Лабораторна робота № 9. Бібліотека стандартних шаблонів

Мета роботи: вивчення і практичне використання бібліотеки стандартних шаблонів.
Теоретичний матеріал лекції, [1, розділ 11], [2, розділи 33-35], [3, розділ 16].

1. Короткі теоретичні відомості

Бібліотека стандартних шаблонів (Standard Template Library, STL) є бібліотекою контейнерних класів C++. До складу бібліотеки входять такі елементи:

• *Контейнер* – об'єкт, який містить інші об'єкти, організовані у вигляді послідовностей (колекції об'єктів). Контейнери поділяють на базові (динамічний вектор, список, черга) та асоціативні (відображення, множини).

• *Ітератор* – забезпечує, доступ до вмісту контейнера. Над літераторами можна виконувати операції, призначені для роботи з вказівниками (інкремент, декремент, розадресування). Розрізняють такі види ітераторів: однонапрямлений, двонапрямлений, введення, виведення, довільного доступу, зворотний.

• *Алгоритм* – це параметризовані зовнішні функція, що забезпечують обчислювальні процедури над елементами контейнерів, наприклад ініціалізація, пошук, сортування, заміна елементів.

• *Алокатор* – забезпечує керування динамічною пам'яттю, виділеною для STL-об'єктів.

• *Функтор* – об'єкт з перевантаженою операцією operator(). Використовується замість вказівника на функцію за потреби передавання функції в іншу функцію через список параметрів.

• *Предикат* – функція булевого типу, яка перевіряє визначені програмістом властивості або відношення між об'єктами. Предикат може бути унарним або бінарним. Типи параметрів предикату відповідають типам об'єктів, інкапсульованих контейнером.

• *Адаптор* – пристосовує елемент бібліотеки для забезпечення різних інтерфейсів. Наприклад, на основі вектора, списку або черги можна створити стек, а на основі списку – чергу.

Для використання алгоритмів та функторів у програмі необхідно під'єднати заголовкові файли:

```
#include <algorithm>
#include <functional>
```

Таблиця 1. Класи бібліотек стандартних шаблонів

Класи	**Файли включення**	**Призначення**
vector	<vector>	динамічний вектор
valarray	<valarray>	масив для математичних обчислень
complex	<complex>	комплексне число
list	<list>	лінійний список
stack	<stack>	стек
queue	<queue>	черга
priority_queue	<queue>	пріоритетна черга
deque	<deque>	черга з двома кінцями
map	<map>	відображення (словник) без повторень елементів
multimap	<map>	відображення з повтореннями елементів
set	<set>	множина без повторень елементів
multiset	<set>	множина з повтореннями елементів
basic_string	<string>	базовий клас для роботи з рядками
string	<string>	спеціалізація basic_string<char>

1.1. Ітератори шаблонних класів

Ітератор використовуються для звернення до елементів контейнера. Оголошуються за допомогою слова `iterator`. Більшість класів STL мають методи `begin()` та `end()`, які повертають значення ітератора відповідно на початок та кінець послідовності елементів контейнера. Ітератори мають властивості вказівників. Для них визначені операції розадресації (*), інкременту (++), декременту(--), цілочисельне збільшення значення (+), різниці ітераторів (-) та операцій порівняння (!=, ==, <, <=,>,>=).

Приклад виведення усіх елементів контейнера за допомогою ітератора і циклу `for`:

```
vector<int> C;
 vector <int>::iterator p;
 for(p=C.begin();p!=C.end;p++) cout<<" "<<*p; cout<<endl;
```

1.2. Шаблонний клас string

Шаблонний клас (контейнер) string призначений для роботи з рядками символів. У контейнері визначено перевантажені конструктори для оголошення об'єкту без ініціалізації, з ініціалізацією об'єкту значенням рядка або іншого об'єкту класу string, наприклад:

```
string s, str("ABCD"), t(str);
```

Створений об'єкт міститиме рядок, розмір якого може динамічно змінюватися. Звертатися до символів рядка можна за допомогою ітератора або індексу. Методи для роботи з символьними рядками:

`str.insert(ofs,s)` – вставити у позицію `ofs` значення `s`, яке може бути об'єктом `string` або рядком символів типу `char *`.

`str.erase(ofs,n)` – вилучити `n` символів з позиції `ofs`.

`str.substr(ofs,s)` – виділити (скопіювати) `n` символів з позиції `ofs`.

`str.copy(x,ofs,n)` – скопіювати `n` символів з позиції `ofs` у рядок `x`.

`str.c_str()` – перетворити рядок символів типу `string` у рядок типу `char`, який закінчується нуль-символом (у стилі С).

`str.find(s,ofs)` – знайти позицію підрядка `s` починаючи із зміщення `ofs`.

`str.push_back(c)` – помістити символ `c` у кінець рядка.

`str.append(s)` – зчепити рядки символів `str` і `s`.

`str1.compare(ofs,n,str2)` – порівняння `n` символів із зміщення `ofs` рядка `str1` з рядком `str2`.

`str.clear()` – витирання усіх символів рядка `str`.

`str.erase(ofs,n)` – витирання `n` символів із зміщенням `ofs` в рядку `str`.

`str.length()`, `str.size()` – визначення довжини рядка `str`.

`str.resize(n, ch)` – змінити довжину рядка, `ch` – символ заповнювач у випадку розширення рядка.

1.3. Шаблонний клас вектор

Шаблонний клас вектор забезпечує роботу з динамічними масивами різних типів. Оголошення об'єкта-вектор:

```
vector<int> v, a(5), b(5,0), c(b);
```

Методи для роботи з об'єктом-вектор:

`push_back()` – розширення базового вектора шляхом занесення елементів у його кінець.

`resize(n, x)` – зміна кількості елементів вектора, `n` – нова кількість елементів, `x` – символ заповнювач.

`size()` - поточна кількість елементів вектора.

`begin()`, `end()` – повертають ітератор на початок, кінець вектора.

`insert(iter, x)` – занесення значення `x` у позицію, яка визначається літератором `iter`.

`erase(iter), erase(iter_from,iter_to` – вилучення елементів з позиції, яка визначається ітератором.

`clear()` – вилучення усіх елементів вектора.

`front(), back()` – повернення посилання на початковий, кінцевий елемент.

Виводити елементи вектора (та інших колекцій) можна поелементно у циклі або застосувати алгоритм `copy`:

```
copy(v.begin(), v.end(), ostream_iterator<float>(cout," "));
```

Фрагмент програми роботи з вектором:

```
vector <float> v;
vector <float>::iterator p,g;
// вектор з випадкових чисел
for(int i=0;i<n;i++) {
    x=(float) rand()/RAND_MAX
    v.push_back(x);
}
// сортування вектора
for(p=v.begin();p!=v.end-1;p++)
   for(q=p+1;q!=v.end();q++)
         if(*p>*q) {x=*p; *p=*q;}
// виведення вектора
for(p=v.begin();p!=v.end;p++) cout<<" "<<*p;
```

На основі вектора можна побудувати різноманітні адаптори, наприклад двовимірні та багатовимірні масиви.

Наприклад, можна організувати матрицю, оголосивши її як вектор, елементами якого є теж вектори:

```
vector <float> v;
vector<float>::iterator p;
float x; int n=3,m=4;
vector <vector<float>>::iterator r;
vector<vector<float>> matrix;
for(int i-0;i<n;i++) {
   for(int j=0;i<m;j++) {
      x=(float) rand/RAND_MAX;
      v.push_back(v); // допоміжний вектор для занесення у матрицю
   }
      matrix.push_back(v);
      v.clear();
}
for(r=matrix.begin();r!=matrix.end();r++) {
   for(p=r->begin();p!=r->end();p++ cout<<" "<<*p;
```

1.4. Алгоритми

Алгоритми використовуються для роботи з елементами контейнерів різних типів. Для використання алгоритмів необхідно включити у текст програми заголовковий файл `#include <algorithm>`.

Алгоритми:

`min_element()` – пошук мінімального елементу, `imin=min_element(v.begin(),v.end());`

`max_element()` – пошук максимального елементу, `imax=max_element(v.begin(),v.end());`

`find()` – пошук заданого елементу, `i=find(v.begin(),v.end(),*imin);`

`remove()` – вилучення елементу, `i=remove(v.begin(),v.end(),*imax);`

`sort()` – сортування вектора, `sort(v.begin(),v.end(),greater(<int>());`

`copy()` – копіювання елементів вектора у інший вектор,
`copy(v.begin(),v.end(),t.begin());`

Бібліотека STL містить і інші корисні алгоритми для роботи з колекціями.

1.5. Масиви значень

Клас valarray призначений для роботи з масивами числових значень. Для роботи з класом valarray необхідно під'єднати заголовковий файл: #include <valarray>.

При створенні масиву вказується кількість елементів та значення для їх ініціалізації, наприклад:

```
int a[5]={-3, 5, 2, -4, 8};
valarrary<int> x(10), y(7,-1), (z(a,5);
```

Визначені у класі valarray методи та операції виконуються над кожним елементом масиву.

Методи призначені для обчислення математичних функцій: abs(), cos() , sin(), tan(), acos() , asin() , atan() , cosh(), sinh(), tanh(), sqrt(), exp(), log(), log10().

У класі valarray перевантажено ряд операцій, які виконуються над кожним елементом масиву або над парами елементів двох масивів: арифметичні (+,-,*,/,%), відношення(==,!=,>,>=,<,<=), порозрядні логічні (&,|,^), зсуву (>>,<<), логічні (&&,||), виділення елемента ([]).

Операції застосовуються до цілісних масивів. Бінарні операції виконуються над масивами з однаковою кількістю елементів. Результат виконання операції повертається як масив значень такої самої розмірності. Операції Відношення повертають масив значень логічного типу, кожен елемент якого є результатом відношення між відповідними парами елементів. Перевантажена операція [] забезпечує звернення до елементів масиву за допомогою індексу.

Приклад обчислення виразу y=(a*b-c)/(a+b+c) над масивами дійсних чисел a,b,c:

```
valarray<double> a(-2,5),b(4,5),c(-7,5),y(5);
y=(a*b-c)/(a+b+c);
for(int i=0;i<y.size();i++) cout<<y[i]<<" ";
```

Інші методи класу valarray:

```
size() – кількість елементів масиву.
resize(n), resize(n,x) – зміна розміру масиву.
min(), max() – мінімальний, максимальний елемент масиву.
sum() – сума елементів масиву.
shift(ofs) – зсув на ofs позицій елементів масиву(ofs>0 – вправо,ofs<0 – вліво).
slice(beg, coutn, end) – виділення підмножини елементів (зрізи).
```

1.6. Стеки

Стек – це облвсть пам'яті з дисципліною доступу LIFO (Last Input – First Output, останній записаний елемент буде прочитаний першим). Стек реалізовано на основі контейнеру дек. За необхідності можна визначити інший базовий контейнер стеку:

Створення стеку:

```
stack<int> s;
stack <int, vector<int> > s2;
stack <int, list<int> > i3;
```

Методи стеку:

push() – занести елемент у стек.

pop() – вилучити елемент зі стеку.

top() – читання елементу з верхівки стеку.

size() – кількість елементів стеку.

empty() – перевірка наявності елементів у стеку.

Приклад роботи із стеком:

```
stack <int> s;
stack <int>::size_type size;
int i=10;
s.push(i);
Ccut<<s.top()<<" ";
```

```
s.pop();
```

1.7. Списки

Шаблонний клас `list` реалізує двонаправлений список елементів. Створення списків:

```
list<int> s, s1(), s2(7, 0), s3(s2);
```

Методи списку:

`push_front()` – занесення елементів у початок списку.
`push_end()` – занесення елементів у кінець списку
`pop_front()` – читання списку з початку.
`pop_back()` – читання списку з кінця.
`begin(), end()` – повертають ітератор на початковий, кінцевий елемент.
`insert(iter,x)` – вставлення елемента у позицію визначену значенням ітератора.
`erase(iter)` – вилучення елемента за значенням ітератора.
`remove(x)` – вилучення елемента `x`.
`clear()` – вилучення всіх елементів списку.
`s1.merge(s2)` – злиття списків `s1, s2`.
`sort()` – сортування списку.
`resize(n),` – зміна розміру списку
`resize(n,x)` – збільшення розміру списку із заповнення розширення символом `x`.

Приклад створення списку і виведення елементів списку:

```
list<int> s;
list<int>::iterator p;
for(int i=0;i<n;i++) s.push_back(i);
for(p=s.begin();p!=s.end();p++) cout << *p <<" ";
```

1.8. Черги і деки

Черги – це організована послідовність елементів з дисципліною доступу FIFO (First Input – First Output). Елементи добавляються в чергу з її кінця, а вилучаються – з початку. За замовчуванням черга будується на основі базового контейнера `deque`. За необхідності можна змінити базовий контейнер черги, наприклад на список:

```
queue<int> q;
queue<char, list<char> > q;
```

Методи черги:

`push(x)` – занесення елемента `x` в кінець черги.
`pop()` – вилучення елемента з початку черги.
`back()` – повернення значення останнього елемента черги.
`front()` – повернення значення першого елемента черги.
`empty()` – перевірка наявності елементів у черзі.
`size()` – поточне число елементів черги.

Приклад створення і роботи з чергою:

```
queue <int> q;
queue <int>::size_type size
q.push(5);
cout<<q.size();
q.pop();
```

Дек – це черга, в якій елементи добавляються і вилучаються з обох кінців. Дек можна порівняти з динамічним вектором, який може зростати і скоручуватися з обох кінців. Оголошення об’єкта шаблонного класу `deque`:

```
deque <int> d, d1(5), d2(5,-1), d3(d2);
```

Методи класу `deque` забезпечують двосторонній доступ до деку.

`push_front(x)` – занесення елемента `x` на початок деку.
`push_back(x)` – занесення елемента `x` у кінець деку.
`pop_front()` - читання елемента з початку деку.
`pop_back()` – читання елемента з кінця деку.
`begin(), end()` – повертають ітератор на початковий, кінцевий елемент деку.
`front(), back()` – читання елементів з початку, кінця деку.
`insert(iter,x)` – добавлення елемента `x` у визначену значенням ітератора позицію.
`erase(iter)` – вилучення елемента за значенням ітератора.
`clear()` –вилучення всіх елементів деку.
`size()` – поточна кількість елементів деку.
`resize(n)` – зміна розміра деку.
`resize(n,x)` – збільшення розміра деку із заповненням розширених елементів значенням `x`.
`empty()` – перевірка наявності елементів у деку.

Приклад створення і роботи з деком:

```
deque <int> d;
dequeue <int>::size_type size
d.push_front(5);
d.push_back(6);
cout<<d.size();
d.pop_back();
d.pop_front();
d.empty();
```

1.9. Асоціативні відображення

Асоціативні контейнери містять елементи, які складаються з двох частин: ключа та елемента даних. Ключ та елемент даних перебувають в асоціативній залежності. За значенням ключа можна відшукати відповідний елемент даних.

Об’єкти відображення (словники) елементів створюються за допомогою шаблонного класу `map`:

```
map<string, int> m;
```

Колекції асоціативних контейнерів зберігаються у пам’яті як бінарне дерево. Конструктори усіх асоціативних контейнерів містять необовязковий параметр, який задає критерій сортування колекції його елементів:

```
map<string, int, less<string> > m1;
map<string, int, greater<string> > m2;
```

Кожний елемент відображення складається з пари значень: ключа та відповідного йому поля даних. Для доступу до пар значень використовується об’єкт шаблонного класу `pair`:

```
pair<string, float> x, *p=new pair<string, float>;
```

Поля пари значень мають назви `first` та `second`. Звертатися до них можна використовуючи назву об’єкта або вказівник: `x.first, x.second, p->first, p->second`.

Пару елементів можна створити за допомогою конструктора класу `pair`:

```
x=pair<string, float> (”Вага”, 15.5);
```

або за допомогою функції `make_pair`:

```
x=make_pair(”Об’єм”, 120.5);
```

Методи відображень:

`begin(), end()` – методи повертають ітератор на початок, кінець колекції елементів
`insert(iter, x)` – вставлення елемента `x` за значенням ітератора
`find(key)` – пошук елемента колекції за ключем.
`lower_bound(key)` – повертає ітератор на перший елемент, який є менший або дорівнює заданому ключу.
`upper_bound(key)` – повертає ітератор на перший елемент, який є більшим за заданий ключ.

`count(key)` – повертає кількість елементів, ключі яких дорівнюють ключу, заданому у списку аргументів.

`erase(iter)` – вилучення елемента за заданим ітератором.

`erase(iter1, iter2)` – вилучення послідовності елементів заданих двома ітераторами.

`clear()` – вилучення усіх елементів.

`empty()` – нявність елементів у колекції.

Приклад створення і роботи з відображенням:

```
map<string,int> m;
map<string,int>::const_iterator p;
typedef pair<string,int> Student;
m.isert(Student(”Курс”, 2));
m.insert(Student(”Курс”,3));
for(p=m.begin();p!=m.end();p++) cout<<p-first<<” ”<<p->second<<endl;
```

1.10. Множини

Множини є різновидністю відображення, у якому значення елемента дорівнює його ключу. Множина задається контейнером шаблонного класу `set`:

```
set<int> s1;
set<int, less<int> > s2;
set<int, greater<int> > s3;
```

У памяті множини зберігаються як бінарне дерево. Для роботи з множинами використовують ті самі методи та операції, що і для класу `map`.

Приклад створення і роботи з множиною:

```
set <int> s1;
set <int>::iterator p,q;
for(int i=0;i<n;i++) s1.insert(i);
for(p=s1.begin();p!=s1.ens();p++) cout<<*p<<” ”;
for each (x in s1) cout <<x<<” ”;
```

1.11. Множини бітів

Множина бітів є об’єктом шаблонного класу `bitset`, але не має ітератора і не вважається контейнером STL-бібліотеки. При оголошенні об’єкта класу `bitset` аргумент шаблону задається константою цілого типу, яка визначає кількість бітів множини:

```
#include <bitset>
bitset<> b0;
bitset<8> b1(5);
string str(”1110011110001110”);
bitset<16> b2(str);
```

Методи для роботи множиною бітів:

`test(n)` – перевірка значення біта у позиції `n`.

`any()` – повертає `true`, якщо хоча б один біт множини встановлено в 1.

`none()` – повертає `true`, якщо у множині немає бітів зі значенням 1.

`set(n)` – встановлює біт з номером `n`.

`set()` – встановлює всі біти множини в 1.

`reset()` – встановлює всі біти множини в 0.

`flip(n)` – інвертує біт з номером `n`.

`flip()` – інвертує всі біти.

`count()` – повертає число одиниць у бітовій множині.

`size()` – повертає загальне число бітів у множині.

`to_string()` – перетворює бітову множину в об’єкт класу `string`.

`to_ulong()` – перетворює бітову множину в ціле число.

Крім методів у класі `bitset` перевантажено операції для роботи з бітовими множинами: порівняння(`==, !=`), інверсії бітів (`~`), комбіновані порозрядні (`&=, |=, ^=`), зсуву (`<<, <<=, >>, >>=`), потокового введення та виведення (`>>,<<`), квадратні дужки. Операція `[]` повертає значення біта у заданій позиції.

Приклад створення і роботи з множиною бітів:

```
#include <bitset>
bitset<8> b1(45),b2(9),b3; cout<<b1<<" "<<b2<<endl;
b3=b1 | b2; cout<<b3<<endl;
b3=b1 & (b1^b2); cout<<b3<<endl;
```

Приклад програми. Розробити телефонну книгу на основі класу Особа (`Person`) та контейнерного класу Множина з повторенням елементів (`multiset`). Ввести прізвище Особи та визначити номер її телефону. Визначити кількість осіб, що мають однакове прізвище та ім'я, і вивести на екран номери їхніх телефонів.

```
#include <iostream>
#include <set>
#include <string>
using namespace std;

// клас особа
class Person {
  string LastName;   // Прізвище
  string FirstName;  // Ім'я
  long PhoneNumber;  // Номер телефону
public:

// конструктор
Person(string LN="", string FN="", long PN=0) :
  LastName(LN), FirstName(FN), PhoneNumber(PN) {}

// перевантажена операція new
void * operator new(size_t, void *ptr) {return ptr;}

// перевантажена операція ==
friend bool operator==(const Person& P1, const Person& P2)
   {return (P1.LastName==P2.LastName) && (P1.FirstName==P2.FirstName)?true:false;}

// перевантажена операція <
friend bool operator<(const Person& P1, const Person& P2)
   {if(P1.LastName==P2.LastName) return (P1.FirstName<P2.FirstName)?true:false;
    return (P1.LastName<P2.LastName)?true:false;}

//перевантажена операція потокового введення
friend istream& operator>>(istream&, Person&);

// перевантажена операція потокового виведення
friend ostream& operator<<(ostream&, Person&);

};

istream& operator>>(istream& is, Person& P)
  {is>>P.LastName>>P.FirstName>>P.PhoneNumber; return is;}

ostream& operator<<(ostream& os, Person& P)
   //{os<<P.LastName<<'\t'<<P.FirstName<<'\t'<<P.PhoneNumber<<endl;return os;}
   {os<<P.LastName<<endl;return os;}

// Головна функція
int main() {
// Множина з повторенням елементів
multiset<Person> S;
```

```
multiset<Person, less<Person> >::iterator i;

// об'єкт класу Person;
Person P;

cout<<"Введіть дані про власників телефонів"<<endl;
while(cin>>P) S.insert(P);   // введення до натиснення Ctrl+Z
cin.clear();                 //очищення потоку введення від ознаки EOF (Ctrl+Z)

cout<<"Число записів="<<S.size()<<endl;

cout<<"Склад множини"<<endl;

//for(i=S.begin(); i!=S.end(); i++) {  cout<<reinterpret_cast<char>(*i)<<endl;}
i=S.begin(); cout <<*i<<endl;


cout<<"Пошук номера телефону"<<endl;
string LN, FN;
cout<<"Введіть прізвище особи"<<endl;
cin>>LN;
cout<<"Введіть ім'я"<<endl;
cin>>FN;

// повторна ініціалізація об'єкта
new(&P) Person(LN,FN,0);
cout<<"Кількість осіб з таким прізвищем та іменем"<<endl;
cout<<S.count(P)<<endl;

cout<<"Всі записи, що відповідають запиту"<<endl;
//for(i=S.lower_bound(P);i!=S.upper_bound(P);i++) cout<<*i<<endl;

return 0;
}
```

Запитання.

1. Які елементи входять до складу бібліотеки стандартних шаблонів?
2. Дати характеристику об’єктам контейнер, ітератор, алгоритм. Типи ітераторів.
3. Дати характеристику об’єктам алокатор, функтор, предикат, адаптер.
4. Шаблонний клас `string` і його методи для роботи з рядками символів.
5. В чому перевага і недоліки використання об’єктів типу `string`.
6. Шаблонний клас `vector` і його методи для роботи з динамічними векторами.
7. Призначення алгоритмів для роботи з колекціями.
8. Масив значень і його методи.
9. Стек і його методи.
10. Список і його методи.
11. Черги і деки та їх методи.
12. Асоціативні відображення.
13. Множини і множини бітів.

Завдання.

1. Написати шаблонну функцію створення, заповнення, сортування і виведення значень динамічного вектора.

2. Написати шаблонну функцію, яка читає текстовий файл і вставляє елементи файла у список.

3. Написати програму яка записує пари номер місяця – назва місяця в асоціативний масив і потім за введеним номером друкує назву місяця.

4. Написати шаблонну функцію послідовного пошуку значення в масиві за заданим

ключем. Функція повертає перший елемент масиву, який відповідає заданому ключу, або виводить повідомлення, що елемент не знайдено.

5. Написати шаблонну функцію сортування одновимірного масиву за зростанням елементів методом бульбашки. Суть методу полягає в порівнянні та перествлянні сусідніх елементів.

6. Написати функцію шаблон, яка перевіряє впорядкованість елементів масиву за зростанням або спаданням.

7. Написати шаблонну функцію формування файлу числових даних у текстовому форматі.

8. Створити шаблонний клас КВАДРАТНА МАТРИЦЯ. Обчислити контрольну суму елементів матриці як суму усіх елементів за модулем 2.

9. Створити шаблонний клас СТЕК. Визначити методи занесення рядків у стек та їх читання зі стеку. Параметризувати стек рядками символів. Знайти найдовший занесений у стек рядок.

10. Створити шаблонний клас для роботи з однонапрямленим лінійним списком. Впорядкувати список за зростанням значень елементів.

11. Створити шаблонний клас для роботи з лінійним списком з подвійними зв’язками. Занести елемент на початок списку. Вилучити елемент з кінця списку.

12. Створити шаблонний клас МНОЖИНА, призначений для збереження елементів і виконання операцій над множинами. Реалізувати класичні операції над множинами: об’єднання, перетину, різниці.

13. Створити шаблонний клас ЧЕРГА З ПРІОРИТЕТАМИ. Занести елемент у чергу. Вилучити з черги елементи з найвищим та з найнижчим пріоритетами.

2. Приклади для самостійної роботи

```
//listing 2 - основні операції над векторами
#include <iostream>
#include <vector>
#include <cctype>
using namespace std;

int main()
{
  vector<char> v(10); // створення вектора з 10 елементів
  unsigned int i;

  // вивід на екран розміру вектора v
  cout << "Розмір = " << v.size() << endl;

  // присвоєння елементам вектора деякі значення
  for(i=0; i<10; i++) v[i] = i + 'a';

  // вивід на екран вмісту вектора
  cout << "Поточний вміст:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  cout << "Розширений вектор\n";
  // добавлення нових елементів в кінець вектора
  for(i=0; i<10; i++) v.push_back(i + 10 + 'a');

  // виводимо на екран розмір вектора v
  cout << "Новий розмір = " << v.size() << endl;

  // виводимо на екран вміст вектора
  cout << "Поточний вміст:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";
```

```
  // змінюємо вміст вектора
  for(i=0; i<v.size(); i++) v[i] = toupper(v[i]);
  cout << "Модифікований вміст:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;

  return 0;
}
```

Розмір = 10
Поточний вміст:
a b c d e f g h i j
Розширений вектор
Новий розмір = 20
Поточний вміст:
a b c d e f g h i j k l m n o p q r s t
Модифікований вміст:
A B C D E F G H I J K L M N O P Q R S T

```
// listing 3 - доступ до елементів вектора за допомогою ітератора
#include <iostream>
#include <vector>
#include <cctype>
using namespace std;

int main()
{
  vector<char> v(10); // створення вектора довжини 10
  vector<char>::iterator p; // створення ітератора
  int i;

  // присвоєння елементам вектора значення
  p = v.begin();
  i = 0;
  while(p != v.end()) {
    *p = i + 'a';
    p++;
    i++;
  }

  // вивід на екран вмісту вектора
  cout << "Початковий вміст:\n";
  p = v.begin();
  while(p != v.end()) {
    cout << *p << " ";
    p++;
  }
  cout << "\n\n";

  // зміна вмісту вектора
  p = v.begin();
  while(p != v.end()) {
    *p = toupper(*p);
    p++;
  }

  // вивід на екран вмісту вектора
  cout << "Модифікований вміст:\n";
  p = v.begin();
  while(p != v.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl;
```

```
  return 0;
}
```
Початковий вміст:
a b c d e f g h i j
Модифікований вміст:
A B C D E F G H I J

//listing 4 - функції вставки і вилучення
```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
  vector<char> v(10);
  vector<char> v2;
  char str[] = "<Vector>";
  unsigned int i;

  // ініціалізуємо вектор v
  for(i=0; i<10; i++) v[i] = i + 'a';

  // копіювання символів із str у v2
  for(i=0; str[i]; i++) v2.push_back(str[i]);

  // вивід на екран вмісту вектора
  cout << "Початковий вміст v:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  vector<char>::iterator p = v.begin();  // ітератор
  p += 2; // встановлення ітератора на 3-й елемент

  // вставка  10-ти символів X у v
  v.insert(p, 10, 'X');

  // вивід на екран вмісту після вставки
  cout << "Розмір після вставки = " << v.size() << endl;
  cout << "Вміст  після вставки:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  // вилучення цих елементів
  p = v.begin();
  p += 2; // встановлення ітератора на 3-й елемент
  v.erase(p, p+10); // вилучення наступних 10 елементів

  // вивід вмісту після вилучення
  cout << "Розмір після вилучення = " << v.size() << endl;
  cout << "Вміст  після вилучення:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  // Вставка вектора  v2 у вектор v
  v.insert(p, v2.begin(), v2.end());
  cout << "Розмір після вставки вектора v2 = ";
  cout << v.size() << endl;
  cout << "Вміст після вставка вектора v2:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;

  return 0;
}
```

```
Початковий вміст v:
a b c d e f g h i j
Розмір після вставки = 20
Вміст  після вставки:
a b X X X X X X X X X c d e f g h i j
Розмір після вилучення = 10
Вміст  після вилучення:
a b c d e f g h i j
Розмір після вставки вектора v2 = 18
Вміст після вставка вектора v2:
a b < V e c t o r > c d e f g h i j
```

```
//listing 5 - вектор, який містить об'єкти класу
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

class DailyTemp {
  int temp;
public:
  DailyTemp() { temp = 0; }
  DailyTemp(int x) { temp = x; }

  DailyTemp &operator=(int x) {
    temp = x; return *this;
  }

  double get_temp() { return temp; }
};

bool operator<(DailyTemp a, DailyTemp b)
{
  return a.get_temp() < b.get_temp();
}

bool operator==(DailyTemp a, DailyTemp b)
{
  return a.get_temp() == b.get_temp();
}

int main()
{
  vector<DailyTemp> v;
  unsigned int i;

  for(i=0; i<7; i++)
    v.push_back(DailyTemp(60 + rand()%30));

  cout << "Температура по Фарангейту:\n";
  for(i=0; i<v.size(); i++)
    cout << v[i].get_temp() << " ";

  cout << endl;

  // Перетворення із шкали Фарангейта у шкалу Цельсія
  for(i=0; i<v.size(); i++)
    v[i] = (int) (v[i].get_temp()-32) * 5/9 ;

  // Перетворення із шкали Фарангейта у шкалу Цельсія
  for(i=0; i<v.size(); i++)
    v[i] = (int) (v[i].get_temp()-32) * 5/9 ;
```

```
  cout << "Температура в градусах цельсія:\n";
  for(i=0; i<v.size(); i++)
    cout << v[i].get_temp() << " ";

  return 0;
}
```

Температура по Фарангейту:
73 76 87 85 83 85 76
Температура в градусах Цельсія:
22 24 30 29 28 29 24

```
//listing 6 - операції над списком
#include <iostream>
#include <list>
using namespace std;

int main()
{
  list<int> lst; // створення порожнього списку
  int i;

  for(i=0; i<10; i++) lst.push_back(i);

  cout << "Розмір = " << lst.size() << endl;

  cout << "Вміст: ";
  list<int>::iterator p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }
  cout << "\n\n";

  // зміна вмісту списку
  p = lst.begin();
  while(p != lst.end()) {
    *p = *p + 100;
    p++;
  }

  cout << "Модифікований вміст: ";
  p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Розмір = 10
Вміст: 0 1 2 3 4 5 6 7 8 9
Модифікований вміст: 100 101 102 103 104 105 106 107 108 109

```
// Listing8 - Функція end().
#include <iostream>
#include <list>
using namespace std;

int main()
{
  list<int> lst; // створення порожнього списку
  int i;
```

```
  for(i=0; i<10; i++) lst.push_back(i);

  cout << "Вивід списку в прямому порядку:\n";
  list<int>::iterator p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }
  cout << "\n\n";

  cout << "Вивід списку в зворотньому порядку:\n";
  p = lst.end();
  while(p != lst.begin()) {
   p--; // зменшення вказівника перед cout, так як останній елемент відповідає
значенню end()-1
    cout << *p << " ";
  }

  return 0;
}
```

Вивід списку в прямому порядку:
0 1 2 3 4 5 6 7 8 9
Вивід списку в зворотньому порядку:
9 8 7 6 5 4 3 2 1 0

//listing 9 - функції push_back() і push_front().

```
#include <iostream>
#include <list>
using namespace std;

int main()
{
  list<int> lst1, lst2;
  int i;

  for(i=0; i<10; i++) lst1.push_back(i);
  for(i=0; i<10; i++) lst2.push_front(i);

  list<int>::iterator p;

  cout << "Вміст списку lst1:\n";
  p = lst1.begin();
  while(p != lst1.end()) {
    cout << *p << " ";
    p++;
  }
  cout << "\n\n";

  cout << "Вміст списку lst2:\n";
  p = lst2.begin();
  while(p != lst2.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Вміст списку lst1:
0 1 2 3 4 5 6 7 8 9
Вміст списку lst2:
9 8 7 6 5 4 3 2 1 0

```
//listing 10 - сортування списку
#include <iostream>
#include <list>
#include <cstdlib>
using namespace std;

int main()
{
  list<int> lst;
  int i;

  // створення списку з випадкових цілих чисел
  for(i=0; i<10; i++)
    lst.push_back(rand()%100);

  cout << "Початковий вміст:\n";
  list<int>::iterator p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl << endl;

  // sort the list
  lst.sort();

  cout << "Відсортований вміст:\n";
  p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст:
83 86 77 15 93 35 86 92 49 21
Відсортований вміст:
15 21 35 49 77 83 86 86 92 93

```
//listing 11 - злиття двох списків
#include <iostream>
#include <list>
using namespace std;

int main()
{
  list<int> lst1, lst2;
  int i;

  for(i=0; i<10; i+=2) lst1.push_back(i);
  for(i=1; i<11; i+=2) lst2.push_back(i);

  cout << "Вміст списку lst1:\n";
  list<int>::iterator p = lst1.begin();
  while(p != lst1.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl << endl;

  cout << "Вміст списку lst2:\n";
  p = lst2.begin();
  while(p != lst2.end()) {
```

```
    cout << *p << " ";
    p++;
  }
  cout << endl << endl;

  // злиття двох файлів
  lst1.merge(lst2);
  if(lst2.empty())
    cout << "lst2 тепер порожній\n";

  cout << "Вміст списку lst1 після злиття:\n";
  p = lst1.begin();
  while(p != lst1.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

```
Вміст списку lst1:
0 2 4 6 8

Вміст списку lst2:
1 3 5 7 9

lst2 тепер порожній
Вміст списку lst1 після злиття:
0 1 2 3 4 5 6 7 8 9
```

```
//listing 12 - список з об'єктами
#include <iostream>
#include <list>
#include <cstring>
using namespace std;

class myclass {
  int a, b;
  int sum;
public:
  myclass() { a = b = 0; }
  myclass(int i, int j) {
    a = i;
    b = j;
    sum = a + b;
  }
  int getsum() { return sum; }

  friend bool operator<(const myclass &o1,
                        const myclass &o2);
  friend bool operator>(const myclass &o1,
                        const myclass &o2);
  friend bool operator==(const myclass &o1,
                        const myclass &o2);
  friend bool operator!=(const myclass &o1,
                        const myclass &o2);
};

bool operator<(const myclass &o1, const myclass &o2)
{
  return o1.sum < o2.sum;
}

bool operator>(const myclass &o1, const myclass &o2)
{
```

```
  return o1.sum > o2.sum;
}

bool operator==(const myclass &o1, const myclass &o2)
{
  return o1.sum == o2.sum;
}

bool operator!=(const myclass &o1, const myclass &o2)
{
  return o1.sum != o2.sum;
}
int main()
{
  int i;

  // створення першого списку
  list<myclass> lst1;
  for(i=0; i<10; i++) lst1.push_back(myclass(i, i));

  cout << "Перший список: ";
  list<myclass>::iterator p = lst1.begin();
  while(p != lst1.end()) {
    cout << p->getsum() << " ";
    p++;
  }
  cout << endl;

  // створення другого списку
  list<myclass> lst2;
  for(i=0; i<10; i++) lst2.push_back(myclass(i*2, i*3));

  cout << "Другий список: ";
  p = lst2.begin();
  while(p != lst2.end()) {
    cout << p->getsum() << " ";
    p++;
  }
  cout << endl;

  // злиття двох списків
  lst1.merge(lst2);
  // вивід на екран списку після злиття
  cout << "Список після злиття: ";
  p = lst1.begin();
  while(p != lst1.end()) {
    cout << p->getsum() << " ";
    p++;
  }
  return 0;
}
```

Перший список: 0 2 4 6 8 10 12 14 16 18
Другий список: 0 5 10 15 20 25 30 35 40 45
Список після злиття: 0 0 2 4 5 6 8 10 10 12 14 15 16 18 20 25 30 35 40 45

//listing 13 - асоціативний масив

```
#include <iostream>
#include <map>
using namespace std;

int main()
{
  map<char, int> m;
  int i;
```

```
  // ввід пари в асоціативний масив
  for(i=0; i<26; i++) {
    m.insert(pair<char, int>('A'+i, 65+i));
  }

  char ch;
  cout << "Введіть ключ (велика баква): ";
  cin >> ch;

  map<char, int>::iterator p;

  // знаходження значення по заданому ключу
  p = m.find(ch);
  if(p != m.end())
    cout << "ASCII-код ключа дорівнює  " << p->second;
  else
    cout << "Ключ не знайдений.\n";

  return 0;
}
```

Введіть ключ (велика баква): F
ASCII-код ключа дорівнює 70

```
//listing 15 - використання асоціативного масиву для створення телефонної книжки
#include <iostream>
#include <map>
#include <cstring>
using namespace std;

class name {
  char str[40];
public:
  name() { strcpy(str, ""); }
  name(char *s) { strcpy(str, s); }
  char *get() { return str; }

};

// визначення оператора < для об'єктів класу name.
bool operator<(name a, name b)
{
   return strcmp(a.get(), b.get()) < 0;
}

class phoneNum {
  char str[80];
public:
  phoneNum() { strcmp(str, ""); }
  phoneNum(char *s) { strcpy(str, s); }
  char *get() { return str; }
};


int main()
{
  map<name, phoneNum> directory;

 // занесення імен і номерів в асоціативний масив
  directory.insert(pair<name, phoneNum>(name("Іван"),
                   phoneNum("555-4533")));
  directory.insert(pair<name, phoneNum>(name("Петро"),
                   phoneNum("555-9678")));
  directory.insert(pair<name, phoneNum>(name("Горпина"),
```

```
            phoneNum("555-8195")));
  directory.insert(pair<name, phoneNum>(name("Параска"),
                   phoneNum("555-0809")));

  // given a name, find number
  char str[80];
  cout << "Введіть ім'я: ";
  cin >> str;

  map<name, phoneNum>::iterator p;

  p = directory.find(name(str));
  if(p != directory.end())
    cout << "Номер телефону: " <<  p->second.get();
  else
    cout << "Таке ім'я відсутнє в книжці.\n";

  return 0;
}
```

Введіть ім'я: Іван
Номер телефону: 555-4533

//listing 16 - алгоритм count()

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <algorithm>
using namespace std;

int main()
{
  vector<bool> v;
  unsigned int i;

  for(i=0; i < 10; i++) {
   if(rand() % 2) v.push_back(true);
   else v.push_back(false);
  }

  cout << "Послідовність:\n";
  for(i=0; i<v.size(); i++)
    cout << boolalpha << v[i] << " ";
  cout << endl;

  i = count(v.begin(), v.end(), true);
  cout << "Заданому предикату <true> відповідає наступна кількість елементів: " <<
i << "\n";

  return 0;
}
```

Послідовність:
true false true true true true false false true true
Заданому предикату <true> відповідає наступна кількість елементів: 7

//listing 17 - алгоритм count_if()

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// унарний предикат, який визначає чи ділиться число на 3
bool dividesBy3(int i)
{
```

```
  if((i%3) == 0) return true;

  return false;
}

int main()
{
  vector<int> v;
  unsigned int i;

  for(i=1; i < 20; i++) v.push_back(i);

  cout << "Послідовність:\n";
  for(i=0; i<v.size(); i++)
    cout << v[i] << " ";
  cout << endl;

  i = count_if(v.begin(), v.end(), dividesBy3);
  cout << "Кількість чисел кратних 3, дорівнює " << i << "\n";

  return 0;
}
```

Послідовність:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Кількість чисел кратних 3, дорівнює 6

//listing 18 - алгоритми remove_copy і replace_copy

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
  char str[] = "Шаблони STL мають великі можливості";
  int i;
  vector<char> v, v2(80);

  for(i=0; str[i]; i++) v.push_back(str[i]);

  //  алгоритм remove_copy
  cout << "Початкова послідовність:\n";
  for(i=0; i<v.size(); i++) cout << v[i];
  cout << endl;

  // вилучення всіх пропусків
  remove_copy(v.begin(), v.end(), v2.begin(), ' ');

  cout << "Результат після вилучення всіх пропусків:\n";
  for(i=0; i<v2.size(); i++) cout << v2[i];
  cout << endl << endl;

  // алгоритм replace_copy
  cout << "Початкова послідовність:\n";
  for(i=0; i<v.size(); i++) cout << v[i];
  cout << endl;

  // заміна пропусків двокрапками
   replace_copy(v.begin(), v.end(), v2.begin(), ' ', ':');

  cout << "Результат після заміни пропусків двокрапками:\n";
  for(i=0; i<v2.size(); i++) cout << v2[i];
  cout << endl << endl;
```

```
  return 0;
}
```
Початкова послідовність:
Шаблони STL мають великі можливості
Результат після заміни пропусків двокрапкаМИ:
Шаблони:STL:мають:великі:можливості

//listing 19 - алгоритм reverse
```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
  vector<int> v;
  unsigned int i;

  for(i=0; i<10; i++) v.push_back(i);

  cout << "Початкова послідовність: ";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;

  reverse(v.begin(), v.end());

  cout << "Зворотня послідовність ";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";

  return 0;
}
```
Початкова послідовність: 0 1 2 3 4 5 6 7 8 9
Зворотня послідовність 9 8 7 6 5 4 3 2 1 0

//listing 20 - алгоритм transform
```
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;

// функція трансформації
double reciprocal(double i) {
  return 1.0/i;
}

int main()
{
  list<double> vals;
  int i;

  // запис значень в список
  for(i=1; i<10; i++) vals.push_back((double)i);

  cout << "Початковий вміст списку vals:\n";
  list<double>::iterator p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  cout << endl;

  // перетворення значень списку  vals
```

```
  p = transform(vals.begin(), vals.end(),
                vals.begin(), reciprocal);

  cout << "Перетворений вміст сриску vals:\n";
  p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст списку vals:
1 2 3 4 5 6 7 8 9
Перетворений вміст сриску vals:
1 0.5 0.333333 0.25 0.2 0.166667 0.142857 0.125 0.111111

//listing 22 - використання унарного функтора

```
#include <iostream>
#include <list>
#include <functional>
#include <algorithm>
using namespace std;

int main()
{
  list<double> vals;
  int i;

  // запис значень в список
  for(i=1; i<10; i++) vals.push_back((double)i);

  cout << "Початковий вміст списку vals:\n";
  list<double>::iterator p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl;

  // перетворення списку з використанням функтора negate
  p = transform(vals.begin(), vals.end(),
                vals.begin(),
                negate<double>());

  cout << "Перетворений вміст списку vals:\n";
  p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст списку vals:
1 2 3 4 5 6 7 8 9
Перетворений вміст списку vals:
-1 -2 -3 -4 -5 -6 -7 -8 -9

//listing 23 - використання бінарного функтора

```
#include <iostream>
#include <list>
```

```
#include <functional>
#include <algorithm>
using namespace std;

int main()
{
  list<double> vals;
  list<double> divisors;
  int i;

  // запис значення в список
  for(i=10; i<100; i+=10) vals.push_back((double)i);
  for(i=1; i<10; i++) divisors.push_back(3.0);

  cout << "Початковий вміст списку vals:\n";
  list<double>::iterator p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  cout << endl;

  // трансформація списку  vals
  p = transform(vals.begin(), vals.end(),
                divisors.begin(), vals.begin(),
                divides<double>()); // виклик бінарного функтора

  cout << "Вміст списку після ділення:\n";
  p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст списку vals:
10 20 30 40 50 60 70 80 90
Вміст списку після ділення:
3.33333 6.66667 10 13.3333 16.6667 20 23.3333 26.6667 30

//listing 25 - створення функтора reciprocal.

```
#include <iostream>
#include <list>
#include <functional>
#include <algorithm>
using namespace std;

// Простий функтор
class reciprocal: unary_function<double, double> {
public:
  result_type operator()(argument_type i)
  {
    return (result_type) 1.0/i;
  }
};

int main()
{
  list<double> vals;
  int i;

  // запис значення в список
```

```
  for(i=1; i<10; i++) vals.push_back((double)i);

  cout << "Початковий вміст списку vals:\n";
  list<double>::iterator p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl;

  // застосування функтора reciprocal
  p = transform(vals.begin(), vals.end(),
                vals.begin(),
                reciprocal()); // виклик функтора

  cout << "Змінений вміст списку vals:\n";
  p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст списку vals:
1 2 3 4 5 6 7 8 9
Змінений вміст списку vals:
1 0.5 0.333333 0.25 0.2 0.166667 0.142857 0.125 0.111111

//listing 26 - редактор зв'язків bind2nd()

```
#include <iostream>
#include <list>
#include <functional>
#include <algorithm>
using namespace std;

int main()
{
  list<int> lst;
  list<int>::iterator p, endp;

  int i;

  for(i=1; i < 20; i++) lst.push_back(i);

  cout << "Початкова послідовність:\n";
  p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl;

  endp = remove_if(lst.begin(), lst.end(),
                   bind2nd(greater<int>(), 8));

  cout << "Результуюча послідовність:\n";
  p = lst.begin();
  while(p != endp) {
    cout << *p << " ";
    p++;
  }

  return 0;
```

```
}
```
Початкова послідовність:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Результуюча послідовність:
1 2 3 4 5 6 7 8

//listing 30 - клас string
```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string str1("Alpha");
  string str2("Beta");
  string str3("Omega");
  string str4;

  // присвоєння значень
  str4 = str1;
  cout << str1 << "\n" << str3 << "\n";

  // зчеплення двох string-рядків
  str4 = str1 + str2;
  cout << str4 << "\n";

  // зчеплення string-рядка з  C-char-рядком
  str4 = str1 + " to " + str3;
  cout << str4 << "\n";

  // порівняння двох strings-рядків
  if(str3 > str1) cout << "str3 > str1\n";
  if(str3 == str1+str2)
    cout << "str3 == str1+str2\n";

  // об'єкту string присвоюється значення c-char
  str1 = "Цей символьний рядок закінчується нульовим байтом.\n";
  cout << str1;

  // створення string об'єкта з c-char рядка
  string str5(str1);
  cout << str5;

  // input a string
  cout << "Введіть символьний рядок: ";
  cin >> str5;
  cout << str5;

  return 0;
}
```
Alpha
Omega
AlphaBeta
Alpha to Omega
str3 > str1
Цей символьний рядок закінчується нульовим байтом.
Цей символьний рядок закінчується нульовим байтом.
Введіть символьний рядок: STL
STL

//listing 31 - функції insert(), erase(), and replace().
```
#include <iostream>
#include <string>
```

```
using namespace std;

int main()
{
  string str1("1234567 символьних рядків в стилі C++.");
  string str2("#POWER STL#");

  cout << "Початкові символьні рядки:\n";
  cout << "str1: " << str1 << endl;
  cout << "str2: " << str2 << "\n\n";

  // функція insert()
  cout << "Вставити  str2 в str1 з 7-ї позиції:\n";
  str1.insert(7, str2);
  cout << str1 << "\n\n";

  // функція erase()
  cout << "Вилучити 11 символів з позиції 7 в str1:\n";
  str1.erase(7, 11);
  cout << str1 <<"\n\n";

  // demonstrate replace
  cout << "Замінити з 7-ї позиції  11 символів в str1 рядком str2:\n";
  str1.replace(7, 11, str2);
  cout << str1 << endl;

  return 0;
}
```

```
Початкові символьні рядки:
str1: 1234567 символьних рядків в стилі C++.
str2: #POWER STL#

Вставити  str2 в str1 з 7-ї позиції:
1234567#POWER STL# символьних рядків в стилі C++.

Вилучити 11 символів з позиції 7 в str1:
1234567 символьних рядків в стилі C++.

Замінити з 7-ї позиції  11 символів в str1 рядком str2:
1234567#POWER STL#льних рядків в стилі C++.
```

```
//listing 32 - пошук символів
#include <iostream>
#include <string>
using namespace std;

int main()
{
  int i;
  string s1 =
    "Швидкий думкою, Сильний тілом, Гарячий серцем";
  string s2;

  i = s1.find("Швидкий");
  if(i!=string::npos) {
    cout << "Знайдено співпадіння в позиції " << i << endl;
    cout << "Залишок:\n";
    s2.assign(s1, i, s1.size());
    cout << s2;
  }
  cout << "\n\n";

  i = s1.find("Сильний");
  if(i!=string::npos) {
```

```
    cout << "Знайдено співпадіння в позиції " << i << endl;
    cout << "Залишок:\n";
    s2.assign(s1, i, s1.size());
    cout << s2;
  }
  cout << "\n\n";

  i = s1.find("Гарячий");
  if(i!=string::npos) {
    cout << "Знайдено співпадіння в позиції " << i << endl;
    cout << "Залишок:\n";
    s2.assign(s1, i, s1.size());
    cout << s2;
  }
  cout << "\n\n";

  // знаходимо останню кому
  i = s1.rfind(",");
  if(i!=string::npos) {
    cout << "Знайдено співпадіння в позиції " << i << endl;
    cout << "Залишок:\n";
    s2.assign(s1, i, s1.size());
    cout << s2;
  }

  return 0;
}
```

```
Знайдено співпадіння в позиції 0
Залишок:
Швидкий думкою, Сильний тілом, Гарячий серцем

Знайдено співпадіння в позиції 29
Залишок:
Сильний тілом, Гарячий серцем

Знайдено співпадіння в позиції 56
Залишок:
Гарячий серцем

Знайдено співпадіння в позиції 54
Залишок:
, Гарячий серцем
```

```
//listing 33 - рядки Strings як контейнери
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int main()
{
  string str1("Обробка Strings-рядків в C++ дуже проста");
  string::iterator p;
  unsigned int i;

  // використання функції  size()
  for(i=0; i<str1.size(); i++)
    cout << str1[i];
  cout << endl;

  // використання ітератора
  p = str1.begin();
  while(p != str1.end())
    cout << *p++;
```

```
  cout << endl;

  // використання алгоритму  count()
  i = count(str1.begin(), str1.end(), 'i');
  cout << "В string-рядку є " << i << " символів <i>\n";

  // використання функції transform() для перетворення малих букв у великі
  transform(str1.begin(), str1.end(), str1.begin(),toupper);
  p = str1.begin();
  while(p != str1.end())
    cout << *p++;
  cout << endl;


  return 0;
}
```

//listing 34 - використання асоціативного масиву символьних рядків
// для імітації телефонної книжки.

```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main()
{
  map<string, string> directory;

  directory.insert(pair<string, string>("Всеволод", "555-4533"));
  directory.insert(pair<string, string>("Богодар", "555-9678"));
  directory.insert(pair<string, string>("Меланія", "555-8195"));
  directory.insert(pair<string, string>("Ксенія", "555-0809"));

  string s;
  cout << "Введіть ім'я: ";
  cin >> s;

  map<string, string>::iterator p;

  p = directory.find(s);
  if(p != directory.end())
    cout << "Номер телефону: " << p->second;
  else
    cout << "Таке ім'я відсутнє в довідникеу.\n";

  return 0;
}
```

Введіть ім'я: Всеволод
Номер телефону: 555-4533