Максим Кузнецов
Игорь Симдянов

bhv®

C++ мастер-класс в задачах и примерах

158 задач и готовых решений!

Тонкости C и C++

Объектно-ориентированное программирование

Библиотека STL

Библиотека ввода-вывода

On-line поддержка

+ CD-ROM

УДК 681.3.06
ББК 32.973.26-018.2
К89

Кузнецов, М. В.

К89 С++. Мастер-класс в задачах и примерах / М. В. Кузнецов, И. В. Симдянов. — СПб.: БХВ-Петербург, 2007. — 480 с.: ил. + CD-ROM

ISBN 978-5-94157-953-2

Книга разбита на две основные части: задачи и решения. Рассматриваются базовые конструкции языка С++, тонкие моменты низкоуровневых операций, объектно-ориентированное программирование, разработка приложений при помощи стандартной библиотеки шаблонов STL, а также прикладные задачи. Особенностью предлагаемых задач и их решений является независимость от платформы и среды программирования, поэтому книга будет интересна как UNIX-, так и Windows-программистам. Компакт-диск содержит листинги всех готовых решений, представленных в книге.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Наталья Таркова*
Зав. редакцией	*Григорий Добин*
Редактор	*Анна Кузьмина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

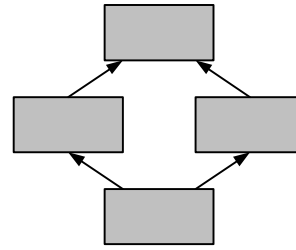
Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.12.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-94157-953-2

Оглавление

ГЛАВА I.3



Функции

Функции являются важнейшей структурной единицей языка C++. Программист может создавать как собственные функции, так и использовать стандартные библиотечные.

Именно с введением функций связывают эру процедурного программирования. Повторяющиеся фрагменты в программе можно оформить в виде отдельной функции, которую можно вызвать одной строкой. В результате разработчик может поддерживать гораздо больший объем кода: во-первых, многострочные последовательности заменяются вызовом одной функции, и текст программы проще воспринимать, во-вторых, монолитную программу можно разбить на отдельные блоки, которые можно разместить в разных файлах. Последнее особенно важно, т. к. программист хорошо ориентируется лишь в том коде, который занимает объем одной-двух страниц. Такую организацию кода гораздо проще выполнить при помощи функций. Таким образом, функции выполняют две важные задачи: обеспечивают повторное использование кода и улучшают читабельность программы.

Замечание

Решения задач данной главы можно найти в каталоге code\3 компакт-диска, поставляемого вместе с книгой.

I.3.1. Подсчет числа вызовов функции

Создайте функцию, которая будет подсчитывать число собственных вызовов и возвращать это значение.

I.3.2. Подсчет среднего значения

Создайте программу, которая будет принимать у пользователя число и выводить среднее арифметическое всех введенных за сеанс чисел. Программа

должна запрашивать у пользователя числа до тех пор, пока пользователь не введет 0. Процедуру подсчета среднего арифметического реализуйте в виде отдельной функции.

Подсказка

Для реализации функции подсчета среднего арифметического следует использовать статические переменные.

I.3.3. Обработка одномерного массива в функции

Пусть имеется массив `arr`, объявление которого представлено в листинге I.3.1. Необходимо создать функцию `squared()`, которая примет массив `arr`, возведет значения его элементов в квадрат. Вторая функция `print_arr()` должна вывести элементы массива в стандартный поток.

Листинг I.3.1. Одномерный массив `arr`

```
int main()
{
  int arr[] = {3, 4, 5, 6, 7, 8, 9, 10};
  return 0;
}
```

I.3.4. Указатель на последний элемент массива

Создайте функцию, которая принимает в качестве аргумента массив `arr` (см. листинг I.3.1) и возвращает указатель на последний элемент массива — выведите содержимое последнего элемента в стандартный поток вывода.

I.3.5. Функция обмена значений двух переменных

Создайте функцию `swap()`, которая будет принимать два параметра и менять местами их значения. Следует реализовать два варианта: с использованием указателей и ссылок. Как правило, для создания такой функции применяется временная переменная или указатель, попробуйте создать функцию, которая использует только два переданных параметра, не привлекая временных переменных.

Замечание

Предполагается, что функция обменивает значениями две переменные типа `int`.

I.3.6. Рекурсивный вызов

Создайте рекурсивную функцию, которая, принимая в качестве значения целое положительное число `number`, выводила бы его и, если оно не равно нулю, вызывала бы сама себя, передавая в качестве параметра число на единицу меньше: `number − 1`.

I.3.7. Переменная сумма

В математике имеется понятие суммы, обозначаемой символом Σ. Пусть имеется переменное число вложенных сумм:

```
Σ Σ Σ Σ ... Σ Σ Σ 1
```

Первая сумма (крайняя левая) пробегает значение от 0 до 1, т. е. она суммирует свой аргумент один раз, вторая сумма от 0 до 2, т. е. суммируем свой аргумент два раза, последняя сумма (крайняя правая) суммирует единицу *n* раз, где *n* — количество сумм в последовательности. Напишите программу, которая будет запрашивать у пользователя число вложенных сумм и возвращать их значение согласно представленной выше формуле. Решите задачу рекурсивно и итеративно (без использования рекурсии).

I.3.8. Допустимо ли выражение *f() = 10.0* ?

Допускается ли использование функции в левой части равенства, если нет, то почему? Если допускается, то при каких условиях?

I.3.9. Предотвращение выхода за границы массива

Создайте функции `put()` и `get()`, которые обращались бы к глобальному массиву `int arr[]` и позволяли присваивать значения его элементам и читать соответственно. Функции не должны допускать выхода за границы массива `arr[]`, в котором должно быть 100 элементов.

I.3.10. Вывод строки в стандартный поток

Создайте функцию `print_str()`, которая выводит в стандартный поток строку. Пусть функция может принимать два аргумента, первый из которых является строкой, а второй необязательный аргумент является количеством символов в строке, которые нужно вывести в стандартный поток. Если второй параметр не передается, в стандартный поток должно выводиться не более 30 символов.

I.3.11. Функции *abs()*, *labs()* и *fabs()*

Функции `abs()`, `labs()` и `fabs()` стандартной библиотеки возвращают абсолютное значение целого (`int`), длинного целого (`long int`) и числа с плавающей точкой (`double`). Почему они не реализованы как одна функция `abs()`, которая принимает аргументы всех трех типов? Реализуйте такую функцию.

I.3.12. Ошибка в перегрузке функции

В листинге I.3.2 представлен код, в котором перегружается функция `low()`, для аргумента типа `double` возвращает косинус, а для аргумента типа `float` — синус. Код содержит ошибку, найдите и устраните ее.

Замечание

Для вызова библиотечных функций `sin()` и `cos()` необходимо подключать заголовочный файл `<math>`.

Листинг I.3.2. Перегрузка функции `low()`

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

float low(float var);
double low(double var);

int main()
{
  cout << low(20) << "\n";

  return 0;
}

float low(float var)
{
  return sin(var);
}

double low(double var)
{
  return cos(var);
}
```

I.3.13. Функция с переменным количеством параметров

Создайте функцию, которая будет принимать произвольное количество аргументов и возвращать в качестве результатов их сумму.

Замечание

В качестве первого аргумента допускается передать количество последующих аргументов.

I.3.14. Указатель на функцию

В листинге I.3.3 представлены две однотипные функции — `f1()` и `f2()`. Создайте указатель на эти функции и вызовите их посредством указателя.

Листинг I.3.3. Функции `f1()` и `f2()`

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

void f1(void)
{
  cout << "Выполняется функция f1()\n";
}
void f2(void)
{
  cout << "Выполняется функция f2()\n";
}

int main()
{
  return 0;
}
```

I.3.15. Обработка функцией элементов массива

Создайте функцию `trig()`, которая принимает массив типа `double` в качестве первого параметра и указатель на библиотечную функцию `sin()` и `cos()` из стандартной библиотеки в качестве второго параметра. В результате работы функции каждый элемент массива должен принять синус или косинус своего старого значения, в зависимости от того указатель на функцию синуса или косинуса передано в качестве второго аргумента функции `trig()`.

Замечание

Для вызова библиотечных функций `sin()` и `cos()` необходимо подключать заголовочный файл `<math>`.

I.3.16. Односвязный список

Используя конструкцию `struct`, создайте односвязный список, содержащий целое число и ссылку на следующий элемент списка (рис. I.3.1). Функция `add_element()` должна добавлять новый элемент в список, функция `delete_element()` удалять элемент из начала списка, а функция `delete_list()` удалять все элементы списка.

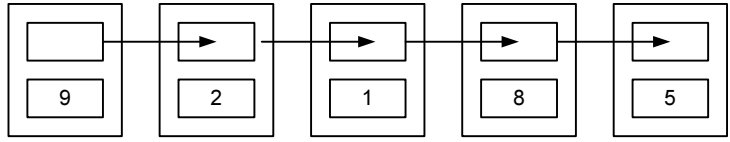


Рис. I.3.1. Односвязный список

Для контроля состояния создайте функцию `print_list()`, которая должна выводить значения элементов с первого элемента до последнего. Так для списка, изображенного на рис. I.3.1, программа должна вывести следующую последовательность цифр:

```
9
2
1
8
5
```

I.3.17. Двухсвязный список

Используя конструкцию `struct`, создайте двухсвязный список, содержащий целое число и ссылку на следующий и предыдущий элемент списка (рис. I.3.2). Создайте набор функций, позволяющих вставлять элемент в начало списка `add_first()`, в конец списка `add_last()`, слева от текущего элемента `add_prev()`, справа от текущего элемента `add_next()`. Кроме того, создайте функции, удаляющие текущий элемент `delete_element()` и весь список `delete_list()`. Для того чтобы двухсвязный список считался полноценным, создайте функцию, которая возвращает указатель на первый `get_first()` и последний `get_last()` элементы списка, а также функцию `get_count()`, подсчитывающую количество элементов в списке.

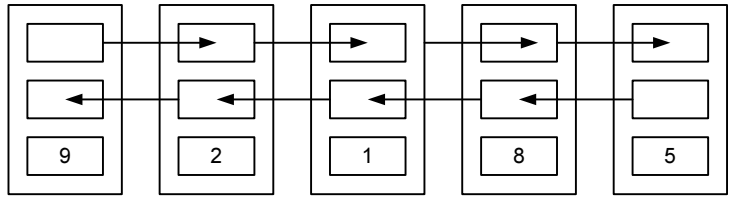


Рис. I.3.2. Двухсвязный список

Для контроля состояния создайте функцию `print_list()`, которая должна выводить значения элементов, начиная с первого элемента и заканчивая последним. Так для списка, изображенного на рис. I.3.2, программа должна вывести следующую последовательность цифр:

```
9
2
1
8
5
```

I.3.18. Создание файла с уникальным именем

Напишите программу, которая будет создавать в текущем каталоге файл с уникальным именем и записывать введенные пользователем строки в файл до тех пор, пока пользователь не введет пустую строку.

I.3.19. Количество строк в файле

Пусть имеется текстовый файл, например, файл созданный при помощи программы из *разд. I.3.18*. Создайте программу, подсчитывающую количество строк в файле.

I.3.20. Вывод случайной строки из файла

Пусть имеется текстовый файл, например, файл созданный при помощи программы из *разд. I.3.18*. Создайте программу, которая будет выводить из файла случайную строку.

I.3.21. Вывод трех случайных строк файла

Создайте программу, которая выводила бы из текстового файла три случайных строки, при этом строки не должны повторяться.

I.3.22. Последние три строки файла

Напишите программу, которая выводила бы из текстового файла три последних строки.

I.3.23. Поиск строки в файле

Создайте программу, которая станет выводить номера строк файла, в которых будет найдено слово, введенное пользователем.

I.3.24. Самая длинная и самая короткая строка в файле

На компакт-диске, поставляемом вместе с книгой, в каталоге code\3 находится стандартный словарь Linux — linux.words. Создайте программу, которая последовательно читает данный файл и выводит самое длинное и самое короткое слово словаря, а также количество символов в нем.

I.3.25. Список слов заданной длины

На компакт-диске, поставляемом вместе с книгой, в каталоге code\3 находится стандартный словарь Linux — linux.words. Создайте программу, которая принимает у пользователя целое число `num` и выводит слова, число символов в которых не превышает, меньше или равно `num`. Подсчитайте количество найденных соответствий.

I.3.26. Поиск слов по первым символам

На компакт-диске, поставляемом вместе с книгой, в каталоге code\3 находится стандартный словарь Linux — linux.words. Создайте программу, которая принимает у пользователя первые символы слова и выводит все найденные соответствия и количество найденных соответствий.

I.3.27. Изменение порядка следования строк в файле

На компакт-диске, поставляемом вместе с книгой, в каталоге code\3 находится стандартный словарь Linux — linux.words. Создайте программу, которая изменяет порядок следования строк в файле, размещая последнюю строку файла первой, предпоследнюю — второй, пред-предпоследнюю — третьей и т. д.

I.3.28. Разбиение файла на части

На компакт-диске, поставляемом вместе с книгой, в каталоге code\3 находится стандартный словарь Linux — linux.words. Создайте программу, которая разбивает содержимое файла linux.words на несколько частей по 1000 слов в каждой и сохраняет каждый фрагмент linux.words в отдельных файлах с именем part и расширением, равным текущему номеру файла, т. е. part.1, part.2, part3, ..., part.100.

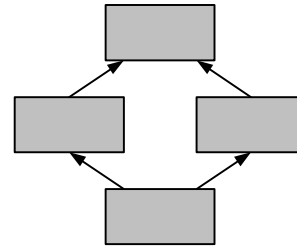
I.3.29. Шаблоны функций

При помощи шаблона функции реализуйте функцию `swap()`, которая будет принимать два параметра произвольного типа и менять местами их значения.

I.3.30. Перегрузка шаблона функций

Создайте два варианта функций `mmin()` и `mmax()`, которые возвращают минимальное и максимальное значения из ряда чисел. Первый вариант должен принимать указатель на массив элементов и количество элементов в массиве, а второй должен принимать лишь два значения. Оба варианта должны использовать шаблоны так, чтобы для аргументов допускалось использование любого базового типа.

ГЛАВА I.4



Объекты и классы

Классы позволяют объединять переменные и обрабатывающий их код в единую структуру. Это поднимает процесс разработки программы на новый уровень абстракции, программист получает возможность программировать не в терминах функций, переменных, файлов, а при помощи абстрактных понятий: можно отправить объект "договор" объекту "принтер", а к объекту "словарь" применить метод поиска. Библиотеки классов позволяют сосредоточиться на решении прикладных задач и думать в терминах прикладной задачи, не увязая в реализации, алгоритмах и машинном представлении программы.

Объектно-ориентированная технология в C++ является чрезвычайно мощным инструментом, сродни бензопилы, однако при неумелом использовании этот инструмент может нанести вред и тому, кто им пользуется. Язык C++, начиная с указателей и заканчивая объектно-ориентированной технологией и библиотекой STL, построен так, чтобы выжать максимум из компьютера — поэтому здесь нет даже намека на "защиту от дурака". Чтобы создавать эффективные программы, следует от начала до конца понимать, что делаешь, зачем и к каким последствиям это приведет.

Замечание

Решения задач данной главы можно найти в каталоге code\4 компакт-диска, поставляемого вместе с книгой.

I.4.1. Чем отличается структура *struct* от класса *class*?

Отличается ли чем-то структура `struct` от класса `class` или данные ключевые слова дублируют друг друга?

I.4.2. Чем отличается объединение *union* от класса *class*?

Назовите сходства и различия объединения `union` и класса `class`.

I.4.3. Константы в классах

Создайте класс `cls`, который бы в качестве одного из значений содержал константу `COUNT`. Объявите два объекта класса `cls`. Поле `COUNT` первого объекта `fst` должно иметь значение 100, а поле `COUNT` второго объекта `snd` — значение 200.

I.4.4. Подсчет количества созданных объектов

Создайте класс, который бы подсчитывал, сколько объектов данного класса создано в настоящий момент.

I.4.5. Найдите ошибку

В листинге I.4.1 приведен код, реализующий класс `cls` и объявляющий объект `obj` данного класса. Однако класс реализован ошибочно. Найдите и исправьте ошибку.

Листинг I.4.1. Определение класса

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

class cls
{
  public:
    void cls(int fst, int snd);
  private:
    int first;
    int second;
};
// Конструктор
void cls::cls(int fst, int snd)
{
  first = fst;
  second = snd;
}
```

```
int main()
{
  cls obj(1,2);

  return 0;
}
```

I.4.6. Использование объекта в нескольких файлах

Определение класса `number` находится в файле myclass.h (листинг I.4.2), реализация методов находится в файле myclass.cpp (листинг I.4.3).

Листинг I.4.2. Файл myclass.h

```
#ifndef MyclassH
#define MyclassH
class number
{
  public:
    number(int value);
    int get_number();
  private:
    int var;
};
#endif
```

Листинг I.4.3. Файл myclass.cpp

```
#include "myclass.h"

number::number(int value)
{
  var = value;
}
int number::get_number()
{
  return var;
}
```

В файлах main.cpp и utils.cpp используется один и тот же объект `num` класса `number`. Как его следует объявить, чтобы в файлах main.cpp и utils.cpp использовался один и тот же объект?

I.4.7. Инициализация объекта при помощи =

В листинге I.4.4 используется объект `obj`, при этом его инициализация производится не посредством передачи параметров конструктору, а при помощи оператора =. Какова должна быть реализация класса `cls`, чтобы код в листинге I.4.4 был рабочим?

Листинг I.4.4. Инициализация объекта при помощи оператора =

```
#include <iostream>
#include "cls.h";
using namespace std;

int main()
{
  cls obj = 12;
  cout << obj.number << "\n";

  return 0;
}
```

I.4.8. Класс с динамическим массивом

Создайте класс, который бы при создании объекта в конструкторе выделял память под целочисленный массив, объем которого задается параметром конструктора, а при уничтожении в деструкторе освобождал бы память. Создайте интерфейс для доступа к членам массива (чтение и запись) с контролем выхода за границы массива.

I.4.9. Класс-интерфейс к файлу

На компакт-диске, поставляемом вместе с книгой, в каталоге code\4 находится стандартный словарь Linux — linux.words. Создайте класс, объект которого в конструкторе открывает этот файл, читает его содержимое в массив, а в деструкторе закрывает файл. Класс должен содержать перегруженный метод `print()`, который должен реализовывать функциональность, представленную в табл. I.4.1.

Таблица I.4.1. *Варианты использования метода* `print()`

Синтаксис метода `print()`	Описание
`print()`	Без параметров метод `print()` должен выводить все содержимое файла linux.words

Таблица I.4.1 *(окончание)*

Синтаксис метода `print()`	Описание
`print(char *str)`	Если метод `print()` принимает в качестве параметра единственную строку `str`, в стандартный поток должны выводиться строки файла linux.words, которые начинаются со строки `str`
`print(int min, int max)`	Если метод принимает два целочисленных параметра, то в стандартный поток должны выводиться слова из файла linux.words, состоящие из количества символов больше или равных `min` и меньше или равных `max`
`print(int count)`	Если метод `print()` принимает единственный целочисленный параметр `count`, в стандартный поток должны выводиться слова, содержащие не больше `count` символов

I.4.10. Постраничная навигация

В листинге I.4.5 представлен класс `part`, позволяющий хранить в каждом объекте класса 10 строк длиной 80 символов. На компакт-диске, поставляемом вместе с книгой, в каталоге code\4 находится стандартный словарь Linux — linux.words. Создайте класс `dict`, который бы читал файл linux.words, разбивал его на части по 10 слов, создавая под каждую из таких частей объект `part`, реализуя массив объектов `part`. На основании этих двух классов создайте программу, которая бы загружала словарь и позволяла пользователю выводить части словаря по индексу массива объектов `part`.

Листинг I.4.5. Класс `part`

```
class part
{
  public:
    static const int NUMBER_STRINGS = 10;
    static const int NUMBER_CHARS   = 80;
    char str[NUMBER_STRINGS][NUMBER_CHARS];
    part()
    {
      for(int i = 0; i < NUMBER_STRINGS; i++)
      {
        str[i][0] = '\0';
      }
    }
```