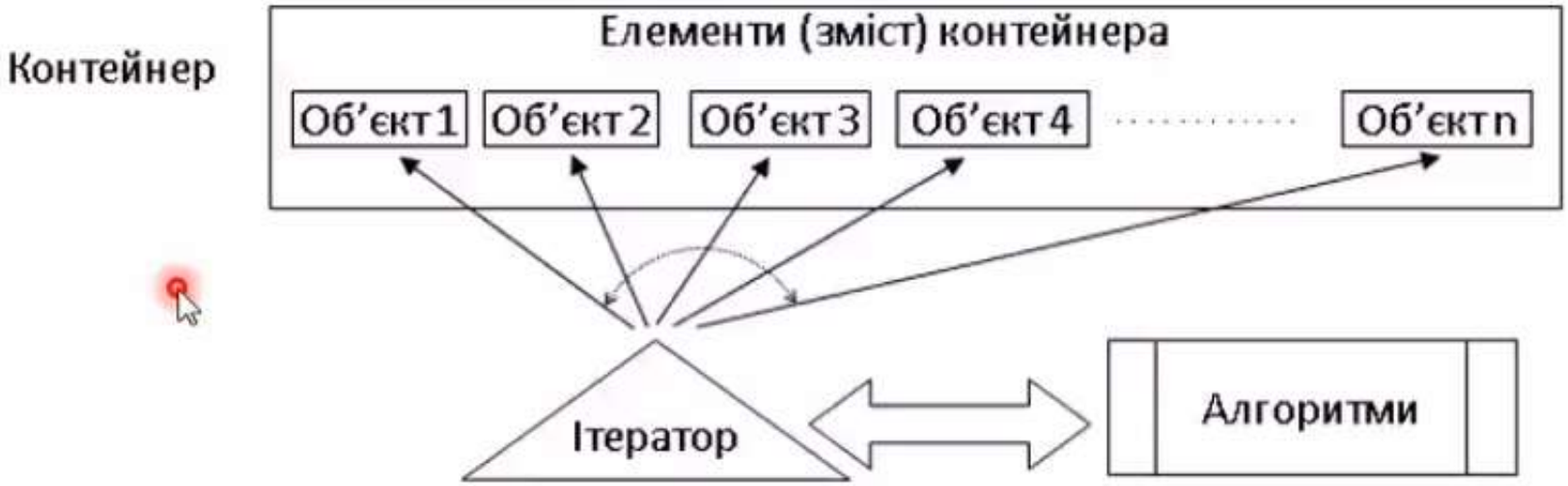
ТЕМА: Принципи будови стандартної бібліотеки шаблонів.

Огляд бібліотеки STL

Стандартна бібліотека шаблонів (скорочено STL) надає шаблонні класи і функції загального призначення, які реалізують багато алгоритмів і структур даних, що часто використовують на практиці. Наприклад, вона забезпечує можливість роботи з векторами, списками, чергами, стеками, відображеннями та іншими структурами, а також визначає різні процедури для роботи з такими об'єктами. Так як бібліотека STL складається з шаблонних класів, алгоритми і структури даних можуть бути застосовані до даних практично будь-якого типу.

Ядро стандартної бібліотеки шаблонів складають три основні елементи: контейнери, ітератори, алгоритми. Вони працюють разом один з одним, забезпечуючи готові розв'язки різних задач програмування. Їх взаємодію можна показати так:



Контейнери – це об'єкти, які складаються з інших об'єктів. Існує декілька різних типів контейнерів. Наприклад, клас `vector` визначає динамічний масив, клас `queue` створює чергу, клас `list` забезпечує роботу з лінійним списком. Такі контейнери в термінології STL називають послідовними контейнерами, бо для перегляду елементів контейнера потрібно послідовно переходити від попереднього елемента до наступного. Крім послідовних контейнерів, бібліотека STL визначає асоціативні контейнери, які дозволяють ефективно знаходити потрібні значення на основі заданих ключів (ідея двійкового дерева або хеш-таблиці). Наприклад, клас `map` забезпечує зберігання і опрацювання значень з унікальними ключами, тобто зберігає пару "ключ/значення" (подібно до словників мови Python).

Кожен контейнерний клас визначає перелік функцій, які можна застосувати до даного контейнера. Наприклад, клас `list` має функції вставлення, викреслення, перестановки, об'єднання елементів, а клас `stack` має функції для додавання і вибору елементів структури даних «стек».

Ітератори – це узагальнення вказівника. Ітератори функціонують подібно до вказівників. Вони дозволяють пересуватись через зміст контейнера так само, як вказівник пересувається елементами масиву.

Ітератори працюють аналогічно до вказівників, наприклад, їх можна інкрементувати, декрементувати, застосувати оператор розіменування адреси *.

В цілому ітератори підтримують операції

```
->, *, +, ++, +=, -, --, -=, <, >, <=, >=, ==, !=, []
```

Існує п’ять типів ітераторів:

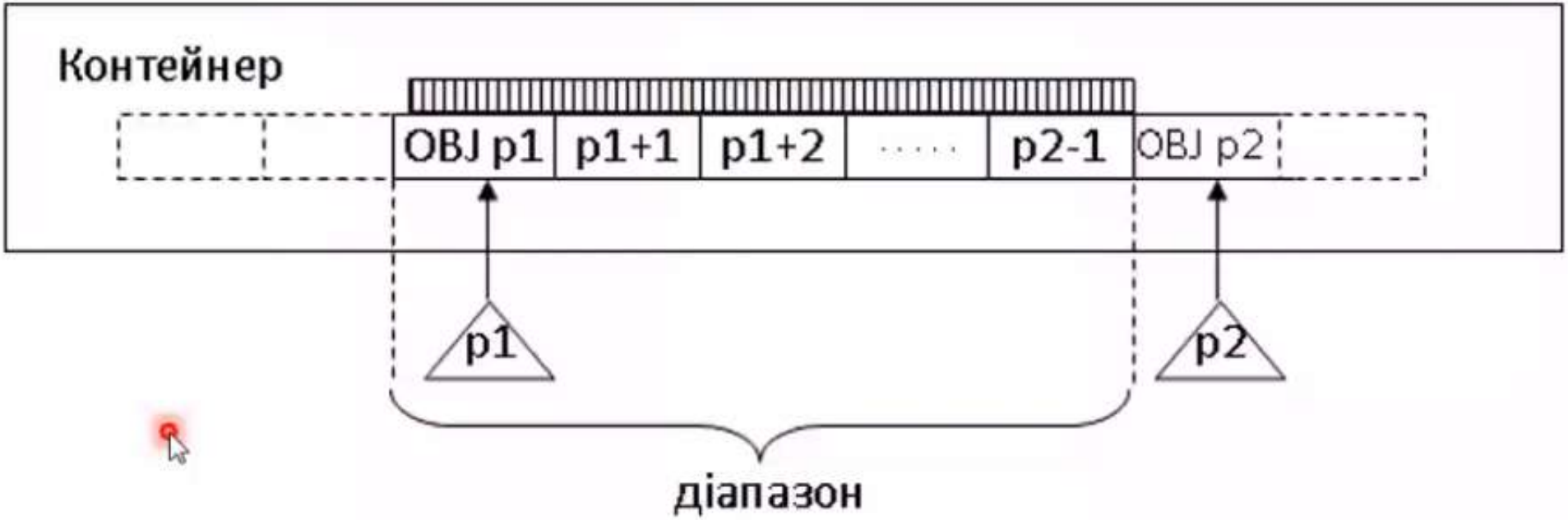
Ітератори	Дозволений доступ
Довільного доступу (random access)	Записують і читають значення; дозволяють організувати довільний доступ до елементів
Двонапрямлені (bidirectional)	Записують і читають значення; забезпечують інкрементальне і декрементальне пересування (вперед і назад)
Однонапрямлені (forward)	Записують і читають значення; забезпечують лише інкрементальне пересування (вперед)
Вхідні (input)	Читають, але не записують значень; забезпечують лише інкрементальне пересування
Вихідні (output)	Записують, але не читають значень; забезпечують лише інкрементальне пересування

Кожен тип ітератора має свою групу допустимих операцій з числа перерахованих вище. А кожен тип контейнера пов’язаний з відповідним типом ітератора, який не можна обирати довільно, бо кожен тип контейнера має свої правила функціонування.

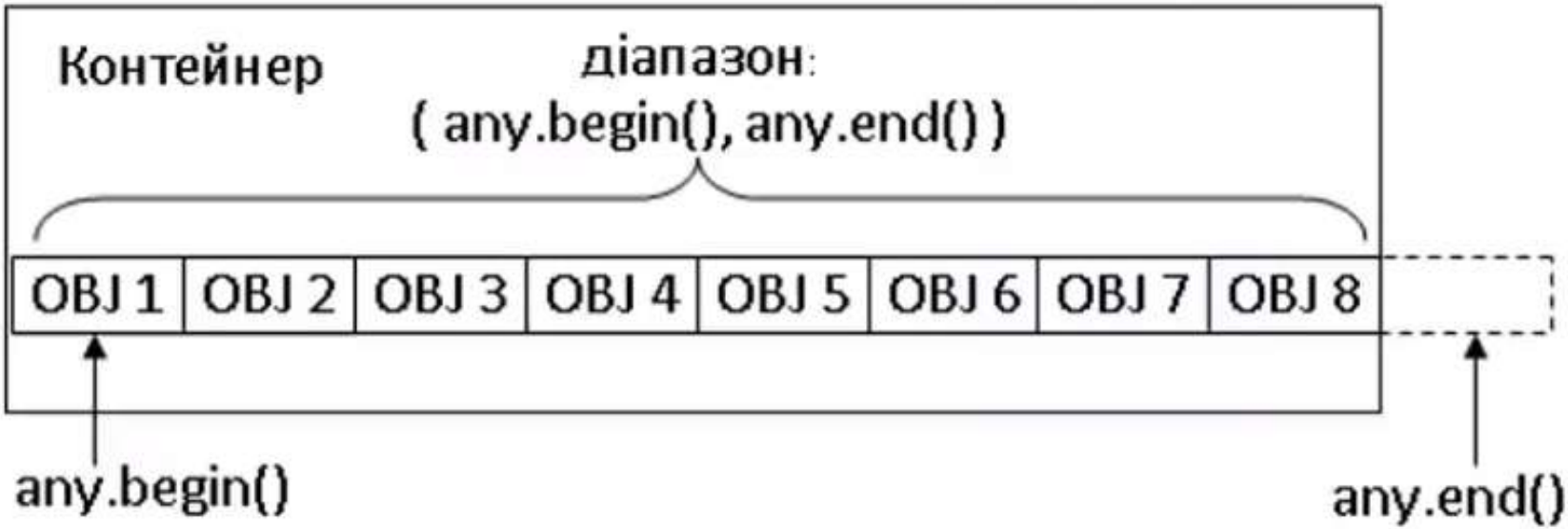
Алгоритми застосовують до контейнерів. Вони надають можливості ініціалізації, сортування, пошуку, перетворення змісту контейнера. Приклади алгоритмів: знайти в контейнері елемент заданого значення; знайти елемент, який відповідає заданому критерію; застосувати задану функцію до групи елементів; об'єднати частини двох контейнерів у третій контейнер; викреслити/замінити групу елементів; сортувати контейнер чи його частину.

Алгоритми можуть опрацьовувати як окремі елементи контейнера, так і цілі групи елементів всередині контейнера за одне виконання. В термінології STL таку групу називають діапазоном. Тому варто дати точне означення поняття діапазону контейнера.

Діапазон – це пара ітераторів `(p1,p2)` така, в якій перший ітератор вказує на початок діапазону, а другий – на елемент, який є наступним за останнім елементом діапазону, тобто, права межа не належить діапазону. Отже, діапазон `(p1,p2)` починається від `p1` і продовжується до `p2` не включаючи `p2`:



Діапазон `(any.begin(),any.end())` включає повний зміст контейнера:

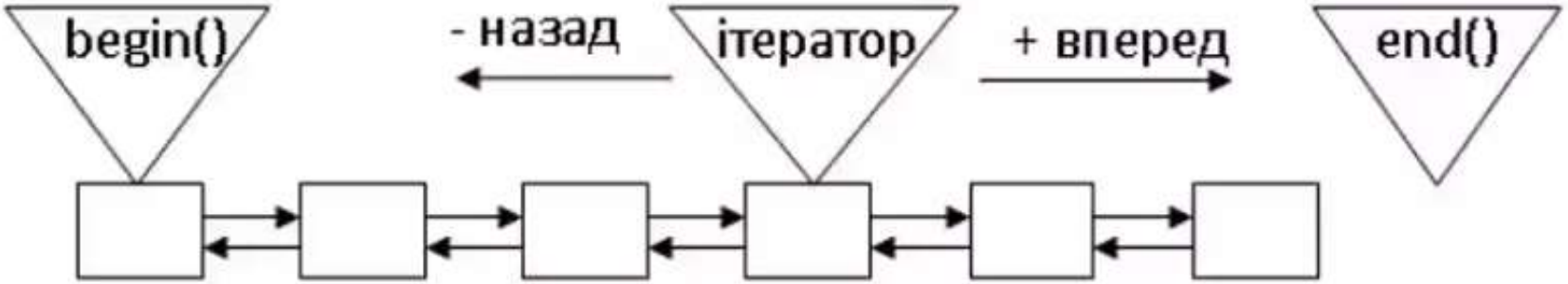


Ітератор `any.end()` вказує на кінець контейнера, що в термінології діапазону означає «елемент поза межами контейнера». При складанні програм запис `any.end()` можна трактувати як місце для наступного нового елемента. Останньому наявному елементу контейнера відповідає значення `any.end()-1`. Якщо значення ітератора під час перегляду контейнера стає рівним значенню функції `end()`, це означає, що всі елементи контейнера переглянуті. Треба бути уважним щодо такої властивості, бо вона в певній мірі нелогічна.

Діапазон `(p1,p1)` вважається порожнім.

Загальна характеристика списків

Контейнерний клас `list` будує і опрацьовує двонапрямлений лінійний список:



Для того, щоб використати в програмі клас `list`, потрібно записати директиву `#include <list>`. Клас є шаблонним, тому оголошення списків має вигляд:

```
list<тип елементів>  ім'я ( параметри конструктора );
```

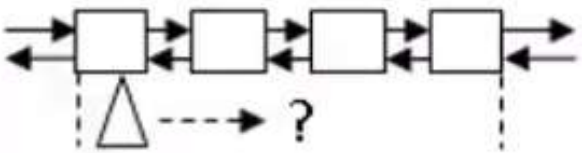
Тоді самі списки можна оголошувати так:

```
list<double> a;  // пустий список дійсних чисел
list<int> b(6,100);  // список 6 цілих чисел, всі значення = 100
list<int> c(b);  // список з елементів іншого списку b
  int data[10] = { 2,4,-3,-8,7,1,0,-9,3,3 };
list<int> d(&data[0], &data[5]);  // список з елементів
   // діапазону, заданого адресами (ітераторами):   2,4,-3,-8,7
```

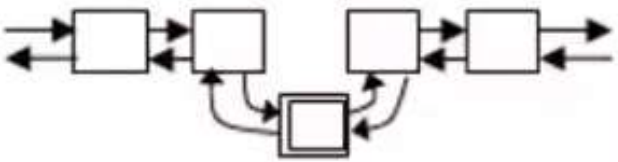
Щоб мати змогу працювати з елементами списку, потрібно визначити один або більше ітераторів виду

```
list<тип елементів>::iterator ім'я ;   // слово iterator є фіксоване
```

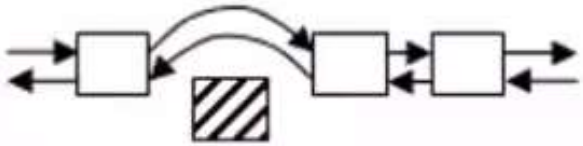
Списки підтримують такі *категорії операцій*: початкове створення списку, копіювання одного списку в інший, пошук елемента, операції над елементами, вставлення нового елемента, викреслення елемента, та інші. Наприклад:



пошук в межах



вставлення



викреслення

Таблиця загальних характеристик операцій (методів) класу `list`:

`iterator begin(),` `iterator end()`	повертає ітератор початку / кінця списку
`iterator insert(...)`	вставлення нових окремих елементів
`void splice(...)`	вставлення групи елементів іншого списку
`void push_back(...),` `void push_front(...)`	вставлення нового елемента до списку в кінець / на початок
`void merge(...)`	об'єднання двох списків в один, зберігаючи впорядкованість
`void pop_back(),` `void pop_front()`	викреслення останнього / першого елемента списку
`iterator erase(...)`	викреслення елементів за ітератором
`void remove(...)`	викреслення всіх елементів заданого значення
`size_type size()`	повертає кількість елементів списку
`інші методи`	

Крім того, в класі `list` визначені оператори порівняння цілих контейнерів

`= =` `!=` `<` `<=` `>` `>=`

які працюють подібно до порівняння текстових рядків `string`.

Найчастіше для роботи з списком застосовують багатократне повторення пари дій:

```
(пересунути ітератор -> операція) ->
(пересунути ітератор -> операція) ->
 і т.д.
```

Зауваження. Функціональне опрацювання елементів списку виконують власними функціями і алгоритмами бібліотеки STL. Для доступу до алгоритмів бібліотеки STL в програму треба включити директиву `#include <algorithm>`.

Приклад задачі застосування списку

Задача. Транспортна компанія забезпечує доставку товарів замовникам і планує перевезення на один тиждень. Кожен рейс доставки будемо визначати для простоти трьома показниками: номер дня тижня; час в годинах і хвилинах; назва товару. Скласти список перевезень, виконати редагування і опрацювання списку, надрукувати результати операцій.

Приклад списку перевезень (файл `rs1.txt`):

```
4   11   30   комплект стільців
3   9   45   холодильник
1   14   30   меблі
1   10   00   віконні блоки
5   16   20   канцтовари і книжки
1   9   00   молочні продукти
3   18   10   кондитерські вироби          <---- кінець файла !
```

Отже, кожен елемент списку буде записом з трьох полів. Для такого запису потрібно визначити і перевантажити всі операції, які можуть бути потрібними в процесі опрацювання списку з боку бібліотеки STL, зокрема:

1) для елемента списку визначити конструктор за замовчуванням (без параметрів);

2) визначити функцію від двох параметрів (бінарний предикат), яка визначає, чи пара елементів 1-2 є *в порядку неспадання* (`true`, якщо `par1<=par2`, `false` інакше);

3) визначити функцію від одного параметра (унарний предикат), яка визначає, чи відповідає елемент потрібній умові (`true/false`). Оскільки в нашій задачі кожен елемент списку є записом, то потрібний доступ до його полів. Всім таким зазначеним умовам щодо функцій відповідають в загальному випадку *дружні функції*, зокрема, перевантажені операції порівняння "<", ">", "!=", "= =", визначені як дружні функції. А заодно їх можна використати у власних функціях безпосередньо.

З свого боку потрібно визначити конструктор з параметрами для будови окремого елемента списку і дружню функцію друкування елемента списку.

Нижче подано повний роздрук тексту програми. Для його аналізу потрібно одночасно переглядати за довідковою системою точне визначення методів класу `list` і алгоритмів STL.

```
// Задача: СПИСОК ТРАНСПОРТНИХ ПЕРЕВЕЗЕНЬ
#include  <iostream>
#include  <fstream>
#include <iomanip>  // використання маніпуляторів з параметрами
#include  <string>
#include <list>  // клас list
#include <algorithm>  // доступ до алгоритмів бібліотеки STL
using  namespace  std;

// назви днів тижня
const string nameday[8] = {
   string("не визначено"),
   string("понеділок"), string("вівторок"), string("середа"),
   string("четвер"), string("п'ятниця"),
   string("субота"), string("неділя")
                              };
```

```
// допоміжна структура - покази часу: година, хвилини
struct ftime {
  int h, m;  // година, хвилини
  ftime()  { h=8; m=0; }  // конструктор без параметрів
            // - час 8:00  [ о 8-ій можна починати роботу ☺ ]
  ftime(int ph, int pm) { h=ph; m=pm; } // констр. з параметрами
  friend ostream & operator<< (ostream & os, const ftime & ob)
  {  // друкувати час
     os << right << setfill(' ') << setw(2) << ob.h << ':' <<
         setfill('0') << setw(2) << ob.m << setfill(' ');
     return os;
  } // friend ostream & operator<<

  bool operator< (const ftime & f2) const
  // const - функція не модифікує об'єкт ("лише для читання")
  { return (h < f2.h) || ((h == f2.h) && (m < f2.m)) ; }

  bool operator> (const ftime & f2) const
  { return (h > f2.h) || ((h == f2.h) && (m > f2.m)) ; }

  bool operator== (const ftime & f2) const
  { return (h == f2.h) && (m == f2.m) ; }

  bool operator!= (const ftime & f2) const
  { return (h != f2.h) || (m != f2.m) ; }
}; // struct ftime
```

```
// основна структура - показники рейсу
struct trans
{
  int Nday;  // номер дня тижня
  string day;  // назва дня тижня
  ftime tm;  // час в годинах і хвилинах
  string ware;  // назва товару

  trans() { Nday=0; day=nameday[Nday]; tm=ftime(); ware="**"; }
       // потрібно мати конструктор без параметрів (вимога STL)

  trans(int d, ftime & f, string & w)  // конструктор "для себе"
  {  if(d<0 || d>7) d=0;  Nday=d; day=nameday[Nday];
     tm=f; ware=w;  }

  // вимога STL: операції порівняння - з двома параметрами
  //    (перевантаження дружніми функціями)
  //    потрібно обрати трактування операцій порівняння
  //    - не обов'язково симетричне !
  // серед операцій порівняння обов'язковою є лише operator<
  friend bool operator< (const trans & o1, const trans & o2)
  { return (o1.Nday < o2.Nday) ||
    ((o1.Nday == o2.Nday) && (o1.tm < o2.tm)) ; }
```

```
// додамо інші операції порівняння "на перспективу"
//   - щоб не повертатись до цього питання ☺

friend bool operator> (const trans & o1, const trans & o2)
{ return (o1.Nday > o2.Nday) ||
  ((o1.Nday == o2.Nday) && (o1.tm > o2.tm)) ; }

friend bool operator== (const trans & o1, const trans & o2)
{ return (o1.Nday == o2.Nday) && (o1.tm == o2.tm) &&
  (o1.ware == o2.ware) ; }

friend bool operator!= (const trans & o1, const trans & o2)
{ return (o1.Nday != o2.Nday) || (o1.tm != o2.tm) ||
  (o1.ware != o2.ware) ; }

friend ostream & operator<< (ostream & os, const trans & ob)
{ // друкувати показники рейсу
  os << ob.day << "   " << ob.tm << "   " << ob.ware << endl;
  return os;
}
```

```
  // додаткові функції опрацювання елемента
  //  - для операцій з списком

  friend void transmit(trans & ob)
  { // перенесення рейсу з понеділка на вівторок
    if(ob.Nday==1) { ob.Nday=2; ob.day=nameday[ob.Nday]; }  }

  friend bool after14(trans & ob)
  { // перевірити час рейсу: чи година 14 і пізніше ?
    return  ob.tm.h >= 14 ; }

}; // struct trans  основна структура - показники рейсу


// списки перевезень визначаємо як глобальні величини
//   - для спрощення доступу

list<trans> van;  // початково список порожній
list<trans> trans14;
```

```
// функція: читати з файла перелік рейсів
//   і побудувати початковий список
void getlistfromfile(char * filename)  // параметр - ім'я файла
{
  trans temp;  // будуємо черговий елемент списку
  int hh, mm;  string st;
  ifstream from(filename);

  // цикл читання рядків до вичерпання файла
  while( ! from.eof() )
  {
    from >> temp.Nday;  temp.day = nameday[temp.Nday];
    from >> hh >> mm;  temp.tm = ftime(hh,mm);
    getline ( from, st );  // st = решта рядка до Enter
    int p = st.find_first_not_of(' ');
            // відкинути префіксні пропуски
    temp.ware.assign(st,p,string::npos);
    van.push_back(temp);  // додати елемент в кінець списку
  }  // у файлі не має бути порожніх рядків

  from.close();
} // void getlistfromfile(char * filename)
```

```
// функція: надрукувати всі елементи списку LT
//    в заданий файл в режимі mode, заголовок title
void putlistintofile(list<trans> & LT, char * filename,
          string title = "----------\n",
          ios::openmode mode=ios::out)
{
  list<trans>::iterator  it; // ітератор перегляду списку
  ofstream outdata(filename, mode);
        // відкрити файл в режимі mode - ios::out або ios::app
  outdata << title;
  for(it = LT.begin(); it != LT.end(); ++it)  outdata << *it;
  outdata.close();
} //  void putlistintofile
```


```
void main()
{
  getlistfromfile("rs1.txt");
  // додати до списку ще один рейс
  van.push_back(trans(3,ftime(15,20),string("цемент,пісок")));
  putlistintofile(van, "out1.txt",
                  "-----Заданий список (остаточно)-----\n");
  // рейси з понеділка перенести на вівторок
  //  (поміняти день тижня)
  for_each(van.begin(), van.end(), transmit);
           // алгоритм for_each бібліотеки STL
  putlistintofile(van, "out1.txt",
     "-----Список з перенесеними рейсами-----\n", ios::app);
     // додати до файла
  // побудувати окремий список рейсів trans14
  //   за часом від 14:00 і пізніше
  trans14.clear();  // спочатку зробити пустим
                    // (можливо, від попередніх даних)
  // переглядаємо список рейсів "вручну" і копіюємо потрібні
  list<trans>::iterator  p; // ітератор перегляду списку
  for(p=van.begin(); p != van.end(); ++p)
    if( p->tm.h >= 14 )  trans14.push_back(*p);
  putlistintofile(trans14, "out1.txt",
                   "-----Рейси після 14:00-----\n", ios::app);
                  // додати до файла
```

```
  // викреслити рейси після 14:00, які перенесли до списку trans14
  list<trans>::iterator  temp = remove_if(van.begin(), van.end(),
          after14);  // алгоритм remove_if бібліотеки STL
  /* алгоритм remove_if насправді зсуває залишені елементи
     до початку списку;  результатом алгоритму є ітератор,
     який вказує на останній залишений елемент, точніше,
     праву межу діапазону з залишеними елементами;
     тому додатково треба вкоротити список до залишених елементів
  */
  van.erase(temp, van.end()); // реальне викресл. елементів списку
  // сортувати решту залишкового списку за днями тижня і часом
  van.sort( ::operator< );  // окремо слово operator є ключовим,
                            //   а  ::operator<  є іменем функції !

  putlistintofile(van, "out1.txt",
     "-----Рейси, які залишились (за часом)-----\n", ios::app);
     // додати до файла
  // якщо список більше не потрібний, його краще очистити
  //  - спрацюють деструктори, якщо є
  van.clear();  trans14.clear();
  cout << "size of lists: " << van.size() << '\t' <<
          trans14.size() << endl;

  system("pause");  // затримати вікно консолі
}  // void main()
```

Підсумковий зміст файла результатів `out1.txt` є такий:

```
-----Заданий список (остаточно)-----
четвер   11:30   комплект стільців
середа    9:45   холодильник
понеділок   14:30   меблі
понеділок   10:00   віконні блоки
п'ятниця   16:20   канцтовари і книжки
понеділок    9:00   молочні продукти
середа   18:10   кондитерські вироби
середа   15:20   цемент,пісок
-----Список з перенесеними рейсами-----
четвер   11:30   комплект стільців
середа    9:45   холодильник
вівторок   14:30   меблі
вівторок   10:00   віконні блоки
п'ятниця   16:20   канцтовари і книжки
вівторок    9:00   молочні продукти
середа   18:10   кондитерські вироби
середа   15:20   цемент,пісок
-----Рейси після 14:00-----
вівторок   14:30   меблі
п'ятниця   16:20   канцтовари і книжки
середа   18:10   кондитерські вироби
середа   15:20   цемент,пісок
-----Рейси, які залишились (за часом)-----
вівторок    9:00   молочні продукти
вівторок   10:00   віконні блоки
середа    9:45   холодильник
четвер   11:30   комплект стільців
```