Введение: библиотека C++

Гаевой, Казменко, Макаров

Санкт-Петербургский Государственный Университет

Понедельник, 21 сентября 2020 года

Ввод и вывод

- Стандартный поток ввода: `cin`
- Стандартный поток вывода: `cout`
- `int n; double x; string s; cin >> n >> x >> s;`
- Пока есть числа: `int x; while (cin >> x) {...}`
- `cout << n << endl;`
- `cout << x << endl;`

В задачах часто просят, например, 9 точных знаков после точки, или относительную погрешность не более 10^{-9}.
Попробуем:

- `double y = 123456789; cout << y << endl;`
- Ответ: `1.23457e+008` — что!?
- Форматирование перед выводом:
  `cout << setprecision (10) << fixed;`
- Ответ: `123456789.0000000000`

Контейнеры

Как организовать хранение данных? Зависит от того, какие операции мы хотим с ними эффективно выполнять.

- Получать элемент по индексу: `vector`
- Добавлять элементы, удалять их и проверять наличие: `set`
- Менять сразу все элементы в заданном диапазоне индексов: ...дерево отрезков?

Посмотрим на основные типы контейнеров в стандартной библиотеке C++ и примеры их использования.

Структура pair

Пара из двух значений любых типов:

- `#include <utility>` (или `#include <algorithm>`)
- `using namespace std;` (или `std::pair`)
- `pair <int, double> p;`
- `p.first = 123;`
- `p.second = 456.789;`
- `pair <int, double> q = {1, 2.5};`
- `if (p < q) {...}`
- пример — точка на плоскости: `pair <int, int> point;`

Более сложные типы:

- `pair <pair <int, int>, string> data;`
- `if (p.first.first > p.first.second) {...}`
- или: `struct Data {int u, v; string id; ...};`

Контейнер vector

Расширяемый массив из значений одинакового типа:

- `#include <vector>`
- `vector <int> v;`
- `v.push_back (1);`
- `vector <int> u (10, 0);`
- `vector <int> w = {1, 2, 3};`
- `if (u < w) {...}`
- `if (v[0] > w[1]) {...}`

Расширяется за линейное время:

- `for (int i = 0; i < n; i++) {v.push_back (i);}`
- $1 + 2 + 3 + \ldots + n = O(n^2)$
- Заводим в 2 (или 1.5) раза больше памяти, когда кончилась
- $1 + 2 + 4 + 8 + 16 + \ldots$ (до первого $\geqslant n$) $= O(n)$

Строка `string`

Строка:

- `#include <string>`
- `string str = "";`
- `str += 'a';` — добавление символа в конец
- `string add (3, 'v');`
- `str = str + add;` — конкатенация
- `str += add;` — почему так быстрее?
- `if (str == "abc") {...}`
- `if (str < add) {...}` — лексикографическое сравнение
- `str = substr (str, start, length);` — подстрока

Контейнер `set`

Множество объектов:

- `#include <set>`
- `set <int> s;`
- `s.insert (1);`
- `s.insert (2); s.insert (2);` — получится одна двойка
- Или: `multiset <int> ms;`
- Поиск: `if (s.find (x) != s.end ())`
- Поиск, короче: `if (s.count (x))`
- Поиск, дольше: `if (ms.count (x))`
- `s.erase (1);`
- Удалить одну двойку: `ms.erase (ms.find (2));`

Внутри — сбалансированное двоичное дерево поиска (red-black tree), все операции происходят за логарифм от размера.

Контейнер map

Отображение (ключ → значение):

- `#include <map>`
- `map <string, int> m;`
- `m["one"] = 1;`
- `m["two"] = 2;`
- Поиск — как в `set`: `if (m.find ("five") != m.end ())`
- `multimap <string, int> mm;`

Внутри — `set` из ключей с дополнительным полем для значения.

Пример

```
1
2
3   #include <iostream>
4   #include <utility>
5   #include <vector>
6
7
8   int main () {
9
10
11      int n;
12      std::cin >> n;
13      std::vector <std::pair <int, int> > points (n);
14      for (auto & p : points)
15          std::cin >> p.first >> p.second;
16      std::pair <double, double> center = {0, 0};
17      for (auto & p : points) {
18          center.first += p.first;
19          center.second += p.second;
20      }
21
22      std::cout << center.first / n << " " <<
23                   center.second / n << std::endl;
24      return 0;
25  }
```

Пример

```
1
2
3   #include <iostream>
4   #include <utility>
5   #include <vector>
6   using namespace std;
7
8   int main () {
9
10
11      int n;
12      cin >> n;
13      vector <pair <int, int> > points (n);
14      for (auto & p : points)
15          cin >> p.first >> p.second;
16      pair <double, double> center = {0, 0};
17      for (auto & p : points) {
18          center.first += p.first;
19          center.second += p.second;
20      }
21
22      cout << center.first / n << " " <<
23              center.second / n << endl;
24      return 0;
25  }
```

Пример

```
1
2   #include <iomanip>
3   #include <iostream>
4   #include <utility>
5   #include <vector>
6   using namespace std;
7
8   int main () {
9
10
11      int n;
12      cin >> n;
13      vector <pair <int, int> > points (n);
14      for (auto & p : points)
15          cin >> p.first >> p.second;
16      pair <double, double> center = {0, 0};
17      for (auto & p : points) {
18          center.first += p.first;
19          center.second += p.second;
20      }
21      cout << setprecision (10) << fixed;
22      cout << center.first / n << " " <<
23              center.second / n << endl;
24      return 0;
25  }
```

Пример

```
 1  #include <fstream>
 2  #include <iomanip>
 3  #include <iostream>
 4  #include <utility>
 5  #include <vector>
 6  using namespace std;
 7
 8  int main () {
 9      auto fin = ifstream ("input.txt");
10      auto fout = ofstream ("output.txt");
11      int n;
12      fin >> n;
13      vector <pair <int, int> > points (n);
14      for (auto & p : points)
15          fin >> p.first >> p.second;
16      pair <double, double> center = {0, 0};
17      for (auto & p : points) {
18          center.first += p.first;
19          center.second += p.second;
20      }
21      fout << setprecision (10) << fixed;
22      fout << center.first / n << " " <<
23              center.second / n << endl;
24      return 0;
25  }
```

Итераторы

Проблема:

- Есть C стандартных контейнеров: обычный массив, `vector`, `set`, ...
- Есть A стандартных алгоритмов: сортировка, разворот, поиск, копирование из одного контейнера в другой, ...
- Каждый алгоритм можно применить к нескольким контейнерам
- Как устроить библиотеку, чтобы не пришлось писать код $C \cdot A$ раз?
- И чтобы новый алгоритм и новый контейнер можно было тоже написать один раз

Разворот

```
 1  #include <algorithm>
 2
 3  using namespace std;
 4
 5
 6  void my_reverse (int * start, int * finish) {
 7      while (start < finish) {
 8          --finish;
 9          swap (*start, *finish);
10          ++start;
11      }
12  }
13
14  int main () {
15      int arr [7] = {0, 1, 2, 3, 4, 5, 6};
16      my_reverse (arr, arr + 7);
17      return 0;
18  }
```

Разворот

```
 1  #include <algorithm>
 2  #include <vector>
 3  using namespace std;
 4
 5  template <typename It>
 6  void my_reverse (It start, It finish) {
 7      while (start < finish) {
 8          --finish;
 9          swap (*start, *finish);
10          ++start;
11      }
12  }
13
14  int main () {
15      vector <int> arr = {0, 1, 2, 3, 4, 5, 6};
16      my_reverse (arr.begin (), arr.end ());
17      return 0;
18  }
```

Разворот

```
 1  #include <algorithm>
 2
 3  using namespace std;
 4
 5  template <typename It>
 6  void my_reverse (It start, It finish) {
 7      while (start < finish) {
 8          --finish;
 9          swap (*start, *finish);
10          ++start;
11      }
12  }
13
14  int main () {
15      int arr [7] = {0, 1, 2, 3, 4, 5, 6};
16      my_reverse (arr, arr + 7);
17      return 0;
18  }
```

Итераторы для ввода и вывода

Итераторы можно обобщить не только на указатели:

```
 1  #include <algorithm>
 2  #include <iostream>
 3  #include <iterator>
 4  #include <vector>
 5  using namespace std;
 6
 7  int main () {
 8      vector <int> v;
 9      copy (istream_iterator <int> (cin),
10            istream_iterator <int> (),
11            back_inserter (v));
12      reverse (v.begin (), v.end ());
13      copy (v.begin (),
14            v.end (),
15            ostream_iterator <int> (cout, " "));
16      return 0;
17  }
```

Ввод: 1 2 3, вывод: 3 2 1 .

Ввести в консоли «конец файла»: обычно `Ctrl+Z` или `Ctrl+D`.

Итераторы для ввода и вывода

Итераторы можно обобщить не только на указатели:

```
 1  #include <algorithm>
 2  #include <iostream>
 3  #include <iterator>
 4  #include <vector>
 5  using namespace std;
 6
 7  int main () {
 8      int n;
 9      cin >> n;
10      vector <int> v (n);
11      copy_n (istream_iterator <int> (cin), n, v.begin ());
12      reverse (v.begin (), v.end ());
13      copy_n (v.begin (), n, ostream_iterator <int> (cout, " "));
14      return 0;
15  }
```

Ввод: 3 1 2 3, вывод: 3 2 1 .

Цикл с итератором

Как посетить все элементы контейнера:

```
 1  #include <iostream>
 2  #include <vector>
 3  using namespace std;
 4
 5  int main () {
 6      vector <int> v = {1, 2, 3, 4, 5, 6, 7};
 7      for (vector <int>::iterator it = v.begin ();
 8           it != v.end (); ++it) {
 9           cout << *it << endl;
10      }
11      return 0;
12  }
```

Цикл с итератором

Как посетить все элементы контейнера:

```
 1  #include <iostream>
 2  #include <vector>
 3  using namespace std;
 4
 5  int main () {
 6      vector <int> v = {1, 2, 3, 4, 5, 6, 7};
 7      for (auto it = v.begin ();
 8           it != v.end (); ++it) {
 9           cout << *it << endl;
10      }
11      return 0;
12  }
```

Цикл с итератором

Как посетить все элементы контейнера:

```
 1  #include <iostream>
 2  #include <vector>
 3  using namespace std;
 4
 5  int main () {
 6      vector <int> v = {1, 2, 3, 4, 5, 6, 7};
 7      for (auto elem : v) {
 8
 9          cout << elem << endl;
10      }
11      return 0;
12  }
```

Цикл с итератором

Как посетить все элементы контейнера:

```
 1  #include <iostream>
 2  #include <set>
 3  using namespace std;
 4
 5  int main () {
 6      set <int> v = {1, 2, 3, 4, 5, 6, 7};
 7      for (auto elem : v) {
 8
 9          cout << elem << endl;
10      }
11      return 0;
12  }
```

Алгоритмы

Примеры алгоритмов, реализованных в стандартной библиотеке:

```
vector <int> v;
```

- `sort (v.begin (), v.end ());` — сортировка
- `reverse (v.begin (), v.end ());` — разворот

```
int x;
```

- `find (v.begin (), v.end (), x);` — поиск
- `binary_search (v.begin (), v.end (), x);` — двоичный поиск
- `lower_bound (v.begin (), v.end (), x);` — тоже двоичный поиск, но возвращает итератор

```
set <int> s;
```

- `find (s.begin (), s.end (), x);` — за какое время?
- `s.find (x);` — более эффективно

Функции высшего порядка

Функция высшего порядка — это функция, которая принимает в качестве аргумента другую функцию.

```
 1  #include <algorithm>
 2  #include <utility>
 3  #include <vector>
 4  using namespace std;
 5
 6
 7
 8
 9
10  int main () {
11      vector <pair <int, int> > p;
12      sort (p.begin (), p.end ());
13
14
15      return 0;
16  }
```

Функции высшего порядка

Функция высшего порядка — это функция, которая принимает в качестве аргумента другую функцию.

```
 1  #include <algorithm>
 2  #include <utility>
 3  #include <vector>
 4  using namespace std;
 5
 6  bool less_p (pair <int, int> a, pair <int, int> b) {
 7      return a.second > b.second;
 8  }
 9
10  int main () {
11      vector <pair <int, int> > p;
12      sort (p.begin (), p.end (), less_p);
13
14
15      return 0;
16  }
```

Функции высшего порядка

Функция высшего порядка — это функция, которая принимает в качестве аргумента другую функцию.

```
1  #include <algorithm>
2  #include <utility>
3  #include <vector>
4  using namespace std;
5
6
7
8
9
10 int main () {
11     vector <pair <int, int> > p;
12     sort (p.begin (), p.end (),
13         [&] (pair <int, int> a, pair <int, int> b)
14         {return a.second > b.second;});
15     return 0;
16 }
```

Функции высшего порядка

Функция высшего порядка — это функция, которая принимает в качестве аргумента другую функцию.

```
1  #include <algorithm>
2  #include <utility>
3  #include <vector>
4  using namespace std;
5
6
7
8
9
10 int main () {
11     vector <pair <int, int> > p;
12     sort (begin (p), end (p),
13         [&] (auto a, auto b)
14         {return a.second > b.second;}); // C++14
15     return 0;
16 }
```

Функции высшего порядка

Функция высшего порядка — это функция, которая принимает в качестве аргумента другую функцию.

```
1  #include <algorithm>
2  #include <utility>
3  #include <vector>
4  using namespace std;
5
6
7
8
9
10 int main () {
11     vector <pair <int, int> > p;
12     auto it = find_if (p.begin (), p.end (),
13         [&] (pair <int, int> a)
14         {return a.first == a.second;});
15     return 0;
16 }
```

Примеры

Следующие программы написаны как цепочки преобразований входных данных.

- Число → последовательность:
  `generate`, `generate_n`, `iota`, ...
- Последовательность → последовательность:
  `copy`, `copy_n`, `copy_if`, `remove_if`, `sort`, `reverse`, `transform`, `for_each`, ...
- Последовательность → число:
  `accumulate`, `min_element`, `max_element`, `find`, `find_if`, ...

Более полный список в документации:

- `en.cppreference.com/w/cpp/algorithm`
- `cplusplus.com/reference/algorithm`

Примеры

Следующие программы написаны как цепочки преобразований входных данных.

```
 1  // print first n Fibonacci numbers
 2  #include <algorithm>
 3  #include <iostream>
 4  #include <iterator>
 5  #include <vector>
 6  using namespace std;
 7
 8  int main () {
 9      int n;
10      cin >> n;
11      vector <int> v (n);
12      int a = 0, b = 1;
13      generate (v.begin (), v.end (), [&] ()
14          {int c = a; a = b; b += c; return c;});
15      copy (v.begin (), v.end (),
16            ostream_iterator <int> (cout, " "));
17      return 0;
18  }
```

Ввод: 10, вывод: 0 1 1 2 3 5 8 13 21 34 .

Примеры

Следующие программы написаны как цепочки преобразований входных данных.

```
 1  // sum for k=1..n of {k*(k+1) if it does not divide by 4}
 2  #include <algorithm>
 3  #include <iostream>
 4  #include <iterator>
 5  #include <numeric>
 6  #include <vector>
 7  using namespace std;
 8
 9  int main () {
10      int n;
11      cin >> n;                                   // 7
12      vector <int> v (n);
13      iota (v.begin (), v.end (), 1);             // 1 2 3 4 5 6 7
14      transform (v.begin (), v.end (), v.begin (),
15          [&] (int x) {return x * (x + 1);}); // 2 6 12 20 30 42 56
16      auto new_end = remove_if (v.begin (), v.end (),
17          [&] (int x) {return x % 4 == 0;});  // 2 6 30 42|30 42 56
18      auto s = accumulate (v.begin (), new_end, 0LL);
19      cout << s << endl;                          // 80
20      return 0;
21  }
```

Ссылка

Все примеры из слайдов можно найти здесь:
`http://acm.math.spbu.ru/~gassa/bachelor-2020/200921_m20`

Вопросы?

Вопросы?