DOI: https://doi.org/10.36910/6775-2524-0560-2022-48-17
УДК 004.415.3
Пех Петро Антонович, к.т.н., доцент
https://orcid.org/0000-0002-6327-3319
Христинець Наталія Анатоліївна, к.т.н., ст. викладач
https://orcid.org/0000-0002-4836-7632
Кучерук Олександр Сергійович, магістр
Луцький національний технічний університет, м. Луцьк, Україна

РЕАЛІЗАЦІЯ АЛГОРИТМІВ СОРТУВАННЯ ЕЛЕМЕНТІВ ПОСЛІДОВНИХ КОНТЕЙНЕРІВ ЗАСОБАМИ С++ БІБЛІОТЕКИ СТАНДАРНИХ ШАБЛОНІВ STL

Пех П.А., Христинець Н.А., Кучерук О.С. Реалізація алгоритмів сортування елементів послідовних контейнерів засобами С++ бібліотеки стандартних шаблонів STL. В статті запропоновані кілька програм на базі різних алгоритмів сортування елементів послідовних контейнерів. Наведені коди програм та результати їх тестування.
Ключові слова: контейнер-вектор, контейнер-черга, контейнер-список, шаблон класу, бібліотека шаблонів

Pekh P., Khrystynets N., Kucheruk O. Implementation of algorithms for sorting elements of sequential containers by means of library of standard templates STL The article offers several programs based on various algorithms for sorting elements of sequential containers. The program codes and results of their testing are given.
Keywords: container-vector, container-deque, container-list, class template, template library

Постановка задачі. Відомо [1,2], що контейнерні класи бібліотеки стандартних шаблонів STL (standard template library) поділяються на послідовні (vector, deque, list) та асоціативні (set та multiset, map та multimap). Кожен з цих класів реалізує певну абстрактну структуру даних. У випадку послідовних контейнерів – це вектори, черги, списки, у випадку асоціативних контейнерів – це множини та карти. Дане дослідження спрямоване лише на роботу з послідовними контейнерами, оскільки у ній розв'язується задача сортування елементів контейнерів за зростанням чи спаданням значень, яка стосується перш за все векторів, черг та списків.

Метою дослідження було розроблення низки програм з використанням алгоритмів бібліотеки STL для реалізації задачі сортування елементів послідовних контейнерів.

Новизна дослідження полягає у підході до програмування алгоритмів сортування саме за допомогою бібліотеки стандартних шаблонів [1, 3-4].

Основна частина. У даній роботі пропонуються три програми, розроблені за єдиною схемою, зображеною на рис. 1. Як можна бачити з цієї схеми, перший розділ програми містить директиви препроцесора, з допомогою яких під'єднуються заголовні файли. Їх призначення подані у вигляді коментарів. Другий розділ програми містить необхідні визначення класів, функцій чи їх шаблонів. Основна частина програми знаходиться у третьому розділі програми. Для кожної програми наведені результати її тестування. Таким чином, достатньо вставити у наведену на рис. 1 схему вміст другого та третього розділів, як ми тут же отримаємо готову програму на базі того чи іншого алгоритму STL. Використовуючи такий підхід, ми розробили програми сортування елементів послідовних контейнерів на базі використання алгоритмів STL: sort та stable_sort (рис. 2 – рис. 4); lower_bound та upper_bound (рис. 5 – рис. 7); binary_search (рис. 8 – рис. 10).

Функції sort і stable_sort сортують елементи діапазону у порядку зростання. Елементи порівнюються або за допомогою оператора менше (<), або за допомогою допоміжної функції compare. Функція sort не гарантує рівність упорядкування елементів, але stable_sort збереже відносний порядок рівних елементів після операції. Це означає, що коли є два рівних елементи a і b, і a передує b, то після сортування елемент a все одно передує елементу b. Функція lower_bond повертає ітератор до першого елемента діапазону [first, last], який не менше (тобто він більше або дорівнює) значення value, або якщо такого елемента немає, повертається ітератор до last. Для цього функція lower_bond порівнює елементи, використовуючи або оператор (<), або допоміжну функцію порівняння compare. Щоб ця функція працювала коректно, діапазон елементів повинен бути попередньо відсортований. Аналогічно працює функція lower_bond. Функція binary_search шукає елементи зі значенням, що дорівнює значенню всередині діапазону [first, last]. Якщо такий елемент знайдений, функція повертає значення true; якщо ні, то повертається false.

```
#include <iostream>  // прототипи функцій для стандартного вводу та виводу в C++
#include <conio.h>   // прототипи функцій для консольного вводу та виводу в C++
#include <string> // прототипи функцій для роботи зі стрічковими даними
#include <windows.h> // прототипи функцій для роботи з кирилицею
#include <algorithm> // прототипи функцій для підключення методів контейнерів
#include <iterator> // прототипи функцій для роботи з ітераторами
#include <vector>  // прототипи функцій для роботи з контейнерами-векторами
#include <deque >  // прототипи функцій для роботи з контейнерами-чергами
#include <list>  // прототипи функцій для роботи з контейнерами-списками
using namespace std; // підключення простору службових імен
//-------------------------------------------------------------------------------------------------------
// Оголошення функцій та шаблонів
//-------------------------------------------------------------------------------------------------------
int main() {
   system("cls"); // очищення екрана від попередньої інформації
   SetConsoleCP(1251); // використання кирилиці при введенні даних
   SetConsoleOutputCP(1251); // використання кирилиці при виведенні даних
  //-------------------------------------------------------------------------------------------------------
   // Основна частина
   //-------------------------------------------------------------------------------------------------------
   return 0;
}
```

Рис. 1. Загальна структура програм, які розроблені у даній статті

```
template<class T>
void print(const T& value) {
   cout << value << " ";
}
char t(char c)  {
   return tolower(c);
}
string toLower(const string& v)  {
   string ret(v);
   transform(ret.begin(), ret.end(), ret.begin(), t);
   return ret;
}
bool descending_caseinsensitive(const string& a, const string& b)  {
   return toLower(b) < toLower(a);
}
bool descending_casesensitive(const string& a, const string& b)  {
   return b < a;
}
bool ascending_caseinsensitive(const string& a, const string& b) {
   return toLower(a) < toLower(b);
}
```

Рис. 2. Визначення шаблонів та функцій програми на базі алгоритмів sort та stable_sort

```
string t[] = { "AB", "dC", "ba", "aB", "BC", "bc", "BA", "cd", "Cd", "ab" };
vector <string> v1(10);
copy(t, t + 10, v1.begin());
cout << "Контейнер-джерело (вектор v1):\n";
cout << "v1: "; for_each(v1.begin(), v1.end(), print<string>); cout << endl;
cout << "\nСортування у порядку зростання:\n";
cout << "Сортування за допомогою функції sort :\n";
sort(v1.begin(), v1.end());
cout << "v1: "; for_each(v1.begin(), v1.end(), print<string>); cout << endl;
cout << "Сортування за допомогою функції stable_sort:\n";
copy(t, t + 10, v1.begin());
stable_sort(v1.begin(), v1.end(), ascending_caseinsensitive);
cout << "v1: "; for_each(v1.begin(), v1.end(), print<string>); cout << endl << endl;
cout << "Сортування у порядку спадання:\n";
cout << " Сортування за допомогою функції sort:\n";
sort(v1.begin(), v1.end(), descending_casesensitive);
cout << "v1: "; for_each(v1.begin(), v1.end(), print<string>); cout << endl;
cout << " Сортування за допомогою функції stable_sort:\n";
copy(t, t + 10, v1.begin());
cout << "v1: "; for_each(v1.begin(), v1.end(), print<string>); cout << endl << endl;
```

Рис. 3. Основна частина програми на базі алгоритмів sort та stable_sort

```
Контейнер-джерело (вектор v1):
v1: AB dC ba aB BC bc BA cd Cd ab
Сортування у порядку зростання:
Сортування за допомогою функції  sort:
v1: AB BA BC Cd aB ab ba bc cd dC
Сортування за допомогою функції  stable_sort:
v1: AB aB ab ba BA BC bc cd Cd dC
Сортування у порядку спадання:
Сортування за допомогою функції  sort:
v1: dC cd bc ba ab aB Cd BC BA AB
Сортування за допомогою функції  stable_sort:
v1: dC cd Cd BC bc ba BA AB aB ab
```

Рис. 4. Результати тестування програми на базі алгоритмів sort та stable_sort

```
template <class T> void print(T start, T end) {
  for (; start != end; ++start) {
    std::cout << *start << " ";  } }
bool compare(int a, int b) {
  return b < a; }
```

Рис. 5. Визначення класу та функцій програми на базі алгоритмів lower_bound та upper_bound

```
int t[] = { 1, 10, 8, 4, 5, 6, 3, 9, 7, 2 };
deque <int> d1(t, t + 10);
cout << "Контейнер-джерело (deque):\n";
cout << "d1: ";    print(d1.begin(), d1.end()); cout << endl;
cout << "\nСортування у порядку зростання:\n";
sort(d1.begin(), d1.end());
cout << "d1: ";    print(d1.begin(), d1.end()); cout << endl;
cout << "Пошук елементів з діапазону [4,6]:\n";
deque<int>::iterator it1 = lower_bound(d1.begin(), d1.end(), 4);
deque<int>::iterator it2 = upper_bound(d1.begin(), d1.end(), 6);
print(it1, it2); cout << endl;
cout << "Пошук одного значення з діапазону за допомогою equal_range:\n";
pair<deque<int>::iterator, deque<int>::iterator> p = equal_range(d1.begin(), d1.end(), 4);
print(p.first, p.second); cout << endl << endl;
cout << "--------------------------------------------------------------------\n";
cout << "\nСортування у порядку спадання:\n";
sort(d1.begin(), d1.end(), compare);
cout << "d1: ";    print(d1.begin(), d1.end()); cout << endl;
cout << "Пошук елементів з діапазону [6,4]:\n";
it1 = lower_bound(d1.begin(), d1.end(), 6, compare);
it2 = upper_bound(d1.begin(), d1.end(), 4, compare);
print(it1, it2); cout << endl;
cout << "Пошук одного значення з діапазону за допомогою equal_range:\n";
p = equal_range(d1.begin(), d1.end(), 4, compare);
print(p.first, p.second); cout << endl;
```

Рис. 6. Основна частина програми на базі алгоритмів lower_bound та upper_bound

```
Контейнер-джерело (deque):
d1: 1 10 8 4 5 6 3 9 7 2
Сортування у порядку зростання:
d1: 1 2 3 4 5 6 7 8 9 10
Пошук елементів з діапазону [4,6]:
4 5 6
Пошук одного значення з діапазону за допомогою equal_range:
4
--------------------------------------------------------------------
Сортування у порядку спадання:
d1: 10 9 8 7 6 5 4 3 2 1
Пошук елементів з діапазону [6,4]:
6 5 4
Пошук одного значення з діапазону за допомогою equal_range:
4
```

Рис. 7. Результати тестування програми на базі алгоритмів lower_bound та upper_bound

```
template <class T> void print(T start, T end) {
   for (; start != end; ++start) {
      std::cout << *start << " ";   } }
bool compare(int a, int b) {
   return b < a; }
```

Рис. 8. Визначення класу та функцій програми на базі алгоритму binary_search

```
int t[] = { 1, 10, 8, 4, 5, 6, 3, 9, 7, 2 };
deque <int> d1(t, t + 10);
cout << "Джерело-контейнер (deque):\n";
cout << "d1: ";    print(d1.begin(), d1.end());cout << endl;
cout << "\nСортування у порядку зростання:\n";
sort(d1.begin(), d1.end());
cout << "d1: ";    print(d1.begin(), d1.end());cout << endl;
cout << "Пошук елемента [5]:\n";
printMessage(binary_search(d1.begin(), d1.end(), 5), 5);
cout << "-------------------------------------------------\n";
cout << "Пошук елемента в контейнері, який належним чином не відсортований:\n";
cout << "Пошук елемента [5] - припускаючи, що контейнер відсортований за спаданням:\n";
printMessage(binary_search(d1.begin(), d1.end(), 5, compare), 5); cout << endl;
cout << "-------------------------------------------------\n";
cout << "\nСортування у порядку спадання:\n";
sort(d1.begin(), d1.end(), compare);
cout << "d1: ";    print(d1.begin(), d1.end());cout << endl;
cout << "Пошук елемента [5]:\n";
printMessage(binary_search(d1.begin(), d1.end(), 5, compare), 5);
```

Рис. 9. Основна частина програми на базі алгоритму binary_search

```
Джерело-контейнер (deque):
d1: 1 10 8 4 5 6 3 9 7 2
Сортування у порядку зростання:
d1: 1 2 3 4 5 6 7 8 9 10
Пошук елемента [5]:
Елемент 5 знайдено!
-------------------------------------------------
Пошук елемента в контейнері, який належним чином не відсортований:
Пошук елемента [5] - припускаючи, що контейнер відсортований за спаданням:
Елемент 5 не знайдено!
-------------------------------------------------
Сортування у порядку спадання:
d1: 10 9 8 7 6 5 4 3 2 1
Пошук елемента [5]:
Елемент 5 знайдено!
```

Рис. 10. Результати тестування програми на базі алгоритму binary_search

Висновок. У статті запропоновані різні варіанти програм, які реалізують алгоритми сортування елементів послідовних контейнерів шляхом використання відповідних методів бібліотеки стандартних шаблонів STL Такий підхід дозволяє значно прискорити процес розроблення програм.

Список бібліографічного опису

1. Х.М.Дейтел, И.Дж.,Дейтел. Как программировать на C++: Пятое издание. М.: ООО «Бином-Пресс», 2011. -1456 с.
2. Архангельский Ф.Я. Программирование в C++Builder. М.: ООО «Бином-Пресс», 2010. -896 с.
3. Пех П.А., Костюк Ю.Ю., Кравченко М.Б. До питання конструювання класів з конструкторами різного типу // Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво" –Луцьк: Видавництво Луцького НТУ. –Вип. 39. -2020. С. 197-203.
4. Пех П.А., Корець Р.С. Реалізація основної функції калькулятора – обчислення виразів – за допомогою Java-класів // Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво" –Луцьк: Видавництво Луцького НТУ. –Вип. 35. -2019. С. 172-176.

References

1. H.M. Deitel, I.J., Deitel. How to program in C++: Fifth edition. M.: "Bynom-Press" LLC, 2011. -1456 p.
2. Arkhangelsky F.Ya. Programming in C++Builder. M.: "Bynom-Press" LLC, 2010. -896 p.
3. Pekh P.A., Kostyuk Yu.Yu., Kravchenko M.B. On the issue of designing classes with different types of constructors // Scientific journal "Computer-integrated technologies: education, science, production" - Lutsk: Publishing House of Lutsk National Technical University. - Issue 39. -2020. P. 197-203.
4. Pekh P.A., Korets R.S. Realization of the main function of the calculator - calculation of expressions - with the help of Java classes // Scientific journal "Computer-integrated technologies: education, science, production" - Lutsk: Publishing House of Lutsk National Technical University. - Issue 35. -2019. P. 172-176.