БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

факультет радиофизики и компьютерных технологий

кафедра информатики и компьютерных систем

Н.В. Серикова

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

к лабораторному практикуму

Библиотека стандартных шаблонов STL

по курсу

«ОСНОВЫ ООП»

2021

МИНСК

Практическое руководство к лабораторному практикуму «БИБЛИОТЕКА STL» по курсу «ОСНОВЫ ООП» предназначено для студентов, изучающих базовый курс программирования на языке С++, специальностей «Радиофизика», «Физическая электроника», «Компьютерная безопасность», «Прикладная информатика».

Руководство содержит некоторый справочный материал, примеры решения типовых задач с комментариями.

Автор будет признателен всем, кто поделится своими сообра­же­ниями по совершенствованию данного пособия.

Воз­можные предложения и замечания можно присылать по адресу:

*E-mail: [Serikova@bsu.by](mailto:Serikova@bsu.by)*

ОГЛАВЛЕНИЕ

[Строки стандартного класса string 4](#_Toc103353602)

[Методы для работы со строками класса string 5](#_Toc103353603)

[Операции для работы со строками класса string 8](#_Toc103353604)

[Библиотека стандартных шаблонов STL 9](#_Toc103353605)

[Векторы 11](#_Toc103353606)

[Списки 14](#_Toc103353607)

[Ассоциативные списки 18](#_Toc103353608)

[Алгоритмы 21](#_Toc103353609)

[Алгоритм вычисления выражений через обратную польскую запись 25](#_Toc103353610)

[ПРИМЕР 1. Объявление и инициализация строки string 28](#_Toc103353611)

[ПРИМЕР 2. Инициализация строки string. Оператор =. Метод assign 29](#_Toc103353612)

[ПРИМЕР 3. Ввод строки string. Оператор >> 30](#_Toc103353613)

[ПРИМЕР 4. Ввод строки string. Метод getline 30](#_Toc103353614)

[ПРИМЕР 5. Длина строки string. Методы length, size 31](#_Toc103353615)

[ПРИМЕР 6. Доступ к элементу строки string. Оператор []. Метод at 32](#_Toc103353616)

[ПРИМЕР 7. Сравнение строк. Операторы сравнения 33](#_Toc103353617)

[ПРИМЕР 8. Сравнение строк. Метод compare 34](#_Toc103353618)

[ПРИМЕР 9. Объединение строк. Оператор +. Метод append 35](#_Toc103353619)

[ПРИМЕР 10. Вставка строки (подстроки) в строку. Метод insert 36](#_Toc103353620)

[ПРИМЕР 11. Замена строки (подстроки) в строке. Метод replace 37](#_Toc103353621)

[ПРИМЕР 12. Удаление подстроки в строке. Метод erase 38](#_Toc103353622)

[ПРИМЕР 13. Выделение подстроки в строке. Метод substr 39](#_Toc103353623)

[ПРИМЕР 14. Обмен содержимого строк. Метод swap, reverse 40](#_Toc103353624)

[ПРИМЕР 15. Поиск подстроки в строке. Метод find 41](#_Toc103353625)

[ПРИМЕР 16. Поиск символа подстроки в строке. Метод find\_first\_of 42](#_Toc103353626)

[ПРИМЕР 17. Поиск символа подстроки в строке. Метод find\_first\_not\_of 43](#_Toc103353627)

[ПРИМЕР 18. Выделение лексем. Методы find\_first\_not\_of, find\_first\_of 44](#_Toc103353628)

[ПРИМЕР 19. Выделение лексем. Чтение из потока 45](#_Toc103353629)

[ПРИМЕР 20. Строки С и С++. Методы copy, c\_str 46](#_Toc103353630)

[ПРИМЕР 21. Класс-контейнер vector. Основные операции 47](#_Toc103353631)

[ПРИМЕР 22. Класс-контейнер vector. Методы insert и erase 49](#_Toc103353632)

[ПРИМЕР 23. Класс-контейнер vector. Хранение объектов пользовательского класса 51](#_Toc103353633)

[ПРИМЕР 24. Класс-контейнер vector. Матрица (вектор векторов) 53](#_Toc103353634)

[ПРИМЕР 25. Класс-контейнер list. Создание, определение числа элементов, просмотр с удалением 56](#_Toc103353635)

[ПРИМЕР 26. Класс-контейнер list. Просмотр элементов списка в прямом и обратном порядке 57](#_Toc103353636)

[ПРИМЕР 27. Класс-контейнер list. Добавление элементов в конец и начало списка 58](#_Toc103353637)

[ПРИМЕР 28. Класс-контейнер list. Сортировка элементов списка 59](#_Toc103353638)

[ПРИМЕР 29. Класс-контейнер list. Слияние списков 60](#_Toc103353639)

[ПРИМЕР 30. Класс-контейнер list. Использование пользовательского класса 61](#_Toc103353640)

[ПРИМЕР 31. Класс-контейнер map. Создание, поиск 63](#_Toc103353641)

[ПРИМЕР 32. Класс-контейнер map. Алгоритм find 64](#_Toc103353642)

[ПРИМЕР 33. Алгоритмы count и count\_if 66](#_Toc103353643)

[ПРИМЕР 34. Алгоритм remove\_copy 67](#_Toc103353644)

[ПРИМЕР 35. Алгоритм reverse 68](#_Toc103353645)

[ПРИМЕР 36. Алгоритм transform 69](#_Toc103353646)

# Строки стандартного класса string

Язык С++ включает в себя новый класс, называемый *string*. Этот класс во многом улучшает традиционный строковый тип, позволяет обрабатывать строки также как данные других типов, а именно с помощью операторов. Он более эффективен и безопасен в использовании, не нужно заботиться о создании массива нужного размера для строковой переменной, класс *string* берет на себя ответственность за управлением памятью.

Если при создании приложения скорость выполнения не является доминирующим фактором, класс *string* предоставляет безопасный и удобный способ обработки строк.

Для работы со строками класса *string* существует множество методов и операций.

Объявление и инициализация строк string.

//создаем пустую строку

//вызов конструктора без аргументов

string s1;

// создаем строку из C-строки

//вызов конструктора с одним аргументом

string s2("aaaa");

// string s2 = "aaaa"; // или так

// создаем строку из C-строки 10 символов

//вызов конструктора с двумя аргументами

string s3("abcdefghijklmnopqrstuvwxyz",10);

// создаем строку из 5 одинаковых символов

string s4(5,’!’);

// создаем строку-копию из строки s3

string s5(s3);

// создаем строку-копию из строки s3

// начиная с индекса 5 не более 3 символов

string s6(s3,5,3);

## Методы для работы со строками класса string

|  |  |  |  |
| --- | --- | --- | --- |
|  | МЕТОД | ЗАПИСЬ | ОПИСАНИЕ |
| 1 | at | at (unsigned n) | доступ к n-му элементу строки |
| 2 | append | append (string &str) | добавляет строку str к концу вызывающей строки  (тоже, что оператор + ) |
|  |  | append (string &str, unsigned pos, unsigned n); | добавляет к вызывающей строке n символов строки str, начиная с позиции pos |
|  |  | append ( char \*sr, unsigned n); | добавляет к вызывающей строке n символов С-строки s |
| 3 | assign | assign (string &str) | присваивает строку str вызывающей строке (тоже, что s2=s1) |
|  |  | assign ( string &str, unsigned pos, unsigned n); | присваивает вызывающей строке n символов строки str, начиная с позиции pos |
|  |  | assign ( char \*sr, unsigned n); | присваивает вызывающей строке n символов C-строки s |
| 4 | capacity | unsigned int capacity (); | возвращает объем памяти, занимаемый строкой |
| 5 | compare | int compare (string &str); | сравнение двух строк, возвращает значение <0, если вызывающая строка лексикографически меньше str, =0, если строки равны и >0, если вызывающая строка больше |
|  |  | int compare (string &str, unsigned pos, unsigned n); | сравнение со строкой str n символов вызывающей строки, начиная с позиции pos; возвращает значение <0, если вызывающая строка лексикографически меньше str, =0, если строки равны и >0, если вызывающая строка больше |
|  |  | int compare (unsigned pos1, unsigned n1, string &str, unsigned pos2, unsigned n2); | n1 символов вызывающей строки, начиная с позиции pos1, сравниваются с подстрокой строки str длиной n2 символов, начиная с позиции pos2; возвращает значение <0, если вызывающая строка лексикографически меньше str, =0, если строки равны и >0, если вызывающая строка больше |
| 6 | copy | unsigned int copy (char \*s, unsigned n, unsigned pos = 0); | копирует в символьный массив s n элементов вызывающей строки, начиная с позиции pos; нуль-символ в результирующий массив не заносится; метод возвращает количество скопированных элементов |
| 7 | c\_str | char \* c\_str() | возвращает указатель на С-строку, содержащую копию вызываемой строки; полученную С-строку нельзя изменить |
| 8 | empty | bool empty(); | возвращает истину, если строка пустая |
| 9 | erase | erase (unsigned pos = 0, unsigned n = npos); | удаляет n элементов, начиная с позиции pos (если n не задано, то удаляется весь остаток строки) npos-самое большое число >0 типа unsigned |
| 10 | find | unsigned int find (string &str, unsigned pos = 0); | ищет самое левое вхождение строки str в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned , если вхождение не найдено |
|  |  | unsigned find (char c, unsigned pos = 0); | ищет самое левое вхождение символа с в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned, если вхождение не найдено |
|  |  | unsigned rfind (char c, unsigned pos = 0); | ищет самое правое вхождение символа с в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned, если вхождение не найдено |
| 11 | find\_first\_of | unsigned find\_first\_of (string &str, unsigned pos = 0); | ищет самое левое вхождение любого символа строки str в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned, если вхождение не найдено |
| 12 | find\_last\_of | unsigned find\_last\_of (string &str, unsigned pos = 0); | ищет самое правое вхождение любого символа строки str в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned, если вхождение не найдено |
| 13 | find\_first\_not\_of | unsigned find\_first\_not\_of (string &str, unsigned pos = 0); | ищет самый левый символ не равный ни одному символу из строки str в вызывающей строке, начиная с позиции pos; возвращает позицию вхождения, или npos(самое большое число >0 типа unsigned, если вхождение не найдено |
| 14 | insert | insert (unsigned pos, string &str); | вставляет строку str в вызывающую строку, начиная с позиции pos |
|  |  | insert (unsigned pos1, string &str, unsigned pos2,  unsigned n); | вставляет в вызывающую строку, начиная с позиции pos1 n символов строки str, начиная с позиции pos2 |
|  |  | insert (unsigned pos, char \* sr, unsigned n); | вставляет в вызывающую строку n символов C-строки s, начиная с позиции pos |
| 15 | length | unsigned length (); | возвращает размер строки |
| 16 | max\_size | unsigned max\_size(); | возвращает максимальную длину строки |
| 17 | replace | replace (unsigned pos, unsigned n, string &str); | заменяет n элементов, начиная с позиции pos вызывающей строки, элементами строки str |
|  |  | replace (unsigned pos1, unsigned n1, string &str, unsigned pos2, unsigned n2); | заменяет n1 элементов, начиная с позиции pos1 вызывающей строки, n2 элементами строки str, начиная с позиции pos2 |
|  |  | replace (unsigned pos, unsigned n1, char \*s, unsigned n2); | заменяет n1 элементов, начиная с позиции pos вызывающей строки, n2 элементами C-строки s |
| 18 | size | unsigned size(); | возвращает размер строки |
| 19 | substr | string substr(unsigned pos = 0, unsigned n = npos); | выделяет подстроку длиной n из исходной строки, начиная с позиции pos |
| 20 | swap | swap (string &str) | обменивает содержимое вызывающей строки и строки str |

## Операции для работы со строками класса string

|  |  |  |  |
| --- | --- | --- | --- |
|  | оператор | ЗАПИСЬ  string s1,s2,s3; | ОПИСАНИЕ |
| 1 | + | s3 = s1+ s2; | конкатенация (сцепление) строк, можно присоединить единичный символ к строке |
| 2 | += | s1 += s2; | конкатенация (сцепление) строк с присвоением результата |
| 3 | = | s2 = s1; | присваивание |
| 4 | == | s2 == s1 | лексикографическое сравнение на равенство строк |
| 5 | != | s2 != s1 | лексикографическое сравнение на неравенство строк |
| 6 | > | s2 > s1 | лексикографическое сравнение строк на > |
| 7 | >= | s2 >= s1 | лексикографическое сравнение строк на >= |
| 8 | < | s2 < s1 | лексикографическое сравнение строк на < |
| 9 | <= | s2 <= s1 | лексикографическое сравнение строк на <= |
| 10 | [] | s1[i] | индексация (обращение к элементу строки) |
| 11 | >> | cin >> s1; | ввод строки (лучше метод getline) |
| 12 | << | cout << s2; | вывод строки |

# Библиотека стандартных шаблонов STL

Библиотека стандартных шаблонов С++ (Standart Template Library) обеспечивает стандартные классы и функции, которые реализуют наиболее популярные и широко используемые алгоритмы и структуры данных.

В частности, в библиотеке STL поддерживаются вектора (vector), списки (list), очереди (queue), стеки (stack). Определены процедуры доступа к этим структурам данных.

Ядро библиотеки образуют три элемента: контейнеры, алгоритмы и итераторы.

Контейнеры – объекты, предназначенные для хранения других объектов. Например, в класса *vector* определяется динамический массив, в классе *queue* – очередь, в классе *list* – линейный список. В каждом классе-контейнере определяется набор функций для работы с этим контейнером. Например, список содержит функции для вставки, удаления, слияния элементов. В стеке – функции для размещения элемента в стек и извлечения элемента из стеа.

Алгоритмы выполняют операции над содержимым контейнеров. Существуют алгоритмы для инициализации, сортировки, поиска, замены содержимого контейнера.

Итераторы – объекты, которые к контейнерам играют роль указателей. Они позволяют получать доступ к содержимому контейнера примерно так же, как указатели используются для доступа к элементам массива. С итераторами можно работать так же как с указателями.

Существуют 5 типов итераторов.

|  |  |
| --- | --- |
| Итератор | Описание |
| Произвольного доступа | Используется для считывания и записи значений. Доступ к элементам произвольный |
| Двунаправленный | Используется для считывания и записи значений. Может проходить контейнер в обоих наравлениях |
| Однонаправленный | Используется для считывания и записи значений. Может проходить контейнер только в одном наравлении |
| Ввода | Используется только для считывания значений. Может проходить контейнер только в одном наравлении |
| Вывода | Используется только для записи значений. Может проходить контейнер только в одном наравлении |

Классы-контейнеры, определенные в STL

|  |  |  |
| --- | --- | --- |
| Контейнер | Описание | Заголовок |
| bitset | Множество битов | <bitset> |
| deque | Двустороняя очередь | <deque> |
| list | Линейный список | <list> |
| map | Ассоциативный список для хранения пар (ключ/ значение), где с каждым ключом связано одно значение | <map> |
| multimap | Ассоциативный список для хранения пар (ключ/ значение), где с каждым ключом связано два или более значений | <map> |
| multiset | Множество, в котором каждый элемент не обязательно уникален | <set> |
| priority-queue | Очередь с приоритетом | <queue> |
| queue | Очередь | <queue> |
| set | Множество, в котором каждый элемент уникален | <set> |
| stack | Стек | <stack> |
| string | Строка символов | <string> |
| vector | Динамический массив | <vector> |

Имена типов элементов, конкретизированных с помощью ключевого слова typedef, входящих в объявление классов-шаблонов:

| Согласованное имя типа | Описание |
| --- | --- |
| size\_type | Интегральный тип, эквивалентный типу size\_t |
| reference | Ссылка на элемент |
| const\_reference | Постоянная ссылка на элемент |
| Iterator | Итератор |
| const\_iterator | Постоянный итератор |
| reverse\_iterator | Обратный итератор |
| const\_reverse\_iterator | Постоянный обратный итератор |
| value\_type | Тип хранящегося в контейнере значения |
| alIocator\_type | Тип распределителя памяти |
| key\_type | Тип ключа |
| key\_compare | Тип функции, которая сравнивает два ключа |
| value\_compare | Тип функции, которая сравнивает два значения |

# Векторы

Шаблон для класса vector:

template <class T, class Allocator = allocator <T>> class vector

Ключевое слово *Allocator* задает распределитель памяти, который по умолчанию является стандартным.

Определены следующие конструкторы:

explicit vector(const Allocator &a = Allocator());

explicit vector(size\_type число, const T &значение = T(),

const Allocator &a = Allocator());

vector(const vector<T,Allocator>&объект);

template <class InIter>vector(InIter начало, InIter конец,

const Allocator &a = Allocator());

Определены операторы сравнения:

== < <= != > >=

Определен оператор []

Функции-члены класса *vector*

| Функция-член | Описание |
| --- | --- |
| template<class lnlter>  void assign (lnlter начало, Inlter конец);  template<class Size, class T>  void assign (Size число,  const T &значение = T());  reference at(slze\_type i);  const\_reference at(size\_type i) const;  reference back();  const\_reference back() const;  iterator begin();  const\_iterator begin() const;  size\_type capacity() const;  void clear();  bool empty() const;  iterator end();  const\_iterator end() const;  iterator erase(iterator i);  iterator erase (iterator начало,  iterator конец);  reference front();  const\_reference front() const;  allocator\_type get\_allocator() const;  iterator insert(iterator i,  const T &значение = T());  void insert(iterator i, size\_type число,  const T &значение);  template<class lnlter>  void insert(iterator i,  Inlter начало, Inlter конец);  size\_type max\_size() const;  reference operator[] (size\_type i) const;  const\_reference operator[]  (size\_type ) const;  void pop\_back();  void push\_back(const T &значение);  reverse\_iterator rbegin();  const\_reverse\_iterator rbegin() const;  reverse\_iterator rend();  const\_reverse\_iterator rend() const;  void reserve(size\_type число);  void resize (size\_type число,  T значение = T();  size\_type size() const;  void swap(vector<T, Allocator> &объект); | Присваивает вектору последовательность, определенную итераторами *начало* и *конец*  Присваивает вектору *число* элементов, причем значение каждого элемента равно параметру *значение*  Возвращает ссылку на элемент, заданный параметром i  Возвращает ссылку на последний элемент вектора  Возвращает итератор первого элемента вектора  Возвращает текущую емкость вектора, т. е. то число элементов, которое можно разместить в векторе без необходимости выделения дополнительной области памяти  Удаляет все элементы вектора  Возвращает истину, если вызывающий вектор пуст, в противном случае возвращает ложь  Возвращает итератор конца вектора  Удаляет элемент, на который указывает итератор *i.* Возвращает итератор элемента, кoтoрый расположен следующим за удаленным  Удаляет элементы, заданные между итераторами *начало* и *конец.* Возвращает итератор элемента, который расположен следующим за последним удаленным  Возвращает ссылку на первый элемент вектора.  Возвращает распределитель памяти вектора  Вставляет параметр *значение* перед элементом, заданным итератором *i.* Возвращает итератор элемента  Вставляет *число* копий параметра *значение* перед элементом, заданным итератором *i*  Вставляет последовательность, определенную между итераторами *начало* и *конец,* перед элементом, заданным итератором *i*  Возвращает максимальное число элементов, которое может храниться в векторе  Возвращает ссылку на элемент, заданный параметром *i*  Удаляет последний элемент вектора  Добавляет в конец вектора элемент, значение которого равно параметру значение  Возвращает обратный итератор конца вектора  Возвращает обратный итератор начала вектора  Устанавливает емкость вектора равной, по меньшей мере, параметру *число* элементов  Изменяет размер вектора в соответствии с параметром *число.* Если при этом вектор удлиняется, то добавляемые в конец вектора элементы получают значение, заданное пара­метром *значение*  Возвращает хранящееся на данный момент в векторе число элементов  Обменивает элементы, хранящиеся в вызывающем векторе, с элементами в объекте *объект* |

# Списки

Шаблон для класса list:

template <class T, class Allocator = allocator <T>> class list

Ключевое слово *Allocator* задает распределитель памяти, который по умолчанию является стандартным.

Определены следующие конструкторы:

explicit list(const Allocator &a = Allocator());

explicit list(size\_type число, const T &значение = T(),

const Allocator &a = Allocator());

list(const list<T,Allocator>&объект);

template <class InIter>list(InIter начало, InIter конец,

const Allocator &a = Allocator());

Определены операторы сравнения:

== < <= != > >=

Функции-члены класса *list*

| Функция-член | Описание |
| --- | --- |
| template<class lnlter>  void assign(lnlter *начало,*Inlter *конец);*  template<class Size, class T>  void assign (Size *число,*  const T &*значение* = T());  reference back();  const\_reference back() const;  iterator begin();  const\_iterator begin() const;  void clear();  bool empty() const;  iterator end();  const\_iterator end() const;  iterator erase(iterator i);  iterator erase(iterator *начало,* iterator *конец);*  reference front();  const\_reference front() const;  allocator\_type get\_allocator() const;  iterator insert(iterator i,  const Т &*значение =* Т());  void insert(iterator i, size\_type *число,*  const T &*значение);*  template<class inIter>  void insert (iterator i, InIter *начало,*  InIter *конец);*  size\_type max\_size() const;  void merge(list<T, Allocator> *&объект);*  template<class Comp>  void merge (list < T, Allocator> &*объект,* Comp *ф\_сравн);*  void pop\_back();  void pop\_front();  void push\_back(const T *&значение);*  void push\_front(const T &*значение);*  reverse\_iterator rbegin();  const\_reverse\_iterator rbegin() const;  void remove(const T &*значение);*  template<class UnPred>  void remove\_if(UnPred *пред);*  reverse\_iterator rend();  const\_reverse\_iterator rend() const;  void resize(size\_type *число,*T *значение =* T());  void reversed;  size\_type size() const;  void sort();  template<class Comp>  void sort Comp *ф\_сравн);*  void splice(iterator *i*,  list<T, Allocator> *&объект);*  void splice(iterator *i,* list<T,  Allocator> *&объект,* iterator *элемент);*  void splice(iterator *i,* list<T,  Allocator> &*объект,* iterator *начало,*  iterator *конец);*  void swap(list<T, Allocator> *&объект);*  void unique();  template<class BinPred>  void unique(BinPred *пред);* | Присваивает списку последовательность, оп­ределенную итераторами *начало* и *конец*  Присваивает списку *число* элементов, причем значение каждого элемента равно параметру *значение*  Возвращает ссылку на последний элемент списка  Возвращает итератор первого элемента спи­ска  Удаляет все элементы списка  Возвращает истину, если вызывающий список пуст, в противном случае возвращает ложь  Возвращает итератор конца списка  Удаляет элемент, на который указывает ите­ратор i. Возвращает итератор элемента, кото­рый расположен следующим за удаленным  Удаляет элементы, заданные между итерато­рами *начало* и *конец.* Возвращает итератор элемента, который расположен следующим за последним удаленным  Возвращает ссылку на первый элемент списка  Возвращает распределитель памяти списка  Вставляет параметр *значение* перед элемен­том, заданным итератором *i.* Возвращает итератор элемента  Вставляет *число* копий параметра *значение* перед элементом, заданным итератором *i*  Вставляет последовательность, определенную между итераторами *начало* и *конец,* перед элементом, заданным итератором i  Возвращает максимальное число элементов, которое может храниться в списке  Выполняет слияние упорядоченного списка, хранящегося в объекте *объект,* с вызываю­щим упорядоченным списком. Результат упо­рядочивается. После слияния список, храня­щийся в объекте *объект* становится пустым. Во второй форме для определения того, явля­ется ли значение одного элемента меньшим, чем значение другого, может задаваться функция сравнения *ф\_сравн*  Удаляет последний элемент списка  Удаляет первый элемент списка  Добавляет в конец списка элемент, значение которого равно параметру *значение*  Добавляет в начало списка элемент, значение которого равно параметру *значение*  Возвращает обратный итератор конца списка  Удаляет из списка элементы, значения кото­рых равны параметру *значение*  Удаляет из списка значения, для которых ис­тинно значение унарного предиката *пред*  Возвращает обратный итератор начала списка  Изменяет размер списка в соответствии с параметром *число.* Если при этом список удлиняется, то добавляемые в конец списка элементы получают значение, заданное пара­метром *значение*  Выполняет реверс (т. е. реализует обратный порядок расположения элементов) вызываю­щего списка  Возвращает хранящееся на данный момент в списке число элементов  Сортирует список. Во второй форме для оп­ределения того, является ли значение одного элемента меньшим, чем значение другого, может задаваться функция сравнения *ф\_сравн*  Вставляет содержимое объекта *объект* в вы­зывающий список. Место вставки определяет­ся итератором *i.* После выполнения операции *объект* становится пустым  Удаляет элемент, на который указывает ите­ратор *элемент,* из списка, хранящегося в объекте *объект,* и сохраняет его в вызываю­щем списке. Место вставки определяется итератором *i*  Удаляет диапазон элементов, обозначенный итераторами *начало* и *конец,* из списка, хранящегося в объекте *объект,* и сохраняет его в вызывающем списке. Место вставки определяется итератором *i*  Обменивает элементы из вызывающего спи­ска с элементами из объекта *объект*  Удаляет из вызывающего списка парные эле­менты. Во второй форме для выяснения уни­кальности элементов используется предикат *пред* |

# Ассоциативные списки

Шаблон для класса map:

template <class key, class T , class Comp=less<Key>,

class Allocator = allocator <T>> class map

Ключевое слово *Allocator* задает распределитель памяти, который по умолчанию является стандартным. *Key* – данные типа ключ, *T* – тип данных, *Comp* – функция для сравнения двух ключей (по умолчанию стандартная объект-функция *less*() ).

Определены следующие конструкторы:

explicit map(const Comp &ф\_сравн = Comp(),

const Allocator &a = Allocator());

map(const map<Key, T, Comp, Allocator>&объект);

template <class InIter>map(InIter начало, InIter конец,

const Comp &ф\_сравн = Comp(),

const Allocator &a = Allocator());

Определены операторы сравнения:

== < <= != > >=

В ассоциативном списке хранятся пары ключ/значение в виде объектов типа *pair*.

Шаблон объекта *pair*:

template <class Ktype, class Vtype> struct pair

{

typedef Ktype первый\_тип; // тип ключа

typedef Vtype второй\_тип; // тип значения

Ktype первый; // содержит ключ

Vtype второй; // содержит значение

// конструкторы

pair();

pair(const Ktype &k, Vtype &v);

template<class A, class B> pair(const <A,B> &объект);

}

Создавать пары ключ/значение можно с помощью функции:

template<class Ktype, class Vtype> pair(Ktype, Vtype>

make\_pair()(const Ktype &k, Vtype &v);

Функции-члены класса *map*

| Функция-член | Описание |
| --- | --- |
| iterator begin();  const\_iterator begin() const;  void clear();  size.type count (const key\_type *&k*)const;  bool empty() const;  iterator end();  const.iterator end() const;  pair<iterator, iterator>  equal\_range (const key\_type pair <const\_iterator, const\_iterator>  equal\_range(const key\_type *&k*)const;  void erase(iterator i);  void erase (iterator *начало,*iterator *конец*);  size\_type erase (const key\_type *&k*);  iterator find. (const key\_type *&k*);  const\_iterator find (const key\_type *&k*)const;  allocator\_type get\_allocator() const;  iterator insert(iterator *i,*  const value\_type *&значение*);  template<class lnlter>  void insert (InIter *начало,*InIter *конец*);  pair<iterator, bool>insert  (const value\_type *&значение*);  key\_compare key\_comp() const;  iterator lower\_bound (const key\_type *&k*);  const.iterator lower\_bound  (const key\_type *&k*)const;  size\_type max\_size() const;  reference operator[] (const key\_type &i);  reverse\_iterator rbegin();  const\_reverse\_iterator rbegin() const;  reverse\_iterator rend();  const\_reverse\_iterator rend() const;  size\_type size() const;  void swap(map<Key, T, Comp,  Allocator> &*объект*);  iterator upper\_bound  (const key\_type *&k*);  const\_iterator upper\_bound  (const key\_type &k) const;  value\_compare value\_comp() const; | Возвращает итератор первого элемента ассо­циативного списка  Удаляет все элементы ассоциативного списка  Возвращает 1 или 0, в зависимости от того, встречается или нет в ассоциативном списке ключ *к*  Возвращает истину, если вызывающий ассо­циативный список пуст, в противном случае возвращает ложь  Возвращает итератор конца ассоциативного списка  Возвращает пару итераторов, которые указы­вают на первый и последний элементы ассо­циативного списка, содержащего указанный ключ *к*  Удаляет элемент, на который указывает ите­ратор *i*  Удаляет элементы, заданные между итерато­рами *начало* и *конец*  Удаляет элементы, соответствующие значе­нию ключа *к*  Возвращает итератор по заданному ключу *к.*  Если ключ не обнаружен, возвращает итера­тор конца ассоциативного списка  Возвращает распределитель памяти ассоциа­тивного списка  Вставляет параметр *значение* на место эле­мента или после элемента, заданного итера­тором *i.* Возвращает итератор этого элемента  Вставляет последовательность элементов, заданную итераторами *начало* и *конец*  Вставляет *значение* в вызывающий ассоциа­тивный список. Возвращает итератор встав­ленного элемента. Элемент вставляется толь­ко в случае, если такого в ассоциативном спи­ске еще нет. При удачной вставке элемента функция возвращает значение pair<iterator, true>, в противном случае — pair<iterator, false>  Возвращает объект-функцию сравнения клю­чей  Возвращает итератор первого элемента ассо­циативного списка, ключ которого равен или больше заданного ключа *к*  Возвращает максимальное число элементов, которое можно хранить в ассоциативном списке  Возвращает ссылку на элемент, соответст­вующий ключу *i* Если такого элемента не су­ществует, он вставляется в ассоциативный список  Возвращает обратный итератор конца ассо­циативного списка  Возвращает обратный итератор начала ассо­циативного списка  Возвращает хранящееся на данный момент в ассоциативном списке число элементов  Обменивает элементы из вызывающего ассо­циативного списка с элементами из объекта *объект*  Возвращает итератор первого элемента ассо­циативного списка, ключ которого больше заданного ключа *к*  Возвращает объект-функцию сравнения зна­чений |

# Алгоритмы

**Алгоритмы библиотеки стандартных шаблонов**

| Алгоритм | Назначение | |
| --- | --- | --- |
| adjacent\_find  binary\_search  copy  copy\_backward  count  count\_if  equal  equal\_range  fill  find  find\_end  find\_first\_of  find\_if  for\_each  generate  generate\_n  includes  inplace\_merge  iter\_swap  lexicographical\_compare  lower\_bound  make\_heap  max  max\_element  merge    min  min\_element  mismatch  next\_permutation  nth\_element  partial\_sort  partial\_sort\_copy  partition  pop\_heap  prev\_permutation  push\_heap  random\_shuffle  remove  remove\_if  remove\_copy  remove\_copy\_if  replace  replace jf  replace\_copy  replace\_copy\_if  reverse  reverse\_copy  rotate  rotate\_copy  search  search\_n  set\_difference  set\_intersection  set\_symmetric\_difference  set\_union  sort  sort\_heap  stable\_partition  stable\_sort  swap  swap\_ranges  transform  unique unique\_copy  upper\_bound | | Выполняет поиск смежных парных элементов в последовательности. Возвращает итератор первой пары  Выполняет бинарный поиск в упорядоченной после­довательности  Копирует последовательность  Аналогична функции сору(), за исключением того, что перемещает в начало последовательности эле­менты из ее конца  Возвращает число элементов в последовательности  Возвращает число элементов в последовательности, удовлетворяющих некоторому предикату  Определяет идентичность двух диапазонов  Возвращает диапазон, в который можно вставить элемент, не нарушив при этом порядок следования элементов в последовательности  Заполняет диапазон заданным значением  Выполняет поиск диапазона для значения и возвра­щает первый найденный элемент  Выполняет поиск диапазона для подпоследователь­ности. Функция возвращает итератор конца подпос­ледовательности внутри диапазона  Находит первый элемент внутри последовательно­сти, парный элементу внутри диапазона  Выполняет поиск диапазона для элемента, для кото­рого определенный пользователем унарный преди­кат возвращает истину  Назначает функцию диапазону элементов  Присваивает элементам в диапазоне значения, воз­ вращаемые порождающей функцией  Определяет, включает ли одна последовательность все элементы другой последовательности  Выполняет слияние одного диапазона с другим. Оба диапазона должны быть отсортированы в порядке возрастания элементов. Результирующая последо­вательность сортируется  Меняет местами значения, на которые указывают два итератора, являющиеся аргументами функции  Сравнивает две последовательности в алфавитном порядке  Обнаруживает первое значение в последовательно­сти, которое не меньше заданного значения  Выполняет пирамидальную сортировку последова­тельности (пирамида, на английском языке heap, — полное двоичное дерево, обладающее тем свойст­вом, что значение каждого узла не меньше значения любого из его дочерних узлов.  Возвращает максимальное из двух значений  Возвращает итератор максимального элемента внутри диапазона  Выполняет слияние двух упорядоченных последова­тельностей, а результат размещает в третьей после­довательности  Возвращает минимальное из двух значений  Возвращает итератор минимального элемента внут­ри диапазона  Обнаруживает первое несовпадение между элемен­тами в двух последовательностях. Возвращает ите­раторы обоих несовпадающих элементов  Образует следующую перестановку (permutation) последовательности  Упорядочивает последовательность таким образом, чтобы все элементы, меньшие заданного элемента Б, располагались перед ним, а все элементы, боль­шие заданного элемента Е, — после него  Сортирует диапазон  Сортирует диапазон, а затем копирует столько эле­ментов, сколько войдет в результирующую последо­вательность  Упорядочивает последовательность таким образом, чтобы все элементы, для которых предикат возвра­щает истину, располагались перед элементами, для которых предикат возвращает ложь  Меняет местами первый и предыдущий перед по­следним элементы, а затем восстанавливает пирамиду  Образует предыдущую перестановку последова­тельности  Размещает элемент на конце пирамиды  Беспорядочно перемешивает последовательность  Удаляет элементы из заданного диапазона  Заменяет элементы внутри диапазона  Меняет порядок сортировки элементов диапазона на обратный  Выполняет циклический сдвиг влево элементов в диапазоне  Выполняет поиск подпоследовательности внутри последовательности  Выполняет поиск последовательности заданного числа одинаковых элементов  Создает последовательность, которая содержит раз­личающиеся участки двух упорядоченных наборов  Создает последовательность, которая содержит одинаковые участки двух упорядоченных наборов  Создает последовательность, которая содержит симметричные различающиеся участки двух упоря­доченных наборов  Создает последовательность, которая содержит объединение (union) двух упорядоченных наборов  Сортирует диапазон  Сортирует пирамиду внутри диапазона  Упорядочивает последовательность таким образом, чтобы все элементы, для которых предикат возвра­щает истину, располагались перед элементами, для которых предикат возвращает ложь. Разбиение на разделы остается постоянным; относительный поря­док расположения элементов последовательности не меняется  Сортирует диапазон. Одинаковые элементы не пе­реставляются  Меняет местами два значения  Меняет местами элементы в диапазоне  Назначает функцию диапазону элементов и сохраня­ет результат в новой последовательности  Удаляет повторяющиеся элементы из диапазона  Обнаруживает последнее значение в последова­тельности, которое не больше некоторого значения |

# Алгоритм вычисления выражений через обратную польскую запись

Обычные алгебраические (логические, математические) выражения можно записывать в виде обратной польской нотации – записи без скобок.

Нотация – «польская» в честь польского математика Яна Лукашевича, который её предложил. Нотация – «обратная» из-за того, что порядок операндов и операций обратный относительно исходного.

Пример:

Выражение Обратная польская нотация

А + (В - С) \* D – F / (G + H) = 🡪 A B C – D \* + F G H + / - =

Если A = 6 B = 4 C = 1 D = 2 F = 3 G = 7 H = 5.

Тогда обратная польская нотация: 6 4 1 – 2 \* + 3 7 5 + / - =

Алгоритм вычисления выражения.

Вычислить обратное польское выражение проще, чем обычное алгебраическое.

Просматриваем все элементы слева направо. Как только встречается операция, выполняем ее по отношению к двум предыдущим элементам, заменяя два элемента одним.

6 4 1 – 2 \* + 3 7 5 + / - = выполняем 4-1=3 🡪

6 3 2 \* + 3 7 5 + / - = выполняем 3\*2=6 🡪

6 6 + 3 7 5 + / - = выполняем 6+6=12 🡪

12 3 7 5 + / - = выполняем 7+5=12 🡪

12 3 12 / - = выполняем 3/12=0.25 🡪

12 0.25 - = выполняем 12-0.25=11.75 🡪

11.75 = результат

Алгоритм преобразования выражения в обратную польскую нотацию.

Необходимо использовать два стека (списка) Х и Y.

Необходимо определить приоритеты операций. Например, так:

операции приоритет

\* / 3

+ - 2

( 1

= 0

1. Просматриваем выражение слева направо. Операнды помещаем в стек X , левые скобки и операции в стек Y.
2. Встретив правую скобку, отыскиваем в стеке соответствующую ей левую. При этом все, что сверху – выталкивается из стека Y и заносится в стек X.
3. Если приоритет очередной операции меньше приоритета операции вершины стека , то операции из вершины стека Y выталкиваются в стек Х, пока не найдём операцию с приоритетом ниже, либо пока стек не окажется пустым .
4. Результат оказывается с стеке X.

Пример преобразования выражения в обратную польскую запись

А + (В - С) \* D – F / (G + H) =

стек X стек Y

1. A

A

2.+

A +

3. (

A ( 🡪 +

4.B

B 🡪 A ( 🡪 +

5. -

B 🡪 A - 🡪 ( 🡪 +

6. C)

C 🡪 B 🡪 A - 🡪 ( 🡪 +

7. )

- 🡪 C 🡪 B 🡪 A +

8. \*

- 🡪 C 🡪 B 🡪 A \* 🡪 +

9. D

D 🡪 - 🡪 C 🡪 B 🡪 A \* 🡪 +

10. –

+ 🡪 \* 🡪 D 🡪 - 🡪 C 🡪 B 🡪 A -

11. F

F 🡪 + 🡪 \* 🡪 D 🡪 - 🡪 C 🡪 B 🡪 A -

12. /

F 🡪 + 🡪 \* 🡪 D 🡪 - 🡪 C 🡪 B 🡪 A / 🡪 -

13. (

F 🡪 + 🡪 \* 🡪 D 🡪 - 🡪 C 🡪 B 🡪 A (🡪 /🡪 -

14. G

G 🡪 F 🡪 + 🡪 \* 🡪 D 🡪 - 🡪 C 🡪 B 🡪 A ( 🡪 / 🡪 -

15. +

G 🡪 F 🡪 + 🡪 \* 🡪 D 🡪 - 🡪 C 🡪 B 🡪 A + 🡪 ( 🡪 / 🡪 -

16. H

H 🡪 G 🡪 F 🡪 + 🡪 \* 🡪 D 🡪 - 🡪 C 🡪 B 🡪 A +🡪 ( 🡪 /🡪 -

17. )

+ 🡪 H 🡪 G 🡪 F 🡪 +🡪 \*🡪 D 🡪 - 🡪 C 🡪 B 🡪 A / 🡪 -

18. =

- 🡪 / 🡪 + 🡪 H 🡪 G 🡪 F 🡪 + 🡪 \*🡪 D🡪 - 🡪 C🡪 B🡪 A

результат в обратном порядке

A B C – D \* + F G H + / - =

## ПРИМЕР 1. Объявление и инициализация строки string

#include <iostream>

#include <string> // строковый класс

using namespace std;

void main()

{

string s1; //создаем пустую строку

string s2("aaaa"); // создаем строку из C-строки

// создаем строку из C-строки 10 символов

string s3("abcdefghijklmnopqrstuvwxyz",10);

// создаем строку из 5 одинаковых символов

string s4(5,’!’);

// создаем строку-копию из строки s3

string s5(s3);

// создаем строку-копию из строки s3

// начиная с индекса 5 не более 3 символов

string s6(s3,5,3);

cout<<"s1= "<<s1<<endl;

cout<<"s2= "<<s2<<endl;

cout<<"s3= "<<s3<<endl;

cout<<"s4= "<<s4<<endl;

cout<<"s5= "<<s5<<endl;

cout<<"s6= "<<s6<<endl;

}

## ПРИМЕР 2. Инициализация строки string. Оператор =. Метод assign

#include <iostream>

#include <string> // строковый класс

using namespace std;

void main()

{

string s1, s2, s3;

// в классе string определены три оператора присваивания:

// string & operator = (const string& str);

// string & operator = (const char\* s);

// string & operator = (char c);

s1 = '1';

s2 = "bbbbbb";

s3 = s2;

cout<<"s1= "<<s1<<endl;

cout<<"s2= "<<s2<<endl;

cout<<"s3= "<<s3<<endl;

// метод assign

s2.assign("ccccccc"); // s2=”cccccccc”

s3.assign(s2); // s3=s2

cout<<"s1 = "<<s1<<endl;

cout<<"s2 = "<<s2<<endl;

cout<<"s3 = "<<s3<<endl;

s2.assign("1234"); // s2=”1234”

// в s3 из s2 3 символа, начиная с 1 позиции

s3.assign(s2,1,3);

cout<<"s2= "<<s2<<endl;

cout<<"s3= "<<s3<<endl;

char s[]="56789";

// присваивает s3 3 символа С-строки

s3.assign(s,3);

cout<<"s= "<<s<<endl;

cout<<"s3= "<<s3<<endl;

}

## ПРИМЕР 3. **Ввод строки string. Оператор >>**

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1;

cout << " Enter a string: ";

//ввод выполняется до первого пробельного символа

cin >> s1;

cout << "You entered: " << s1 << endl;

}

## ПРИМЕР 4. Ввод строки string. Метод getline

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1;

cout << "Enter a string: ";

getline(cin,s1); //ввод строки

cout << "You entered: " << s1 << endl;

cin.get(); // удаление из потока символа ‘\n’.

cout << " Enter a string: ";

// свой разделитель для ввода строки

getline(cin,s1,'&');

cout << "You entered: " << s1 << endl;

cin.get(); // удаление из потока символа ‘&’.

}

## ПРИМЕР 5. Длина строки string. М**етоды length, size**

#include <iostream>

#include <string>

using namespace std;

void main()

{

string st("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

cout << " " << st.length()<< " " << st.size() <<

" "<<st.max\_size()<<endl;

st = "Good Morning";

cout << " " << st.length()<< " " << st.size() <<

" "<<st.max\_size()<<endl;

st = "Hello";

cout << " " << st.length()<< " " << st.size() <<

" "<<st.max\_size()<<endl;

}

## ПРИМЕР 6. **Доступ к элементу строки string. Оператор []. Метод at**

#include <iostream>

#include <string>

using namespace std;

void main()

{

string st("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

int i;

for (i = 0; i < st.length(); i++)

cout << st[i];

// если i выходит за пределы строки,

// то поведение не определено

st = "Good Morning";

for (i = 0; i < st.length(); i++)

cout << st.at(i);

// если i выходит за пределы строки,

// метод возвращает исключение типа out\_of\_rang

st = "Hello";

for (i = 0; i < st.length(); i++)

cout << st[i];

}

## ПРИМЕР 7. Сравнение строк. Операторы сравнения

Вводим строки в цикле, выход – ввод “пустой” сроки.

Вывод на экран результатов сравнения двух строк.

#include <iostream>

#include <string>

using namespace std;

void main()

{

string st1, st2;

cout << "Enter a string \n";

getline(cin,st1); // ввод 1 строки

while (true)

{

cout << "Enter a string \n";

getline(cin,st2); // ввод 2 строки

cin.get();

if ( st2 == "")

break; // выход из цикла

cout << endl<< st2 ;

// операторы лексикографического сравнения строк

if (st2 == st1)

cout << " = ";

else

if (st2 < st1)

cout <<" < ";

else

cout << " > ";

cout << st1<<endl ;

st1 = st2;

}

}

## ПРИМЕР 8. Сравнение строк. Метод compare

Вводим строки в цикле, выход – ввод “пустой” сроки.

Вывод на экран результатов сравнения двух строк.

#include <iostream>

#include <string>

using namespace std;

void main()

{ string st1, st2;

cout << "Enter a string \n";

getline(cin,st1); // ввод 1 строки

while (true)

{

cout << "Enter a string \n";

cin.get();

getline(cin,st2); // ввод 2 строки

if ( !st2.compare("")) break; // выход из цикла

cout << endl<< st2 ;

// лексикографическое сравнение строк

if (!st2.compare(st1)) cout <<" = ";

else

if (st2.compare(st1)<0) cout <<" < ";

else cout << " > ";

cout << st1<<endl ;

cout << endl <<st2[1]<<st2[2];

// лексикографическое сравнение подстрок

if (!st2.compare(1,2,st1,2,3))

cout <<" = ";

else

if (st2.compare(1,2,st1,2,3)<0)

cout <<" < ";

else

cout << " > ";

cout << st1[2]<<st1[3]<<st1[4]<<endl ;

st1 = st2;

}

}

## ПРИМЕР 9. Объединение строк. Оператор +. Метод append

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1("11");;

string s2("2222");

string s3 ("333333");

string s4("44444444");

s1 += s2; // добавить s2 к s1

cout<<"s1 = s1 + s2 = "<<s1<<endl;

s4 = s1 + s2; // добавить s1 к s2

cout<<"s4 = s1 + s2 = "<<s4<<endl;

s4 = s4 + '!'; // добавить s1 к s2

cout<<"s4 = s4 + ! = "<<s4<<endl;

s2.append(s1); // добавить s1 к s2

cout<<"s2 + s1 = "<< s2<<endl;

// добавить к s3 2 символа строки s1 со 1 позиции

s3.append(s4,1,2);

cout<<"s3 + s4 = "<< s3<<endl;

char s[] = "56789";

// добавить к s3 2 символа С-строки s

s3.append(s,2);

cout<<"s3 + s = "<< s3<<endl;

}

## ПРИМЕР 10. Вставка строки (подстроки) в строку. Метод insert

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1("1111111");;

string s2("23456789");

s1.insert(3,s2); // вставить s2 в s1 с 3 позиции

cout<<"s1="<< s1<<endl;

s1 = "1111111";

s2 = "23456789";

// вставить 4 символа s2 со 2 позиции в s1 с 3 позиции

s1.insert(3,s2,2,4);

cout<<"s1="<< s1<<endl;

s1 = "1111111";

char s[] = "23456789";

// вставить 4 символа s в s1 с 3 позиции

s1.insert(3,s,4);

cout<<"s1 = "<< s1<<endl;

}

## ПРИМЕР 11. Замена строки (подстроки) в строке. Метод replace

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1("1111111");;

string s2("23456789");

// замена 2 символов в s1 с 3 позиции элементами s2

s1.replace(3,2,s2);

cout<<"s1="<< s1<<endl;

s1 = "1111111";

s2 = "23456789";

// замена 2 символов в s1 с 3 позиции 1 символом

// из 4 позиции строки s2

s1.replace(3,2,s2,4,1);

cout<<"s1="<< s1<<endl;

s1 = "1111111";

char s[] = "23456789";

// замена 2 символов в s1 с 3 позиции 4 символами s

s1.replace(3,2,s,4);

cout<<"s1="<< s1<<endl;

}

## ПРИМЕР 12. Удаление подстроки в строке. Метод erase

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1("123456789");

// удаление 2 символов в s1 с 3 позиции

s1.erase(3,2);

cout<<"s1="<< s1<<endl;

// удаление всех символов в s1 с 3 позиции

s1 = "123456789";

s1.erase(3);

cout<<"s1="<< s1<<endl;

s1 = "123456789";

s1.erase(); // удаление всех символов s1

cout<<"s1="<< s1<<endl;

}

## ПРИМЕР 13. Выделение подстроки в строке. Метод substr

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1("123456789");

string s2;

// s2 – подстрока s1 из 2 символов с 3 позиции

s2 = s1.substr(3,2);

cout<<"s2="<< s2<<endl;

s1 = "123456789";

// s2 – подстрока s1 всех символов с 3 позиции

s2 = s1.substr(3);

cout<<"s2="<< s2<<endl;

s1 = "123456789";

s2 = s1.substr(); // s2=s1

cout<<"s2="<< s2<<endl;

}

## ПРИМЕР 14. Обмен содержимого строк. Метод swap, reverse

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1("123456789");

string s2("abcdef");

s1.swap(s2);

cout<<"s1="<< s1<<endl;

cout<<"s2="<< s2<<endl;

string s3("12345678");

reverse(s3.begin(),s3.end());

cout<< s3 <<endl;

}

## ПРИМЕР 15. Поиск подстроки в строке. Метод find

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1("123123123");

string s2("12");

unsigned k;

// поиск подстроки s2 в строке s1 с 4 позиции

k = s1.find(s2,4);

cout<<" "<< k<<endl;

// поиск подстроки s2 в строке s1 с 7 позиции

k = s1.find(s2,7);

cout<<" "<< k<<endl;

// поиск символа '1' в строке s1 с 4 позиции

k = s1.find('1',4);

cout<<" "<< k<<endl;

// поиск символа '1' в строке s1 с 4 позиции

k = s1.rfind('1',4);

cout<<" "<< k<<endl;

}

## ПРИМЕР 16. Поиск символа подстроки в строке. Метод find\_first\_of

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1("1234561728");

string s2("12");

char s3[100]="124";

unsigned k;

// поиск первого любого символа из подстроки s2 в строке s1

// с 4 позиции

k = s1.find\_first\_of(s2,4);

cout<<" "<< k<<endl; // 6

// поиск первого любого символа из count первых

// символов подстроки s3 в строке s1 с 4 позиции

k = s1. find\_first\_of (s3,4,2);

cout<<" "<< k<<endl; // 6

// поиск первого любого символа из подстроки s3 в

// строке s1 с 4 позиции

k = s1. find\_first\_of (s3,4);

cout<<" "<< k<<endl; // 6

// поиск символа '1' в строке s1 с 4 позиции

k = s1. find\_first\_of ('1',4);

cout<<" "<< k<<endl; // 6

}

## ПРИМЕР 17. Поиск символа подстроки в строке. Метод find\_first\_not\_of

#include <iostream>

#include <string>

using namespace std;

void main()

{

string s1("1234561728");

string s2("12");

char s3[100]="124";

unsigned k;

// поиск первого символа отличного от любого символа

// из подстроки s2 в строке s1 с 6 позиции

k = s1.find\_first\_not\_of(s2,6);

cout<<" "<< k<<endl; // 7

// поиск первого символа отличного от любого символа

// из count первых символов из подстроки s3

// в строке s1 с 6 позиции

k = s1. find\_first\_not\_of (s3,6,2);

cout<<" "<< k<<endl; // 7

// поиск первого любого символа отличного от любого символа // из подстроки s3 в строке s1 с 6 позиции

k = s1. find\_first\_not\_of (s3,6);

cout<<" "<< k<<endl; // 7

// поиск первого символа отличного от символа '1'

// в строке s1 с 6 позиции

k = s1. find\_first\_not\_of ('1',6);

cout<<" "<< k<<endl; // 7

}

## ПРИМЕР 18. Выделение лексем. Методы find\_first\_not\_of, find\_first\_of

#include <iostream>

#include <string>

using namespace std;

#define OUT

void lexem(string str, string delim, OUT string& res)

{

unsigned int wordBegin = 0, wordEnd = 0;

// позиция начала лексемы

wordBegin = str.find\_first\_not\_of(delim, wordEnd);

// позиция конца лексемы

wordEnd = str.find\_first\_of(delim, wordBegin);

if (wordEnd >= str.length())

wordEnd = str.length();

while (wordBegin < str.length())

{ // выделение лексемы

string word = str.substr(wordBegin, wordEnd - wordBegin);

if (res.length())

res += " ";

res += word; // результирующая строка

// позиция начала лексемы

wordBegin = str.find\_first\_not\_of(delim, wordEnd);

// позиция конца лексемы

wordEnd = str.find\_first\_of(delim, wordBegin);

if (wordEnd >= str.length())

wordEnd = str.length();

}

}

void main()

{

string delim(" .,;!?-:"); // строка разделителей

string s1, s2;

cout << "Enter a string \n";

getline(cin,s1); // ввод строки

lexem(s1, delim, OUT s2);

cout << s2 << endl; // вывод результатов

}

## ПРИМЕР 19. Выделение лексем. Чтение из потока

#include <iostream>

#include <string>

using namespace std;

#define OUT

void lexem(string str, string delim, OUT string& res)

{

// заменяем все символы в строке str из delim на пробелы

for (int i = 0; i < str.length(); i++)

{

char c = str[i];

if (delim.find(str[i]) < delim.length())

str[i] = ' ';

}

stringstream stream(str);

string word;

while (stream >> word) // читаем из потока следующую лексему

{

if (res.length())

res += ' ';

res += word;

}

}

void main()

{

string delim(" .,;!?-:"); // строка разделителей

string s1, s2;

cout << "Enter a string \n";

getline(cin,s1); // ввод строки

lexem(s1, delim, OUT s2);

cout << s2 << endl; // вывод результатов

}

## ПРИМЕР 20. Строки С и С++. Методы copy, c\_str

#include <iostream>

#include <string>

using namespace std;

void main()

{

string str("1234567890");

char s[80];

int k;

// копируем 3 символа str c 5 позиции в s

k = str.copy(s,3,5);

cout<<s<<" "<<k<<endl;

// копируем 3 символа str c 5 позиции в s

k = str.copy(s,3,5);

s[3]='\0';

cout<<s<<" "<<k<<endl;

cout<<str.c\_str()<<endl;

}

## ПРИМЕР 21. Класс-контейнер vector. Основные операции

Основные операции над вектором.

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> v; // создание вектора нулевой длины

// вывод на экран размера исходного вектора v

cout << "Размер = " << v.size() << endl;

// помещение значений в конец вектора,

// по мере необходимости вектор будет расти

for (int i = 0; i < 10; i++)

v.push\_back(i);

// вывод на экран текущего размера вектора v

cout << "Новый размер = " << v.size() << endl;

// вывод на экран содержимого вектора v

// доступ к содержимому вектора

// с использованием оператора индекса

cout << "Текущее содержимое:\n";

for (unsigned int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

// помещение новых значений в конец вектора,

// и опять по мере необходимости вектор будет расти

for (int i = 0; i < 10; i++)

v.push\_back(i + 10);

// вывод на экран текущего размера вектора

cout << "Новый размер = " << v.size() << endl;

// вывод на экран содержимого вектора

cout << "Текущее содержимое:\n";

for (unsigned int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

// изменение содержимого вектора

for (unsigned int i = 0; i < v.size(); i++)

v[i] = v[i] + v[i];

// вывод на экран содержимого вектора

cout << "Удвоенное содержимое:\n";

for (unsigned int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

// доступ к вектору через итератор

vector<int>::iterator p = v.begin();

while (p != v.end))

{

cout << \*p << " ";

++p; // префиксный инкремент итератора быстрее

// постфиксного

}

return 0;

}

## ПРИМЕР 22. Класс-контейнер vector. Методы insert и erase

Демонстрация функций вставки и удаления элементов.

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> v(5, 1); // создание пятиэлементного вектора

// из единиц

// вывод на экран исходных размера и содержимого вектора

cout << "Размер = " << v.size() << endl;

cout << "Исходное содержимое:\n";

for (int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

vector<int>::iterator p = v.begin();

p += 2; // p указывает на третий элемент

// вставка в вектор на то место,

// куда указывает итератор p десяти новых элементов,

// каждый из которых равен 9

v.insert(p, 10, 9);

// вывод на экран размера

// и содержимого вектора после вставки

cout << "Размер после вставки = " << v.size() << endl;

cout << "Содержимое после вставки:\n";

for (unsigned int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

// удаление вставленных элементов

p = v.begin();

p += 2; // указывает на третий элемент

v.erase(p, p + 10);

// удаление следующих десяти элементов

// за элементом, на который указывает итератор p

// вывод на экран размера

// и содержимого вектора после удаления

cout << "Размер после удаления = " << v.size() << endl;

cout << "Содержимое после удаления:\n";

for (unsigned int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

return 0;

}

## ПРИМЕР 23. Класс-контейнер vector. Хранение объектов пользовательского класса

Хранение в векторе объектов пользовательского класса.

#include <iostream>

#include <vector>

using namespace std;

class Demo // пользовательский класс

{

double d;

public:

Demo()

{

d = 0.0;

}

Demo(double x)

{

d = x;

}

Demo& operator=(double x)

{

d = x;

return \*this;

}

double getd() const

{

return d;

}

};

// операция <

bool operator<(const Demo& a, const Demo& b)

{

return a.getd() < b.getd();

}

// операция =

bool operator==(const Demo& a, const Demo& b)

{

return a.getd() == b.getd();

}

int main()

{

vector<Demo> v;

for (int i = 0; i < 10; i++)

v.push\_back(Demo(i/3.0));

// добавить элемент в конец вектора

for (unsigned int i = 0; i < v.size(); i++)

cout << v[i].getd() << " ";

cout << endl;

for (unsigned int i = 0; i < v.size(); i++)

v[i] = v[i].getd() \* 2.1;

for (unsigned int i = 0; i < v.size(); i++)

cout << v[i].getd() << " ";

return 0;

}

## ПРИМЕР 24. Класс-контейнер vector. Матрица (вектор векторов)

Создание матрицы как вектора векторов.

#include <iostream>

#include <vector>

using namespace std;

// функция вывода матрицы на экран

void vyvod(vector<vector<int>> vec)

{

for (int i = 0; i < vec.size(); i++)// Цикл по строкам

{ // Цикл по элементам в строке

for (int j = 0; j < vec[i].size(); j++)

cout << vec[i][j] << ' ';

// Вывод элементов i строки вектора

cout << endl;

}

cout << endl;

}

// функция инициализации матрицы случайными числами

void vvod(vector<vector<int>> &vec, int m)

{

for (int i = 0; i < vec.size(); i++)// Цикл по строкам

{ // Цикл по элементам в строке

for (int j = 0; j < vec[i].size(); j++)

vec[i][j] = rand() % m;

}

}

// функция инициализации вектора случайными числами

void vvod\_vec(vector<int> &vec, int m)

{

for (int j = 0; j < vec.size(); j++)

vec[j] = rand() % m;

}

// --------------------------------------------------

int main()

{

int n, m; // переменные, отвечающие за размер матрицы

// --------------------------- 1 --------------------

cin >> n >> m; // Ввод размеров матрицы

vector<vector<int>> vec1; // нет памяти для матрицы

for (int i = 0; i < n; i++)

{

vector <int> temp; // вектор-строка (памяти нет)

for (int j = 0; j < m; j++) // добавляем элементы

temp.push\_back(rand() % m);// в строку

vec1.push\_back(temp); // добавляем строку в матрицу

}

vyvod(vec1); // вывод матрицы

// --------------------------- 2 ---------------------

cin >> n >> m; // Ввод размеров матрицы

vector<vector<int>> vec2; // нет памяти для матрицы

for (int i = 0; i < n; i++)

{

vector<int> temp(m); // одномерный вектор-строка

// из m элементов

vvod\_vec(temp, m); // инициализация

vec2.push\_back(temp); // добавляем строку в матрицу

}

vyvod(vec2); // вывод матрицы

// --------------------------- 3 -------------------

cin >> n >> m; // Ввод размеров матрицы

vector<vector<int>> vec3(n, vector <int>(m, 0));

// Объявление матрицы на n строк по m элементов

// заполнение 0

vyvod(vec3); // вывод матрицы

vvod(vec3, m); // инициализация

vyvod(vec3); // вывод матрицы

// --------------------------- 4 ----------------

cin >> n >> m; // Ввод размеров матрицы

vector<vector<int>> vec4(n, vector <int>(m));

// Объявление матрицы на n строк по m элементов

vvod(vec4,m); // инициализация

vyvod(vec4); // вывод матрицы

// --------------------------- 5 ---------------

cin >> n >> m; // Ввод размеров векторов

vector<vector<int>> vec5(n);

// Объявление матрицы на n строк

for (size\_t i = 0; i < n; ++i)

{ // для каждой строки выделяем память на m элементов

vec5[i] = vector<int>(m);

vvod\_vec(vec5[i],m); // инициализация

}

vyvod(vec5); // вывод матрицы

// --------------------------- 6 ---------------

cin >> n >> m; // Ввод размеров векторов

vector < vector <int> > vec6(n);

// Объявление матрицы на n строк

for (size\_t i = 0; i < n; ++i)

{ // для каждой строки выделяем память на m элементов

vec6[i].resize(m);

vvod\_vec(vec6[i],m); // инициализация

}

vyvod(vec6); // вывод матрицы

// --------------------------- 7 ----------------

// вывод элементов массива через итераторы

cin >> n >> m; // Ввод размеров векторов

vector<vector<int>> vec7(n, vector<int>(m, 1));

// Объявление матрицы на n строк по m элементов

// заполнение 1

for (vector<vector<int>>::iterator it1 = vec7.begin();

it1 != vec7.end(); ++it1)

{

for (vector<int>::iterator it2 = (\*it1).begin();

it2 != (\*it1).end(); ++it2)

cout << \*(it2) << ' ';//Вывод элементов i строки

cout << endl;

}

return 0;

}

## ПРИМЕР 25. Класс-контейнер list. Создание, определение числа элементов, просмотр с удалением

Основные операции списка: создание, определение числа элементов, просмотр элементов с удалением.

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<char> lst; // создание пустого списка

for (int i = 0; i < 10; i++)

lst.push\_back('A' + i); // добавить в конец списка

// число элементов в списке

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

cout << "Содержимое: ";

while (!lst.empty()) // пока список не пуст

{

list<char>::iterator p;

p = lst.begin(); // итератор первого элемента списка

cout << \*p;

lst.pop\_front(); // удаление первого элемента списка

}

return 0;

}

## ПРИМЕР 26. Класс-контейнер list. Просмотр элементов списка в прямом и обратном порядке

Просмотр элементов списка в прямом и обратном порядке без удаления.

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<char> lst;

for (int i = 0; i < 10; i++)

lst.push\_back('A' + i); // добавить в конец списка

// число элементов в списке

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

// просмотр элементов списка

cout << "Содержимое: ";

// итератор первого элемента списка

list<char>::iterator p = lst.begin();

cout << "---->: ";

while(p != lst.end()) // итератор конца списка

{

cout << \*p;

++p;

}

cout << endl;

// просмотр элементов списка в обратном порядке

// итератор последнего элемента списка

list<char>::reverse\_iterator rp = lst.rbegin();

cout << "<----: ";

while(rp != lst.rend()) // итератор конца списка

{

cout << \*rp;

++rp; // это строка не меняется,

// обратный итератор задаёт направление

}

cout << endl;

return 0;

}

## ПРИМЕР 27. Класс-контейнер list. Добавление элементов в конец и начало списка

Элементы можно размещать не только начиная с начала списка, но также и начиная с его конца. Создается два списка, причем во втором порядок организации элементов обратный первому.

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<char> lst;

for (int i = 0; i < 10; i++)

lst.push\_back('A' + i); // добавить в конец списка

cout << "Размер прямого списка =" << lst.size()<< endl;

cout << "Содержимое прямого списка: ";

// Удаление элементов из первого списка

// и размещение их в обратном порядке во втором списке

list<char> revlst;

while (!lst.empty())

{ // получить первый элемент в списке

char element = lst.front();

cout << element;

lst.pop\_front(); // удаление первого элемента списка

revlst.push\_front(element); //добавить в начало списка

}

cout << endl;

cout << "Размер обратного списка = ";

cout << revlst.size() << endl;

cout << "Содержимое обратного списка: ";

for (list<char>::iterator p = revlst.begin();

p != revlst.end(); ++p)

cout << \*p;

return 0;

}

## ПРИМЕР 28. Класс-контейнер list. Сортировка элементов списка

Сортировка списка.

#include <iostream>

#include <list>

#include <cstdlib>

using namespace std;

int main()

{

list<char> lst;

// заполнение списка случайными символами

for (int i = 0; i < 10; i++)

lst.push\_back('A' + (rand()%26));

cout << "Исходное содержимое: ";

list<char>::iterator p = lst.begin();

while (p != lst.end())

{

cout << \*p;

++p;

}

cout << endl;

// сортировка списка

lst.sort();

cout << "Отсортированное содержимое: ";

p = lst.begin();

while (p != lst.end())

{

cout << \*p;

++p;

}

return 0;

}

## ПРИМЕР 29. Класс-контейнер list. Слияние списков

Слияние двух списков.

#include <iostream>

#include <list>

using namespace std;

int main()

{ list<char> lst1, lst2;

int n = 0, m = 0;

cin>> n >> m;

for (int i = 0; i < n; i += 2)

lst1.push\_back('A' + i);

for (int i = 1; i < m; i += 2)

lst2.push\_back('A' + i);

cout << "Содержимое первого списка: ";

list<char>::iterator p = lst1.begin();

while (p != lst1.end())

{ cout << \*p; ++p;

}

cout << endl;

cout << "Содержимое второго списка: ";

p = lst2.begin();

while (p != lst2.end())

{ cout << \*p; ++p;

}

cout << endl;

// Слияние двух списков

lst1.merge(lst2);

if (lst2.empty())

cout << "Теперь второй список пуст\n";

cout << "Содержимое первого списка после слияния:\n";

p = lst1.begin();

while (p != lst1.end())

{

cout << \*p;

++p;

}

return 0;

}

## ПРИМЕР 30. Класс-контейнер list. Использование пользовательского класса

Использование в списке объектов пользовательского класса.

#include <iostream>

#include <list>

#include <cstring>

using namespace std;

const int MAX = 40;

class Project // пользовательский класс

{

char name[MAX];

int days\_to\_completion;

public:

Project()

{

strcpy\_s(name, MAX, " ");

days\_to\_completion = 0;

}

Project(const char\* n, int d)

{

strcpy\_s(name, MAX, n);

days\_to\_completion = d;

}

void add\_days(int i)

{

days\_to\_completion += i;

}

void sub\_days(int i)

{

days\_to\_completion -= i;

}

bool completed() const

{

return !days\_to\_completion;

}

void report() const

{

cout << name << ": ";

cout << days\_to\_completion;

cout << " day to finish" << endl;

}

};

int main()

{

list<Project> proj;

proj.push\_back(Project("compile", 35));

proj.push\_back(Project("exel", 190));

proj.push\_back(Project("STL", 1000));

// вывод проектов на экран

for (list<Project>::iterator p = proj.begin();

p != proj.end(); ++p)

p->report();

// увеличение сроков выполнения первого проекта

list<Project>::iterator p = proj.begin();

p->add\_days(10);

// последовательное завершение первого проекта

do

{

p->sub\_days(5);

p->report();

} while (!p->completed());

return 0;

}

## ПРИМЕР 31. Класс-контейнер map. Создание, поиск

Иллюстрация возможностей ассоциативного списка.

#include <iostream>

#include <map>

using namespace std;

int main()

{

map<char, int> m;

// размещение пар в ассоциативном списке

for (int i = 0; i < 10; i++)

m.insert(pair<char, int>('A' + i, i));

char ch = 0;

cout << "Введите ключ: ";

cin >> ch;

map<char, int>::iterator p;

// поиск значения по заданному ключу

p = m.find(ch);

if (p != m.end())

cout << p -> second;

else

cout << "Такого ключа в ассоциативном списке нет\n";

return 0;

}

## ПРИМЕР 32. Класс-контейнер map. Алгоритм find

Ассоциативный список слов и антонимов.

#include <iostream>

#include <map>

#include <cstring>

using namespace std;

class word // пользовательский класс слов word

{

char str[20];

public:

word() { strcpy(str, ""); }

word(char \*s) { strcpy(str, s); }

char \*get() { return str; }

};

// для объектов типа word определим оператор < (меньше),

// чтобы его можно было использовать как ключ

// в контейнере map

bool operator<(word a, word b)

{

return strcmp(a.get(), b.get()) < 0;

}

class opposite // пользовательский класс слов opposite

{

char str[40];

public:

opposite() { strcmp(str, ""); }

opposite(char \*s) { strcpy(str, s); }

char \*get() { return str; }

};

int main()

{

map<word, opposite> m;

// размещение в ассоциативном списке слов и антонимов

m.insert(pair<word, opposite>(word("да"),

opposite("нет")));

m.insert(pair<word, opposite>(word("хорошо"),

opposite("плохо")));

m.insert(pair<word, opposite>(word("влево"),  
 opposite("вправо")));

m.insert(pair<word, opposite>(word("вверх"),

opposite("вниз")));

// поиск антонима по заданному слову

char str[80];

cout << "Введите слово: ";

cin >> str;

map<word, opposite>::iterator p;

p = m.find(word(str));

if (p != m.end())

cout << "Антоним: " << p->second.get();

else

cout << "Такого слова в ассоциативном списке нет\n";

cout << "ob1: " << ob1.geta() << endl;

cout << "ob2: " << ob2.geta() << endl;

return 0;

}

## ПРИМЕР 33. Алгоритмы count и count\_if

Демонстрация алгоритмов count и count\_if.

#include <iostream>

#include <vector>

#include <algoritm>

using namespace std;

/\* унарный предикат, который определяет,

является ли значение четным \*/

bool even(int x)

{

return !(x%2);

}

int main()

{

vector<int> v;

for (int i = 0; i < 20; i++)

{

if (i%2)

v.push\_back(1);

else

v.push\_back(2);

}

cout << "Последовательность: ";

for (int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

int n = count(v.begin(), v.end(), 1);

cout << n << " количество элементов равных 1\n";

n = count\_if(v.begin(), v.end(), even);

cout << n << " количество четных элементов\n";

return 0;

}

## ПРИМЕР 34. Алгоритм remove\_copy

Демонстрация алгоритма remove\_copy.

#include <iostream>

#include <vector>

#include <algoritm>

using namespace std;

int main()

{

vector<int> v, v2(20);

for (int i = 0; i < 20; i++)

{

if (i%2) v.push\_back(1);

else v.push\_back(2);

}

cout << "Последовательность: ";

for (int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

// удаление единиц

remove\_copy(v.begin(), v.end(), v2.begin(), 1);

cout << "Результат: ";

for (int i = 0; i < v2.size(); i++)

cout << v2[i] << " ";

cout << endl;

return 0;

}

## ПРИМЕР 35. Алгоритм reverse

Демонстрация алгоритма reverse.

#include <iostream>

#include <vector>

#include <algoritm>

using namespace std;

int main()

{

vector<int> v;

for (int i = 0; i < 10; i++)

v.push\_back(i);

cout << "Исходная последовательность: ";

for (int i = 0; i < v.size(); i++)

cout << v[i] << " ";

cout << endl;

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

cout << "Обратная последовательность: ";

for (int i = 0; i < v.size(); i++)

cout << v[i] << " ";

return 0;

}

## ПРИМЕР 36. Алгоритм transform

Пример использования алгоритма *transform*.

#include <iostream>

#include <list>

#include <algoritm>

using namespace std;

// Простая функция модификации

int xform(int i)

{

return i \* i; // квадрат исходного значения

}

int main()

{

list<int> x1;

// размещение значений в списке

for (int i = 0; i < 10; i++)

v.push\_back(i);

cout << "Исходное содержимое списка x1: ";

list<int>::iterator p = x1.begin();

while (p != x1.end())

{

cout << \*p << " ";

++p;

}

cout << endl;

// модификация элементов списка x1

p = transform(x1.begin(), x1.end(), x1.begin(), xform);

cout << "Модифицированное содержимое списка x1: ";

p = x1.begin();

while (p != x1.end())

{

cout << \*p << " ";

++p;

}

return 0;

}