Übungspaket 5

Objektmodellierung in C++

Übungsziele:

1. Modellieren eines Objektes in C++
2. Implementieren einer Klasse in C++

Literatur:

C++-Skript[1], Kapitel 19

Semester:

Wintersemester 2020/21

Betreuer:

Theo und Ralf

Synopsis:

In diesem Übungspaket werden wir uns mit der Modellierung von Objekten in der Programmiersprache C++ beschäftigen. Dabei werden anfangs die einzelnen syntaktischen Elemente der Sprache C++ wiederholen. Im Anwendungsteil werden wir wiederunm das Problem eines Eingabepuffers mit „unbegrenzter“ Länge bearbeiten. Diesmal werden wir aber ein C++ Programm entwickeln.

[1] www.amd.e-technik.uni-rostock.de/ma/rs/lv/hoqt/script.pdf

Teil I: Stoffwiederholung

Aufgabe 1: Das Klassenkonzept in C++

Wie (Schlüsselwort) beginnt eine Klasse?	`class`
Wie definiert man eine Klasse?	`class <name> { attribute/methoden };`
Welche Zugriffsmodifikatoren kennst du?	`private` und `public`
Wie spezifiziert „man“ Attribute?	`private`
Wie spezifiziert „man“ Methoden?	meist `public`, manchmal auch `private`
Wie definiert man Objekte?	`class-name object_1, object_2, ... ;`
Wie heißt der Konstruktor?	Wie die Klasse
Wie heißt der Konstructor der Klasse `C`?	`C()`
Wie heißt der Destruktor?	Wie die Klasse aber mit vorangestellter „`~`“
Wie heißt der Destructor der Klasse `C`?	`~C()`
Welchen Präfix erhält eine Methode bei ihrer Implementierung?	Klassenname gefolgt von „`::`“
Notiere ein Beispiel	`class C: int C::m() { ... }`
Wie ruft man eine Methode auf?	`obj.methode()` bzw. `obj->methode()`
Notiere ein Beispiel	`o.m()` bzw. `o->m()`
Wie kann eine Methode auf die Attribute ihres Objektes zugreifen?	Mittels des `this`-Zeigers
Wer erzeugt diesen Zeiger?	Der Compiler, er übergibt: `& o` → `this`
Wie erzeugt man Objekte dynamisch?	`CLASS *p = new CLASS (....)`
Warum nimmt man nicht `malloc()`?	Sonst wird der Konstruktor nicht aufgerufen
Wie gibt man diese Objekte wieder frei?	`delete objekt_zeiger`

Aufgabe 2: Methodenaufruf

Erkläre mit eigenen Worten, wie in `C++` eine Methoden aufgerufen wird. Erläuter dabei die Begriffe *Objekt*, *Methode* und `this`-*Zeiger*.

Durch die Klassendefinition besteht ein Objekt aus verschiedenen Attributen. Zusätzlich hat das Objekt Zugriff auf verschiedene Methoden, die ebenfalls durch die Klassendefinition festgelegt sind. Den Aufruf `obj.m()` einer Methode `m()` im Kontext des Objektes `obj` wandelt der Compiler selbstständig wie folgt um: `G++: obj.m( & obj )` Dieser erste Parameter '`& obj`' ist innerhalb der Methode `m()` mittels des `this`-Zeigers verwendbar.

Teil II: Quiz

Aufgabe 1: Attribute, Methoden, Gültigkeitsbereiche

Für die anstehenden Quizzaufgaben betrachten wir folgendes C++ Programm:

```
 1 #include <stdio.h>
 2
 3 class C {
 4           private:
 5              int i;
 6              int j;
 7           public:
 8              void print();
 9              void set( int i, int b );
10              C( int a );
11         };
12
13 C::C( int a )
14    {
15       i = a; j = a + 4;
16    }
17
18 void C::set( int i, int b )
19      {
20         i = j + b;
21         this->i = i; this->j = b - 1;
22      }
23
24 void C::print()
25      {
26         printf( "i=%2d j=%2d\n", i, j );
27      }
28
29 int main( int argc, char **argv )
30     {
31        C obj( 2 );
32        obj.print();
33        obj.set( 3, 4 );
34        obj.print();
35        return 0;
36     }
```

Schau dir das Programm gut an und versuche zu verstehen, wie es abgearbeitet wird.

Beantworte die folgenden Fragen zu obigem Programm.

Welche Attribute sind in `C` definiert?	`i` und `j`
Welche Methoden sind in `C` definiert?	`print()`, `set()` und `C`
Welche Elemente in `C` sind `private`?	Die Attribute `i` und `j`
Elemente in `C` sind `public`?	Die Methoden `print()`, `set()` und `C`
Welche Werte ahben `i` und `j` nach Zeile `31`?	`obj.i =  2, obj.j = 6`
Welche Werte ahben `i` und `j` nach Zeile `33`?	`obj.i = 10, obj.j = 3`
Kann man in `main()` auf `i` und `j` zugreifen?	Nein
Warum ist dies so	Sie sind als `private` spezifiziert
Kann man in `main()` die Methoden aufrufen?	Ja
Warum ist dies so	Sie sind als `public` spezifiziert
Was ist `C( int a )`?	Das ist der Konstruktor

Teil III: Fehlersuche

Aufgabe 1: Praktische Fehlersuche

Dr. Class hat gerade angefangen, in C++ zu programmieren. Leider hat er hier und da noch ein paar Tippfehler, bei deren Korrektur wir ihm helfen sollten. Sein erstes Programm sieht wie folgt aus:

```
 1 #include <stdio.h>
 2
 3 cLass C {
 4           private
 5              int i;
 6           Public:
 7              void print();
 8              void C( int a );
 9         } My_Class;
10
11 C( int a )
12      {
13         (*this).i = a;
14      }
15
16 void C: :print()
17      {
18         printf( "i=%2d\n", this.i );
19      }
20
21 int main( int argc, char **argv )
22     {
23        CC obj( 2 );
24        obj.print();
25        printf( "i = %d\n", obj.i );
26        return 0;
27     }
```

Zeile	Fehler	Erläuterung	Korrektur
3	`cLass`	„`class`“ schreibt sich in Kleinbuchstaben.	`class`
4	`private`	Hinter `private` muss ein Doppelpunkt „`:`“ folgen.	`private:`
6	`Public:`	„`public`“ schreibt sich in Kleinbuchstaben.	`public:`

Zeile	Fehler	Erläuterung	Korrektur
8	`void C(int a)`	Der Konstruktor `C()` hat *keinen* (Rückgabe-) Typ.	`C( int a )`
11	Klassenname fehlt	Bei der Implementierung müssen vor dem Methodennamen der Klassenname nebst zweier Doppelpunkte stehen.	`C::C(...)`
16	`: :`	Die beiden Doppelpunkte sind eine lexikalische Einheit und müssen zusammen geschrieben werden.	`::`
18	`this.i`	`this` ist ein Zeiger. Der Zugriff auf die Elemente erfolgt mittels „`->`“.	`this->i`
23	`class CC`	Der Klassenname lautet `C`, ohne „`class`“.	`C`
25	`obj.i`	Da das Attribut `i` `private` spezifiziert ist, kann `main()` nicht darauf zugreifen.	

Programm mit Korrekturen:

```
 1 #include <stdio.h>
 2
 3 class C {
 4           private:
 5              int i;
 6           public:
 7              void print();
 8              C( int a );
 9         };
10
11 C::C( int a )
12      {
13         (*this).i = a;
14      }
15
16 void C::print()
17      {
18         printf( "i=%2d\n", this->i );
19      }
20
21 int main( int argc, char **argv )
22     {
23        C obj( 2 );
24        obj.print();
25        return 0;
26     }
```

Teil IV: Anwendungen

In diesem Anwendungsteil löse wir die selbe Aufgabe, die wir bereits in Übungspaket 4 gelöst haben. Dabei ging es um die Implementierung eines Puffers, der sich bei Bedarf vergrößert und somit nicht überlaufen kann. Diesmal entwickeln wir allerdings eine C++-Lösung.

Aufgabe 1: Eingabepuffer beliebiger Länge in C++

1. Aufgabenstellung

An dieser Stelle verweisen wir auf Übungspaket 4, Aufgabe 1 des Anwendnugsteils, da wir dort die Aufgabenstellung bereits detailiert besprochen haben.

2. Entwurf

Im ersten Entwurfsschritt müssen wir definieren, welche Attribute und Methoden unsere C++-Klasse `CBUF` haben soll. Diese Klassendefinition werden wir in der Datei `cbuf.h` ablegen. Die Implementierung aller Methoden werden wir wie üblich in der korrespondiedenden Datei `cbuf.cpp` plazieren.

Wie im Skript beschrieben, werden sollten wir die Attribute `private` und die Methoden `public` spezifizieren. Die oben erwähnte C-Realisierung stellt einen guten Ausgangspunkt für diese Teilaufgabe dar:

3. Kodierung

Klassendefinition in `cbuf.h`:

```
 1  class CBUF {
 2          private:
 3             int len;                  // length incl. '\0'
 4             int size;                 // total size of buf
 5             char *buf;                // the acutal buffer
 6
 7          public:
 8              CBUF ();                 // constructor
 9             ~CBUF ();                 // destructor
10             void  cb_reset ();
11             char *cb_addc  ( char c );
12             char *cb_addstr( const char *str );
13             char *cb_buf   ();
14         };
```

Klassenimplementierung in `cbuf.cpp` **(sehr kompakt gesetzt):**

```
 1 #include <stdio.h>       // fprintf()
 2 #include <stdlib.h>      // exit()
 3 #include <string.h>      // memcpy()
 4 #include "cbuf.h"
 5 #define INCR    128      // extending increment of buf
 6
 7 static void test_memory( void *p, const char *fname )
 8        {
 9           if ( p )
10              return ;            // everything all right
11           fprintf(stderr, "%s: out of memory\n", fname);
12           exit( 1 );             // exit program
13        }
14
15 CBUF::CBUF()
16        {
17           test_memory(buf = new char[size=INCR], "CBUF");
18           this->cb_reset();
19        }
20 CBUF::~CBUF(){ delete this->buf; }
21
22 void  CBUF::cb_reset(){ this->buf[this->len=0] = '\0'; }
23 char *CBUF::cb_buf(){ return this->buf; }
24
25 char *CBUF::cb_addc( char c )
26      {
27         if ( this->len + 1 >= this->size )
28         {
29            char *new_buf = new char [this->size + INCR];
30            test_memory( new_buf, "CBUF::cb_addc()" );
31            memcpy( new_buf, buf, len + 1 ); // incl. '\0'
32            delete buf; size += INCR; buf = new_buf;
33         }
34         buf[ len++ ] = c; buf[ len ] = '\0';
35         return this->buf;
36      }
37
38 char *CBUF::cb_addstr( const char *str )
39      {
40         while( *str )
41            this->cb_addc( *str++ );
42         return this->buf;
43      }
```

Ein Testprogramm `main.cpp`**:**

```
 1 #include <stdio.h>       // printf()
 2
 3 #include "cbuf.h"        // class definition
 4
 5 int main( int argc, char **argv )
 6     {
 7         CBUF *p = new CBUF;
 8         p->cb_addc( 'h' );
 9         p->cb_addc( 'i' );
10         p->cb_addstr( " you" );
11         p->cb_addc( '!' );
12         printf( "buf= '%s'\n", p->cb_buf() );
13         p->cb_reset();
14         printf( "buf= '%s'\n", p->cb_addstr( "servus" ) );
15     }
Ausgabe: buf= 'hi you!'
         buf= 'servus'
```