Тема 17

Алгоритмы STL

Определение последовательности

Последовательность –набор однотипных элементов данных, для которых определено понятие «следующий элемент последовательности» (это понятие определено для всех элементов, кроме последнего).

Примеры последовательностей:

- массивы
- контейнерные классы из библиотеки STL
- строки (класс string)
- потоки

Итераторы, указатели, интервалы

- Итератор – инструмент для доступа к элементам последовательности
- Указатель (имя массива) – частный случай итератора, если в качестве последовательности используется массив
- Интервал – идущие подряд элементы последовательности. Интервал задается парой итераторов. Первый итератор связан с первым элементом интервала, второй итератор – с первым элементом, следующим за интервалом. Второй итератор может быть недействительным

Примеры итераторов и интервалов

Сортировка массива

```
long M[50];
// заполнение массива

sort(M, M+50);
```

Сортировка контейнера vector

```
vector <long> M;
// заполнение контейнера

sort(M.begin(), M.end());
```

Операции над итераторами

Общие операции

```
*i – доступ к элементу последовательности,
  связанному с итератором i
i++   ++i    i = j    i == j    i != j
```

Входные итераторы

```
t = *i
```

Выходные итераторы

```
*i = t
```

Двусторонние итераторы

```
i--  --i
```

Итераторы произвольного доступа

```
i+n   i-n   i+=n   i<j
```

Случаи возникновения недействительных итераторов

- итератор не был инициализирован;
- итератор указывает на конец последовательности;
- элемент контейнера, с которым он связан, удален;
- контейнер, с которым он связан, изменил размеры или уничтожен.

Функциональные классы

- Функциональный класс – класс, среди методов которого имеется переопределенный оператор вызова функции

Пример функционального класса

```
class isbest{
public:
 int operator() (int a, int b){
     return (a>b || a%10==0 ? a : b);
 }
};
…
isbest b;
int x, y;
cout << b(x, y);
```

Часто используемые виды функциональных объектов

бинарная функция	T3 (T1, T2)
унарная функция	T3 (T1)
бинарный предикат	bool (T1, T2)
унарный предикат	bool (T1)

Шаблоны стандартных функциональных объектов

Название	Вид	Результат
plus	T (T, T)	x + y
minus	T (T, T)	x – y
multiplies	T (T, T)	x * y
divides	T (T, T)	x / y
modulus	T (T, T)	x % y
negate	T (T)	- x
equal_to	bool (T, T)	x == y
not_equal_to	bool (T, T)	x != y
greater	bool (T, T)	x > y
less	bool (T, T)	x < y
greater_equal	bool (T, T)	x >= y
less_equal	bool (T, T)	x <= y

Отрицатели и связыватели

Название	Вид	Результат
not1	bool (bool(T))	отрицание унарного предиката
not2	bool (bool (T, T))	отрицание бинарного предиката
bind2nd	bool(bool (T t1, T t2), const T t3)	преобразует бинарный предикат в унарный, подставляя значение t3 вместо t2 и t1 соответственно
bind1st	bool(bool (T t1, T t2), const T t3)	

Пример использования отрицателей

```
struct isbest : binary_function <int, int, bool> {
bool operator() (int a, int b) const {
     return (a>b && a%10==0 ? true : false);}
};

int M[20];
```

Желаем получить:

```
sort(M, M+20, isbest()); //сортирует по «лучшести»
sort(M, M+20, not2(isbest())); //сортирует по
                               //«худшести»
```

Пример использования отрицателей (продолжение)

```
srand(375294625);
  for (int i=0; i<20; i++)
   M[i] = rand()%500;
  sort(M, M+20);
  for (int i=0; i<20; i++)
   cout << M[i] << " ";
  cout << endl;
```

Результат:

17 23 32 33 33 61 80 100 168 178 276 300 324 342 373 388 392 411 434 469

Пример использования отрицателей (продолжение)

```
srand(375294625);
  for (int i=0; i<20; i++)
   M[i] = rand()%500;
  sort(M, M+20, isbest());
  for (int i=0; i<20; i++)
   cout << M[i] << " ";
  cout << endl;
```

Результат:

300 168 411 33 392 324 23 373 61 32 434 33 178 276 469 100 80 388 17 342

Пример использования отрицателей (продолжение)

```
srand(375294625);
  for (int i=0; i<20; i++)
   M[i] = rand()%500;
  sort(M, M+20, not2(isbest()));
  for (int i=0; i<20; i++)
   cout << M[i] << " ";
  cout << endl;
```

Результат: **ошибка выполнения**

Причина: для a==b isbest (a, b) и isbest (b, a) возвращают **true**

Пример использования отрицателей (продолжение)

Другие функции «лучшести»

```
{return (a>=b || a%10==0 ? true : false); }
```

Результат:

```
sort(M, M+20, isbest()); // ошибка выполнения
sort(M, M+20, not2(isbest())); // нормально
```

```
{ if (a==b) return false;
return ((a>b) && (a%10==0) ? true : false); }
```

Результат:

```
sort(M, M+20, isbest()); // нормально
sort(M, M+20, not2(isbest())); // ошибка
  выполнения
```

Использование функциональных объектов в алгоритмах

алгоритм count_if определяет число элементов интервала, для которых верен заданный унарный предикат (функция либо функциональный объект)

```
int count_if (интервал, унарный_предикат)
```

Примеры использования функциональных объектов

Задача: подсчитать количество элементов вектора, описанного как

```
vector <long> d;
```

больших пяти.

Первый вариант решения (функция)

```
bool gr5(long a) { return (a>5); }
…
cout << count_if (d.begin(), d.end(), gr5)
     << endl;
```

Примеры использования функциональных объектов

Второй вариант решения (функциональный объект)

```
class _gr5 {
public:
 bool operator() (long a) { return (a>5); }
};
…
cout << count_if (d.begin(), d.end(), _gr5())
     << endl;
```

Примеры использования функциональных объектов

Третий вариант решения (функциональный объект, выражение вместо константы)

```
class _gr {
   long _n;
public:
   _gr(long an) { _n = an; }
   bool operator() (long a) { return (a>_n); }
};
…
long k;
…
cout << count_if (d.begin(), d.end(), _gr(k))
     << endl;
```

Примеры использования функциональных объектов

Четвертый вариант решения (стандартные функциональные объекты и связыватели)

```
long k;
…
cout << count_if (d.begin(), d.end(),
   bind2nd(greater <long> (), 5)) << endl;
```

Немодифицирующие алгоритмы

- не изменяется ни последовательность, ни её элементы;
- передаётся интервал последовательности, заданный парой итераторов;
- контроля за выходом за границы последовательности нет.

Немодифицирующие алгоритмы

Описание последовательности для последующих примеров:

```
typedef deque <long> ldeque;
typedef ldeque::iterator lit;
ldeque d;
lit it1, it2, it3;
```

Немодифицирующие алгоритмы: count, count_if

считают количество значений последовательности, равных заданному, либо количество значений, для которых справедлив заданный предикат.

Примеры:

```
cout << count(d.begin(), d.end(), 15);
// считает количество элементов, равных 15

cout << count_if(d.begin(), d.end(),
  bind2nd(less <long> (), 15));
// считает количество элементов, меньших 15
```

Немодифицирующие алгоритмы: find, find_if

возвращают итератор на первый элемент последовательности, равный заданному, либо для которого справедлив заданный унарный предикат.

Пример:

```
// найти в деке d начальный элемент массива p
cout << *(find(d.begin(), d.end(), p[0]));
// неверно, поскольку элемента может и не быть
// правильный вариант использования алгоритма:
if ((it1 = find(d.begin(), d.end(), p[0]))
                    == d.end())
  cout << "Not found";
else
  cout << *it1;
```

Немодифицирующие алгоритмы: find, find_if

```
// Вывести в поток cout все значения дека,
// большие 7:

it1 = d.begin();
while (it1 != d.end()) {
  it2 = find_if(it1, d.end(),
                bind2nd(greater<long>(), 7));
  if (it2 == d.end())
     break;
  cout << *it2 << " ";
  it1=(++it2);
}
```

Немодифицирующие алгоритмы: for_each

вызывает для каждого элемента последовательности заданную callback-функцию вида со спецификацией T1 (T), где T – тип хранящихся элементов, а T1 чаще всего – **void**.

Пример:

```
//вывести в стандартный поток остаток от
// деления каждого элемента дека на 7
void mod7(long a) { cout << a%7 << " "; }
for_each(d.begin(), d.end(), mod7);
cout << endl;
```

Немодифицирующие алгоритмы: equal

сравнивает две последовательности на попарное совпадение элементов.

Формат алгоритма equal:

```
bool equal(нач1, кон1, нач2);
bool equal(нач1, кон1, нач2, предикат);
```

- два первых параметра – итераторы, задающие первую последовательность;
- третий параметр задаёт начальный элемент второй последовательности (её размер равен размеру первой);
- четвёртый параметр – бинарный предикат, задающий правила сравнения

Немодифицирующие алгоритмы: equal

Пример:

```
int A1[] = { 3, 1, 4, 1, 5, 9, 3 };
int A2[] = { 3, 1, 4, 2, 8, 5, 7, 2, 5 };
const int N = sizeof(A1) / sizeof(int);
cout << "Result of comparison: "
                << equal(A1, A1 + N, A2) << endl;
  // результат – ложь

cout << "Result of comparison: "
                << equal(A1, A1 + 3, A2) << endl;
  // результат - истина
```

Немодифицирующие алгоритмы: mismatch

сравнивает две последовательности на попарное совпадение элементов и возвращает пару итераторов на первые несовпадающие элементы.

Формат алгоритма mismatch:

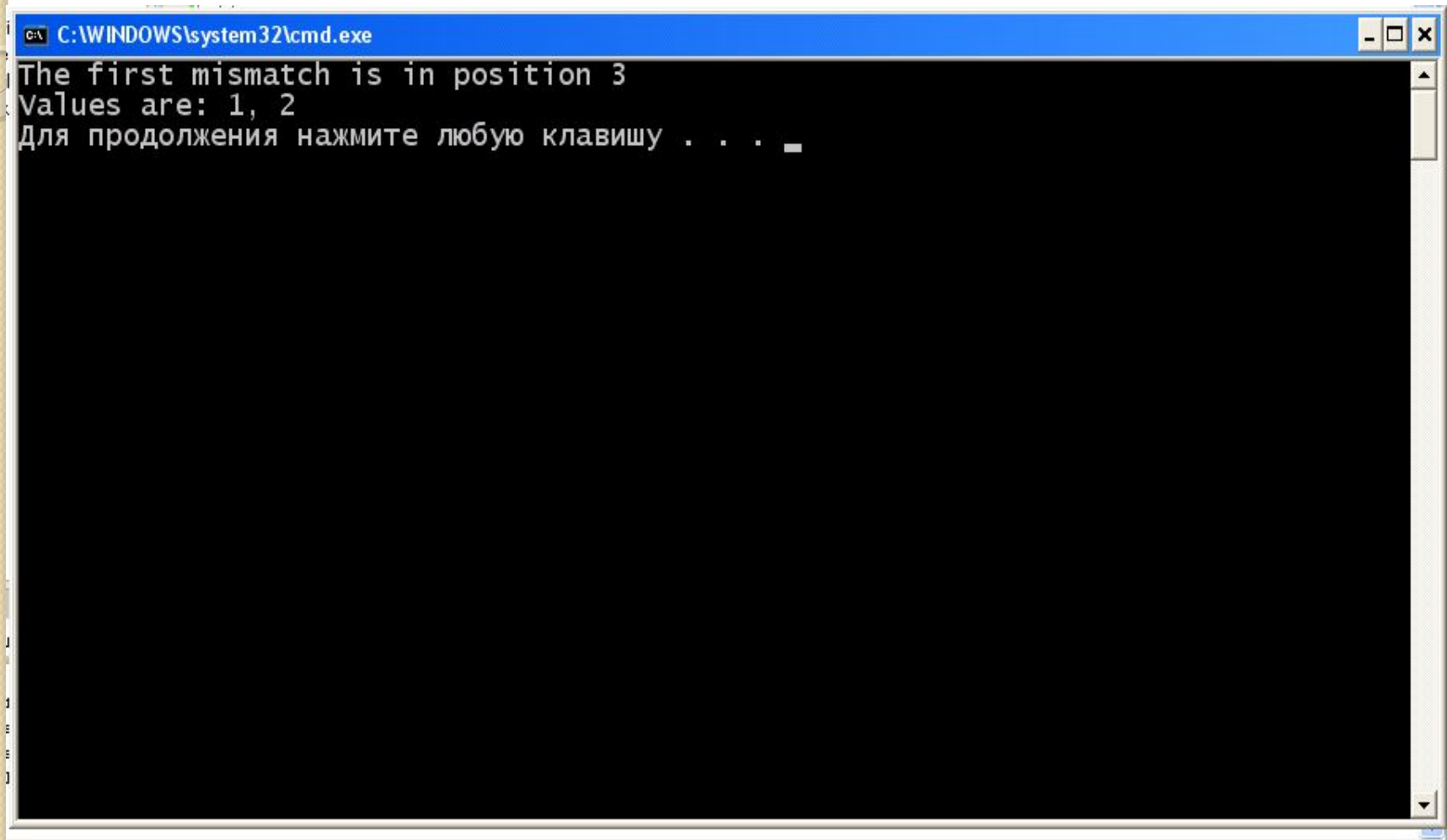
```
pair <итератор1, итератор2> mismatch (нач1,
  кон1, нач2);
pair <итератор1, итератор2> mismatch (нач1,
  кон1, нач2, предикат);
```

Немодифицирующие алгоритмы: mismatch

Пример:

```
int A1[] = { 3, 1, 4, 1, 5, 9, 3 };
int A2[] = { 3, 1, 4, 2, 8, 5, 7, 2, 5 };
const int N = sizeof(A1) / sizeof(int);


pair<int*, int*> result = mismatch(A1, A1 + N,
  A2);
if (result.first != A1 + N) {
  cout << "The first mismatch is in position "
            << result.first - A1 << endl;
  cout << "Values are: " << *(result.first)
            << ", " << *(result.second) <<
  endl;
}
```

Немодифицирующие алгоритмы: mismatch



Немодифицирующие алгоритмы: search, search_n

Алгоритм search находит первое вхождение в первую последовательность второй последовательности (в качестве подпоследовательности) и возвращает итератор на первый совпадающий элемент.

Алгоритм search_n находит в последовательности первую подпоследовательность из нескольких одинаковых элементов с заданным значением и возвращает итератор на ее начало.

Формат алгоритмов:

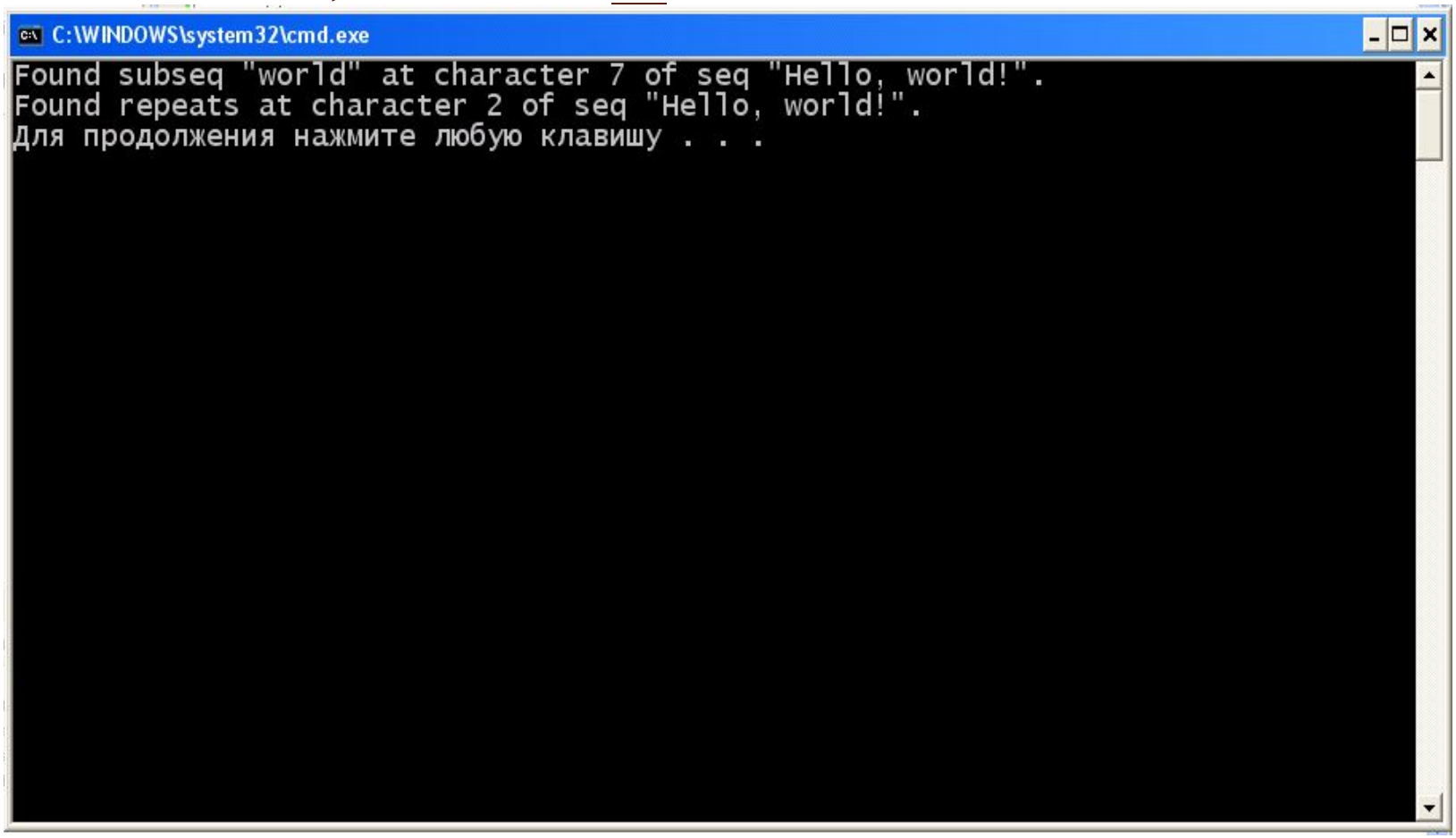
```
итератор search (нач1, кон1, нач2, кон2);
итератор search_n (нач1, кон1, количество,
  значение);
```

Немодифицирующие алгоритмы: search, search_n

Пример:

```
char S1[] = "Hello, world!";
char S2[] = "world";
const int N1 = sizeof(S1) - 1;
const int N2 = sizeof(S2) - 1;

char* p = search(S1, S1 + N1, S2, S2 + N2);
printf("Found subseq \"%s\" at character %d of
  seq \"%s\".\n", S2, p - S1, S1);

p = search_n(S1, S1 + N1, 2, 'l');
printf("Found repeats at character %d of seq
  \"%s\".\n", p - S1, S1);
```

Немодифицирующие алгоритмы: search, search_n



Модифицирующие алгоритмы

- структура последовательности не изменяется, изменяются только её элементы;
- после «удаления» некоторых элементов освободившееся место заполняется мусором;
- передаётся интервал последовательности, заданный парой итераторов;
- контроля за выходом за границы последовательности нет.

Модифицирующие алгоритмы: copy, copy_backward

Алгоритмы поэлементно копируют входную последовательность1 в выходную последовательность2 (которая задана только одним итератором!). Первый алгоритм перемещает элементы в сторону увеличения итераторов выходной последовательности, второй – в сторону уменьшения

Формат алгоритмов:

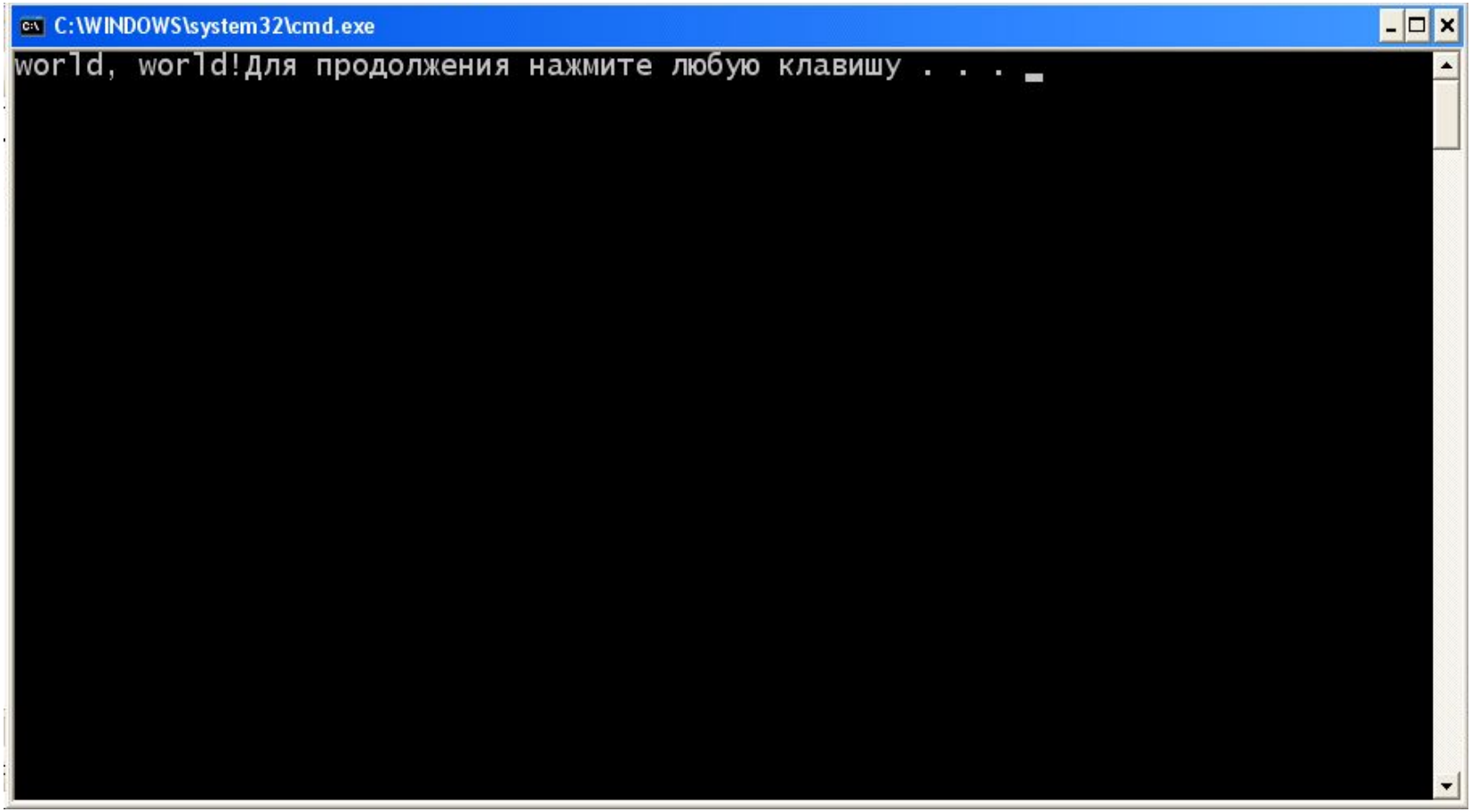
```
кон2 copy (нач1, кон1, нач2);
нач2 copy_backward (нач1, кон1, кон2);
```

Модифицирующие алгоритмы: copy, copy_backward

Пример:

```
char S1[] = "Hello, world!";
char S2[] = "world";
const int N1 = sizeof(S1) - 1;
const int N2 = sizeof(S2) - 1;

copy(S2, S2+N2, S1);
// вместо этого можно записать
// copy_backward(S2, S2+N2, S1+N2);
copy(S1, S1+N1, ostream_iterator <char> (cout));
```

Модифицирующие алгоритмы: copy, copy_backward



Модифицирующие алгоритмы: fill, fill_n

Эти алгоритмы позволяют заполнить последовательность (или несколько ее элементов) заданным значением.

Формат алгоритмов:

```
void fill (интервал, значение);
void fill_n (нач, количество, значение);
```

Примеры:

```
fill(d.first(), d.last(), 113);
fill_n(d.first(), 5, 113)
```

Модифицирующие алгоритмы: generate, generate_n

Алгоритмы заполняют элементы последовательности значениями функции-генератора, которая указывается при вызове алгоритма. Функция-генератор не имеет параметров, а тип возвращаемого значения соответствует типу элементов последовательности.

Формат алгоритмов:

```
void generate (интервал, генератор);
void generate_n (нач, количество, генератор);
```

Модифицирующие алгоритмы: generate, generate_n

Пример:

```
// заполнить последовательность случайными числами
// из заданного интервала
class my_gen {
  long __a, __b;
public:
  my_gen(long a, long b) {
   srand((unsigned) time(NULL));
   __a=a; __b=b;
  }
  long operator() () {
     return __a+(rand()%(__b-__a));
   }
};
…
generate(d.begin(), d.end(), my_gen(1000, 1500));
```

Модифицирующие алгоритмы: replace, replace_if

Эти алгоритмы заменяют элементы последовательности с заданным старым значением (или удовлетворяющие заданному условию) на новое значение.

Пример:

```
// заменить в последовательности элементы,
// большие 100, на 100
replace_if(d.begin(), d.end(),
        bind2nd(greater<long>(), 100), 100);
```

Модифицирующие алгоритмы: remove, remove_if

Эти алгоритмы «удаляют» элементы последовательности с заданным значением (или удовлетворяющие заданному условию), перенося оставшиеся элементы в начало последовательности. Освободившееся место заполняется «мусором». Функции возвращают итератор на начало мусора.

Формат алгоритмов

```
итератор remove (интервал, значение)
итератор remove_if (интервал, условие)
```

Модифицирующие алгоритмы: remove, remove_if

Примеры:

Удалить из последовательности все элементы, большие 5, и вывести в стандартный поток cout оставшиеся элементы:

```
copy(d.begin(),
      remove_if(d.begin(), d.end(),
      bind2nd(greater <long> (), 5)),
  ostream_iterator <long> (cout, " "));
```

Для физического удаления этих элементов можно записать следующий оператор:

```
d.erase(remove_if(d.begin(), d.end(),
       bind2nd(greater<long>(), 5)), d.end());
```

Модифицирующие алгоритмы: unique

Эти алгоритмы «удаляют» повторяющиеся элементы последовательности, оставляя только первый элемент. Освободившееся место заполняется «мусором». Функции возвращают итератор на начало мусора.

Формат алгоритма

```
итератор unique (интервал)
итератор unique (интервал, условие_совпадения)
```

Пример:

```
copy(d.begin(),
      unique(d.begin(), d.end()),
ostream_iterator <long> (cout, " "));
```

Алгоритмы, связанные с сортировкой и поиском

- sort
- stable_sort (порядок одинаковых элементов не изменяется)
- partial_sort (сортируются первые несколько элементов, задаётся позиция элемента, до которого выполняется сортировка)
- nth_element (последовательность разбивается на две, в первой содержится n элементов, и каждый элемент первой подпоследовательности не превосходит любого элемента второй, задаётся позиция элемента, до которого выделяется первая последовательность)
- binary_search (поиск с помощью дихотомии в отсортированном массиве)

Алгоритмы, связанные с сортировкой и поиском

Примеры:

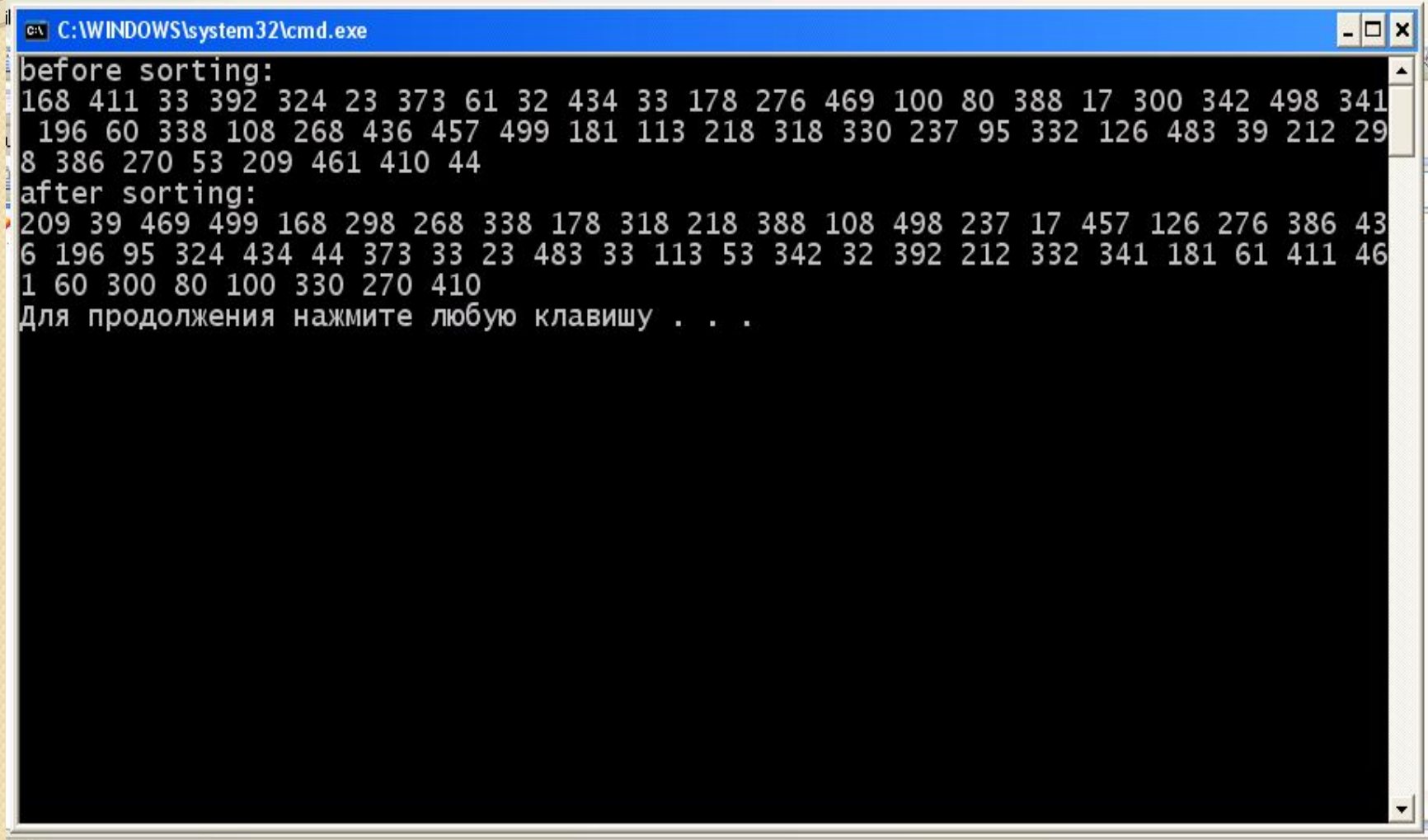
```
struct isbest : binary_function<int, int, bool>
  {
bool operator() (int a, int b) const {
   int _a=a%10, _b=b%10;
   if (_a==_b) return false;
     return (_a>_b);
 }
};

long M[50];
…
```

Алгоритмы, связанные с сортировкой и поиском

```
srand(375294625);
for (int i=0; i<50; i++)
  M[i] = rand()%500;
cout << "before sorting: " << endl;
for (int i=0; i<50; i++)
  cout << M[i] << " ";
cout << endl;
sort(M, M+50, isbest());
cout << "after sorting: " << endl;
for (int i=0; i<50; i++)
  cout << M[i] << " ";
cout << endl;
```

Алгоритмы, связанные с сортировкой и поиском

Результат:

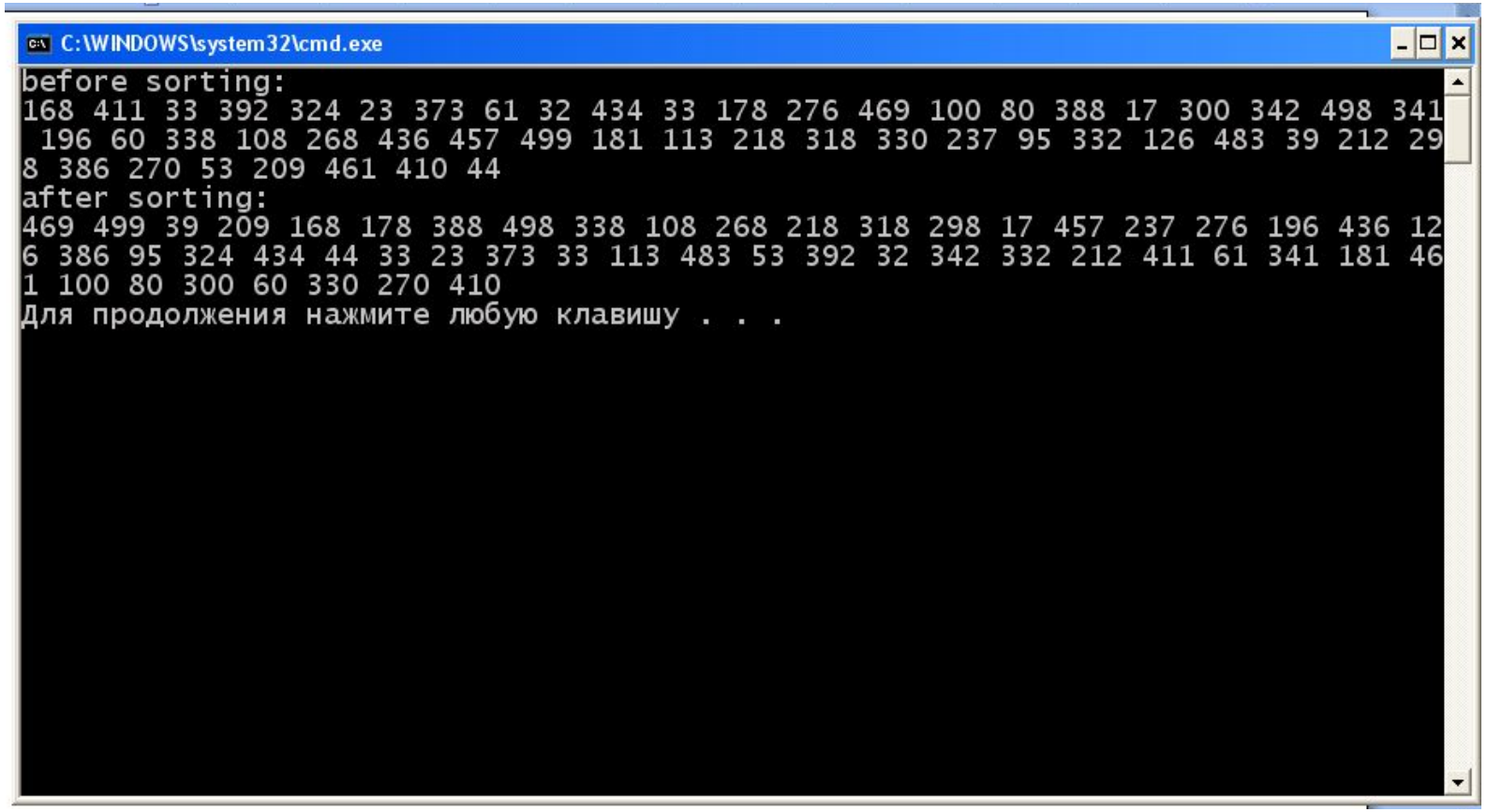


Алгоритмы, связанные с сортировкой и поиском

```
srand(375294625);
for (int i=0; i<50; i++)
   M[i] = rand()%500;
cout << "before sorting: " << endl;
for (int i=0; i<50; i++)
   cout << M[i] << " ";
cout << endl;
stable_sort(M, M+50, isbest());
  cout << "after sorting: " << endl;
  for (int i=0; i<50; i++)
   cout << M[i] << " ";
cout << endl;
```

Алгоритмы, связанные с сортировкой и поиском

Результат:

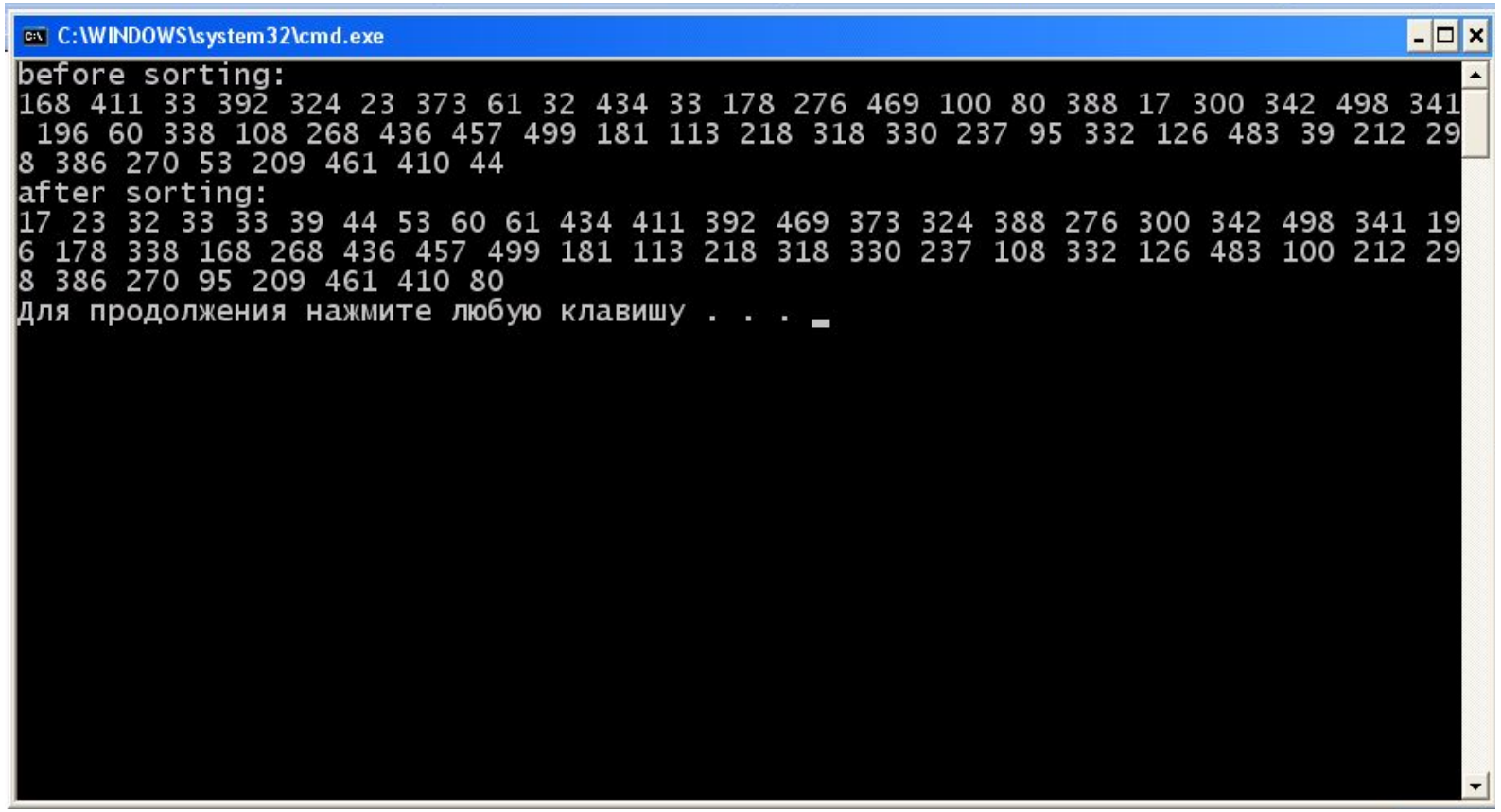


Алгоритмы, связанные с сортировкой и поиском

```
srand(375294625);
for (int i=0; i<50; i++)
   M[i] = rand()%500;
cout << "before sorting: " << endl;
for (int i=0; i<50; i++)
   cout << M[i] << " ";
cout << endl;
partial_sort(M, M+10, M+50);
  cout << "after sorting: " << endl;
  for (int i=0; i<50; i++)
   cout << M[i] << " ";
cout << endl;
```

Алгоритмы, связанные с сортировкой и поиском

Результат:

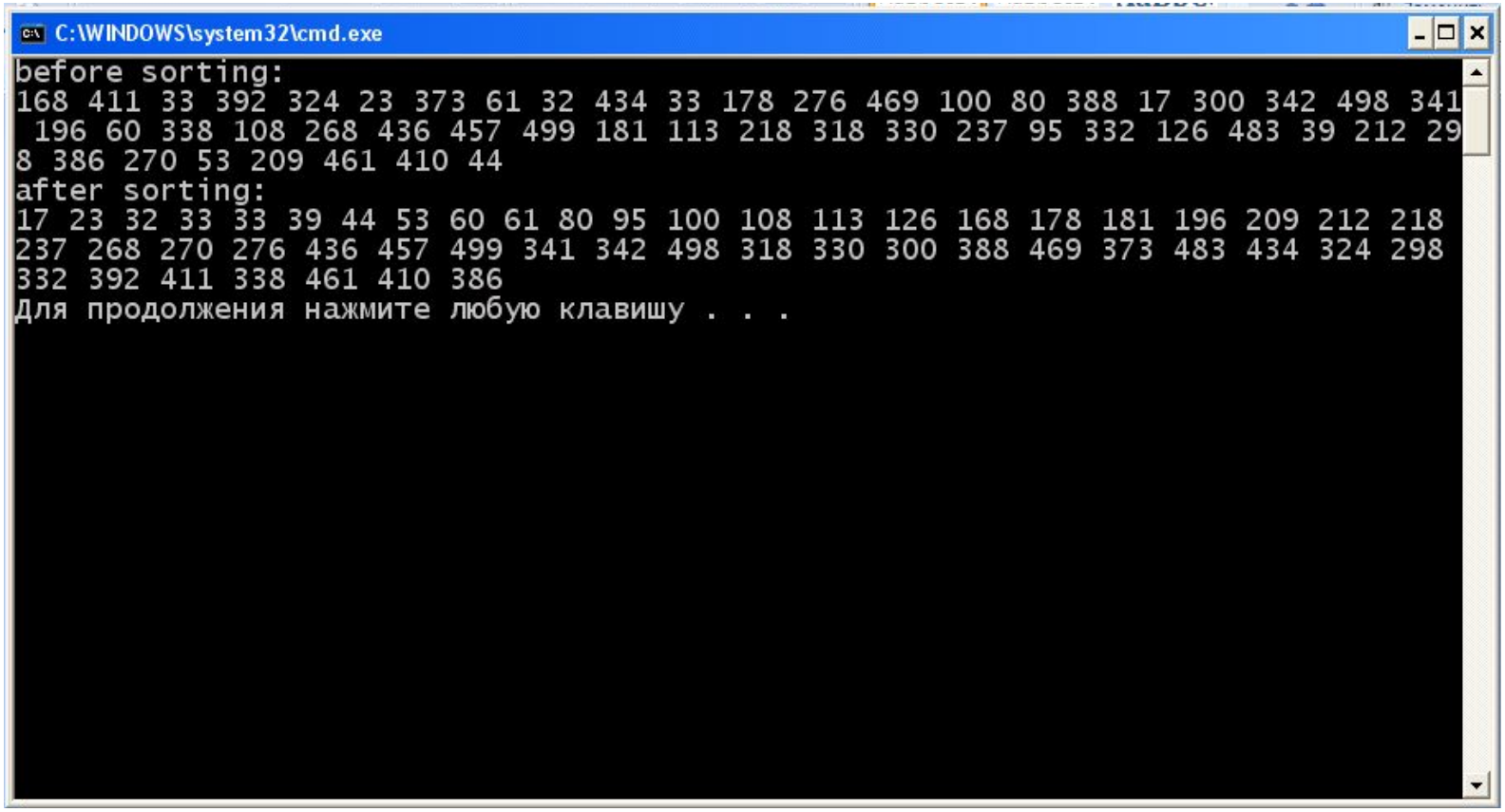


Алгоритмы, связанные с сортировкой и поиском

```
srand(375294625);
for (int i=0; i<50; i++)
   M[i] = rand()%500;
cout << "before sorting: " << endl;
for (int i=0; i<50; i++)
   cout << M[i] << " ";
cout << endl;
nth_element(M, M+10, M+50);
  cout << "after sorting: " << endl;
  for (int i=0; i<50; i++)
   cout << M[i] << " ";
cout << endl;

// сортировка не обязательно должна получиться!
```

Алгоритмы, связанные с сортировкой и поиском

Результат:



Алгоритмы, связанные с сортировкой и поиском

```
srand(375294625);
for (int i=0; i<50; i++)
  M[i] = rand()%500;
cout << "before sorting: " << endl;
for (int i=0; i<50; i++)
  cout << M[i] << " ";
cout << endl;
sort(M, M+50);
 cout << "after sorting: " << endl;
for (int i=0; i<50; i++)
  cout << M[i] << " ";
cout << endl;
 srand((unsigned) time(NULL));
for (int i = 1; i <= 10; ++i) {
  int k = rand()%500;
  cout << "Searching for " << k << ": " <<
       binary_search(M, M+50, k) ? "present" :
       "not present") << endl;
```

Алгоритмы, связанные с сортировкой и поиском

Результат: