Лабораторная работа №8
Тема: «Функции»

Цель работы: сформировать навыки и умения обработки структурированных типов данных, организованных в виде функций.

Время выполнения: 4 часа.

Теоретические сведения

Функции — это блоки кода, выполняющие определенные операции. Если требуется, функция может определять входные параметры, позволяющие вызывающим объектам передавать ей аргументы. При необходимости функция также может возвращать значение как выходное. Функции полезны для инкапсуляции основных операций в едином блоке, который может многократно использоваться. В идеальном случае имя этого блока должно четко описывать назначение функции. Следующая функция принимает два целых числа от вызывающего объекта и возвращает их сумму; a и b — это параметры типа int.

```
int sum(int a, int b)
{
   return a + b;
}
```

Функция может вызываться или вызываться из любого количества мест в программе. Значения, передаваемые функции, являются аргументами, типы которых должны быть совместимы с типами параметров в определении функции.

```
int main()
{
   int i = sum(10, 32);
   int j = sum(i, 66);
   cout << "The value of j is" << j << endl; // 108
}
```

Практических ограничений на длину функции нет, но хороший дизайн предназначен для функций, которые выполняют одну четко определенную задачу. Сложные алгоритмы лучше разбивать на более короткие и простые для понимания функции, если это возможно.

Функции, определенные в области видимости класса, называются функциями-членами. В C++, в отличие от других языков, функции можно

также определять в области видимости пространства имен (включая неявное глобальное пространство имен). Такие функции называются свободными или не-членами; они широко используются в стандартной библиотеке.

Функции могут быть перегружены. Это означает, что разные версии функции могут иметь одно и то же имя, если они отличаются числом и/или типом формальных параметров.

Части объявления функции

Минимальное объявление функции состоит из возвращаемого типа, имени функции и списка параметров (который может быть пустым), а также необязательных ключевых слов, которые предоставляют дополнительные инструкции для компилятора. В следующем примере показано объявление функции:

```
int sum(int a, int b);
```

Определение функции состоит из объявления и тела, который представляет собой весь код между фигурными скобками:

```
int sum(int a, int b)
{
   return a + b;
}
```

Объявление функции, за которым следует точка с запятой, может многократно встречаться в разных местах кода программы. Оно необходимо перед любыми вызовами этой функции в каждой записи преобразования. По правилу одного определения, определение функции должно фигурировать в коде программы лишь один раз.

При объявлении функции необходимо указать:

Тип возвращаемого значения, который указывает тип значения, возвращаемого функцией, или void значение, если значение не возвращается. В C++ является допустимым типом возвращаемого значения, auto который указывает компилятору вывести тип из оператора return. В C++ decltype(auto) также разрешено. Дополнительные сведения см. в подразделе "Выведение возвращаемых типов" ниже.

Имя функции, которое должно начинаться с буквы или подчеркивания и не может содержать пробелы. Как правило, символы подчеркивания в начале имен функций стандартной библиотеки указывают на частные функции-члены

или функции, не являющиеся членами, которые не предназначены для использования в коде.

Список параметров, заключенный в скобки. В этом списке через запятую указывается нужное (возможно, нулевое) число параметров, задающих тип и, при необходимости, локальное имя, по которому к значениям можно получить доступ в теле функции.

Необязательные элементы объявления функции:

constexpr — указывает, что возвращаемое значение функции является константой, значение которой может быть определено во время компиляции.

```
constexpr float exp(float x, int n)
{
   return n == 0 ? 1 :
      n % 2 == 0 ? exp(x * x, n / 2) :
      exp(x * x, (n - 1) / 2) * x;
};
```

Спецификация компоновки или externstatic.

```
//Declare printf with C linkage.
extern "C" int printf( const char *fmt, ... );
```

Дополнительные сведения см. в разделе Единицы перевода и компоновка.

inline — отдает компилятору команду заменять каждый вызов функции ее кодом. Подстановка может улучшить эффективность кода в сценариях, где функция выполняется быстро и многократно вызывается во фрагментах, являющихся критическими для производительности программы.

```
inline double Account::GetBalance()
{
   return balance;
}
```

Выражение noexcept , указывающее, может ли функция вызывать исключение. В следующем примере функция не создает исключение, если is_pod выражение имеет truезначение .

```
#include <type_traits>

template <typename T>
T copy_object(T& obj) noexcept(std::is_pod<T>) {...}
```

Для получения дополнительной информации см. noexcept.

(Только функции-члены) Cv-квалификаторы, которые указывают, является const ли функция или volatile.

(Только функции-члены) virtual, overrideили final. virtual — указывает, что функция может быть переопределена в производном классе. override — означает, что функция в производном классе переопределяет виртуальную функцию. final означает, что функция не может быть переопределена в любом последующем производном классе.

Применение к функции-члену означает, что функция не связана ни с какими экземплярами объектов класса.

Квалификатор ссылки, который указывает компилятору, какую перегрузку функции следует выбрать, когда параметр неявного объекта (*this) является ссылкой rvalue или ссылкой lvalue.

Определения функций

Определение функции состоит из объявления и тела функции, заключенных в фигурные скобки, которые содержат объявления переменных, операторы и выражения. В следующем примере показано полное определение функции:

```
int foo(int i, std::string s)
{
  int value {i};
  MyClass mc;
  if(strcmp(s, "default") != 0)
  {
      value = mc.do_something(i);
  }
  return value;
}
```

Переменные, объявленные в теле функции, называются локальными. Они исчезают из области видимости при выходе из функции, поэтому функция никогда не должна возвращать ссылку на локальную переменную.

```
MyClass& boom(int i, std::string s)
{
  int value {i};
  MyClass mc;
  mc.Initialize(i,s);
  return mc;
```

```
}
```

Функции const и constexpr

Можно объявить функцию-член как const, чтобы указать, что функции запрещено изменять значения каких-либо элементов данных в классе. Объявляя функцию-член как const, вы помогаете компилятору обеспечить константкорректность. Если кто-то по ошибке пытается изменить объект с помощью функции, объявленной как const, возникает ошибка компилятора.

Объявите функцию как constexpr, если значение, которое она создает, можно определить во время компиляции. Функция constexpr обычно выполняется быстрее, чем обычная функция.

Шаблоны функций

Шаблоны функций подобны шаблонам классов. Их задача заключается в создании конкретных функций на основе аргументов шаблонов. Во многих случаях шаблоны могут определять типы аргументов, поэтому их не требуется явно указывать.

```
template<typename Lhs, typename Rhs>
auto Add2(const Lhs& lhs, const Rhs& rhs)
{
   return lhs + rhs;
}

auto a = Add2(3.13, 2.895); // a is a double
auto b = Add2(string{ "Hello" }, string{ " World" }); // b is a std::string
```

Параметры и аргументы функций

У функции имеется список параметров, в котором через запятую перечислено необходимое (возможно, нулевое) число типов. Каждому параметру присваивается имя, по которому к нему можно получить доступ в теле функции. Шаблон функции может указывать дополнительные параметры типа или значения. Вызывающий объект передает аргументы, представляющие собой конкретные значения, типы которых совместимы со списком параметров.

По умолчанию аргументы передаются функции по значению, то есть функция получает копию передаваемого объекта. Для больших объектов создание копии может быть дорогостоящим и не всегда требуется. Чтобы аргументы передавались по ссылке (в частности, ссылке lvalue), добавьте квантификатор ссылки в параметр

```
void DoSomething(std::string& input){...}
```

Если функция изменяет аргумент, передаваемый по ссылке, изменяется исходный объект, а не его локальная копия. Чтобы предотвратить изменение такого аргумента функцией, квалифицируйте параметр как const&:

void DoSomething(const std::string& input){...}

C++: Для явной обработки аргументов, передаваемых по rvalue-reference или lvalue-reference, используйте double-ampersand в параметре , чтобы указать универсальную ссылку:

void DoSomething(const std::string&& input){...}

Функция, объявленная с одним ключевым словом void в списке объявления параметров, не принимает аргументов, если ключевое слово void является первым и единственным членом списка объявлений аргументов. Аргументы типа void в другом месте списка создают ошибки. Пример:

```
// OK same as GetTickCount()
long GetTickCount( void );
```

Несмотря на то, что нельзя указывать void аргумент, за исключением описанного здесь, типы, производные от типа void (например, указатели void на и массивы) могут отображаться в любом месте списка объявлений аргументов void.

Аргументы по умолчанию

Последним параметрам в сигнатуре функции можно назначить аргумент по умолчанию, т. е. вызывающий объект сможет опустить аргумент при вызове функции, если не требуется указать какое-либо другое значение.

```
int DoSomething(int num,
   string str,
   Allocator& alloc = defaultAllocator)
{ ... }

// OK both parameters are at end
int DoSomethingElse(int num,
   string str = string{ "Working" },
   Allocator& alloc = defaultAllocator)
{ ... }

// C2548: 'DoMore': missing default parameter for parameter 2
int DoMore(int num = 5, // Not a trailing parameter!
   string str,
```

```
    Allocator& = defaultAllocator)
{...}
```

Функция не может возвращать другую функцию или встроенный массив; однако он может возвращать указатели на эти типы или лямбда-выражение, которое создает объект функции. За исключением этих случаев, функция может возвращать значение любого типа, который находится в области действия, или она может не возвращать значение. В этом случае типом возвращаемого значения является void.

Завершающие возвращаемые типы

"Обычные" возвращаемые типы расположены слева от сигнатуры функции. Конечный тип возвращаемого значения находится в правой части сигнатуры и предшествует оператору -> . Завершающие возвращаемые типы особенно полезны в шаблонах функций, когда тип возвращаемого значения зависит от параметров шаблона.

```
template<typename Lhs, typename Rhs>
auto Add(const Lhs& lhs, const Rhs& rhs) -> decltype(lhs + rhs)
{
   return lhs + rhs;
}
```

При auto использовании в сочетании с конечным типом возвращаемого значения он просто служит заполнителем для выражения типа decltype и сам не выполняет выведение типа.

Локальные переменные функции

Переменная, объявленная в теле функции, называется локальной или просто локальной. Нестатические локальные элементы видны только в теле функции и, если они объявлены в стеке, выходят за пределы области действия при выходе функции. При создании локальной переменной и возвращении ее по значению компилятор обычно может выполнить оптимизацию именованного возвращаемого значения, чтобы избежать ненужных операций копирования. Если локальная переменная возвращается по ссылке, компилятор выдаст предупреждение, поскольку любые попытки вызывающего объекта использовать эту ссылку произойдут после уничтожения локальной переменной.

В C++ локальные переменные можно объявлять, как статические. Переменная является видимой только в теле функции, однако для всех экземпляров функции существует только одна копия переменной. Локальные статические объекты удаляются во время завершения, определенного

директивой atexit. Если статический объект не был создан, так как поток управления программы обошел его объявление, попытка уничтожить этот объект не предпринимается.

Выведение возвращаемых типов (C++)

В C++ можно использовать auto , чтобы указать компилятору выводить тип возвращаемого значения из текста функции, не указывая конечный тип возвращаемого значения. Обратите внимание, что auto всегда выводит значение, возвращаемое по значению. Используйте auto&&, чтобы дать компилятору команду выведения ссылки.

В этом примере auto будет выведено как неконстная копия значения суммы lhs и rhs.

```
template<typename Lhs, typename Rhs>
auto Add2(const Lhs& lhs, const Rhs& rhs)
{
   return lhs + rhs; //returns a non-const object by value
}
```

Обратите внимание, что auto не сохраняет константность типа, который он выводит. Для функций пересылки, возвращаемое значение которых должно сохранять константность или ссылочность аргументов, можно использовать decltype(auto) ключевое слово , которое использует decltype правила вывода типов и сохраняет все сведения о типе. decltype(auto) может использоваться как обычное возвращаемое значение с левой стороны или как завершающее возвращаемое значение.

В следующем примере показано decltype(auto) использование для обеспечения идеальной пересылки аргументов функции в возвращаемом типе, который неизвестен до создания экземпляра шаблона.

```
template<typename F, typename Tuple = tuple<T...>, int... I>
decltype(auto) apply_(F&& f, Tuple&& args, index_sequence<I...>)
{
   return std::forward<F>(f)(std::get<I>(std::forward<Tuple>(args))...);
}

template<typename F, typename Tuple = tuple<T...>,
   typename Indices = make_index_sequence<tuple_size<Tuple>::value >>
  decltype( auto)
   apply(F&& f, Tuple&& args)
{
```

```
    return apply_(std::forward<F>(f), std::forward<Tuple>(args), Indices());
}
```

Возврат нескольких значений из функции

Существует несколько способов возврата нескольких значений из функции:

Инкапсулируйте значения в именованном объекте класса или структуры. Требует, чтобы определение класса или структуры было видимым для вызывающего объекта:

```
#include <string>
#include <iostream>

using namespace std;

struct S
{
    string name;
    int num;
};

S g()
{
    string t{ "hello" };
    int u{ 42 };
    return { t, u };
}

int main()
{
    S s = g();
    cout << s.name << " " << s.num << endl;
    return 0;
}
```

Возвращает объект std::tuple или std::p air:

```
#include <tuple>
#include <string>
#include <iostream>
```

```
using namespace std;

tuple<int, string, double> f()
{
   int i{ 108 };
   string s{ "Some text" };
   double d{ .01 };
   return { i,s,d };
}

int main()
{
   auto t = f();
   cout << get<0>(t) << " " << get<1>(t) << " " << get<2>(t) << endl;

   // --or--

   int myval;
   string myname;
   double mydecimal;
   tie(myval, myname, mydecimal) = f();
   cout << myval << " " << myname << " " << mydecimal << endl;

   return 0;
}
```

Используйте структурированные привязки. Преимущество структурированных привязок заключается в том, что переменные, в которых хранятся возвращаемые значения, инициализируются одновременно с объявлением, что в некоторых случаях может быть значительно эффективнее. В операторе auto[x, y, z] = f(); квадратные скобки вводят и инициализируют имена, которые находятся в области действия для всего блока функций.

```
#include <tuple>
#include <string>
#include <iostream>

using namespace std;

tuple<int, string, double> f()
```

```
{
   int i{ 108 };
   string s{ "Some text" };
   double d{ .01 };
   return { i,s,d };
}
struct S
{
   string name;
   int num;
};

S g()
{
   string t{ "hello" };
   int u{ 42 };
   return { t, u };
}

int main()
{
   auto[x, y, z] = f(); // init from tuple
   cout << x << " " << y << " " << z << endl;

   auto[a, b] = g(); // init from POD struct
   cout << a << " " << b << endl;
   return 0;
}
```

Помимо использования самого возвращаемого значения, можно "возвращать" значения, определяя любое количество параметров, используемых по ссылке, чтобы функция может изменять или инициализировать значения объектов, предоставляемых вызывающим объектом.

Указатели функций

Как и в C, в C++ поддерживаются указатели на функции. Однако более типобезопасной альтернативой обычно служит использование объекта-функции.

Рекомендуется использовать для typedef объявления псевдонима для типа указателя функции при объявлении функции, возвращающей тип указателя функции. Например.

```
typedef int (*fp)(int);
fp myFunction(char* s); // function returning function pointer
```

Если этого не сделать, правильный синтаксис объявления функции можно вывести из синтаксиса декларатора для указателя функции, заменив идентификатор (fp в приведенном выше примере) на имя функций и список аргументов следующим образом:

```
int (*myFunction(char* s))(int);
```

Предыдущее объявление эквивалентно объявлению, использующим typedef ранее.

C++ позволяет указать несколько функций с одним и тем же именем в одной области. Эти функции называются перегруженными функциями или перегрузками. Перегруженные функции позволяют предоставлять различную семантику для функции в зависимости от типов и количества ее аргументов.

Например, рассмотрим функцию print , которая принимает std::string аргумент . Эта функция может выполнять задачи, отличные от функций, которые принимают аргумент типа double. Перегрузка не позволяет использовать такие имена, как print_string или print_double. Во время компиляции компилятор выбирает перегрузку для использования в зависимости от типов и количества аргументов, переданных вызывающим. При вызове print(42.0)void print(double d) вызывается функция . При вызове print("hello world")вызывается перегрузка void print(std::string) .

Можно перегружать как функции-члены, так и свободные функции. В следующей таблице показано, какие части объявления функции C++ использует для различения групп функций с одинаковым именем в одной области.

```
// function_overloading.cpp
// compile with: /EHsc
#include <iostream>
#include <math.h>
#include <string>

// Prototype three print functions.
int print(std::string s);           // Print a string.
```

```
int print(double dvalue);           // Print a double.
int print(double dvalue, int prec);  // Print a double with a
                                     //  given precision.
using namespace std;
int main(int argc, char *argv[])
{
   const double d = 893094.2987;
   if (argc < 2)
   {
      // These calls to print invoke print( char *s ).
      print("This program requires one argument.");
      print("The argument specifies the number of");
      print("digits precision for the second number");
      print("printed.");
      exit(0);
   }

   // Invoke print( double dvalue ).
   print(d);

   // Invoke print( double dvalue, int prec ).
   print(d, atoi(argv[1]));
}

// Print a string.
int print(string s)
{
   cout << s << endl;
   return cout.good();
}

// Print a double in default precision.
int print(double dvalue)
{
   cout << dvalue << endl;
   return cout.good();
}

//  Print a double in specified precision.
//  Positive numbers for precision indicate how many digits
```

```
//  precision after the decimal point to show. Negative
//  numbers for precision indicate where to round the number
//  to the left of the decimal point.
int print(double dvalue, int prec)
{
   // Use table-lookup for rounding/truncation.
   static const double rgPow10[] = {
      10E-7, 10E-6, 10E-5, 10E-4, 10E-3, 10E-2, 10E-1,
      10E0, 10E1,  10E2,  10E3,  10E4, 10E5,  10E6 };
   const int iPowZero = 6;

   // If precision out of range, just print the number.
   if (prec < -6 || prec > 7)
   {
      return print(dvalue);
   }
   // Scale, truncate, then rescale.
   dvalue = floor(dvalue / rgPow10[iPowZero - prec]) *
      rgPow10[iPowZero - prec];
   cout << dvalue << endl;
   return cout.good();
}
```

В приведенном выше коде показаны перегрузки print функции в области файла.

Аргумент по умолчанию не считается частью типа функции. Поэтому он не используется при выборе перегруженных функций. Две функции, которые различаются только в своих аргументах, считаются множественными определениями, а не перегруженными функциями.

Аргументы по умолчанию нельзя указать для перегруженных операторов.

Индивидуальные задания к лабораторной работе №8

Задание 1. Используя методы создайте программу согласно варианту.

1. Дано натуральное число n. Выясните, можно ли представить данное число в виде произведения трех последовательных натуральных чисел.
2. Дано натуральное число m. Укажите все тройки натуральных чисел x, y и z, удовлетворяющие следующему условию: $m = x^3 + y^3 + z^3$.
3. Среди всех четырехзначных номеров машин, определите количество номеров, содержащих только три одинаковые цифры.

4. Среди всех четырехзначных номеров машин, определите количество номеров, содержащих три или более одинаковых цифры.

5. Среди всех четырехзначных номеров машин, определите количество номеров машин, содержащих только две одинаковые цифры.

6. Среди всех четырехзначных номеров машин, определите количество номеров машин, содержащих только две или более одинаковых цифры.

7. Напишите программу нахождения, следующего за данным совершенного числа. Совершенным называется число, сумма делителей которого, не считая самого числа, равна этому числу. Первое совершенное число 6 (6 = 1 + 2 + 3).

8. Проверьте, является ли данное натуральное число простым.

9. Дано натуральное число P. Напишите программу нахождения всех натуральных чисел, не превосходящих P, которые можно представить в виде произведения двух простых чисел.

10. Дано натуральное число P, заданное в восмеричной системе счисления. Напишите программу перевода этого числа в двоичную систему счисления. Встроенный метод перевода не использовать.

11. Дано натуральное число P, заданное в шестнадцатеричной системе счисления. Переведите его в двоичную систему счисления. Встроенный метод перевода не использовать.

12. Дано натуральное число P. Найдите все «совершенные» числа, не превосходящие P. Совершенным, называется число, сумма делителей которого, не считая самого числа, равна этому числу. Первое совершенное число 6 (6 = 1 + 2 + 3).

13. Дано натуральное n-значное число P. Проверьте, является ли данное число палиндромом (перевертышем).

14. Дано натуральное число P. Проверьте, кратно ли P трем, используя признак делимости на 3.

15. Дано натуральное число P. Проверьте, кратно ли P одиннадцати, используя признак делимости на 11 (знакопеременная сумма его цифр делится на 11).

16. Дано натуральное число P. Найдите все простые числа, не превосходящие числа P.

17. Дано натуральное число P. Найдите все делители числа P.

18. Дано натуральное число P. Найдите сумму цифр числа P.

19. Дано натуральное число P. Выбросите из записи числа P цифры 0, оставив прежним порядок остальных цифр.

20. Дано натуральное число P. Проверьте, кратно ли число P девяти, используя признак делимости на 9.

21. Дана обыкновенная дробь $\frac{m}{n}$. Сократите данную дробь.

22. Напишите программу сложения двух обыкновенных несократимых дробей $\frac{m}{n}$ и $\frac{p}{q}$. Результат представить в виде несократимой

дроби.

23. Напишите программу вычитания двух обыкновенных несократимых дробей $\frac{m}{n}$ и $\frac{p}{q}$. Результат представить в виде несократимой дроби.

24. Напишите программу умножения двух обыкновенных несократимых дробей $\frac{m}{n}$ и $\frac{p}{q}$. Результат представить в виде несократимой дроби.

25. Напишите программу деления двух обыкновенных несократимых дробей $\frac{m}{n}$. и $\frac{p}{q}$. Результат представить в виде несократимой дроби.

26. Данное натуральное число N переведите из десятичной системы счисления в двоичную. Встроенный метод перевода не использовать.

27. Данное натуральное число N замените суммой квадратов его цифр. Произведите K таких замен.

28. Дано натуральное число n. Найдите все меньшие n числа Мерсена. Простое число называется числом Мерсена, если оно может быть представлено в виде 2p – 1, где p – тоже простое число.

29. Дано натуральное число N. Найдите все составные натуральные числа, меньшие N.

Задание 2. Используя перегрузку методов, создайте программу согласно варианту.

1.
а) для сложения целых чисел;
б) для сложения комплексных чисел.
2.
а) для сложения вещественных чисел;
б) для сложения комплексных чисел.
3.
а) для умножения целых чисел;
б) для умножения комплексных чисел.
4.
а) для вычитания целых чисел;
б) для вычитания комплексных чисел.
5.
а) для умножения вещественных чисел;
б) для умножения комплексных чисел.
6.
а) для вычитания вещественных чисел;
б) для вычитания комплексных чисел.

7.

а) для деления целых чисел;

б) для деления комплексных чисел.

8.

а) по номеру года выдает его название по старояпонскому календарю;

б) по названию месяца выдает знак Зодиака.

9.

а) для сложения десятичных дробей;

б) для сложения обыкновенных дробей.

10.

а) для вычитания десятичных дробей;

б) для вычитания обыкновенных дробей.

11.

а) для умножения десятичных дробей;

б) для умножения обыкновенных дробей.

12.

а) для деления десятичных дробей;

б) для деления обыкновенных дробей.

13.

а) для преобразования десятичной дроби в обыкновенную;

б) для преобразования обыкновенной дроби в десятичную.

14.

а) для вычисления натурального логарифма;

б) для вычисления десятичного логарифма.

15.

а) целые числа возводит в степень n;

б) из десятичных чисел извлекает корень степени n.

16.

а) для перевода часов и минут в минуты;

б) для перевода минут в часы и минуты.

17.

а) для последовательности целых чисел находит среднее арифметическое;

б) для строки находит количество букв, содержащихся в ней.

18.

а) для последовательности целых чисел находит максимальный элемент;

б) для строки находит длину самого длинного слова.

19.

а) для последовательности целых чисел находит минимальный элемент;

б) для строки находит длину самого короткого слова.

20.

а) для последовательности целых чисел находит количество четных элементов;

б) для строки находит количество слов, начинающихся на букву «а».

21.

а) для последовательности целых чисел находит количество отрицательных элементов;

б) для строки находит количество слов, заканчивающихся и начинающихся на одну и ту же букву.

22.

а) для последовательности целых чисел находит количество нечетных элементов;

б) для строки находит количество слов в ней.

23.

а) для последовательности, начинающейся на четное число, выполняет циклический сдвиг влево на количество элементов равное первому элементу последовательности.

б) для последовательности, начинающейся на нечетное число, выполняет циклический сдвиг вправо на количество элементов равное последнему элементу последовательности.

24.

а) для последовательности целых чисел удаляет все четные элементы из последовательности;

б) для строки удаляет все четные слова.

25.

а) для последовательности удаляет все четные элементы;

б) для последовательности удаляет все элементы, заключенные между двумя нулевыми элементами.

26.

а) для последовательности целых чисел находит количество простых чисел;

б) для строки находит количество слов, заканчивающихся и начинающихся на одну и ту же букву.

27.

а) для последовательности целых чисел находит количество нечетных элементов;

б) для строки находит количество пробелов в ней.

28.

а) для последовательности, начинающейся на четное число, выполняет обрезку всех элементов после четного числа.

б) для последовательности, начинающейся на нечетное число, выполняет обрезку всех элементов, начиная с 4-го символа после нечетного числа.

29.

а) для последовательности целых чисел удаляет все четные элементы из последовательности;

б) для строки удаляет все нечетные слова.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить теоретическую часть лабораторной работы.

2. Реализовать индивидуальное задание по вариантам, представленные в теоретических сведениях, сделать скриншоты работающих программ. Написать комментарии.

3. Написать отчет, содержащий:

1. Титульный лист, на котором указывается:

а) полное наименование министерства образование и название учебного заведения;

б) название дисциплины;

в) номер практического занятия;

г) фамилия преподавателя, ведущего занятие;

д) фамилия, имя и номер группы студента;

е) год выполнения лабораторной работы.

2. Индивидуальное задание из раздела «Теоретические сведения» с кодом, комментариями и скриншотами работающих программ.

3. Построение блок-схем.