C++
för den som tror sej kunna C++

Sven Rosvall
sven-e@lysator.liu.se

Programming Research Ltd

- Hjälper utvecklare med bättre kvalitet billigare.
- Utvecklingsprocesser.
  Fokus på kodinspektion.
- Verktyg för statisk analys av kod.

`http://www.prqa.co.uk`

Innehåll ikväll

- Standard C++
- Vanliga och ovanliga fel
- Styrkor
- Effektivitet
- Praktisk användning av C++

Standard C++

- Standarden antagen Juni 1998
  ISO/IEC 14882
- Arbetet idag

Vanliga och ovanliga fel

- Resurser
- Polymorfism
- Exceptions
- Konverteringar

Resurser

- Dynamiskt minne, fil-handtag, ...
- Ofta problem med vem som ansvarar för skapande och borttagande.
- Mycket tid går åt att spåra resurser som hanteras fel.

Exempel: class String

```
class String {
public:
  String(const char * = 0);
private:
  char * rep;
};

String::String(const char *value)
{
  rep = new char[strlen(value)+1] ;
  strcpy(rep, value) ;
}
```

Exempel: class String

```
class String {
public:
  String(const char * = 0);
  ~String();
private:
  char * rep;
};

String::~String()
{
  delete [] rep;
}
```

Exempel: class String

```
class String {
public:
  String(const char * = 0);
  ~String();
  operator const char *() const;
private:
  char * rep;
};

String::operator const char *() const
{
  return rep;
}
```

Exempel: class String

```
void doNothing(String a) { }
void work1()
{
  String s(“hej”);
  doNothing(s);
  std::cout << s << std::endl;      // ???
}

String doHello() { return “hej”; }
void work2()
{
  String s;
  s = doHello();
  std::cout << s << std::endl;      // ???
}
```

Exempel: class String

```
class String {
public:
...
 String(const String &);
 const String & operator=(const String &);
private:
  char * rep;
};

String::String(const String &value)
{
  rep = new char[strlen(value.rep)+1] ;
  strcpy(rep, value.rep) ;
}
```

Exempel: class String

```
class String {
public:
...
  String(const String &);
  const String & operator=(const String &);
private:
  char * rep;
};

const String & String::operator=(const String &value)
{
  delete [] rep;
  rep = new char[strlen(value.rep)+1] ;
  strcpy(rep, value.rep) ;
  return *this;
}
```

Exempel: class String

```
void work()
{
  String a(“hej”);
  a = a;                // Bang!
}
```

Exempel: class String

```
const String & String::operator=(const String &value)
{
  if (&value != this) {
    delete [] rep;
    rep = new char[strlen(value.rep)] ;
    strcpy(rep, value.rep) ;
  }
  return *this;
}
```

Binära operatorer

```
String a(“hej”);
String b(“ världen”);
String c;

c = a + b ;
```

Binära operatorer

```
class String {
public:
...
  const String & operator+=(const String&);
  String operator+(const String&);
...
};

String String::operator+(const String &right)
{
  String temp(*this);
  temp += right;
  return temp;
}
```

Binära operatorer

```
class String {
public:
...
  const String & operator+=(const String&);
  String operator+(const String&);
...
};

const String & String::operator+=(const String &right)
{
  char* temp = new char[strlen(rep)+strlen(right.rep)+1];
  strcpy(temp, rep);
  strcat(temp, right);
  delete rep;
  rep = temp;
  return *this;
}
```

Binära operatorer

```
String a(“hej”);
String b(“ världen”);
String c;

c = a + b;          // funkar fint!
c = a + “ alla”;    // funkar fint!
c = “hela” + b;     // funkar inte!
```

Binära operatorer

```
class String {
public:
...
  const String & operator+=(const String&);
...
};
String operator+(const string &, const String&);

String operator+(const String &left, const String &right)
{
  String temp(left);
  temp += right;
  return temp;
}
```

Använd std::string istället för char*

- Slipper tänka på dynamiskt minne
- Slipper tänka på tillräckligt utrymme

Dynamiskt minne

- Hur länge finns objektet?
- Vem tar bort objektet?
- Vem mer kan tänkas vara intresserad av objektet?

std::auto_ptr

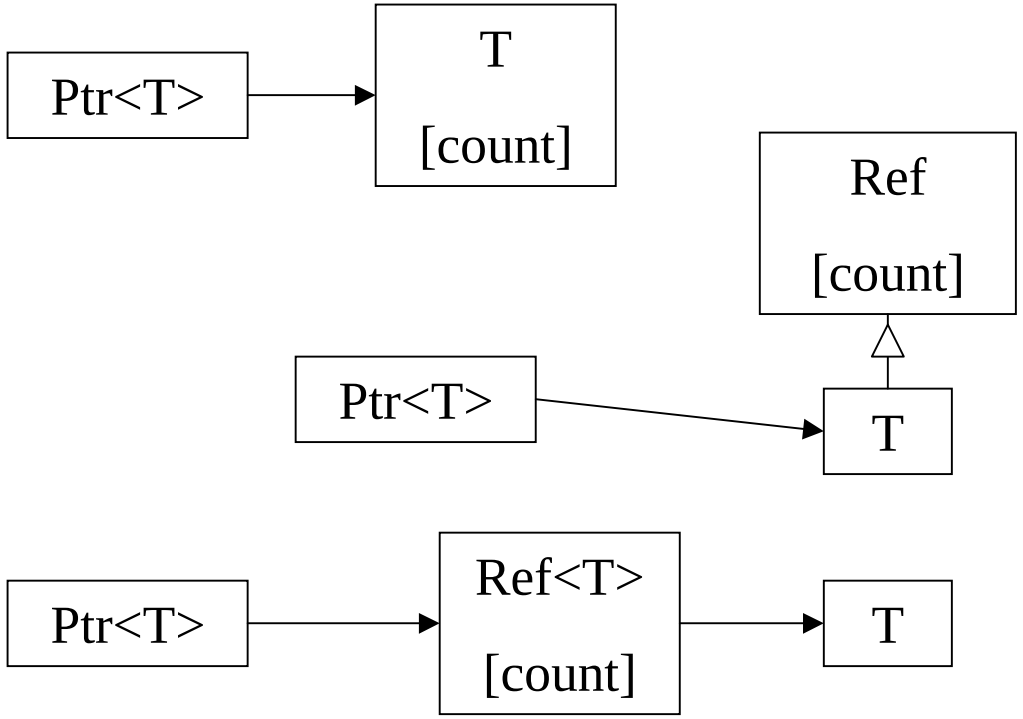
- Ansvarar för ett dynamiskt objekt
- När auto_ptr går försvinner städar den bort sitt objekt.

```
#include <memory>

void f()
{
  std::auto_ptr<char> p = new char ;
  *p = ‘x’ ;
  std::cout << “x=“ << *p << std::endl;
}     // p is deleted.
```

Referensräknande pekare

- Flera pekare till ett objekt
- Ingår tyvärr inte i C++-biblioteket
- Ett antal varianter:



Begränsningar

- Cirkulära listor.
- Måste fortfarande tänka på ägarskap.
- Ibland lever en rå pekare eller referens kvar.

Polymorfism

- Ett antal olika typer behandlas lika.
- Gemensamt interface.
- Kräver access via pekare eller referens
- Kräver omtanke i design

Polymorfism

```
class B {
public:
  B() ;
  virtual void dosomething() const;
};

class D : public B {
public:
  D(string) ;
  virtual void dosomething() const;
private:
  string s;
};
```

Polymorfism via handtag

```
void f(B b)        { b.dosomething(); }
void g(const B &b) { b.dosomething(); }

D d(“hello”);
f(d);           // pass by value => Slicing!
g(d);           // pass by reference => OK.
```

Polymorfism via handtag

```
B *bp = new D(2);  // create D object
delete bp;         // destroy object
```

Polymorfism via handtag

```
B *bp1 = new D;
B *bp2 = new D;
*bp2 = *bp1 ;       // copy D objects?
```

Polymorfism

```
class B {
public:
  B() ;
  virtual ~B() ;
  virtual void dosomething() const;
protected:
  const B & operator=(const B &);
};
class D : public B {
public:
  D(string) ;
  virtual ~D() ;
  virtual void dosomething() const;
  const D & operator=(const D &);
private:
  string s;
};
```

Exceptions

- Används för felhantering
- Kan dock ställa till med problem

Exempel: exceptions

```
void f()
{
  char *p = new char;
  *p = ‘x’;
  std::cout << “x=“ << *p << std::endl;
  delete p;
}
```

Exempel: exceptions

```
void f()
{
  char *p = new char;
  *p = ‘x’;
  try {
    std::cout << “x=“ << *p << std::endl;’
  } catch (...) {
    delete p;
    throw;
  }
  delete p;
}
```

Exempel: exceptions

```
#include <memory>

void f()
{
  std::auto_ptr<char> p = new char;
  *p = ‘x’;
  std::cout << “x=“ << *p << std::endl;
}     // p is deleted.
```

Tankenöt

http://www.peerdirect.com/resources/gotw020.html

Without being given any additional information, how many execution paths could there be in the following code?

```
String EvaluateSalaryAndReturnName( Employee e )
{
  if( e.Title() == "CEO" || e.Salary() > 100000 )
  {
    cout << e.First() << " " << e.Last() << " is overpaid" << endl;
  }
  return e.First() + " " + e.Last();
}
```

Design av exceptions

- Klasshierarki
- Låt gärna exception-objekt innehålla information
- Exception-objekt skall aldrig kasta exceptions

Hantering av exceptions

- Fånga exceptions via referens
- Ange sub-exceptions före bas-exceptions

Konverteringar

- Kopiering via basklass
- Oönskade konverteringar
- Flertydliga konverteringar

Oönskade konverteringar

```
void work1()
{
  String s(“4”);
  if (s != “4”) {
    std::cout << “Inte fyra” << std::endl;
  }
}
```

Exempel: class String

```
void work1()
{
  String s(“4”);
  if (s != String(“4”)) {
    std::cout << “Inte fyra” << std::endl;
  }
}
```

Oönskade konverteringar

```
class A {
public:
  A(const class B&);
};

class B {
public:
  operator A() const;
};

void work()
{
  void f(A) ;
  B b;
  f(b);
}
```

Styrkor

- Objektorientering
- Stark typning
- Biblioteket

Stark typning

- Funktionssignaturer måste alltid ges.
- Även länkaren kontrollerar typer.
- Templates hjälper till.

Exempel: max(T, T)

```
template <typename T>
T max(T a, T b)
{
  return (a<b)?b:a;
}

a = max(”foo”, 1.0);   // Felaktiga typer.
```

Exempel: Matriser

- Matris-operationer kan bara utföras på matriser med kompatibla storlekar
- Kolla storlekarna tidigt.

Exempel: Matriser (1)

```
template <int r, int c, typename T=double>
class matrix { ... };
template <int r, int c, typename T=double>
  matrix<r,c,T> operator+(const matrix<r,c,T>&,
                          const matrix<r,c,T>&);
template <int r1, int c1, int c2, typename T=double>
  matrix<r1,c1,T> operator*(const matrix<c1,c2,T>&,
                            const matrix<c2,r1,T>&);
```

Exempel: Matriser (2)

```
matrix<3,4> m1, m2, r1;
matrix<4,2> v;
matrix<2,3> r2;
r1 = m1 + m2;
r2 = m1 * v;
```

Biblioteket

- Containers
- Algoritmer
- Streams och strängar

Containers

- Sekvenser
  - **`list`** / **`vector`** / **`deque`**
  - **`queue`** / **`stack`** / **`priority_queue`**
- Associativa tabeller
  - **`set`** / **`map`** / **`multiset`** / **`multimap`**

Iteratorer

- Iteratorer istället för pekare
- Varje container har sin egen iterator
- Iteratorer för olika container har samma grundfunktion
- Vissa container har iteratorer med mer funktionalitet

Typer av iteratorer

- Input iterator
- Output iterator
- Forward iterator
- Bidirectional iterator
- Random access iterator

Exempel: Iterator

```
#include <vector>

int count_values(vector<int> & container, int value)
{
  int count = 0;
  for (vector<int>::const_iterator iter = container.begin();
       iter != container.end();
       ++iter)
  {
    if (*iter == value)
    {
      ++count;
    }
  }
  return count;
}
```

Algoritmer

for_each find find_if find_end find_first_of adjacent_find count count_if mismatch equal search search_n copy copy_backward swap_ranges iter_swap transform replace replace_if replace_copy replace_copy_if fill fill_n generate generate_n remove remove_if remove_copy remove_copy_if unique unique_copy reverse reverse_copy rotate rotate_copy random_shuffle partition stable_partition sort stable_sort partial_sort partial_sort_copy nth_element lower_bound upper_bound equal_range binary_search merge inplace_merge includes set_union set_intersection set_difference set_symmetric_difference push_heap pop_heap make_heap sort_heap min max min_element max_element lexicographical_compare next_permutation prev_permutation

Exempel: Algoritm

```
#include <vector>
#include <algorithm>

void work()
{
  std::vector<int> vi;
  doSomething(vi);
  int count = std::count(vi.begin(), vi.end(), 4);
}
```

Exempel: Algoritm

```
#include <vector>
#include <algorithm>
#include <string>

bool isPalindrome(const std::string &s);

void work()
{
  std::vector<std::string> vs;
  doSomething(vs);
  int count = std::count_if(vs.begin(), vs.end(),
                            isPalindrome);
}
```

Exempel: Algoritm

```
class contains {
public:
  contains(const std::string &s) : m_s(s) {}
  operator()(const std::string &s);
private:
  std::string m_s;
};

void work()
{
  std::vector<std::string> vs;
  doSomething(vs);
  int count = std::count_if(vs.begin(), vs.end(),
                            contains(“foo”);
}
```

Exempel: Fakultet

```
class incr_generator {
  int count;
public:
  explicit incr_generator(int init_value)
         : count(init_value) {}
  int operator()() { return count++; }
};

int fak(int n)
{
  std::vector<int> v(n);
  std::generate(v.begin(), v.end(), incr_generator(1));
  return std::accumulate(v.begin(), v.end(),
                         1, std::multiplies<int>());
}
```

Exempel: Fakultet (2)

```
int fak(int n)
{
  std::vector<int> v(n);
  v[0] = 1;
  std::transform(v.begin(), v.end()-1, v.begin()+1,
                 std::bind2nd(std::plus<int>(), 1)) ;
  return std::accumulate(v.begin(), v.end(),
                         1, std::multiplies<int>());
}
```

Egna algoritmer

```
#include <vector>
#include <list>
#include <iostream>

template <typename Iterator>
void do_something(Iterator first, Iterator last)
{
  std::cout << "Generic do_something" << std::endl ;
}

int main()
{
  std::list<int> li ;
  do_something(li.begin(), li.end()) ;

  std::vector<int> vi ;
  do_something(vi.begin(), vi.end()) ;
}
```

Egna algoritmer

```
#include <iterator>

template <typename Iterator>
void do_something(Iterator first, Iterator last)
{
  do_something(first, last,
               std::iterator_traits<Iterator>::iterator_category()) ;
}
template <typename Iterator, typename IteratorTag>
void do_something(Iterator first, Iterator last, IteratorTag tag)
{
  std::cout << "Generic do_something" << std::endl ;
}
template <typename Iterator>
void do_something(Iterator first, Iterator last,
                  std::random_access_iterator_tag tag)
{
  std::cout << "Random access do_something" << std::endl ;
}
```

Iterator-adaptorer

- reverse_iterator
- insert_iterator
  - insert_iterator
  - back_insert_iterator
  - front_insert_iterator
- stream iterators
  - istream_iterator
  - ostream_iterator
  - istreambuf_iterator
  - ostreambuf_iterator

Effektivare programmering

- Eliminering av temporära objekt
- Inlining
- Snabbare kompileringar

Temporära objekt

```
String operator+(String left,   // copy parameter
                 String right)  // copy parameter
{
  String temp = left ;  // create temporary ; copy parameter
  temp += right ;
  return temp ;          // copy to return value
}

String a, b, c;
c = a + b ;       // 4 temporaries!
```

Temporära objekt (2)

```
String a ;
String c = a + ”hello” ;       // c = a + String(”hello”)
```

Lösning:

```
String operator+(const String &, const char *);
String operator+(const char *, const String &);
```

Inlining

- Endast råd till kompilatorn
- Det måste finnas riktiga funktioner för virtuella funktioner eller funktioner som någon tar adressen till.
- Länkaren måste ta hand om inline-ade funktioner som inte inline-as.
- Vissa små funktioner kan bli stora.

Snabbare kompileringar

- Forward-deklarationer

  istället för #include

- Pimpl-klasser

  Klass med interface använder en pekare till en klass med implementationen.

- Protokoll-klasser

  En basklass implementerar interface. Implementationen finns i en ärvd klass.

Felmeddelanden

```
std::list<smart_ptr<object>> ptrlist ;
```

Visual C++ (6.0) säger:

```
error.cpp(8) : syntax error : missing ’,’ before
   identifier ’ptrlist’
error.cpp(8) : ’ptrlist’ : undeclared identifier
error.cpp(8) : syntax error : missing ’>’ before ';’
```

GNU C++ 2.95 säger:

```
error.cpp:8: warning: ‘>>’ should be ‘> >’ in template
   class name
```

Felmeddelanden

```
void foo(int) ;
std::list<std::string> ls ;
foo(ls) ;
```

Visual C++ (6.0) säger:

```
error.cpp(3) : ’foo’ : cannot convert parameter 1 from
’class std::list<class std::basic_string<char,struct
std::char_traits<char>,class std::allocator<char> >,class
std::allocator<class std::basic_string <char,struct
std::char_traits<char>,class std::allocator<char> > > >’
to 'int’
```

GNU C++ 2.95 säger:

```
error.cpp:3:‘class list<basic_string<char,string_char_trai
ts<char>, __default_alloc_template<false,0> >,__default_al
loc_template<false,0> >’ used where a ‘int’ was expected
```

Fakultet som konstant

```
template <int n> struct fak { enum { C = n * fak<n-1>::C }; };
template <> struct fak<1> { enum { C = 1 }; };

const int F5 = fak<5>::C;
```

Referenser

- ISO/IEC 14882 Programming Languages - C++
- Effective C++ av Scott Meyers
  `http://www.aristeia.com`
- The C++ Standard Library av Nicolai M. Josuttis
- Guru of the Week
  `http://www.peerdirect.com/resources/gotw.html`
- Programming Research
  `http://www.prqa.co.uk`
- Denna föreläsning
  `http://www.lysator.liu.se/~sven-e/c++`