1 Usage of member functions

1.1 Rvalues, lvalues, references, and const qualifier

We have seen that a lvalue is an expression corresponding to value and its location in memory, which means that it can be modified. A rvalue is just a value and one can not modify it.

Thus, passing a reference to a rvalue is meaningless. Nevertheless, for performances, we can pass an intermediate result as a `const` reference.

```
1  double nothing1(double x) {}
2  double nothing2(double &x) {}
3  double nothing3(const double &x) {}
4
5  int main(int argc, char **argv) {
6    nothing1(3+4);
7    nothing2(3+4);
8    nothing3(3+4);
9  }
```

```
1  /tmp/chose.cc: In function ‘int main(int, char **)’:
2  /tmp/chose.cc:7: initializing non-const ‘double &’ with ‘int’ will use a temporary
3  /tmp/chose.cc:2: in passing argument 1 of ‘nothing2(double &)’
```

1.2 Member functions can be called through standard functions

A very elegant way to offer nice functions an operators, and still use the data-field protection principles is to design a set of member functions with privileged access to the data field, and then, a set of standard functions and operators, which call the member functions.

```
1  #include <iostream>
2
3  class ForSureNonNull {
4    double value;
5  public:
6    ForSureNonNull(double v) {
7      if(v == 0) { cerr << "ARE YOU CRAZY?!?!!?\n"; abort(); }
8      value = v;
9    }
10
11   double getValue() const {
12     return value;
13   }
14 };
15
16 double sum(const ForSureNonNull &n1, const ForSureNonNull &n2) {
17   return n1.getValue() + n2.getValue();
18 }
19
20 int main(int argc, char **argv) {
21   ForSureNonNull x(15);
22   double k = sum(x, x);
23 }
```

1.3 Overloading the << operator

The usage of `cout` is very practical. The operators re-definition allows us to define our own << operator.

As we have seen, `cout` is of type `ostream`, and an expression such as :

```
1  cout << a << b << c;
```

Will be evaluated from left to right as :

```
1  ( (cout << a) << b) << c;
```

so, the left operand of << will always be an `ostream` and the right operand will be whatever we want :

```
1  ostream &operator << (ostream &s, const ForSureNonNull &x) {
2    return (s << x.getValue());
3  }
```