

C++-Crashkurs

Matthias Huck, Markus Freitag
{huck,freitag}@i6.informatik.rwth-aachen.de

Softwareprojektpraktikum Maschinelle Übersetzung SS 2011

Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik 6
Computer Science Department
RWTH Aachen University, Germany

Aufbau eines C++-Programms

Beispiel:

helloworld.cc

```
1 #include <string>
2 #include <iostream>
3
4 // hello world
5 int main(int argc, char **argv) {
6     std::string str("Hello World");
7     std::cout << str << std::endl;
8     return 0;
9 }
```

Kompilieren: `g++ helloworld.cc -o helloworld`

Bemerkungen: <code>#include</code>	Einbinden von Headerdateien
<code>argv</code>	Kommandozeilenargumente
<code>::</code>	Bereichsoperator
<code><<</code>	Ausgabeoperator

Benutzerdefinierte Datentypen

charstack.h

```

1 // Schnittstelle
2 #ifndef __CHARSTACK_H__
3 #define __CHARSTACK_H__
4
5 class charStack {
6     private:
7         char *s;
8         unsigned int max;
9         unsigned int num;
10
11     public:
12         charStack(unsigned int n);
13         ~charStack();
14         void push(char elem);
15         char pop();
16         bool empty();
17 };
18
19 #endif

```

charstack.cc

```

1 // Implementierung
2 #include "charstack.h"
3
4 charStack::charStack(unsigned int n) {
5     s = new char[max=n];
6     num = 0;
7 }
8
9 charStack::~~charStack() {
10     delete [] s;
11 }
12
13 void charStack::push(char elem) {
14     s[num++] = elem;
15 }
16
17 char charStack::pop() {
18     return s[--num];
19 }
20
21 bool charStack::empty() {
22     return num == 0;
23 }

```

Getrennte Übersetzung

main.cc

```

1 #include <iostream>
2 #include "charstack.h"
3
4 int main() {
5     charStack s(100);
6     s.push('6');
7     s.push('i');
8     while ( !s.empty() ) {
9         std::cout << s.pop();
10    }
11    std::cout << std::endl;
12    return 0;
13 }

```

Makefile

```

1 #!/usr/bin/gmake
2
3 CC      :=      g++
4 LN      :=      g++
5
6 OBJ     =      main.o charstack.o
7
8 charstack:    $(OBJ)
9               $(LN) -o charstack $(OBJ)
10
11 %.o: %.cc
12           $(CC) -c $(CFLAGS) $< -o $@

```

► charstack.cc **und** main.cc **werden getrennt kompiliert:**

► g++ -c charstack.cc -o charstack.o

► g++ -c main.cc -o main.o

► **Objektdateien** charstack.o **und** main.o **werden zu einem ausführbaren Programm gelinkt:**

► g++ -o charstack charstack.o main.o

Zeiger und Adressen

```
#include <iostream>

int main() {
    int a, b;
    int *ptr;    // Zeiger auf int

    a = 1000;
    ptr = &a;    // speichert die Adresse von a in ptr
    b = *ptr;    // dereferenziert ptr und weist Inhalt b zu
    *ptr = 2000; // ändert den Wert von a

    printf("b: %d \n", b);    // 1000
    printf("a: %d \n", a);    // 2000
    return 0;
}
```

► & ist der Adressoperator

► * ist der Dereferenzierungsoperator

Argumentübergabe (1)

call by value

```
void noswap(int x,
            int y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

```
void e() {
    int x = 1, y = 2;
    noswap(x, y);
    // x == 1; y == 2;
}
```

- ▶ jedes formale Argument wird durch das korrespondierende aktuelle Argumente initialisiert
- ▶ die Methode manipuliert die Werte der lokalen Kopien

call by reference

```
void swap(int &x,
          int &y) {
    int tmp = x;
    x = y;
    y = tmp;
}
```

```
void f() {
    int x = 1, y = 2;
    swap(x, y);
    // x == 2; y == 1;
}
```

- ▶ die Referenz ist ein alternativer Name für ein Objekt
- ▶ die Methode manipuliert die Werte der beim Aufruf übergebenen Variablen

Argumentübergabe (2)

konstante Referenz

```
int swap1(const int &x,  
          int &y) {  
    int tmp = y;  
    y = x;  
    return tmp;  
}
```

```
void g() {  
    int x = 1, y = 2;  
    x = swap1(x, y);  
    // x == 2; y == 1;  
}
```

- der Wert eines als konstante Referenz übergebenen Arguments kann durch die Methode nicht geändert werden

Zeiger als Funktionsargumente

```
void swap2(int *x,  
           int *y) {  
    int tmp = *x;  
    *x = *y;  
    *y = tmp;  
}
```

```
void h() {  
    int x = 1, y = 2;  
    swap2(&x, &y);  
    // x == 2; y == 1;  
}
```

Dynamischer Speicher

```
struct Person {  
    std::string name_;  
    unsigned int alter_;  
    Person(const std::string &name, const unsigned int &alter) :  
        name_(name), alter_(alter) {}  
};  
  
int main() {  
    Person *p = new Person("Niklaus Wirth", 76);  
    printf("%d \n", p->alter_);    // 76  
    delete p;  
}
```

- ▶ `new` erzeugt Objekt im dynamischen Speicher (Heap)
- ▶ ein mit `new` erzeugtes Objekt existiert unabhängig von dem Gültigkeitsbereich, in dem es angelegt wurde, so lange, bis es explizit mit `delete` zerstört wird
- ▶ keine automatische Speicherbereinigung (Garbage Collection) in C++

Generische Programmierung (1)

stack.h

```

1 // Schnittstelle
2 #ifndef __STACK_H__
3 #define __STACK_H__
4
5 template <class T> class stack {
6     private:
7         T *s;
8         unsigned int max;
9         unsigned int num;
10
11     public:
12         stack(unsigned int n);
13         ~stack();
14         void push(T elem);
15         T pop();
16         bool empty();
17 };
18 #endif

```

Der Template-Mechanismus ermöglicht das Angeben von Typen als Parameter für eine Klasse oder Funktion

stack.cc

```

1 // Implementierung
2 #include "stack.h"
3
4 template <class T> stack<T>::stack(unsigned int n) {
5     s = new T[max=n];
6     num = 0;
7 }
8
9 template <class T> stack<T>::~~stack() {
10     delete [] s;
11 }
12
13 template <class T> void stack<T>::push(T elem) {
14     s[num++] = elem;
15 }
16
17 template <class T> T stack<T>::pop() {
18     return s[--num];
19 }
20
21 template <class T> bool stack<T>::empty() {
22     return num == 0;
23 }

```

Generische Programmierung (2)

main.cc

```

1 #include <iostream>
2 #include "stack.cc"
3
4 int main() {
5     stack<char> s1(100);
6     stack<int> s2(100);
7     stack<stack<int>*> s3(100);
8     s1.push('i');
9     s2.push(2);
10    s2.push(4);
11    s3.push(&s2);
12    std::cout << s1.pop();
13    std::cout << s3.pop()->pop();
14    std::cout << s2.pop() << std::endl;
15    return 0;
16 }
```

- Template-Instanziierung erzeugt eine Klassendeklaration aus einer Template-Klasse und einem Template-Parameter

- Der Compiler generiert die benötigten Spezialisierungen, in diesem Fall `stack<char>`, `stack<int>` und `stack<stack<int>*>`
- Dazu muss die komplette Implementierung eingebunden werden: `#include "stack.cc"`
- 1. Alternative: Sowohl Schnittstelle als auch Implementierung in der Headerdatei
- 2. Alternative: Angabe aller erforderlichen Spezialisierungen in `stack.cc`

Template-Instanziierung (stack.cc)

```

template class stack<char>;
template class stack<int>;
template class stack<stack<int>*>;
```

Standard Template Library (STL)

Die C++-Standardbibliothek stellt eine Vielzahl parametrisierbarer Container-
datentypen und generischer Algorithmen bereit.

Container:

<code>vector<T></code>	eindimensionales Feld
<code>list<T></code>	doppelt verkettete Liste
<code>queue<T></code>	Warteschlange (FIFO)
<code>stack<T></code>	Keller (LIFO)
<code>map<key, value></code>	assoziatives Feld
<code>set<T></code>	Menge

außerdem: `dqueue`, `priority_queue`, `multimap`, `multiset`

Algorithmen:

<code>for_each()</code>	für jedes Element eine Operation ausführen
<code>find()</code>	Auftreten eines Wertes finden
<code>reverse()</code>	die Reihenfolge der Elemente umkehren
<code>sort()</code>	Elemente sortieren
<code>min_element()</code>	das kleinste Element einer Sequenz liefern
<code>max_element()</code>	das größte Element einer Sequenz liefern

und viele mehr ...

Container und Iteratoren

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4 #include <map>
5 #include <algorithm>
6
7 using namespace std;
8
9 int main() {
10     vector<int> v(10, 7);           // Vektor mit 10 int-Elementen, alle initialisiert mit 7
11     v[9] = 5;                      // Neubelegung des Elements an der letzten Position
12     reverse(v.begin(), v.end());   // Umkehrung der Reihenfolge der Elemente
13     for (vector<int>::iterator i=v.begin(); i!=v.end(); ++i) {
14         if ( *i == 5 ) *i = 42;
15         cout << *i << endl;        // Zugriff auf Elemente mittels des Iterators
16     }
17
18     map<string, int> m;              // assoziatives Feld, das string-Schlüssel auf int-Werte abbildet
19     m["Wirth"] = 76;                // Hinzufügen von Schlüssel-Wert-Paaren
20     m["Knuth"] = 72;
21     int age = m["Wirth"];           // Zugriff auf einen Wert über den Schlüssel
22     for (map<string, int>::const_iterator ci=m.begin(); ci!=m.end(); ++ci) {
23         cout << ci->first << ": " << ci->second << endl;
24     }
25 }
```

Ein- und Ausgabe mit Streams

```

1  #include <iostream>
2  #include <string>
3  #include <fstream>
4  #include <sstream>
5
6  using namespace std;
7
8  int main(int argc, char **argv) {
9
10     if (argc != 2) {
11         cout << "usage: " << argv[0]
12             << " <inputfile>" << endl;
13         return EXIT_FAILURE;
14     }
15
16     char *in_filename=argv[1];
17
18     ifstream f_in(in_filename);
19     if ( !f_in.good() ) {
20         cerr << "ERROR: Opening file '"
21             << in_filename << "' failed."
22             << endl;
23         return EXIT_FAILURE;
24     }
25     string line, token;
26
27     while ( getline(f_in,line) ) {
28         istringstream ist(line);
29         cout << "<line>" << endl;
30         while ( ist >> token ) {
31             cout << token << endl;
32         }
33         cout << "</line>" << endl;
34     }
35
36     if ( !f_in.eof() ) {
37         cerr << "ERROR: Reading file '"
38             << in_filename << "' failed."
39             << endl;
40         return EXIT_FAILURE;
41     }
42     f_in.close();
43
44     return EXIT_SUCCESS;
45 }

```

Literatur

Bücher:

B. Stroustrup The C++ Programming Language, Addison-Wesley

A. Alexandrescu Modern C++ Design, Addison-Wesley

S. Lippman et al. C++ Primer, Addison-Wesley

S. Meyers Effective C++, Addison-Wesley

S. Meyers More Effective C++, Addison-Wesley

N. M. Josuttis The C++ Standard Library, Addison-Wesley

B. Eckel Thinking in C++, Web-Download:

<http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html>

Web:

<http://www.cppreference.com/>

