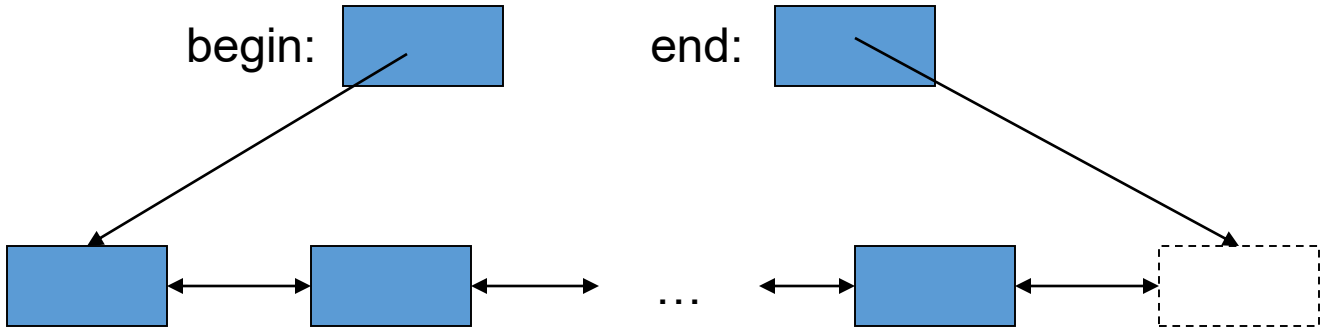
Библиотека стандардних шаблона (Standard template library – STL)
Мапе и алгоритми

Основни модел

- Пар итератора одређује секвенцу
  - Почетак (beginning) – показује на први елемент (ако га уопште има)
  - Крај (end) – показује **иза последњег** елемента



- Итератор је тип који подржава итераторске операције:
  - == Да ли итератори показују на исти елемент? (за одређивање краја)
  - * Добави вредност
  - ++ Пређи на следећи елемент
- Итератори могу подржавати још неке операције, нпр.:

  --, +, [ ] ...

Пример: Сумирање

```
template<class In, class T>
T accumulate(In first, In last, T init)
{
 while (first != last) {
    init = init + *first;
    ++first;
 }
 return init;
}
```


```
int sum = accumulate(v.begin(), v.end(), 0);
```

Пример: Сумирање

```
void f(vector<double>& vd, int* p, int n)
{
 double sum = accumulate(vd.begin(), vd.end(), 0.0);
 // шта ако ставимо 0.0F?

 int si = accumulate(p, p+n, 0);
 // p+n је (отприлике) исто што и &p[n]

 long s1 = accumulate(p, p+n, long(0)); // или 0L
 double s2 = accumulate(p, p+n, 0.0);
 long s3 = accumulate(p, p+n, 0); // ?

 //ово је чест идиом: јасније се види тип
 double ss = 0;
 ss = accumulate(vd.begin(), vd.end(), ss);
}
```

Пример: Сумирање

```
// нећемо само сабирање да користимо
// хоћемо било коју операцију

template<class In, class T, class BinOp>
T accumulate(In first, In last, T init, BinOp op)
{
 while (first != last) {
     init = op(init, *first); // “init op *first”
     ++first;
 }
 return init;
}
```

Пример: ~~Сумирање~~ - Производ

```
void f(list<double>& ld)
{
  double product = accumulate(ld.begin(), ld.end(), 1.0,        ???          );
  //
  //
  //
  // …
}
```

Пример: ~~Сумирање~~ - Производ

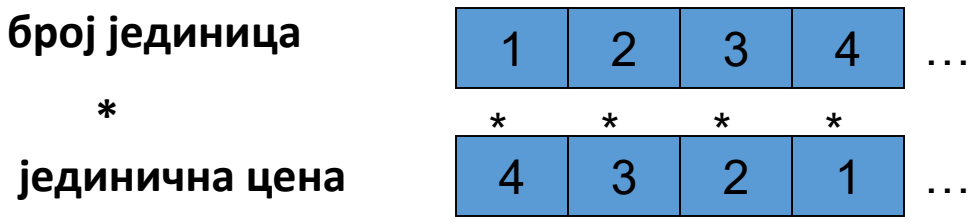
```
#include <functional>

void f(list<double>& ld)
{
  double product = accumulate(ld.begin(), ld.end(), 1.0, multiplies<double>());
  // исто што и
  // multiplies<double> mulOp;
  // double product = accumulate(ld.begin(), ld.end(), 1.0, mulOp);
  // …
}


// multiplies је део стандардне библиотеке
```

Пример: Акумулирање

```
struct Record {
  int units;
  double unit_price;
  // …
};

double price(double v, const Record& r)
{
  return v + r.unit_price * r.units;
}

void f(const vector<Record>& vr) {
  double total = accumulate(vr.begin(), vr.end(), 0.0, price);
  // …
}
```

Скаларни производ

```
template<class In, class In2, class T>
T inner_product(In first, In last, In2 first2, T init)
{
 while(first != last) {
      init = init + (*first) * (*first2);
       ++first;
       ++first2;
 }
 return init;
}
```



Пример

```
vector<double> dow_price;
dow_price.push_back(81.86);
dow_price.push_back(34.69);
dow_price.push_back(54.45);
// …
vector<double> dow_weight;
dow_weight.push_back(5.8549);
dow_weight.push_back(2.4808);
dow_weight.push_back(3.8940);
// …
double dj_index = inner_product(
     dow_price.begin(), dow_price.end(),
     dow_weight.begin(),
     0.0);
```

Пример

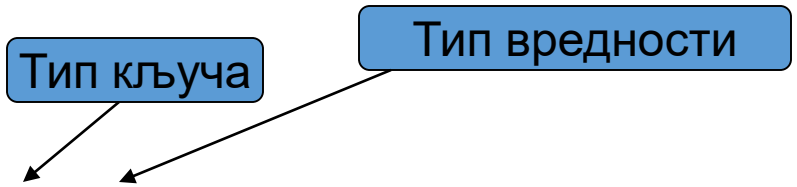
```
vector<double> dow_price = {
  81.86, 34.69, 54.45,
  // …
};
vector<double> dow_weight = {
  5.8549, 2.4808, 3.8940,
  // …
};

double dj_index = inner_product(
      dow_price.begin(), dow_price.end(),
      dow_weight.begin(),
      0.0);
```

Подизање скаларног производа

```
template<class In, class In2, class T, class BinOp, class BinOp2 >
T inner_product(In first, In last, In2 first2, T init, BinOp op, BinOp2 op2)
{
  while (first != last) {
       init = op(init, op2(*first, *first2));
       ++first;
       ++first2;
  }
  return init;
}
```

Мапа (пресликавање, асоцијативни низ)

- Код вектора, индексира се целим бројем
- Код мапе, може се индексирати било којим типом




```
int main()
{
 map<string, int> words;
 string s;
 while (cin >> s)
  ++words[s]; // words[s] += 1;

 for (const auto&  p : words)
       cout << p.first << ": " << p.second << "\n";
}
```

- Шта ради овај програм?

Улаз за програм који броји појављивање речи

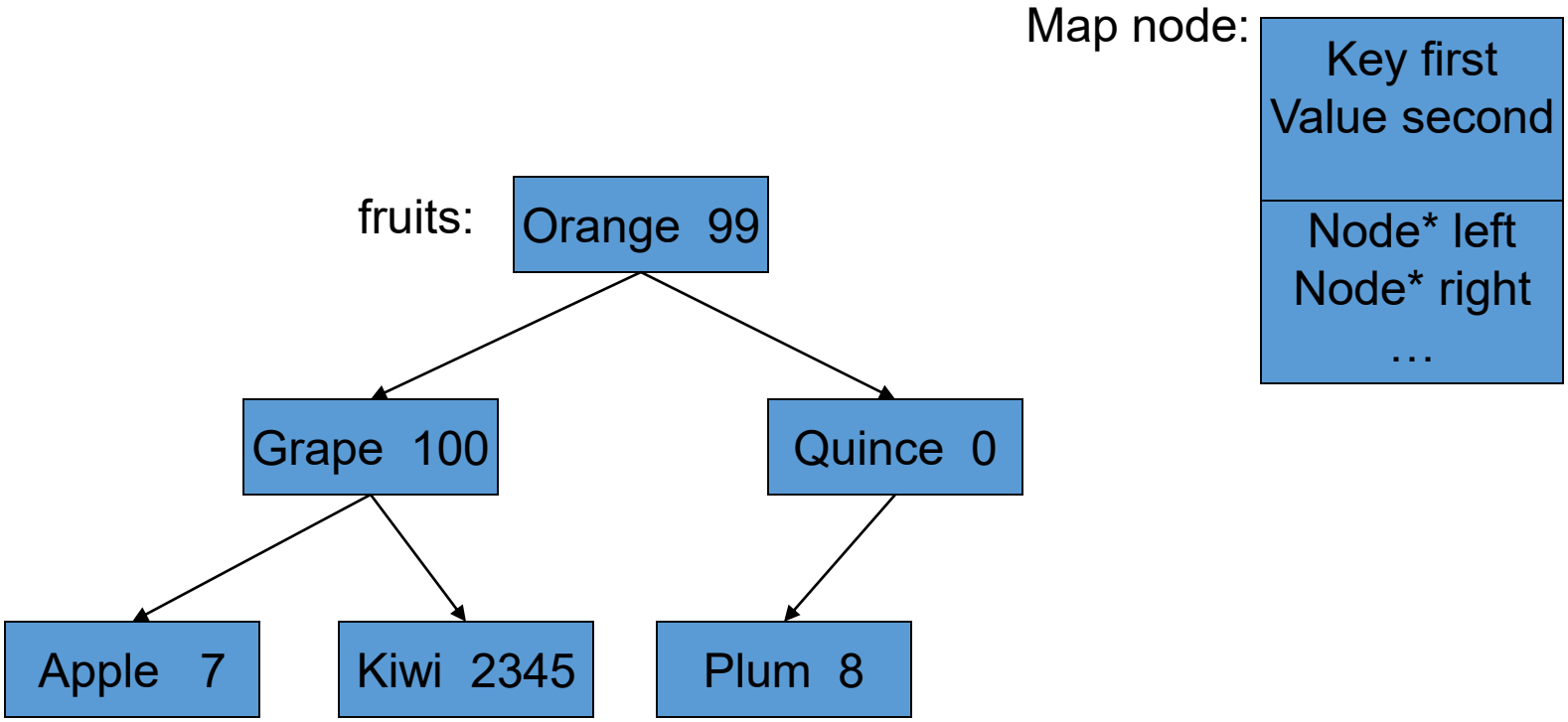
This lecture and the next presents the STL (the containers and algorithms part of the C++ standard library). It is an extensible framework dealing with data in a C++ program. First, I present the general ideal, then the fundamental concepts, and finally examples of containers and algorithms. The key notions of sequence and iterator used to tie containers (data) together with algorithms (processing) are presented. Function objects are used to parameterize algorithms with “policies”.

Излаз

(data): 1
(processing): 1
(the: 1
C++: 2
First,: 1
Function: 1
I: 1
It: 1
STL: 1
The: 1
This: 1
a: 1
algorithms: 3
algorithms.: 1
an: 1
and: 5
are: 2
concepts,: 1
containers: 3
data: 1
dealing: 1
examples: 1
extensible: 1
finally: 1
framework: 1
fundamental: 1
general: 1
ideal,: 1
in: 1
is: 1
iterator: 1
key: 1
lecture: 1
library).: 1
next: 1
notions: 1
objects: 1
of: 3
parameterize: 1
part: 1
present: 1
presented.: 1
presents: 1
program.: 1
sequence: 1
standard: 1
the: 5
then: 1
tie: 1
to: 2
together: 1
used: 2
with: 3
“policies”.: 1

Мапа

- Мапа је имплементирана као балансирано бинарно стабло претраге (уређено балансирано бинарно стабло)
  - Релација поретка је **<**



Мапа

```
// уочите сличност са вектором и листом

template<class Key, class Value> class map {
  // …
  using value_type = pair<Key,Value>;
  using iterator = ???;
  using const_iterator = ???;

  iterator begin();
  iterator end();

  Value& operator[ ](const Key&);
  iterator find(const Key& k);

  void erase(iterator p);
  pair<iterator, bool> insert(const value_type&);
};
```

Примери употребе мапе

```
double alcoa_price = dow["AA"];
double boeing_price = dow["BO"];


if (dow.find("INTC") != dow.end())
 cout << "Intel is in the Dow\n";


for (const auto& p : dow) {
 const string& symbol = p.first;
 cout << symbol  << '\t' << p.second << '\t' <<
    dow_name[symbol] << '\n';
}
```

Скаларни производ са мапом

```
double value_product(
 const pair<string,double>& a,
 const pair<string,double>& b)
{
 return a.second * b.second;
}

double dj_index =
 inner_product(dow.begin(), dow.end(),
               dow_weight.begin(),
               0.0,
               plus<double>(),
               value_product
          );
```

Алгоритми

- Алгоритми у STL-у
  - Раде над једном или више секвенци
    - Обично саопштених у облику парова итератора
  - Примају једну или више параметризујућих операција
    - Обично у облику функтора (функцијских објеката)
    - Прихватају се и обичне функције
  - Грешка или неуспех се обично пријављују враћањем итератора на крај секвенце

Корисни алгоритми у STL-у

- `r = find(b, e, v)`
- `r = find_if(b, e, p)`
- `x = count(b, e, v)`
- `x = count_if(b, e, p)`
- `sort(b, e)`
- `sort(b, e, p)`
- `copy(b, e, b2)`
- `unique_copy(b, e, b2)`
- `merge(b, e, b2, e2, r)`
- `r = equal_range(b, e, v)`
- `equal(b, e, b2)`

Копирање

```
template<class In, class Out>
Out copy(In first, In last, Out res)
{
  while (first != last) *res++ = *first++;
                  // скраћени запис овог кода:
                  // *res = *first; ++res; ++first;
  return res;
}

void f(vector<double>& vd, list<int>& li)
{
  if (vd.size() < li.size()) error("target container too small");
  copy(li.begin(), li.end(), vd.begin());

  sort(vd.begin(), vd.end());
  // …
}
```

Итератори улазног и излазног тока

```
ostream_iterator<string> oo(cout);

*oo = "Hello, ";  // cout << "Hello, "
++oo;             // спреми се за следећи испис
*oo = "world!\n"; // cout << "world!\n"

istream_iterator<string> ii(cin);

string s1 = *ii; // cin >> s1
++ii;            // спреми се за следеће читање
string s2 = *ii; // cin >> s2
```

Прављење речника – коришћењем вектора

```
int main()
{
   string from, to;
   cin >> from >> to;

   ifstream is(from);
   ofstream os(to);

   istream_iterator<string> ii(is);
   istream_iterator<string> eos;
   ostream_iterator<string> oo(os, "\n"); // сваки пут додај "\n"

   vector<string> b(ii, eos);
   sort(b.begin(), b.end());
   unique_copy(b.begin(), b.end(), oo);
}
```

Прављење речника – коришћењем вектора

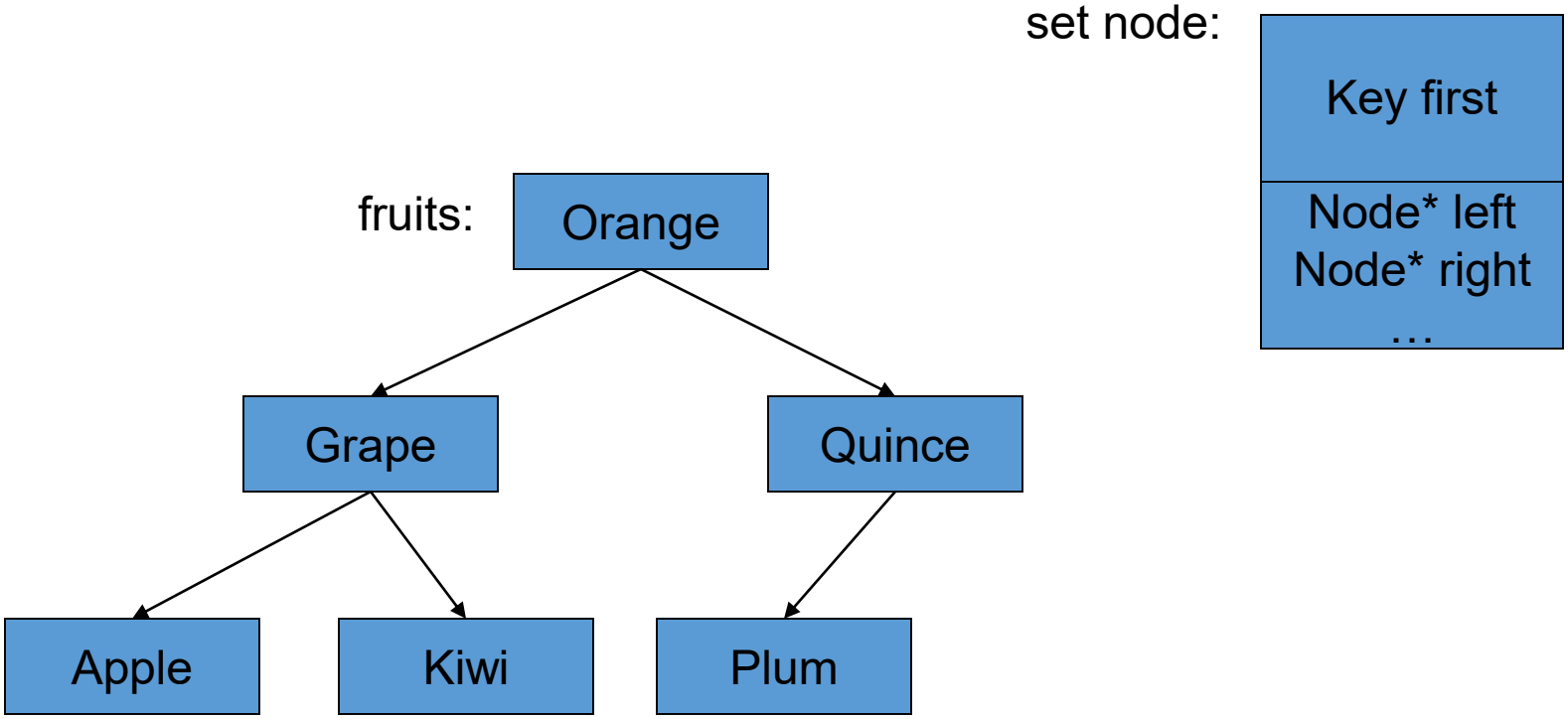
- Имамо додатног, беспотребног посла:
  - Зашто уопште уписивати дупликате у вектор?
  - Зашто уређујемо вектор?
  - Зашто избацујемо дупликате при испису?
- Много би боље било да сваку реч коју прочитамо одмах проверимо да ли има дупликат и ставимо на право место у речник.

Прављење речника – коришћењем скупа (сета)

```
int main()
{
   string from, to;
   cin >> from >> to;

   ifstream is(from);
   ofstream os(to);

   istream_iterator<string> ii(is);
   istream_iterator<string> eos;
   ostream_iterator<string> oo(os, "\n");

   set<string> b(ii, eos);
   copy(b.begin(), b.end(), oo);
}
```

Сет (скуп)

- Сет је такође балансирано бинарно стабло претраге
  - Релација поретка је <



copy_if()

```
template<                                   >
    copy_if(                                    )
{









}
```

copy_if()

```
template<class   , class    , class     >
    copy_if(   first,    last,     dest,       p)
{




}
```

copy_if()

```
template<class In, class Out, class Pred>
Out copy_if(In first, In last, Out dest, Pred p)
{

                                        !=, *, ++, ()


}
```

copy_if()

```
template<class In, class Out, class Pred>
Out copy_if(In first, In last, Out dest, Pred p)
{
 while (first != last) {
     if (p(*first)) { *dest = *first; ++dest; }
     ++first;
 }
 return dest;
}
```

copy_if()

```
template<class In, class Out, class Pred>
Out copy_if(In first, In last, Out dest, Pred p)
{
 while (first != last) {
     if (p(*first)) *dest++ = *first;
     ++first;
 }
 return dest;
}
```

Неки стандардни функтори

- <functional>
  - Бинарни
    - plus, minus, multiplies, divides, modulus
    - equal_to, not_equal_to, greater, less, greater_equal, less_equal, logical_and, logical_or
  - Унарни
    - negate
    - logical_not