Einführung in die Programmierung mit C++

Kontrollfluss

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

Inhalt

Bedingungen und Bedingte Zuweisung

Verzweigungen

- if[-else]
- switch-case
- goto

Schleifen

- while
- do-while
- for

Ausnahmen

Inhalt

Bedingungen und Bedingte Zuweisung

Verzweigungen
- if[-else]
- switch-case
- goto

Schleifen
- while
- do-while
- for

Ausnahmen

Speichern alternativer Werte in x in Abhängigkeit der Gültigkeit einer Bedingung c vom Typ **bool**.

Beispiel

```
x=c ? 1 : 0;
```

x=1 für c==**true**; x=0 für c==**false**;

- relationale (bzw. Vergleichs-) Operatoren: ==, !=, >, <, >=, <=
- logische Operatoren:
  - y=!x

x	y
0	1
1	0

  - z=x&&y

x	y	z
0	0	0
1	0	0
0	1	0
1	1	1

  - z=x||y

x	y	z
0	0	0
1	0	1
0	1	1
1	1	1

Inhalt

- **if**–[**else**]
- **switch**–**case**
- **goto**

Bedingte Verzweigung

if–[else]

- Durchführung einer Anweisung in Abhängigkeit der Gültigkeit einer Bedingung c vom Typ **bool**

```
1 | if (c) x=1;
```

- Durchführung mehrerer Anweisungen in Abhängigkeit der Gültigkeit von c

```
1 | if (c) { x=1; y=1; }
```

- Durchführung mehrerer Anweisungen in Abhängigkeit der Gültigkeit von c; aternative Durchführung mehrerer Anweisungen bei Ungültigkeit der Bedingung

```
1 | if (c) { x=1; } else { x=2; }
```

→ Tafelbild: Kontrollflussgraphen

Bedingte Verzweigung

Beispiel


```
1  int i;
2  cin >> i;
3  if (i>0) { std::cout << "GEWINNER" << std::endl; }
4  else if (i<0)
5    std::cout << "VERLIERER" << std::endl;
6  else {
7    std::cout << "LANGWEILER";
8    std::cout << std::endl;
9  }
```

Beachte: Schachtelung und Kombination mit anderen Kontrollflussstrukturen möglich.

AUFWACHEN!

Was wird ausgegeben?

```
1  int main() {
2    int i; bool b=false;
3    std::cin >> i;
4    if (i>0)
5      b=!b;
6    else
7      b=b&&true;
8    if (b||false) std::cout << "Mojn";
9    std::cout << "!" << std::endl;
10   return 0;
11 }
```

Wach?

```
1  int main() {
2    int i; bool b=false;
3    std::cin >> i;
4    if (i>0)
5      b=!b;
6    else
7      b=b&&true;
8    if (b||false) std::cout << "Mojn";
9    std::cout << "!" << std::endl;
10   return 0;
11 }
```

i>0 ? Mojn! : !

Fallunterscheidung

switch−case


```
1 std::cout << "+";
2 switch (i) {
3   case 0 : std::cout << "i==0";
4   case 1 : std::cout << "i==1"; break;
5   default : std::cout << "i!=0&&i!=1";
6 }
7 std::cout << "+";
```

generiert Ausgaben

`+i==0i==1+`

für i=0,

`+i==1+`

für i=1 und

`+i!=0&&i!=1+`

sonst.

- Einstiegsmarken in Sequenz von Anweisungen in Abhängigkeit des Wertes eines ganzzahligen Ausdrucks
- Reihenfolge beliebig
- Verlassen der switch-Anweisung mittels break
- alternative Einstiegsmarke default
- Schachtelung und Kombination mit anderen Kontrollflussstrukturem möglich.

→ Tafelbild: Kontrollflussgraph

... wir erinnern uns, dass **char** auch ein ganzzahliger Datentyp ist (Stichwort: ASCII)

```
1  char c; std::cin >> c;
2  switch (c) {
3    case '0' :
4      std::cout << "1" << std::endl;
5      break;
6    case '1' :
7      std::cout << "0" << std::endl;
8      break;
9    default : std::cout << c << std::endl;
10 }
```

Fallunterscheidung

switch mit/ohne break


```
1 int i; std::cin >> i;
2 switch (i) {
3   case 1 : std::cout << "1" << std::endl; break;
4   std::cout << "2" << std::endl;
5   default : std::cout << "3" << endl;
6 }
```

generiert Ausgabe

- 1 falls i==1
- 123 falls i==1 und **break** fehlt
- 3 falls i!=1

→ Tafelbild: Kontrollflussgraph

Fallunterscheidung

switch ohne default


```
1 int i; std::cin >> i;
2 switch (i) {
3   case 1 : std::cout << "1"; break;
4   std::cout << "2"; break
5   case 3 : std::cout << "3";
6 }
```

generiert Ausgabe

- 1 falls i==1
- 12 falls i==1 und erstes **break** fehlt
- 3 falls i==3

→ Tafelbild: Kontrollflussgraph

AUFWACHEN!

Was wird ausgegeben?

```
1  ...
2  int main() {
3    int i=2;
4    switch (i) {
5      default : std::cout << "3";
6      case 1 : std::cout << "1";
7      std::cout << "2" << std::endl;
8      break;
9    }
10   return 0;
11 }
```

Wach?

```
1  ...
2  int main() {
3    int i=2;
4    switch (i) {
5      default : std::cout << "3";
6      case 1 : std::cout << "1";
7      std::cout << "2" << std::endl;
8      break;
9    }
10   return 0;
11 }
```

312

WACH BLEIBEN!

Geben sie ein äquivalentes **if**-Konstrukt für

```
1 int i; std::cin >> i;
2 switch (i) {
3    case 2 : std::cout << "2" << std::endl;
4    default : std::cout << "default" << std::endl;
5    case 1 : std::cout << "1" << std::endl; break;
6 }
```

an!

Noch Wach?

```
1  int i; std::cin >> i;
2  switch (i) {
3    case 2 : std::cout << "2" << std::endl;
4    default : std::cout << "default" << std::endl;
5    case 1 : std::cout << "1" << std::endl; break;
6  }
```

=

```
1  int i; std::cin >> i;
2  if (i==2)
3    std::cout << "2" << std::endl << "default" << std::endl << "1" << std::endl;
4  else if (i==1)
5    std::cout << "1" << std::endl;
6  else
7    std::cout << "default" << std::endl << "1" << std::endl;
```

→ Tafelbild: Kontrollflussgraph

```
1 #include<iostream>
2
3 int main() {
4   goto l1;
5   std::cout << "o" << std::endl;
6 l1: return 0;
7 }
```

Wenn sie es scheinbar benötigen, dann ist es an der Zeit, den zu implementierenden Algorithmus neu zu überdenken.

Sie können immer ohne goto auskommen. Leser Ihres Quelltexts werden es Ihnen danken...

→ Tafelbild: Kontrollflussgraph

Inhalt

- **while**
- **do−while**
- **for**

Schleifen

while

```
1  int i=0;
2  while (i<10) {
3    std::cout << i;
4    if (i==7) break;
5    i++;
6    continue;
7    std::cout << i;
8  }
```

generiert Ausgabe
01234567

- Schleifenkörper wird solange wiederholt, wie getestete Schleifenbedingung gültig ist.
- 0 bis ∞ Schleifendurchläufe sind möglich.
- break verlässt die Schleife.
- continue beginnt bei Gültigkeit der Schleifenbedingung eine neue Iteration; bei Ungültigkeit wird die Schleife verlassen.
- Schachtelung und Kombination mit anderen Kontrollflussstrukturem möglich.

→ Tafelbild: Kontrollflussgraph

Fakultät: $f = n!$

```
1  // naumann@stce.rwth-aachen.de
2
3  #include<iostream>
4  #include<cassert>
5
6  int main() {
7    int n,m=1,f=1;
8    std::cin >> n;
9    assert(n>=0);
10   while (m<=n) f*=m++;
11   std::cout << f << std::endl;
12   return 0;
13 }
```

→ Optional: Inspektion mit gdb

```
1  int i=0;
2  do {
3     std::cout << i;
4     if (i==7) break;
5     i++;
6     continue;
7     std::cout << i;
8  }
9  while (i<10);
```

generiert Ausgabe
01234567

- Schleifenkörper wird solange wiederholt, wie getestete Schleifenbedingung gültig ist.
- 1 bis ∞ Schleifendurchläufe sind möglich.
- break verlässt die Schleife.
- continue beginnt bei Gültigkeit der Schleifenbedingung eine neue Iteration; bei Ungültigkeit wird die Schleife verlassen.
- Schachtelung und Kombination mit anderen Kontrollflussstrukturem möglich.

→ Tafelbild: Kontrollflussgraph

Fakultät: $f = n!$

```
1  // naumann@stce.rwth-aachen.de
2
3  #include<iostream>
4  #include<cassert>
5
6  int main() {
7    int n,m=1,f=1;
8    std::cin >> n;
9    assert(n>=0);
10   do {
11     f*=m++;
12   } while (m<=n);
13   std::cout << f << std::endl;
14   return 0;
15 }
```

→ Optional: Inspektion mit gdb

AUFWACHEN!

Konvertiere

```
1 do {
2   B
3 } while (c);
```

in eine **while**-Schleife und

```
1 while(c) {
2   B
3 }
```

in eine **do**−**while**-Schleife.

Wach?

Konvertiere

```
1 do { B } while (c);
```

in eine **while**-Schleife und

```
1 while(c) { B }
```

in eine **do**−**while**-Schleife.

```
1 B
2 while (c) { B }
```

```
1 do
2   if (c) B
3 while (c)
```

→ Tafelbild: Kontrollflussgraph

Schleifen

for


```
1 for (int i=0;i<10;i++) {
2   std::cout << i;
3   if (i==7) break;
4   continue;
5   std::cout << i; // tot ...
6 }
```

generiert Ausgabe
01234567

- Spezielle **while**-Schleife
- Kombination von Deklaration / Initialisierung des Schleifenindex mit dessen iterativer Modifikation und dem Schleifentest
- Gültigkeitsbereich des Schleifenindex ist auf Schleife beschränkt.
- break verlässt die Schleife.
- continue beginnt bei Gültigkeit der Schleifenbedingung eine neue Iteration; bei Ungültigkeit wird die Schleife verlassen.
- Schachtelung und Kombination mit anderen Kontrollflussstrukturem möglich.

→ Tafelbild: Kontrollflussgraph

Fakultät: $f = n!$

```
1  // naumann@stce.rwth-aachen.de
2
3  #include<iostream>
4  #include<cassert>
5
6  int main() {
7    int n,f=1;
8    std::cin >> n;
9    assert(n>=0);
10   for(int m=1;m<=n;m++) f*=m;
11   std::cout << f << std::endl;
12   return 0;
13 }
```

→ Optional: Inspektion mit gdb

Ist der Schleifenindex außerhalb der Schleife deklariert, d.h. nicht nur innerhalb der Schleife gültig, so behält er im Falle von **break** seinen aktuellen Wert, z.B.

```
1  int i;
2  for (i=0; i<10; i++)
3    if (i==6) break;
4  std::cout << i << std::endl;
```

generiert Ausgabe

```
6
```

continue führt zur vorzeitigen Inkremetierung des Schleifenindex gepaart mit einer Reevaluation der Schleifenbedingung. Bei Ungültigkeit letzterer wird die Schleife verlassen, z.B.

```
1  int i;
2  for (i=0; i<10; i++) {
3    if (i<9) continue;
4    std::cout << i << std::endl;
5  }
6  std::cout << i << std::endl;
```

generiert Ausgabe

```
9
10
```

→ Optional: Inspektion mit gdb

Im allgemeinen können mehrere Schleifenvariablen deklariert / initialisiert werden.

Ebenso kann die Inkrementierung eine Sequenz von durch Komma separierten Zuweisungen enthalten.

Z.B. generiert

```
1 for (int i=0,j=4; i<3; i++,j+=3)
2   std::cout << i << "<" << j << std::endl;
```

die Ausgabe

0<4
1<7
2<10

Beides trägt nicht zwingend zum besseren Verständnis des Quelltextes bei.

Inhalt

Fehlerbehandlung in C++ kann (und sollte in den meisten Fällen) durch Ausnahmen implementiert werden.

Diese werden mittels **throw** aktiviert und in einem **try−catch** Kontrollflusskonstrukt behandelt, z.B.

```
1  #include<iostream>
2
3  int main() {
4    try {
5      int i; std::cin >> i;
6      if (i<0) throw std::string("<0");
7    }
8    catch ( std::string s ) {
9      std::cout << s << std::endl;
10   }
11   return 0;
12 }
```

Ausnahmen können beliebige Typen haben, hier z.B. Zeichenketten der Standardbibliothek (std::string).

Zusammenfassung

Bedingungen und Bedingte Zuweisung

Verzweigungen

- if[-else]
- switch-case
- goto

Schleifen

- while
- do-while
- for

Ausnahmen