THINKING IN C++, VOL 1, CHAPTER 16 AND VOL 2, CHAPTER 1

TOM OLSSON

Innehåll

Date: 2013-08-31 15:04:52.

1. Bibliotek med mallklasser, går det att dölja källkoden.

cpp_template_class_05.aw2

Vad gäller i generellt för mall-kod, är det möjligt att komplett gömma källkoden för ett bibliotek skrivit med hjälp av mallar (templates)? Jämför med vanliga biblioteksfunktioner som sparas kompilerad i ett bibliotek.

a) Ja, det fungerar alltid.
b) Nej, det går alldrig."
c) Ja, fast om export-egenskapen används."

2. Läsning av en fil

cpp_io_02.aw2

Om du vill öppna en fil för att LÄSA denna, vilken klass skall du då använda?

a) istringstream"
b) ostringstream
c) istream"
d) ofstream"
e) ostream"
f) ifstream

3. Instansiering av mallen

cpp_template_class_06.aw2

När instansieras vanligtvis en mall?

a) Vid exekveringen.
b) Vid länkningen.
c) Vid kompileringstillfället.

4. När är multipelt arv ej lämpligt?

cpp_multi_inheritance_01.aw2

Under vilka villkor rekommenderas ej multipelt arv?

a) När två eller flera klasser har samma virtuella funktionsnamn.
b) Vet ej.
c) När mer än ett objekt kan härledas från CObject
d) När en klass innehåller ett objekt av samma typ som den kan härledas från.

5. Vilken är illegal?

cpp_template_class_01.aw2

Vilken av följande är illegal?

a) template <class T, class S> int func(T x, S y)
b) template <class T> T func(T x)
c) template <class T> func(T x)
d) template <class T> int func(T x)

6. Polymorfism

cpp_template_class_04.aw2

Ideen med mallfunktioner/mallklasser är detta på något sätt relaterat till polymorfism? Ideen bakom polymorfism är: Ett gränssnitt, multipla metoder.

a) Ja, polymorfism vid kompileringstillfället.
b) Endast om malltyperna är objekt.
c) Nej

7. TICPP volym1 sidan 723-784: Introduction to templates

ticpp_ch16_01.aw2

Svara med R för rätt/sant och F för fel/falskt.

a) Mallklasser innebär att den totala kodmängden blir mindre.
b) Ett problem med behållarklasser är vem som äger objekten i behållaren.
c) Mallklasser är ett sätt att återanvända objektkod."
d) Behållarklasser är en väsentlig del av OOP och ersätter t ill stora delar användning av den primitiva array-klassen.
e) Ett iterator-objekt skapas för att traversera en behållare med objekt.
f) En objekt av en iteratorklass används för att hålla reda på totala antalet objekt i behållaren.
g) En funktionsmall används för att implementera en generisk algoritm.
h) Ett iterator-objekt använder minne proportionellt mot antalet objekt i behållaren.
i) Ett sätt att kunna implementera behållarklasser i språk som SmallTalk och Java är att låta alla object ärva egenskaperna från en basklass Object.
j) Nyckelordet template anger för kompilatorn att en eller flera ospecificerade typer kommer att användas.
k) Kompilatorn allokerar minne för en mallklass vid instantieringen av denna (ett objekt skapas).
l) Ett viktigt tillämpningsområde för mall-klasser (templates) är så kallade behållarklasser (container classes).

8. När krävs mallparametrar?

cpp_template_class_03.aw2

När måste en mallfunktion (template function) uttryckligen ha mall-parametrar?

a) Alltid.
b) När mallens typer inte kan härledas.
c) Aldrig, mallens typer kan alltid härledas.

9. Mallfunktionen print3

cpp_template_functions_02.aw2

Kod:

```
#include <iostream>
using namespace std;

class myClass
{
        //
};

int main()
{
        int a,b,c;
        double e,f,g;
        myClass obj1, obj2, obj3;

        //
        print(a,b,c);
        print(a,e,obj1);
}
```

Vilken av nedanstående mall-funktioner skulle fungera med ovanstående program?

a)

```
template <typename T1, typename T2, typename T3>
void print3(T1 x, T2 y, T3 z)
{
        cout << x << endl;
        cout << y << endl;
        cout << z << endl;
}
```

b) Ingen av alternativenav de övriga alternativen fungerar.

c)
```
template <typename T1, typename T2>
void print3(T1 x, T1 y, T2 z)
{
        cout << x << endl;
        cout << y << endl;
        cout << z << endl;
}
```

d)
```
template <typename T>
void print3(T x, T y, T z)
{
        cout << x << endl;
        cout << y << endl;
        cout << z << endl;
}
```

10. Minnestorlek

cpp_template_class_07.aw2

Vilket av följande beskriver ett potentiellt resultat av att använda mallar?

a) Långsammare program"
b) Namngivningen i debuggern är ganska dålig.
c) Storleken på den exekverbara filen ökar relativt kodbasen.

11. Vad är multipelt arv?

cpp_multi_inheritance_02.aw2

Multipelt arv kan bäst beskrivas som:

a) En klass som kan härledas från två eller flera klasser.
b) Alla de övriga är fel.
c) Multipelt härledda klasser i en lång kedja från basklassen, arv i fler än ett led.
d) En klass som kan härledas från en eller flera klasser.

12. Kast av ett heltal

cpp_exception_01.aw2

Kod:

```
#include <iostream>
using namespace std;

void g( int &i )
{
        if ( i < 0 )
                throw 99; //Error Code
        else
                i=i+4;;
}

void f(int &x)
{
        x--;
        g(x);
}

int main ()
{
        int j=0;

          try
          {
                f(j);
                cout << "j=" << j << endl;
          }
          catch (int codenumber)
          {
                cout << "Error: " << codenumber << endl;
          }
}
```

Vad kommer att skrivas ut då ovanstående program exekveras?

a) `Error: 99`
`j=-1`

b) Error: 99

c) j=0

d) j=4
e) j=3

13. TICPP volym2: Exceptions

ticpp_v2_ch01_01.aw2

Svara med R för rätt/sant och F för fel/falskt.

a) Det är bättre att fånga ett undantag genom ett värde iställer för via en referens.
b) Det går inte att skriva någon egen terminate()-funktion.
c) Klasserna logic_error och runtime_error kan härledas till basklassen exception.
d) Ett undantag behöver ej fångas om man inte vill det.
e) Det är alltid bättre att skapa en egen undantagsklass som ej kan härledas till basklassen exception.
f) En god regel är att använda standard-undantagsklasser där det är relevant istället för att skapa egna undantagsklasser.
g) Att förbättra möjligheterna att återhämta sig från fel som uppstår under exekveringen är ett av de mest kraftfulla sätten att nå en ökad robusthet för ett program.
h) Ett undantag bör generellt vara en sällsynt händelse.
i) Att kasta ett undantag är ett sätt att ta hand om felet där det uppstår.
j) Klassen Out_of_range kan härledas till klassen runtime_error.
k) Med catch(...) fångas alla typer av undantag.
l) En fördel med undantagshantering är att det separerar problemlösningen ifrån fehanteringen.
m) Undantag som sker i en konstruktor är ej helt triviala att hantera om den gör någon form av resursallokering.
n) En god regel är att ej kasta ett undantag i en destruktor.
o) Undantagshantering är svaret på alla felhanteringsproblem.
p) Ett undantag som ej fångas på någon nivå kommer att kommer att göra att biblioteksfunktionen terminate() anropas automatiskt.

14. Division med 0

cpp_exception_02.aw2

Kod:

```
#include <iostream>
#include <string>
using namespace std;
```

```
class DivZero
{
        public:
                DivZero() { message="Division by Zero!";}
                void print() const { cout << message << endl
        private:
                string message;
};

int divide (int top, int bottom)
{
          if (bottom == 0)
                  throw DivZero();
          return top/bottom;
}

int main(int argc, char* argv[])
{
          try
          {
                  cout << divide (3,0) << endl;
                  cout << divide (4,2) << endl;
          }
          catch (DivZero error)
          {
                    error.print();
          }
}
```

Vad blir utskriften då ovanstående program exekveras?

a) Division by Zero!

b)
```
inf
2
```

c)
```
Division by Zero!
2
```

15. Skrivning till en fil

cpp_io_01.aw2

Om du vill öppna en fil för att SKRIVA till denna, vilken klass skall du då använda?

a) istream"
b) istringstream"
c) ostream"
d) ostringstream
e) ofstream "
f) ifstream"