UE MOBJ [4L103]

Jean-Paul CHAPUT
Jean-Paul.Chaput@lip6.fr

SESI

2017-2018

II.3.2

```
int main ( int argc, char* argv ) {
  Complex a ( 1, 1 );
  Complex b ( 2, 2 );
  Complex c ( 3, 3 );
  Complex m;

  d = ~a;
  m = a*b + c;
  cout << "result:" << m << endl;

  return 0;
}
```

II.3.2

```
class Complex {
  public:
    Complex  operator~ () const;
    Complex  operator+ ( const Complex& ) const;
    Complex  operator* ( const Complex& ) const;
    Complex& operator= ( const Complex& );
  // ...
};

Complex  Complex::operator~ () const
           { return Complex(re_,-im_); }
Complex  Complex::operator+ ( const Complex& b ) const
           { Complex r; r.re_ = re_+b.re_; r.im_ = im_+b.im_;
             return r; }
Complex  Complex::operator* ( const Complex& b ) const
           { Complex r; r.re_ = re_*b.re_ - im_*b.im_;
                        r.im_ = im_*b.re_+re_*b.im_;
             return r; }
Complex& Complex::operator= ( const Complex& b )
           { re_ = b.re_; im_ = b.im_; return *this; }
```

II.3.3

```
namespace std {
  class ostream {
    public:
      ostream& operator<< (ostream&,int);
      ostream& operator<< (ostream&,float);
      ostream& operator<< (ostream&,const char*);
      ostream& operator<< (ostream&,const std::string&);
  };
}

int main(int argc, char* argv) {
  int            i = 1;
  float          j = 2.0;
  std::string    k = "trois";

  std::cout << "i:" << i << "␣j:" << j << "␣k:" << k << std::endl;
//std::cout.operator<<( "i:").operator<<(i)
//         .operator<<(" j:").operator<<(j);
//         .operator<<(" k:").operator<<(k);
//         .operator<<(std::endl);
}
```

II.3.3

```
ostream& operator<< ( ostream& o, const Complex& a )
{ a.print(o); return o; }

int main ( int argc, char* argv ) {
  Complex a ( 1, 1 );
  Complex d;

  d = ~a;
  cout << "conjugate:" << d << endl;

// operator( cout.operator<<( "conjugate:"), d )
//    .operator<<( endl );

  return 0;
}
```

III.1

```
template<class T>
  T tableMax ( T* table, int size ) {
    T max = table[0];
    for ( int i=1 ; i<size ; i++ ) {
      if (table[i] > max) max = table[i];
    }
    return max;
  }
```

III.1

```
int main ( int argc, char* argv ) {
  int  t1[4] = { 0, 1, 2, 3 };
  int  t2[2] = { 4, 5 };
  char s1[5] = "abcd";

  cout << tableMax<int >(t1,4) << endl;
  cout << tableMax<char>(s1,4) << endl;

  cout << tableMax(t2,2) << endl;
}
```

III.1

```
class Element {
  private:
    long  value_;
  friend bool operator> ( Element& lhs, Element& rhs );
};

bool operator> (Element& lhs, Element& rhs)
{ return lhs.value_ > rhs.value_; }

int main ( int argc, char* argv ) {
  Element t1[4] = { 0, 1, 2, 3 };

  cout << tableMax(t1,4) << endl;
}
```

III.2.1

```
template<typename T>
class Vector {
  private:
           T*        table_;
           size_t    size_;
           size_t    capacity_;
  private:
           void      resize_     ( size_t newcapacity );
  public:
                     Vector      ();
                     Vector      ( const Vector& );
                    ~Vector      ();
  public:
    inline size_t    size        ();
    inline size_t    capacity    ();
           void      reserve     ( size_t );
           void      push_back   ( T );
           void      pop_back    ();
           T&        back        ();
    const  T&        back        () const;
           T&        operator[]  ( size_t );
};
```

III.2.2

```
template<typename T>
Vector<T>::Vector () : table_   (NULL)
                     , size_    (0)
                     , capacity_(0)
{ }

template<typename T>
Vector<T>::~Vector ()
{ if (table_) delete [] table_; }

template<typename T>
T& Vector<T>::operator[] (size_t index)
{
  static T notFound;
  if (index < size_) return table_[index];
  return notFound;
}
```

III.2.2

```
template<typename T>
void  Vector<T>::resize_ ( size_t newcapacity )
{
  if (newcapacity <= capacity_) {
    cerr << "[ERROR]␣Vector::resize_()␣cowardly␣refusing␣to␣shrink␣("
         << capacity_ << "␣to␣" << newcapacity << ")" << endl;
    return;
  }

  T* newtable = new T [newcapacity];
  for ( size_t i=0 ; i<size_ ; ++i ) newtable[i] = table_[i];

  if (table_) delete [] table_;

  table_    = newtable;
  capacity_ = newcapacity;
}
```

III.2.2

```
template<typename T>
void  Vector<T>::push_back ( T element ) {
  if (size_ == capacity_) {
    size_t newcapacity = (capacity_)?(capacity_*2):2;
    resize_( capacity_ );
  }
  table_[ size_++ ] = element;
}

template<typename T>
void  Vector<T>::pop_back () { if (size_) --size_; }

template<typename T>
T&  Vector<T>::back () {
  static T notFound;
  return (size_) ? table_[size_-1] : notFound;
}
```

III.2.2

```
#include "Vector.h"
void printVectorInt ( const Vector<int>& v ) {
  for ( size_t i=0 ; i<v.size() ; ++i )
    cout << "v[" << i << "]␣=␣" << v[i] << endl;
}

int main (int argc, char* argv) {
  Vector<int> v;
  for ( size_t i=0 ; i<10 ; ++i )
    v.push_back(i);
  printVectorInt( v );
  return 0;
}
```

III.3.1

```
#include <vector>
#include "Box.h"

int main ( int argc, char* argv[] ) {
  vector<Box> boxes;

  Box b1 ( "b1", 0, 0, 10, 10 );
  Box b2 ( "b2", 5, 5, 20, 20 );

  boxes.push_back( b1 );  // L'element est copie.
  boxes.push_back( b2 );
  for ( size_t i=0 ; i<boxes.size() ; ++i )
    cout << "boxes[" << i << "]␣=␣" << boxes[i] << endl;
}
```

III.3.1

```
#include <vector>
#include "Box.h"

int main ( int argc, char* argv[] ) {
  vector<Box*> boxes;

  Box b1 ( "b1", 0, 0, 10, 10 );
  Box b2 ( "b2", 5, 5, 20, 20 );

  boxes.push_back( &b1 );   // L'element *n'est pas*
                            // copie.
  boxes.push_back( &b2 );
  for ( size_t i=0 ; i<boxes.size() ; ++i )
    cout << "boxes[" << i << "]␣=␣" << *boxes[i] << endl;
}
```

III.3.2

```
  bool    empty      ();
  size_t  size       ();
  void    resize     ( size_t size );
  size_t  capacity   ();
  size_t  max_size   ()
  void    clear      ();
  T&      front      ();
  T&      back       ();
  void    push_back  ( const T& element );
  void    pop_back   ();
// Pas dans <vector>.
  void    push_front ( const T& element );
  void    pop_front  ();
```

III.4

```
int   i     = 0;
char* table = new char [10];
for ( char* p=table ; p!=table+10 ; ++p ) // Remplissage
  (*p) = '0'+i;
for ( char* p=table ; p!=table+10 ; ++p ) // Affichage
  std::cout << *p;
std::cout << std::endl

// Le meme code, avec des iterateurs.
vector<char> v;
for ( int j=0 ; j<10 ; ++j )             // Remplissage
  v.push_back('0'+j);
vector<char>::iterator iv = v.begin();
for ( ; iv != v.end() ; ++iv )           // Affichage
  std::cout << (*iv);
std::cout << std::endl
```

III.4.1

```
vector<char>::iterator  beg = v.begin();
vector<char>::iterator  end = v.end();
vector<char>::iterator  pos = v.insert(beg,'R');
vector<char>::iterator  pos = v.erase (beg);

// Parcours inverse.
vector<char>::reverse_iterator iv = v.rbegin();
for ( ; iv != v.rend() ; ++iv )
  std::cout << (*iv);
std::cout << std::endl
```

III.4.4

```
std::map<std::string,Box> m;
m["machin"] = Box(0,0,1,1);
m["bidule"] = Box(0,0,2,2);
m["truc"  ] = Box(0,0,3,3);
std::map<std::string,Box>::iterator im = m.find("truc");
if (im != m.end()) {
  std::cout << "Key:"   << (*im).first
            << "␣value" << (*im).second << endl;
  m.erase(im);
}
for ( im = m.first() ; im != m.end() ; ++im )
  std::cout << "Key:"   << (*im).first
            << "␣value" << (*im).second << endl;
```

III.4.5

```
void myFind ( const std::map<std::string,Box>& m ) {
  std::map<std::string,Box>::const_iterator im = m.find("truc");
  if (im != m.end())
    std::cout << "Key:"   << (*im).first
              << "␣value" << (*im).second << endl;
  else
    std::cout << "Not␣found" << std::endl;
}

std::map<std::string,Box> m;
m["machin"] = Box(0,0,1,1);
m["bidule"] = Box(0,0,2,2);
m["truc"  ] = Box(0,0,3,3);
myFind ( m );
```

III.4.6

```
class CompareByY2 {
  public:
    bool operator() ( const Box& lhs, const Box& rhs )
    { return lhs.getY2() < rhs.getY2(); }
};


std::vector<Box> v;
v.push_back( Box(0,0,3,3) );
v.push_back( Box(0,0,2,2) );
v.push_back( Box(0,0,1,1) );

CompareByY2 cmp;      // cmp est un *objet* ...
if ( cmp(v[0],v[1]) ) // qui peut etre appele comme une *fonction*.
  cout << "v[0]␣est␣inferieur␣a␣v[1]" << endl;

sort( v.begin(), v.end(), CompareByY2() );
```