = '\n';
}
```

oder besser ...

```
for(int i=0,end=strlen(text);i<end;++i){
   if(text[i] == ' ') text[i] = '\n';
}
```

oder ...

```
for(char *t=text;*t;++t){
   if(*t == ' ') *t = '\n';
}
```

Die break-Anweisung

- Die Anweisung `break` verursacht, dass der aktuelle (innerste) `for`,`while` oder `case`-Block verlassen wird

Beispiel **break**

```
#include <string>
int main(){
  std::string str;
  std::cin >> str;
  for(int i=0;true;++i){
     if(!str[i]) break;
     str[i] = toupper(str[i]);
  }
}
```

Die continue-Anweisung

- Verursacht einen Sprung an das Ende der aktuellen Schleife
- Nur innerhalb des Rumpfes von `for`, `while` und `do-while`-Schleifen
- Ermöglicht flachere und übersichtlichere Klammerstruktur

Beispiel **continue**

```
#include <string>
#include <iostream>
int main(){
  std::string str; std::cin >> str;
  for(int i=0;true;i++){
    if(!str[i]) break;
    if(str[i] >= 'A' && str[i] <= 'Z') continue;
    if(str[i] >= 'a' && str[i] <= 'z') str[i] += ('A'-'a');
  }
  std::cout << str << std::endl;
}
```

switch-case

- Für Fallunterscheidungen

Syntax: swtich-case

```
switch(Ausdruck){
  case C1: Anweisungen
  case C2: Anweisungen
  ...
  default: Anweisungen
}
```

- Tatsächlich kann der Block auch nur eine einzelne Anweisung sein (dies ist aber selten sinnvoll)
- `Ausdruck` muss ein Ganzzahl-Typ, ein sog. Integral-Type, sein (`bool`, `char`, `wchar_t`, `(un)signed int`, oder `short`)
- `C1`, `C1`, ... müssen konstante Ganzzahl-Ausdrücke sein.

Anmerkungen zu switch-case

- Sollten in der Anweisungsliste einer case Marke Variablen deklariert werden, so muss die Anweisungsliste in einem Block zusammengefasst werden (Grund: nicht klar definierbare Lebenszeit des Variablenbezeichners)
- **Wichtig:** Die einzelnen Anweisungsteile müssen explizit mit einem `break`-Statement beendet werden.
- Ansonsten werden alle folgenden Anweisungsteile mitausgeführt

switch-case Beispiel

Beispiel: switch-case

```
void handle_event(const MouseEvent &evt){
   switch(evt.getType()){
      case PRESS_EVENT:
         global_mouse_down = true;
         break;
      case RELEASE_EVENT:
         global_mouse_down = false;
         break;
      case MOVE_EVENT:{
         int x = evt.getX(), y= evt.getY();
         global_mouse_pos = Point(x,y);
         break;
      }
      default;
         return;
   }
   update_visualization();
}
```

switch-case - Anmerkungen

- Manchmal möchte man in Fallunterscheidung lokale Variablen anlegen
- Da `break` optional ist kann es hierzu zu Überspringen von Variablendeklarationen komment

```
switch(evt.getType()) {
  case PRESS_EVENT:
    int x = evt.x(); int y = evt.y();
    // ...
  break;
  case MOVE_EVENT:
    int x = evt.x(); int y = evt.y(); // geht nicht
    x = evt.x(); y = evt.y();         // geht auch nicht
    // ...
  break;
}
```

switch-case - Anmerkung

- Lösung: lokale Blöcke benutzen

```
switch(evt.getType()) {
  case PRESS_EVENT:
  {
    int x = evt.x(); int y = evt.y();
    // ...
  }
  break;

  case MOVE_EVENT:
  {
    int x = evt.x(); int y = evt.y(); // geht!!
    // ...
  }
  break;
}
```

Die goto-Anweisung

- Sehr unschönes C-Relikt
- Wird in C++ sehr selten (und ungern) gesehen

Beispiel: goto

```
int main(){
  int i = 0;
  for (; i < 100; ++i) {
    if (i == 10){
      goto MARKE;
    }
  }
  MARKE:
  std::cout << i << std::endl;
  return 0;
}
```

- **Besser:** Verwendung von Exceptions (später)

Anmerkungen zu goto

- `MARKE` und `goto MARKE` müssen in der gleichen Funktion stehen
- Andere Sprünge mittels `setjmp` und `longjmp` (verfügbar durch den Header `<csetjmp>`)
- Diese werden z.B. zur Fehlerbehandlung in C-Bibliotheken verwendet
- Sind aber noch gefährlicher
- In C++: Leichtere Ausnahmenbehandlung mittels `try-catch`-Blöcken und dem `throw`-Operator (später)