Object Oriented Software Design Exceptions and Templates Giuseppe Lipari http://retis.sssup.it/-lipari Scuola Superiore Sant Anna - Pisa March 27, 2013 G. Lipari (Scuola Superiore Sant Anna) Exceptions and Templates March 27, 2013

```
O It is possible to put more catch blocks in sequence
O they will be processed in order, the first one that catches the exception is the last one to execute

try {
// this code can generate exceptions
// last exceptions of ExcType1
} catch (ExcType1&e1) {
// all exceptions of ExcType2
} catch (...) {
// every exception
}

G. Upat (Scools Superiore Sant Anna)

Exceptions and Templates

March 27, 2013 6/65
```

```
Outline

Exceptions
Cleanup

Generic code

Templates

Standard Template Library
Associative Arrays

Advanced templates
Exercises

G. Licatt (Scools Superiore Sant'Anna)
Exceptions and Templates

March 27, 2013, 2/45
```



```
Try/catch

An exception object is thrown by the programmer in case of an error condition

An exception object can be caught inside a try/catch block

Try {

// this code can generate exceptions

| catch (ExcTypell el) {

// all exceptions of ExcTypel are handled here

| If the exception is not caught at the level where the function call has been performed, it is automatically forwarded to the upper layer

Until it finds a proper try/catch block that cathes it

or until there is no upper layer (in which case, the program is aborted)
```

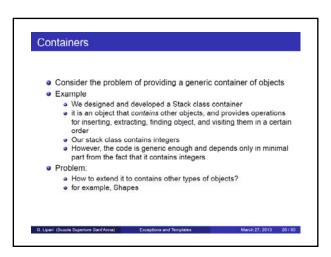
```
    It is possible to specify which exceptions a function might throw, by listing them after the function prototype
    Exceptions are part of the interface!

| void f(int a) throw(Excl, Exc2, Exc3); | void h() throw();

| f() can only throw exception Exc1, Exc2 or Exc3
| g() can throw any exception
| h() does not throw any exception
| h() does not throw any exception
| CLipar (Goods Superiore SanrAnns) | Exceptions and Templates | March 27, 2013 | 10/65
```



When writing code with exceptions, it's particularly important that you always ask, "If an exception occurs, will my resources be properly cleaned up?" Most of the time you're fairly safe, but in constructors there's a particular problem: if an exception is thrown before a constructor is completed, the associated destructor will not be called for that object. Thus, you must be especially diligent while writing your constructor. The difficulty is in allocating resources in constructors. If an exception occurs in the constructor, the destructor doesn't get a chance to deallocate the resource. see exceptions/rawp.cpp



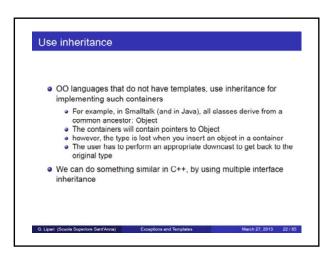
To prevent such resource leaks, you must guard against these "raw" resource allocations in one of two ways:
You can catch exceptions inside the constructor and then release the resources
You can place the allocations inside an object's constructor, and you can place the deallocations inside an object's destructor.
The last technique is called Resource Acquisition Is Initialization (RAII for short) because it equates resource control with object lifetime.
Example: exception_wrap.cpp

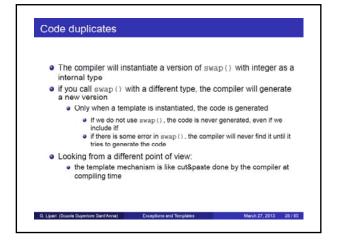
O. Ligani (Souda Supercer SerrAnca)

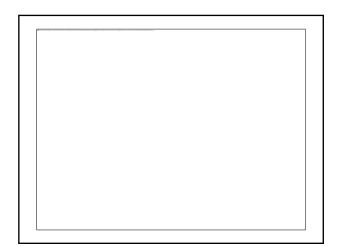
Disceptions and Templetes

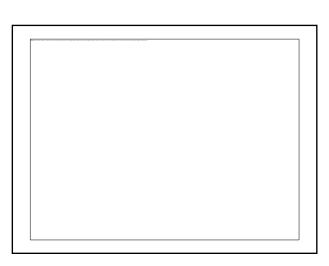
March 27, 2013

17/85

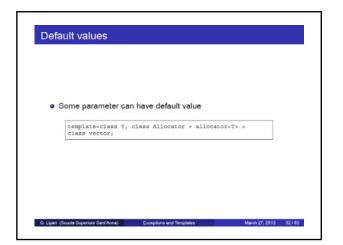


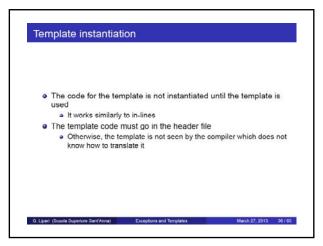






The template mechanism resembles the macro mechanism in C We can do the same in C by using pre-processing macros: ### define awap(type, a, b) { type tmp; tmp=a; a=b; b=tmp; } int x = 5; int y = a; awap(int, x, y); in this case, the C preprocessor substitutes the code it works only if the programmer knows what he is doing The template mechanism does something similar but the compiler performs all necessary type checking O Leaf (Goole Operiors Serf Arose) Exceptions and Templates Macch 27, 2013 27/55

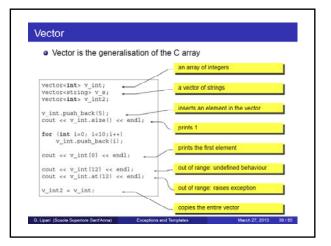


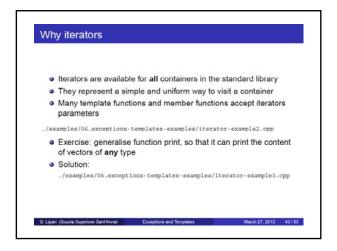


```
The STL is provided with the compiler
It contains generic code (templates) with
containers (vector, list, deque, map, set)
algorithms (sort, foreach, etc.)
I/O streams (cout, cin, fstreams, etc.)
string
Recently, with the new standard, many more features have been added (will see a selection later in the course)

Cueptons and Templates

March 27, 2013 307 65
```





lterators • Iterators are a generic way to access elements in a container, according to a predefined order • The iterator is usually a class provided by the container itself • It can be seen as a pointer to the elements of the container • begin() returns an iterator to the first element • end() returns the iterator pointing beyond the last element of the array • it is possible to use ++ and - to increment/decrement the iterator (i.e. move to the next/previous element) • it is possible to access the pointed element by using the dereferencing operator★ vector<int> v; vector<int> v;

```
Lists

• The STL also provides the simple linked list we have seen in the course
• In the STL, the template parameter indicates the data type

list-tint> list;
for (int i=0; l<10; i++)
list.push_back(i);
// going through all elements
list-tint>:riterator i = lst.begin();
int sum = 0;
while (i|=lst.end()) {
    sum == *i;
    i++;
}

O Uper (Coools Superiors Sant/Anne)

Cooptons and Templates

March 27, 2013 45/65
```

```
Iterators

Interators

Int main()
{
    int main()
    {
        int main()    {
            vector-int+ v - {2, 4, 6, 8}, vec
```

```
There are five types of iterators, depending on the functionality they provide:

The difference consists in the type of operations that are supported:

all types support operator ++ and *
input supports copy construction and copy, operator ->, equality == and inequality !=

output supports assignment as Ivalue (to the left of an assignment operator)

forward is as input and output, but also supports default constructor

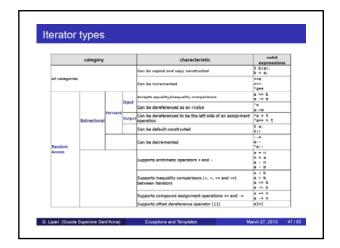
bidirectional is as forward, but it also supports operator -

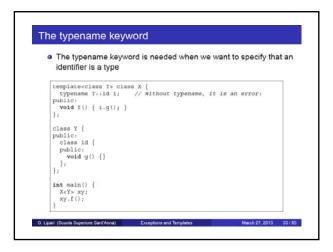
random is as bidirectional, but also supports operator -

random is as bidirectional, but also supports operator -

random is as bidirectional, but also supports operator -

random is one of the companion of the co
```





Associative array generalize the concept of array
Two subtypes: sets and maps

• est-key- and multiset-key- contain ordered sets of objects
• in set-key- the key must be unique
• in subtiset-keys, the same key can be inserted several times

• map-key, value> and multimap-key, value> contains pairs <key, value>, where key is the "index" in the array
• in map-key, value>, each different key must be associated one unique value
• map-key, value>, several values can be associated to the same key

• Lipan (Cookla Coperiore Carr'Array)

Coceptions and Templates

March 27, 2013 207/85

if a type referred to inside template code is qualified by a template type parameter, you must use the typename keyword as a prefix,
 unless it appears in a base class specification or initializer list in the same scope (in which case you must not).

O Ligari (Scoola Superiore Cent'Anne)

Coceptions and Templates

March 27, 2013

54/65

Usage

• The typical example of usage is for iterators

template<class T, template<class U, class = allocator<U> > class Seq*

void printseq(iseq<0.6 seq) {
 for(typename Seq<0.7::terator b = seq.begin();
 b != seq.end();
 cut < < sb+ < endl;
}

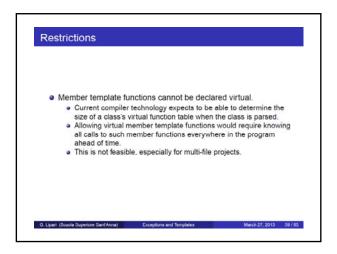
int main() {
 // Process a vector
 vector<int> V;
 v.push back(1);
 printSeq(v);
 // Process a list
 list-int lst;
 lat.push back(1);
 lot.push back(3);
 printSeq(lat);
}

O.Liper(Gooode Ouperlove CertAnne)

Coceptions and Templates

March 27,2013 55/65

An example for the complex class | template<typename T> class complex { public: template<class X> complex(const complex<X>\$\delta\$); }; complex<float> \(\pi(1, 2); complex<double> \(\pi(2); complex<double> \(\pi(2); complex<double> \(\pi(2); complex double> \(\pi(2); complex double> \(\pi(2); complex double) On the declaration of w, the complex template parameter T is double and X is float. Member templates make this kind of flexible conversion easy. On the declaration of the complex templates make this kind of flexible conversion easy.



| Int data[5] = { 1, 2, 3, 4, 5 }; | vector<int> v1(data, data+5); | vector<double> v2(v1.begin(), v1.end()); | As long as the elements in v1 are assignment-compatible with the elements in v2 (as double and int are here), all is well | The vector class template has the following member template constructor: | template<class InputIterator | | vector(InputIterator first, InputIterator last, const Allocator& = Allocator()); | InputIterator is interpreted as vector<int>::iterator

```
Function templates

The standard template library defines many function templates in algorithm

agorithm

sort, find, accumulate, fill, binary_search, copy, etc.

An example:

Sinclude calgorithms

int i, j;

int z = min<int>(i, j);

Type can be deducted by the compiler

But the compiler is smart up to a certain limit ...

Int z = min(x, j); // x is a double, error, not the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<ach description of the same types

int z = min<
```

```
Another example

template<class T> class Outer {
  public:
    template<class R> class inner {
    public:
       void f();
    };
};

template<class T> template<class R>
  void Outer<T>::Inner<R>::f() {
       cout << "Outer == " << typeid(T).name() << endl;
       cout << "Full Inner == " << typeid(*this).name() << endl;
       cout << "Full Inner == " << typeid(*this).name() << endl;
    }

int main() {
    Outer<int>::Inner<bool> inner;
    inner.f();
}
O. Upan (Souola Cuperiors CertArna)

Comptions and Templates

March 27, 2013 50/65
```