Formation C++ Ubisoft - Module 6

Romain Arcila[1,2]
Charles de Rousiers[1]

10 mai 2009

[1] Inria Grenoble
[2] Liris -CNRS Lyon

Plan

Introduction

- Suite du module précédent.
- Template nécessaire.

Plan

- TR1 : Technical Report 1
- Boost : librairie "presque" standard

Plan

TR1 : Who - When

- ISO/IEC TR 19768, C++ Library Extensions.
- Mais ce n'est pas une norme : Draft.
- Proposition (quasi-accepté) pour le prochain standard (C++0X).

$\Rightarrow$ Implémentation non obligatoire (pour être conforme a la norme)

Tr1 : Where

- Visual Studio
- Gcc
- Boost (principale source d'inspiration pour le TR1)

Aucune de ces implémentation n'est complète et entièrement conforme (actuellement).

TR1 : What

- **Utilitaire**.
- **Programmation fonctionnelle**.
- **MetaProgramming**.
- Maths.
- **Conteneur**.
- RegExp.

Reference Wrapper

- Basé Boost.
- Header : `<functional>`.
- Fournit : `ref()`, `cref()`, `reference_wrapper`
- Permet de passer des références plutôt que des copies dans les templates.

RW Exemple

Code : RW

```
void f2(int &a) { a++; }

template< class Funct, class Arg >
void g( Funct f, Arg t ) { f(t); }

g(f2,a);
// valeur de a ?
```

RW Exemple

Code : RW

```
void f(int a) { a++; }
void f2(int &a) { a++; }

template<typename t>
void ft(t a) { f2(a); }

template< class Funct, class Arg >
void g( Funct f, Arg t ) { f(t); }
```

RW Exemple

Code : Exmple 2

```
int a=0;
f(ref(a)); // appel normal => f avec copie de a                    0
f2(ref(a)); // appel normal => f2 avec reference sur a             1
ft(a); // appel ft avec une copie de a => f2 ne modifie pas a      1
ft(ref(a)); // appel ft avec RW(a) => f2 a une reference sur a     2
g(f2,a); // g prend une copie de a ...                            2
g(f2,ref(a)); // g prend un RW(a)                                  3
```

cref

Version constante de ref() $\Rightarrow$: renvoie une référence constante.

cref

- Pour les passages de référence constant.
- Optimisation.

Règle habituelle du passage de paramètre.

- Classe renvoyée par `ref` et `cref`.
- Peu de raison de l'instanciée manuellement.
- Interface :

Code : Interface

```
template<class T>
class reference_wrapper {
  reference_wrapper(T& __indata): _M_data(&__indata) {}
  reference_wrapper(const reference_wrapper<T>& __inref): _M_data(__inref._M_data) { }

  reference_wrapper&
  operator=(const reference_wrapper<T>& __inref) {
    _M_data = __inref._M_data;
    return *this;
  }

  operator T&() const { return this->get(); }
  T& get() const{ return *_M_data; }
}; // Note : __inref._M_data contient l'objet
```

SmartPointer

- Basé boost.
- Pointeur intelligent utilisant le RIIA
- Header : memory
- `shared_ptr, weak_ptr...`

- Allocation gérée par l'utilisateur
- Désallocation gérée le `shared_ptr`
- `shared_ptr` contient un pointeur et un compteur
  - Destruction de `shared_ptr` : compteur décrémenter
  - Copie, Affectation : compteur incrémenter
- Compteur à 0 : désallocation.

SmartPointer Exemple

Code : SharedPointer

```
void f(shared_ptr<int> p) { // copie: compteur incrementer

} // destruction de p: compteur decrementer

{
  shared_pointer<int> a(new int);
  f(a);
  int* pi=a.get(); // principalement pour des raisons de compatibilite
  // a detruit => compteur passe a 0 => contenu de a detruit
}
```

Shared Pointer

Shared Pointer

- Creation d'un `shared_ptr` : ne pas le faire lors de l'appel de fonction.

```
// bad code
{
  f(shared_pointer<int> a(new int));
}
```

- utiliser le plus possible le `shared_ptr` et non le pointeur contenu (minimiser .get()).

Functionnelle

- Concerne les fonctions
- Header : functional

Polymorphic Function Wrapper

- basé boost function.
- Pointeur de fonction généralisé : peut contenir des fonctions de tous types.
- Permet de faire passer des pointeurs de fonctions dans les fonctions plus facilement.
- Utilisation conjointe avec bind et lambda.

Polymorphic Function Wrapper II

Code : PFW

```
int f(int& a) { return ++a; }

struct C {
  int f(int &a) { return ++a; }
  static int fs(int &a) { return ++a; }
};

struct incr { int operator()(int &a) { return ++a; } };
```

PFW Exemple 2

Code : PFW Exemple 2

```
int a=0;
incr i;
// permet de creer une fonction renvoyant un int et prenant un int
typedef function<int (int&)> MyFunction;
// appel de la fonction f par f1
MyFunction f1(f);
cout << f1(a) << endl;                                    1
// appel de la methode statique C::f par fs
MyFunction fs(C::fs);
cout << fs(a) << endl;                                    2
// Appel de incr ::() automatique...
MyFunction fc2(i);
cout << fc2(a) << endl;                                   3
// Defaut: copie de i=> ref pour avoir une reference de i
MyFunction fcref(ref(i));
cout << fcref(a) << endl;                                 4

function<int (C,int&)> fc(&C::f);
cout << fc(C(),a) << endl;                                5
```

Bind

- Basé Boost.
- Generalisation de bind1st et bind2nd de la STL.
- Permet de fixer le (les) arguments d'une fonction dans un nouvel objet appelable $\Rightarrow$ très utile lors de parcours de collections.
- Plus facile d'utilisation : utilisation de variable désignant le numéro du parametre : $\__1$ représente le 1er argument de l'appel.

Bind Exemple

Code : Bind Exemple

```
int f(int& a) { return ++a; }
int f(int& a, int b) { return a+=b; }
int g(int& a) {return ++a;}

struct F {
  int operator()(int a, int b) { return a - b; }
};
```

PFW Exemple 2

Code : Bind Exemple 2

```
// Utilisation de base
int a=0; int b=5; F fo; int x = 104; int x2 = 8;
// retourne une fonction dont l'argument lors de l'appel
// sera le 1er => f(a)
bind(f,_1)(a); cout << a << endl;                                   1
// appel f(a,_1) => f(a,5)
bind(f,a,_1)(5); cout << a << endl;                                 1
//appel f( reference sur a,5)
bind(f,ref(a),_1)(5); cout << a << endl;                            6
// appel f(b,a)
bind(f,_1,_2)(b,a); cout << a << endl;                              6
// appel f(a,b)
bind(f,_2,_1)(b,a); cout << a << endl;                              17
// appel f(g(a))
bind(f,bind(g,_1))(ref(a)); cout << a << endl;                      19

// Utilisation de la valeur de retour
// Par defaut: specifier le type : bind<int> :retour int
cout << bind<int>(fo, _1, _1)(x) << endl;                           0
// Si la classe contient un typedef result_type : bind() suffit
// std :: less contient result_type , bind<int> non necessaire
cout << bind(std::less<int>(), _1, 9)(x2) << endl;                  1
```

Bind : Remarques

- Sert énormément combiné avec `mem_fn` et `for_each`
- Operateurs surchargés : permet la comparaison des résultats d'un bind, ainsi que les expressions logiques.

  ```
  if (bind(f,_1)(a)<bind(f,_1)(b) ...
  ```

- Combinable avec function.

  ```
  int a; function<int (int,int)> fu=bind(f,_1,a);... fu(2);
  ```

- Basé Boost.
- Construire un functor a partir d'une méthode de classe.
- Amélioration de `mem_fun` et `mem_fun_ref` (mais pas `ptr_fun`)
  - `mem_fun` : pointeur
  - `mem_fun_ref` : reference
  - autre ?

`mem_fn` : pointeur, reference, autre (nécessite la méthode `get_pointer()`)

mem_fn Exemple

Code : mem_fn Exemple

```
class Shape {
  public:
  Shape(int a): _a(a) {}
  void draw() const { cout << _a << endl;}
  void draw2(int vec) const { cout << _a+vec << endl;}
  private:
  int _a;
};

void f(Shape& s) {
  cout << "Drawing s" << endl;
}
```

mem_fn Exemple 2

Code : mem_fn 2

```
vector<Shape> vs; vector<Shape*> vsp; vector<shared_ptr<Shape> > vssp;
for (unsigned int i=0; i < 10; ++i) {
  vs.push_back(Shape(i));
  vsp.push_back(new Shape(i));
  vssp.push_back(new Shape(i));
}


for_each(vs.begin(), vs.end(), mem_fun_ref(&Shape::draw));
for_each(vsp.begin(), vsp.end(), mem_fun(&Shape::draw));

// Erreur !
// for_each(vs.begin(), vs.end(), mem_fun(&Shape::draw));
// for_each(vsp.begin(), vsp.end(), mem_fun_ref(&Shape::draw));
// for_each(vssp.begin(), vssp.end(), mem_fun(&Shape::draw));
// for_each(vssp.begin(), vssp.end(), mem_fun_ref(&Shape::draw));

for_each(vs.begin(), vs.end(), mem_fn(&Shape::draw));
for_each(vsp.begin(), vsp.end(), mem_fn(&Shape::draw));
for_each(vssp.begin(), vssp.end(), mem_fn(&Shape::draw));
for_each(vssp.begin(), vssp.end(), bind(mem_fn(&Shape::draw2),_1,5));
for_each(vs.begin(), vs.end(), ptr_fun(f));
for_each(vs.begin(), vs.end(), f);
```

MetaProgramming

- Basé Boost.
- Permet de savoir si un type a certaine propriétée.
- Transformation sur les types.
- Tres utile avec boost static assert et concept

Principe

```
// cas general :
template<bool>
struct is_void { static bool value=false; };
//  specialisation
template<>
struct is_void<void>{ static bool value=false ;}
```

-
- Dans Tr1, l'attribut value contient le résultat de la requete.
- Dans Tr1, l'operateur () renvoie `true_type` si la propriété est vraie et `false_type` sinon.

SFINAE

- Substitution Failure is no an error
- Si une substitution est fausse, on continue !

SFINAE

```
struct Test {
  template<class A> void test(A* a) {
    cout << "pointer" << endl;
  }
  void test(...) {
    cout << "not_pointer" << endl;
  }
};
Test t;
int a;  int* b;
t.test(a);
t.test(b);
```

- Utilise pour les traits !

MetaProgramming Exemple Simple

Code : MetaProg

```
typedef char True;
class FalseT {char a[20];}; typedef FalseT False;

template <class T>
True is_same(T*,T*);
False is_same(...);

template<class A, class B>
struct SameType {
  static A* a;
  static B* b;
  static const bool value=(sizeof(True)==sizeof(is_same(a,b)));
};

class X {};
class Y {};

X x;
Y y;

cout << SameType<X,X>::value << endl;
cout << SameType<X,Y>::value << endl;
```

MetaProgramming Exemple Simple

Code : MetaProg

```
struct IsRef {
  void operator()(const true_type&) {cout << "is__a_ref" << endl;}
  void operator()(const false_type&) {cout << "is_not_a_ref" << endl;}
};

template<bool>
struct Test {
  void operator()(){cout << "Im_a_ref"<<endl;}
};

template<>
struct Test<false> {
  void operator()(){cout << "Im_not_a_ref"<<endl;}
};

int
main() {
  int a;
  int& b=a;

  IsRef()(is_reference<int&>());
  IsRef()(is_reference<int>());
  Test<is_reference<int>::value>()();
  Test<is_reference<int&>::value>()();
  return 0;
}
```

MetaProgramming Exemple

Code : MetaProg

```
// memcpy est normalement plus rapide, mais necessite certaines contraintes : avoir un operateur= simple
// note: true_type=integral_constant<bool, true>
//       false_type=integral_constant<bool, false>
namespace detail{

  template<typename I1, typename I2, bool b>
  I2 copy_imp(I1 first, I1 last, I2 out, const boost::integral_constant<bool, b>&){
    while(first != last) {
      *out = *first;
      ++out;
      ++first;
    }
    return out;
  }

  template<typename T>
  T* copy_imp(const T* first, const T* last, T* out, const boost::true_type&) {
    memcpy(out, first, (last-first)*sizeof(T));
    return out+(last-first);
  }
}

template<typename I1, typename I2>
 inline  I2 copy(I1 first, I1 last, I2 out){
  typedef typename std::iterator_traits<I1>::value_type value_type;
  return detail::copy_imp(first, last, out, boost::has_trivial_assign<value_type>());
}
```

Random

- Header : random.
- Ajout de nouveaux générateur (v. TR1 Draft).

Fonction

- Header : cmath.
- Ajout de nouveaux fonctions (v. TR1 draft).

Tuple

- Aggregation de plusieurs objets de type différents. (extension de `std::pair`).
- Basé Boost.
- Header : tuple
- Aide a la creation avec `make_tuple(...)`
- la fonction `get<int I>(tuple)` permet de recuperer le Ieme attibut.
- les tuples et la fonction `tie` permettent de simuler les retour multiples pour les fonctions.

Tuple Exemple

Code : Tuple

```
tuple<int&,double&> f(int& a, double& b) {
  return make_tuple(ref(a),ref(b)); // make_tuple <int, double> use ref()  for  reference
}

tuple<int,double> g() {
  return make_tuple(1,2.);
}
```

Tuple Exemple 2

Code : Tupe Exemple 2

```
tuple<int,double,string> t(2,2.0);
cout << get<0>(t) << " " << get<1>(t)<< " " <<          2 2
get<2>(t) << endl;
t=make_tuple(0,0.0,"aaa");
cout << get<0>(t) << " " << get<1>(t)<< " " <<          0 0 aaa
get<2>(t) << endl;
get<0>(t)=2;
cout << get<0>(t) << " " << get<1>(t)<< " " <<          2 0 aaa
get<2>(t) << endl;
int a=0; int a2=0; double b2=0.;
tuple<int&,double,string> t2(a,2.0);
get<0>(t2)=2;
cout << a << endl;                                      2

tuple<int&,double&> t3(f(a2,b2));
tuple<int&,double&> t4(f(a2,b2));
cout << (t3 < t4 ) << " " << (t3==t4) << endl;          0 1
int a3=0;
double b3=0.;
tie(a3,b3)=g();
cout << a3 << " " << b3 << endl;                        1 2
a3=0;
tie(ignore,b3)=g();
cout << a3 << " " << b3 << endl;                        0 2
//  cout << t3 << endl; isn't working
```

Array

- Header : array
- Basé Boost.
- Tableau taille constante avec interface compatible STL.

Array Exemple 2

Code : array Exemple 2

```
void display(int a) { cout << a << " "; }

void c_func(int* a, unsigned s) {
  for (unsigned i=0; i < s; ++i)
    a[i]++;
}

array<int,4> a;
typedef array<int,4>::iterator iter;
cout << a.size() << endl;                          4
int c=0;
// utilisable comme conteneur STL
for (iter i=a.begin(); i < a.end(); ++i,++c)
  *i=c;
for_each(a.begin(), a.end(), display);             0 1 2 3
cout << endl;
// data() renvoie le pointeur type C
c_func(a.data(), a.size());
for_each(a.begin(), a.end(), display);             1 2 3 4
cout << endl;
```

Hashmap

- Header : `unordered_map`, `unordered_set`
- Utilisation très proche de map et set.
- Différence avec un map (et un set)
  - Non ordonné.
  - Hachage : accès plus rapide

HashMap Exemple

Code :Hashmap base

```
// type de la cle, type de la valeur
unordered_map<string,string> s;
typedef unordered_map<string,string>::iterator iter;
s["this"]="that"; // ajout d'element
s["this2"]="that2";
s["this3"]="that3";
s["this4"]="that4";

// recuperation de la valeur
cout << s["this"] << endl;                                    that
// element non present: ajout avec valeur par defaut
cout << s["tada"] << endl;                                    ""
// iteration comme un map...
for (iter i=s.begin(); i != s.end(); ++i)
  cout << "K:" <<i->first                                     K:this2,that2 K:this4,that4
    << "," << i->second << "␣";                  K:this3,that3 K:this,that K:tada,
```

HashMap Exemple

Code : Interface

```
template<class _Key, class _Tp,
 class _Hash = hash<_Key>,
 class _Pred = std::equal_to<_Key>,
 class _Alloc = std::allocator<std::pair<const _Key, _Tp> > >
 class unordered_multimap
```

`hash` : calcul de la cle et `equal_to` : comparation, par defaut ==

HashMap Exemple

Code : Interface

```
template<typename _Tp>
struct hash : public std::unary_function<_Tp, size_t> {
  size_t
  operator()(_Tp __val) const;
};
```

RegExp

- Header : regex
- Implémentation des expressions rationelles.
- Basé Boost.
- Syntaxe Posix, Posix étendu et Perl.

RegExp Exemple

Code : RegEx

```
void is_email_valid(const std::string& email) {

  // definition d'une expression rationelle
  regex p("(\\w+)(\\.|_)?(\\w*)@(\\w+)(\\.(\\w+))+");
  smatch m;

  // recherche: si trouve: true, info dans match
  if (regex_match(email, m, p)) {
    cout << "User is : " ;
    for (unsigned int i =1; i < m.size()-3; ++i)
      cout << m[i];
    cout << " and domain is " << m[m.size()-3];
    cout << match[m.size()-2] << endl ;
  } else
    cout << "Mail is invalid" << endl;
}
string email1 = "marius.bancila@domain.com";
string email2 = "mariusbancila@domain.com";
string email4 = "marius@domain";
```

```
is_email_valid(email1);
is_email_valid(email2);
is_email_valid(email4);
```

```
User is : marius.bancila and domain is domain.com
User is : mariusbancila and domain is domain.com
Mail is invalid
```

TR2

- Unicode
- XML & HTML
- Socket
- Module

Plan

Boost - What's ?

- Ensemble de librairies reconnues pour leurs qualités.
- License permissive.
- Deux types :
  - abstraction OS : filesystem, thread...
  - fonctionnalités : date, container, algorithmes...
- Source "d'inspiration" pour le TR1 et le TR2.

Boost - What's, Where

- www.boost.org
- Développment communautaire.
- Dans ce cours : fonctionnalité (un peu)

Variant

- Principe : Union.
- Contient un élément courant, peut contenir différents types.
- Fonctionne avec les classes utilisateurs.
- Header (principal) : variant.hpp

Variant Exemple

Code : Variant Exemple

```
// union contenant soit un int soit un string
variant< int, string > u("hello world");
cout << u << endl; // affiche hello world
u="abc";
// recupere le champs en tant que string
get<string>(u)+="a"; // affiche abca
cout << u << endl;
```

Variant Exemple

Code : Variant Exemple

```
struct my_visitor : public static_visitor<variant<int,string> >{
  variant<int,string> operator()(int i) const {
    return variant<int,string>(i+1);
  }
  variant<int,string> operator()(const string & str) const {
    return  variant<int,string>(str+"test");
  }
};

variant< int, string > u("abca");
cout << u << endl;
// le visitor travaille avec le type courant donc ()( string &)
variant<int,string> result = apply_visitor( my_visitor(), u );
cout << result << endl; // affiche abcatest
u=2; // change le type courant
cout << u << endl; // affiche 2
// le visitor travaille avec le type courant donc ()( int &)
result = apply_visitor( my_visitor(), u );
cout << result << endl; // affiche 3
```

Variant Exemple

Code : Variant Exemple

```
// marche avec tous les types qui ont +
struct generic_visitor : public static_visitor<>{
  template<typename T>
  void operator()(T& t) const {
    t+=t;
    cout << t << endl;
  }
};

int main() {
  variant< int, string > u("hello world");
  boost::apply_visitor( generic_visitor(), u ); // affiche hello worldhello world
  return 0;
}
```

Any

- Principe : Stocke un objet de n'importe quel type.
- Runtime.
- Header (principal) : any.hpp

Any - Exemple

Code : Any Exemple

```
struct Test {
  int a;
  Test(): a(0) {}
  int& getA() { return a; }
};

any a; // cree un any ne contenant rien
a=2; // any contient 2
// recupere la valeur comme etant un int
cout << any_cast<int>(a) << endl;
// a.type() renvoie la structure typeinfo du type contenu:
assert(a.type()==typeid(int));
a="aa";
// throw bad_any_cast si le type contenu n'est pas le type demande
try {
  cout << any_cast<int>(a) << endl;
} catch (bad_any_cast) {
  cout << "Does not contain an int" << endl;
}

assert(a.type()==typeid(string));
a=Test(); // fonctionne avec les types utilisateurs
any_cast<Test&>(a).getA()+=4;
cout << any_cast<Test&>(a).getA() << endl; // affiche 4
```

Any

- Runtime : cout.
- Système de propriété générique.

Operators

- Fournit les opérateur (quasi-) automatiquement.
- Grande Granularité.

Operator Exemple

Operator Exemple

```
// addable<T>, substractable<T> : fournit l'addition et la soustraction entre deux T
// dividable2 <T,Y>, mulitplicable<T,Y> : fournit la multiplication et la divisition entre T et Y
template <class T>
class point: addable< point<T>, subtractable< point<T>
    , dividable2< point<T>, T, multipliable2< point<T>, T> > > > { // note: operateur sont chaines
    T x_; T y_;
public :
  point(const T& x, const T& y):x_(x), y_(y) {}
  T x() const {return x_; }
  T y() const {return y_; }

  // a partir de OP=, fournit OP
  point<T>& operator+=(const point<T>& p) {... }
  point<T>& operator-=(const point<T>& p) {... }
  point<T>& operator*=(const T& t) {... }
  point<T>& operator/=(const T& t) {... }
};

template <class T>
T length(const point<T> p) { return sqrt(p.x()*p.x() + p.y()*p.y()); }
```

Operator Exemple

Operator Exemple

```
const point<float> right(0, 1);
const point<float> up(1, 0);
const point<float> pi_over_4 = up + right;
const point<float> pi_over_4_normalized = pi_over_4 / length(pi_over_4);
```

Operator Exemple

Operator Exemple

```
// arithmetic : addition et soustraction , multiplicative : mulitplication et addition
template <class T>
class point2: arithmetic< point2<T>, multiplicative2< point2<T>, T> > {
    T x_; T y_;
public :
  point2(const T& x, const T& y):x_(x), y_(y) {}
  T x() const {return x_; }
  T y() const {return y_; }

  point2<T>& operator+=(const point2<T>& p) {... }
  point2<T>& operator-=(const point2<T>& p) {... }
  point2<T>& operator*=(const T& t) {... }
  point2<T>& operator/=(const T& t) {... }
};
```

Operator Suite

- granularite,
- version avec 1 ou 2 types différents
- operation, comparaisons...

static assert

- Assertion à la compilation
- Scope : namespace, fonction et classe.
- Prend en parametre un booleen.
- Permet de vérifier des propriétés à la compilation.

static_assert Exemple I

Code : static_assert I

```
template<class T>
struct Test { BOOST_STATIC_ASSERT((sizeof(T)>2)); };
int
main() {

  BOOST_STATIC_ASSERT(true);
  // BOOST_STATIC_ASSERT(false); fails
  Test<long> a;
  // Test<char> a; fails
  return 0;
}
```

static_assert Exemple II

Code : static_assert II

```
template<class T, class T2>
T2& downcast(T& t, const T2& t2) {
  BOOST_STATIC_ASSERT((is_base_of<T,T2>::value));
  return dynamic_cast<T2&>(t);
}

struct X { virtual void print() const { cout << "Im a X" << endl; } };

struct Y: public X { virtual void print() {cout << "Im a Y" << endl; } };

class Z {};

int
main() {
  X* x=new Y();
  Y* y= new Y();
  Z* z= new Z();
  downcast(*x,*y).print();
  // downcast(*x,*z).print(); // Erreur : Z n'herite pas de X.
  return 0;
}
```

- Activé, désactivé des fonctions uniquenement si le paramètre template répond à certains critères.
- Permet de Spécialisé plus finement (Exemple que pour les types numériques).
- `enable_if_c` active : prend un parametre booleen et le type de retour
- `disable_if_c` active : prend un parametre booleen et le type de retour

enable_if Exemple

Code : enable_if

```
// Cette fonction ne fonctionne que pour les classes
template <class T>
typename enable_if_c<is_class<T>::value, T>::type // type==T
trivial_ex(const T& t) { cout << typeid(T).name()<< " is a class" << endl; }

class X {
};

int
main() {

  //  trivial_ex (5); // la fonction n'existe pas pour les types primitifs .
  trivial_ex(X());

  return 0;
}
```

enable_if Exemple II

Code : enable_if II

```
template<typename T> struct has_plus_operator { static const bool value=false; };
template<> struct has_plus_operator<int> { static const bool value=true; };

// Si le type possede un operateur + : on l'utilise
template<class T> typename enable_if_c<has_plus_operator<T>::value,T>::type
add(const T& e1, const T& e2) { return e1+e2; }
// sinon on utilise la methode add
template<class T> typename enable_if_c<!has_plus_operator<T>::value,T>::type
add(const T& e1, const T& e2) { return e1.add(e2); }

struct Test {
  int a;
  Test(int _a): a(_a) {}
  Test operator+(const Test& t1) const {
    Test tmp(*this); return tmp+=t1.a;
  }
};
template<> struct has_plus_operator<Test> { static const bool value=true; };

struct Test2 {
  int a;
  Test2(int _a): a(_a) {}
  Test2 add(const Test2& t1) const {
    Test2 tmp(*this); return tmp+=a;
  }
};
```

Code : enable_if Exemple II suite

```
Test t1(1);  Test t2(2);
Test2 t3(1);  Test2 t4(2);

cout << add(1,2)<< endl; // utilise +
cout << add(t1,t2).a << endl; // utilise +
cout << add(t3,t4).a << endl; // utilise .add
```

BCCL

- Boost Concept Check Library
- Contrainte des types dans les templates : nom du parametre, pas de controle.
  $\Rightarrow$ Message d'erreur tardif, avec les templates dépliés.
- BCCL : specifier les contraintes qu'un type doit respecter pour être utilisé dans des templates.
- But :
  - détecter les erreurs les plus tot possible. (utilisateur).
  - faciliter l'écriture de nouvelle contraintes.

Exemple sans concept

Template Fails

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

int main() {
  std::vector<std::complex<float> > v;
  // ne fonctionne pas car les complex ne possede pas <
  std::stable_sort(v.begin(), v.end());
  return 0;
}
```

Exemple sans concept : Sortie

Template Fails

```
     /usr/include/c++/4.3/bits/stl_algo.h: In function 'void std::__insertion_sort(_RandomAccessIterator, _Ra
______/usr/include/c++/4.3/bits/stl_algo.h:3059:___ instantiated _from_'void_std ::  __inplace_stable_sort  (_RandomAccessI
     /usr/include/c++/4.3/bits/stl_algo.h:4982: instantiated from 'void std::stable_sort(_RAIter, _RAIter) [wit
______boost_metaprog_fails.cc:7:___ instantiated _from_here
______/usr/include/c++/4.3/bits/stl_algo.h:1757:_error:_no_match_for_'operator<' in '__val < __first.__gnu_cxx::__n
______/usr/include/c++/4.3/bits/stl_algo.h:_In_function_'void_std :: __merge_without_buffer (  _BidirectionalIterator  ,_ _B
     /usr/include/c++/4.3/bits/stl_algo.h:3065: instantiated from 'void std::__inplace_stable_sort(_RandomAcc
______/usr/include/c++/4.3/bits/stl_algo.h:4982:___ instantiated _from_'void_std :: stable_sort (_RAIter,__RAIter)_[with__

     /usr/include/c++/4.3/bits/stl_algo.h:2679: error: no match for 'operator<'_in_'__middle.__gnu_cxx:: __normal_ite
     /usr/include/c++/4.3/bits/stl_algo.h: In function 'void std::__unguarded_linear_insert(_RandomAccessIter
______/usr/include/c++/4.3/bits/stl_algo.h:1763:___ instantiated _from_'void_std ::  __insertion_sort  (_RandomAccessIterat
     /usr/include/c++/4.3/bits/stl_algo.h:3059: instantiated from 'void std::__inplace_stable_sort(_RandomAcc
______/usr/include/c++/4.3/bits/stl_algo.h:4982:___ instantiated _from_'void_std :: stable_sort (_RAIter,__RAIter)_[with__
     boost_metaprog_fails.cc:7:  instantiated from here
     /usr/include/c++/4.3/bits/stl_algo.h:1718: error: no match for 'operator<'_in_'__val_<___next.__gnu_cxx:: __nor
     /usr/include/c++/4.3/bits/stl_algo.h: In function '_FIter std::lower_bound(_FIter, _FIter, const _Tp&) [wi
______/usr/include/c++/4.3/bits/stl_algo.h:2691:___ instantiated _from_'void_std :: __merge_without_buffer (  _Bidirectionall
     /usr/include/c++/4.3/bits/stl_algo.h:3065: instantiated from 'void std::__inplace_stable_sort(_RandomAcc
______boost_metaprog_fails.cc:7:___ instantiated _from_here
______/usr/include/c++/4.3/bits/stl_algo.h:4982:___ instantiated _from_'void_std :: stable_sort (_RAIter,__RAIter)_[with__
     boost_metaprog_fails.cc:7:  instantiated from here
     /usr/include/c++/4.3/bits/stl_algo.h:2093: error: no match for 'operator<'_in_'__middle.__gnu_cxx:: __normal_ite
     /usr/include/c++/4.3/bits/stl_algo.h: In function '_FIter std::upper_bound(_FIter, _FIter, const _Tp&) [wi
______/usr/include/c++/4.3/bits/stl_algo.h:2698:___ instantiated _from_'void_std :: __merge_without_buffer (  _Bidirectionall
     /usr/include/c++/4.3/bits/stl_algo.h:3065: instantiated from 'void std::__inplace_stable_sort(_RandomAcc
```

Exemple sans concept : Sortie II

Template Fails

```
      /usr/include/c++/4.3/bits/stl_algo.h:2191: error: no match for 'operator<' in '__val < __middle.__gnu_cxx::__n
      /usr/include/c++/4.3/bits/stl_algo.h: In function '_OIter std::merge(_IIter1, _IIter1, _IIter2, _IIter2, _
      /usr/include/c++/4.3/bits/stl_algo.h:2566: instantiated from 'void std::__merge_adaptive(_BidirectionalIterato
      /usr/include/c++/4.3/bits/stl_algo.h:3016: instantiated from 'void std::__stable_sort_adaptive(_RandomAc
      /usr/include/c++/4.3/bits/stl_algo.h:4984: instantiated from 'void std::__stable_sort(_RAIter, _RAIter) [with__
      boost_metaprog_fails.cc:7:   instantiated from here
      /usr/include/c++/4.3/bits/stl_algo.h:4868: error: no match for 'operator<' in '__first2.__gnu_cxx::__normal_iter
      /usr/include/c++/4.3/bits/stl_algo.h: In function '_BidirectionalIterator3 std::__merge_backward(_Bidir
      /usr/include/c++/4.3/bits/stl_algo.h:2572: instantiated from 'void std::__merge_adaptive(_BidirectionalIterato
      /usr/include/c++/4.3/bits/stl_algo.h:3016: instantiated from 'void std::__stable_sort_adaptive(_RandomAc
      /usr/include/c++/4.3/bits/stl_algo.h:4984: instantiated from 'void std::__stable_sort(_RAIter, _RAIter) [with__
      boost_metaprog_fails.cc:7:   instantiated from here
      /usr/include/c++/4.3/bits/stl_algo.h:2468: error: no match for 'operator<' in '*__last2 < __last1.__gnu_cxx::
      /usr/include/c++/4.3/bits/stl_algo.h: In function '_OIter std::merge(_IIter1, _IIter1, _IIter2, _IIter2, _
      /usr/include/c++/4.3/bits/stl_algo.h:2877: instantiated from 'void std::__merge_sort_loop(_RandomAccessIterat
      /usr/include/c++/4.3/bits/stl_algo.h:2962: instantiated from 'void std::__merge_sort_with_buffer(_Random
      /usr/include/c++/4.3/bits/stl_algo.h:3013: instantiated from 'void std::__stable_sort_adaptive(_RandomAcces
      /usr/include/c++/4.3/bits/stl_algo.h:4984: instantiated from 'void std::stable_sort(_RAIter, _RAIter) [wit
____
```

Exemple sans concept : Sortie III

Template Fails

...

Exemple avec concept : Sortie

Template Fails

```
      boost/concept_check.hpp: In destructor 'boost::LessThanComparable<TT>::~
      LessThanComparable() [with TT = std::complex<float>]':
      boost/concept/detail/general.hpp:29:   instantiated from 'static void boost::
      concept::requirement<Model>::failed() [with Model = boost::
      LessThanComparable<std::complex<float> >]'
      boost/concept/requires.hpp:30:   instantiated from 'boost::_requires_<void
      (*)(boost::LessThanComparable<std::complex<float> >)>'
      bad_error_eg.cpp:8:   instantiated from here
      boost/concept_check.hpp:236: error: no match for 'operator<' in '((boost::
      LessThanComparable<std::complex<float> >*)this)->boost::
      LessThanComparable<std::complex<float> >::a < ((boost::
      LessThanComparable<std::complex<float> >*)this)->boost::
      LessThanComparable<std::complex<float> >::b'
```

Concept : utilisation

Template Fails

```
// classe CTest necessite T == T
template<class T>
class CTest {
  BOOST_CONCEPT_ASSERT((boost::EqualityComparable<T>));
};

// fonction  necessite T == T
template<class T>
void CFonction() {
  BOOST_CONCEPT_ASSERT((boost::EqualityComparable<T>));
}

class C {};

int main() {
  CTest<std::complex<float> > t; // ok
  CTest<C> t2; // fails
  CFonction<std::complex<float> >(); // ok
  CFonction<C>(); // fails
  return 0;
}
```

Concept : plusieurs contraintes

Template Fails

```
template<class T>
class CTest {
  // liste de contrainte
  BOOST_CONCEPT_ASSERT((boost::EqualityComparable<T>));
  BOOST_CONCEPT_ASSERT((boost::LessThanComparable<T>));
};

template<class T>
void CFonction() {
  // liste de contrainte
  BOOST_CONCEPT_ASSERT((boost::EqualityComparable<T>));
  BOOST_CONCEPT_ASSERT((boost::LessThanComparable<T>));
}

int main() {
  //CTest<std::complex<float> > t; //fails
  CTest<float> t2;
  //CFonction<std::complex<float> >(); // fails
  CFonction<float>();
  return 0;
}
```

Concept : 1er problemes

- Les contraintes sont exprimées dans le corps de la classe/ fonctions.
- Classe : ok
- Fonctions : ... pas idéal

$\Rightarrow$ BOOST_CONCEPT_REQUIRES

Concept Require

Concept Require

```
template<typename Iter>
// BOOST_CONCEPT_REQUIRES(
// (( contrainte1 )) (( contrainte2 )) ... (( contrainteX )),
// ( type_de_retour ))
// Note: les parent\'eses sont importantes
BOOST_CONCEPT_REQUIRES(
((boost::Mutable_RandomAccessIterator<Iter>))
((boost::LessThanComparable<typename boost::Mutable_RandomAccessIterator<Iter>::value_type>)),
(void))
my_sort(Iter , Iter );

int main() {
  std::vector<int> v;
  std::vector<complex<float> > v2;
  my_sort(v.begin(),v.end());
  my_sort(v2.begin(),v2.end());
  return 0;
}
```

Ecriture de Concept

- On connait les outils pour les utiliser et en partie les écrire.
- Comment Ecrire les concepts ?

Concept : Ecriture de nouvelles contraintes

Concept Part1

```
template <class X>
// on herite de contraintes existante pour beneficier de ces contraintes
// un InputIterator est donc Assignable et Comparable
class InputIterator: Assignable<X>, EqualityComparable<X> {
  typedef std::iterator_traits<X> t;
  public:
  typedef typename t::value_type value_type;
  typedef typename t::difference_type difference_type;
  typedef typename t::reference reference;
  typedef typename t::pointer pointer;
  typedef typename t::iterator_category iterator_category;

  // On verfie les contraintes necessaires .
  BOOST_CONCEPT_ASSERT((SignedInteger<difference_type>));
  BOOST_CONCEPT_ASSERT((Convertible<iterator_category, std::input_iterator_tag>));
};
```

Comment tester plus en détails, notamment les contraintes non existantes ?

Concept Usage

`BOOST_CONCEPT_USAGE` permet de tester du code.

Concept Usage

```
BOOST_CONCEPT_USAGE(InputIterator) {
  X j(i);   // necessite constructeur de copie
  same_type(*i++,v); // le deferencement doit avoir le meme type que v ( detailler fonctionnemnt)
  X& x = ++j; // doit avoir ++ et ++ doit renvoye une reference.
}

 private :
X i;
value_type v;

template <typename T>
void same_type(T const&, T const&);
```

Important : X et v sont des attributs sont des attributs

Concept Last

- Bien définir les concepts.
- voir `boost::concept` reference pour les concepts existants.

Lambda

- Support des fonctions anonymes et lambda function.
- Utilisable avec les autres librairies de Boost et la STL.
- Meme syntaxe que bind `_x` représente le Xieme paramètre.

Lambda base

Lambda Exemple

```
int i=0; int j=0;
int m=2;

// creer la fontion anonyme a un argument (_1) {return _1+2} et l'appel
// avec i en parametre
cout << (_1+2)(i) <<endl;
// une fonction lambda peut etre passe dans un std :: function
function<int (int)> f2= _1+2;
cout << f2(i) << endl;
// sequence d' instruction dans les fonctions anonymes: separateur :",", il est conseiller de parenteser .
// la derniere instruction est renvoye
// i et j n'ont modifie, ce sont des copie qui sont passe dans la fonction anonyme
cout << (_1+=2,_2+=1, _1+_2)(i,j) << endl;
// see for taking a reference on i .
```

Lambda

- Support des tests.
- Support des boucles
- Support des switch
- .... avec 2 syntaxes disponibles !
- var() sert a referencé une variable externe.
- constant sert a signaler que l'argument est constant.

Lambda base

Lambda Exemple

```
int i=0; int m=2;

i=2;
i=if_then_else_return((var(i)==2), // i_t_e_r(test, alors, sinon)
        (cout << constant("ok\n"),_1=4),
        (cout << constant("not_ok\n"),_1=5))(m);
cout << m << endl;
cout << i << endl;

i=3;
if_(var(i)==2)[(cout << constant("ok\n"),_1=4)] // if_(test)[alors].else_(sinon)
    .else_[(cout << constant("not_ok\n"),_1=5)](m);
cout << m << endl;
cout << i << endl;
```

Lambda base

- utilisable avec la STL et boost : permet de booster la STL.

Lambda base

Lambda Exemple

```
void fill(vector<int> &v) {
  v.push_back(1); v.push_back(2);
  v.push_back(3); v.push_back(4);
}

int incr(int x,int y) { return x+y; }

vector<int> a; fill(a);
for_each(a.begin(), a.end(), _1+=1); // incremente le contenu de a
for_each(a.begin(), a.end(), _1=bind(incr,_1,2)); //  a[step]=incr(a[step]+2)
for_each(a.begin(), a.end(), cout<<_1 << constant('\n')); // affiche le contenu

i=0;
vector<int> b; fill(b);
// si (i est pair), b[step]=0 sinon b[step]=b[step]*2 Note: i incrementer a chaque tour
for_each(b.begin(), b.end(),
        (if_(var(i)%2==0)
           [(_1=0)].else_[(_1=_1*2)],
        (var(i)++)));
for_each(b.begin(), b.end(),cout << _1<< constant('\n'));
```

Lambda base

- utilisable avec la STL et boost : permet de booster la STL.
- $\Rightarrow$ Permet la programmation fonctionnel et de generalise les pointeur de fonctions.

Lambda base

Lambda Exemple

```
vector<int> c; fill(c);
apply(c, (_1+_2), 2); // passe la fonction a apply
for_each(c.begin(), c.end(),cout << _1<< constant('\n'));

apply(c, (_1*_2), 2);
for_each(c.begin(), c.end(),cout << _1<< constant('\n'));
```

Tour 1 : String

- conversion
- Formattage a la C typesafe
- Algo

Tour 2 : Container

- Bimap
- Graph
- Librairie traitement image d'adobe
- Container Intrusive

Tour 4 : Metaprogramming

- MPL
- Fusion
- Proto

Tour 5 : Thread, Reseaux

- Asio
- Interporcess
- MPI
- Thread

Tour 5 : Et plein d'autres choses

- Preprocesseur C
- Parametres nommés
- Retour Optionel
- Scope Exit
- ...

Plan

Boost

- Librairie très puissantes et complète.
- Influence importante pour les extensions de la librairie standard.
- Influence importante pour les extensions du langage.

C++0x

- Prochaine version C++ : C++0A.
- Enorme Mise a jour du langage
- Enorme mise de la librairie (TR1 notamment).

C++0x II

- support des concepts, lambda, typeof dans le langage
- Beaucoup de chose obsolete dans Boost
- Cependant Boost reste interessant (y compris ce qui est presente aujourd'hui) : le support des compiler pour la norme C++0X n'est pas encore la .