Министерство образования и науки РФ

федеральное государственное бюджетное
образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный технологический институт
(технический университет)»

Кафедра систем автоматизированного
проектирования и управления

Чистякова Т. Б., Новожилова И. В., Антипин Р. В.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ
на примере объектов химической технологии

Учебное пособие

Допущено учебно-методическим объединением вузов по университетскому политехническому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки «Информатика и вычислительная техника»

Санкт-Петербург
Издательсво СПбГТИ(ТУ)
2012

УДК 681.3.062

Чистякова, Т. Б. Программирование на языке высокого уровня на примере объектов химической технологии : учебное пособие / Т. Б. Чистякова, И. В. Новожилова, Р. В. Антипин. – СПб.: Изд-во СПбГТИ(ТУ), 2012. – 232 с.

Учебное пособие посвящено изложению основных принципов объектно-ориентированного программирования, технологическим средствам разработки программного обеспечения, представлению основных структур программирования, а также краткому обзору языка программирования С++.

В учебное пособие включены задания по изучению языка программирования высокого уровня С++, технологических средств разработки программного обеспечения и созданию программ для объектов химической технологии в инструментальной среде разработки приложений на языке С++.

Учебное пособие предназначено для подготовки бакалавров и магистров техники и технологии по направлению «Информатика и вычислительная техника» и соответствует разделам рабочих программ дисциплин «Программирование» и «Технология разработки программного обеспечения».

Учебное пособие допущено учебно-методическим объединением вузов по университетскому политехническому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки «Информатика и вычислительная техника».

Ил. 13, табл. 20, библиогр. назв. 11

Рецензенты:

Яковлев С. А., профессор кафедры автоматизированных систем обработки информации и управления Санкт-Петербургского государственного электротехнического университета «ЛЭТИ» им. В. И. Ульянова (Ленина), д-р техн. наук, проф.

Викторов В. К., зав. кафедрой информационных систем в химической технологии Санкт-Петербургского государственного технологического института (технического университета), д-р техн. наук, проф.

Утверждено на заседании учебно-методической комиссии факультета информатики и управления

ISBN 978-5-905240-50-8

Рекомендовано к изданию РИСо СПбГТИ(ТУ).

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Для изучения программирования на языках высокого уровня необходимо рассмотреть понятия программирование, язык программирования и парадигмы программирования.

Программирование – процесс и искусство создания компьютерных программ и/или программного обеспечения с помощью языков программирования.

Язык программирования – формальная знаковая система, предназначенная для записи программ. Программа обычно представляет собой некоторый алгоритм в форме, понятной для исполнителя (например, компьютера). Язык программирования определяет набор лексических, синтаксических и семантических правил, используемых при составлении компьютерной программы. Он позволяет программисту точно определить то, на какие события будет реагировать компьютер, как будут храниться и передаваться данные, а также какие именно действия следует выполнять над этими данными при различных обстоятельствах.

Программирование сочетает в себе элементы искусства, фундаментальных наук (прежде всего математики) и инженерии. В более широком смысле программирование – процесс создания программ, то есть разработка программного обеспечения. Программирование включает в себя: 1) анализ; 2) проектирование – разработка алгоритма; 3) кодирование и компиляцию – написание исходного текста программы и преобразование его в исполнимый код с помощью компилятора; 4) тестирование и отладку – выявление и устранение ошибок; 5) сопровождение.

Программы имеют два основных аспекта: набор алгоритмов и набор данных, которыми оперируют. Эти два аспекта оставались неизменными за всю историю программирования, зато отношения между ними (**парадигма программирования**) менялись. **Парадигма программирования** – это парадигма, определяющая стиль программирования, иначе говоря – некоторый цельный набор идей и рекомендаций, определяющих стиль написания программ. Существует достаточно большое количество парадигм программирования, которые являются подклассами других парадигм или имеют пересечения с парадигмами одного уровня, в связи с этим трудно дать четкую классификацию. Можно выделить основные парадигмы верхнего уровня и относящиеся к ним парадигмы.

Основные парадигмы программирования:

– императивное программирование – описывает процесс вычисления в виде инструкций, изменяющих состояние программы. Императивная программа очень похожа на приказы, выражаемые повелительным наклонением в естественных языках, то есть это последовательность команд, которые должен вы-

полнить компьютер. К разновидностям императивного программирования относятся процедурное (основанная на концепции вызова процедуры, содержащей последовательность шагов для выполнения, языки: Basic, Fortran и др.), структурное (представление программы в виде иерархической структуры блоков, языки: C, Pascal, Algol и др.), объектно-ориентированное (основными концепциями являются понятия объектов и классов, языки: Java, C#, C++, Python, Ruby и др.), автоматное программирование (программа или её фрагмент осмысливается как модель какого-либо формального автомата, может быть реализовано на разных языках императивного программирования) и аспектно-ориентированное (основано на идее разделения функциональности, особенно сквозной функциональности, для улучшения разбиения программы на модули, языки: AspectJ, AspectC++, Aspect.NET и др.) программирование;

– декларативное программирование – способ программирования, описывающий не "как решить задачу", а "что нужно получить". В декларативном программировании мы скорее сообщаем компьютеру, что собой представляет проблема, описываем спецификацию программы. Наиболее важными разновидностями декларативного программирования, являются функциональное (рассматривает процесс вычисления как получение значения математически описанной функции, представляет собой последовательность инструкций и не имеют глобального состояния, языки: Haskell, Erlang, Python, Lisp, Ruby и др.) и логическое (основано на выводе новых фактов из данных фактов согласно заданным логическим правилам, языки: Prolog, Haskell, Erlang и др.) программирование.

ООП является в настоящее время основой всей индустрии прикладного программирования благодаря выигрышу в конкурентной борьбе с альтернативными технологиями программирования. В промышленном программировании только в системном программировании позиции ООП еще не очень сильны. С другой стороны, нельзя считать, что ООП во всех случаях является наилучшей из методик программирования.

Одним из наиболее распространенных объектно-ориентированных языков высокого уровня является C++, созданный Бьерном Страуструпом. Язык C++ стал популярным благодаря тому, что он полностью унаследовал и расширил возможности языка C, который до сих пор часто используется в задачах системного программирования. Популярность C++ привела к тому, что “чистый” объектно-ориентированный язык Java был разработан на его основе, с избавлением от процедурного наследия C. На основе языков Java и C++ был разработан другой популярный сейчас “чистый” объектно-ориентированный язык программи-рования C#. Язык высокого уровня C++ является одним из самых “мощных” языков программирования, включает преимущества как объектно-ориентированного программирования, так и более низкоуровнего процедурного программирования. За столь широкие возможности приходится платить более

высокой сложность языка в отличие от “чистых” объектно-ориентированных языков программирования. Понимание основ языка высокого уровня C++ дает возможность простого перехода на другие основанные на нем языки.

Все примеры текстов программ являются переносимыми, их можно использовать с любым компилятором C++, способным реализовать на компьютере консольное приложение.

В данном учебном пособии представлены задания по изучению основных принципов объектно-ориентированного программирования; технологическим средствам разработки программного обеспечения; языка программирования C++; созданию простых программ для объектов химической технологии в инструментальной среде разработки приложений на языке C++.

КОНЦЕПЦИЯ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

ООП основная методика программирования 90-х годов. По мере развития вычислительной техники возникали разные методики программирования. На каждом этапе создавался новый подход, который помогал программистам справляться с растущим усложнением программ. Первые программы создавались посредством ключевых переключателей на передней панели компьютера. Очевидно, что такой способ подходит только для очень небольших программ. Затем был изобретен язык ассемблера, который позволял писать более длинные программы. Следующий шаг был сделан в 1950 году, когда был создан первый язык высокого уровня Fortran.

Используя язык высокого уровня, программисты могли писать программы до нескольких тысяч строк длиной. Для того времени указанный подход к программированию был наиболее перспективным. Однако язык программирования, легко понимаемый в коротких программах, когда дело касалось больших программ, становился нечитабельным (и неуправляемым). Избавление от таких неструктурированных программ пришло после изобретения в 1960 году языков структурного программирования. К ним относятся языки Algol, C и Pascal. Структурное программирование подразумевает точно обозначенные управляющие структуры, программные блоки, отсутствие (или, по крайней мере, минимальное использование) инструкций GOTO, автономные подпрограммы, в которых поддерживается рекурсия и локальные переменные. Сутью структурного программирования является возможность разбиения программы на составляющие ее элементы. Используя структурное программирование, средний программист может создавать и поддерживать программы свыше 50 000 строк длиной.

Хотя структурное программирование, при его использовании для написания умеренно сложных программ, принесло выдающиеся результаты, даже оно оказывалось несостоятельным тогда, когда программа достигала определенной длины. Чтобы написать более сложную программу, необходим был новый подход к программированию. В итоге были разработаны принципы объектно-ориентированного программирования. ООП аккумулирует лучшие идеи, воплощенные в структурном программировании, и сочетает их с мощными новыми концепциями, которые позволяют оптимально организовывать ваши программы. ООП позволяет вам разложить проблему на составные части. Каждая составляющая становится самостоятельным объектом, содержащим свои собственные коды и данные, которые относятся к этому объекту. В этом случае

вся процедура в целом упрощается, и программист получает возможность оперировать с гораздо большими по объему программами.

ООП основано на трех основополагающих концепциях, называемых инкапсуляцией, полиморфизмом и наследованием.

Инкапсуляция *(encapsulation)* – это механизм, который объединяет данные и код, манипулирующий с этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. В ООП код и данные могут быть объединены вместе; в этом случае говорят, что создается так называемый "черный ящик". Когда коды и данные объединяются таким способом, создается *объект (object).* Другими словами, объект – это то, что поддерживает инкапсуляцию.

Внутри объекта коды и данные могут быть *закрытыми (private)* для этого объекта или *открытыми (public).* Закрытые коды или данные доступны только для других частей этого объекта. Таким образом, закрытые коды и данные недоступны для тех частей программы, которые существуют вне объекта. Если коды и данные являются открытыми, то, несмотря на то, что они заданы внутри объекта, они доступны и для других частей программы. Характерной является ситуация, когда открытая часть объекта используется для того, чтобы обеспечить контролируемый интерфейс закрытых элементов объекта.

На самом деле объект является переменной определенного пользователем типа. Может показаться странным, что объект, который объединяет коды и данные, можно рассматривать как переменную. Однако применительно к ООП это именно так. Каждый элемент данных такого типа является составной переменной.

Полиморфизм *(polymorphism)* – это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к ООП, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. Например, для языка С, в котором полиморфизм поддерживается недостаточно, нахождение абсолютной величины числа требует трех различных функций: abs()**,** labs() и fabs(). Эти функции подсчитывают и возвращают абсолютную величину целых, длинных целых и чисел с плавающей точкой соответственно. В объектно-ориентированных языках программирования каждая из этих функций может быть названа abs() и выполнять свою функцию в зависимости от используемого типа данных. Тип данных, который используется при вызове функции, определяет, какая конкретная версия функции действительно выполняется. Это называется *перегрузкой функций (function overloading).*

В более общем смысле, концепцией полиморфизма является идея "один интерфейс, множество методов". Это означает, что можно создать общий ин-

терфейс для группы близких по смыслу действий. Преимуществом полиморфизма является то, что он помогает снижать сложность программ, разрешая использование того же интерфейса для задания *единого класса действий.* Выбор же *конкретного действия,* в зависимости от ситуации, возлагается на компилятор. Вам, как программисту, не нужно делать этот выбор самому. Нужно только помнить и использовать общий интерфейс. Пример из предыдущего абзаца показывает, как, имея три имени для функции определения абсолютной величины числа вместо одного, обычная задача становится более сложной, чем это действительно необходимо.

Полиморфизм может применяться также и к операторам. Фактически во всех языках программирования ограниченно применяется полиморфизм, например, в арифметических операторах. Так, в С, символ + используется для складывания целых, длинных целых, символьных переменных и чисел с плавающей точкой. В этом случае компилятор автоматически определяет, какой тип арифметики требуется. В ООП вы можете применить эту концепцию и к другим, заданным вами, типам данных. Такой тип полиморфизма называется *перегрузкой операторов (operator overloading).*

Ключевым в понимании полиморфизма является то, что он позволяет вам манипулировать объектами различной степени сложности путем создания общего для них стандартного интерфейса для реализации похожих действий.

Наследование *(inheritance)* – это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. Наследование является важным, поскольку оно позволяет поддерживать концепцию *иерархии классов (hierarchical classification).* Применение иерархии классов делает управляемыми большие потоки информации. Например, подумайте об описании жилого дома. Дом – это часть общего класса, называемого строением. С другой стороны, строение – это часть более общего класса – конструкции, который является частью еще более общего класса объектов**.** В каждом случае порожденный класс наследует все, связанные с родителем, качества и добавляет к ним свои собственные определяющие характеристики. Без использования иерархии классов, для каждого объекта пришлось бы задать все характеристики, которые бы исчерпывающе его определяли. Однако при использовании наследования можно описать объект путем определения того общего класса (или классов), к которому он относится, с теми специальными чертами, которые делают объект уникальным.

ПРОСТАЯ ПРОГРАММА НА С++

Самая маленькая программа на С++ выглядит так: main () { }.

В этой программе определяется функция, называемая main, которая не имеет параметров и ничего не делает. Фигурные скобки { и } используются в С++ для группирования операторов. В данном случае они обозначают начало и конец тела (пустого) функции main. В каждой программе на С++ должна быть своя функция main(), и программа начинается с выполнения этой функции.

Обычно программа выдает какие-то результаты. Вот программа, которая выдает приветствие Hello, World!:

```
#include <iostream.h>
int main ()
{
    //Самая простая программа на С++
    cout << "Hello, World!\n";
    /*Выход из функции main и завершение программы*/
}
```

Строка #include <iostream.h> сообщает транслятору, что надо включить в программу описания, необходимые для работы стандартных потоков ввода-вывода, которые находятся в iostream.h. Без этих описаний выражение

```
cout << "Hello, World!\n";
```

не имело бы смысла. Операция << ("выдать") записывает свой второй параметр в первый параметр. В данном случае строка "Hello, World!\n" записывается в стандартный выходной поток cout. Строка – это последовательность символов, заключенная в двойные кавычки. Два символа: обратной дробной черты \ и непосредственно следующий за ним – обозначают некоторый специальный символ. В данном случае \n является символом конца строки (или перевода строки), поэтому он выдается после символов Hello,world! Двойной слеш (//) отмечает начало однострочного комментария, который игнорируется компилятором и служит только для пояснения текста программы. Многострочные комментарии включаются между символами /* и */.

Целое значение, возвращаемое функцией main(), если только оно есть, считается возвращаемым системе значением программы. Если ничего не возвращается, система получит какое-то "мусорное" значение.

1 ЛЕКСИЧЕСКИЕ ОСНОВЫ ЯЗЫКА С++

Язык С++ строг в отношении использования идентификаторов: любой идентификатор объекта, с которым манипулирует программа, должен быть предварительно описан. Прописные и строчные буквы в тексте программы на языке С++ различаются. Это правило касается как ключевых слов и идентификаторов, так и препроцессорных директив.

1.1 Алфавит языка

– В алфавит языка входят: прописные латинские буквы A..Z; строчные латинские буквы a..z; арабские цифры 0..9; символ подчеркивания _ (рассматривается как буква).

Все эти символы используются для образования ключевых слов и имён языка. Имя есть последовательность букв и цифр, начинающихся с буквы и не являющаяся ключевым словом (_ в начале имени ставить не рекомендуется).

– Знаки пунктуации и специальные символы, представленные в таблице 1.

Таблица 1 – Знаки пунктуации и специальные символы

Символы	Наименование	Символы	Наименование
,	запятая	{	открывающая скобка
.	точка	}	закрывающая скобка
;	точка с запятой	<	меньше
:	двоеточие	>	больше
?	знак вопроса	[	открывающая скобка
'	апостроф	]	закрывающая скобка
!	восклицательный знак	#	номер или решетка
|		%	процент
/	слэш	&	амперсанд
\	обратный слэш	^	НЕ-логическое
~	тильда	-	минус
*	звездочка	=	равенство
(	открывающая скобка	"	кавычки
)	закрывающая скобка	+	плюс

– Пробельные символы. К этой группе относятся пробел, символы табуляции, перевода строки, возврат каретки, перевода страницы. Эти символы отделяют друг от друга лексемы языка.

1.2 Типы данных

Все типы данных можно разделить на две категории: скалярные и составные, как показано на рисунке 1.

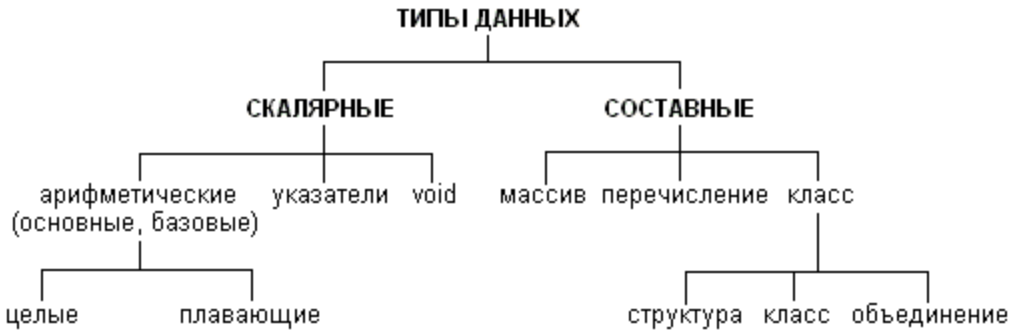


Рисунок 1 – Категории типов данных

Скалярные типы данных представлены в таблице 2.

Таблица 2 – Скалярные типы данных

Тип данных	Размер в байтах	Диапазон	Примеры
char	1	от -128 до 127	'S'
signed char (сокращенно char)	1	от-128 до 127	23
unsigned char	1	от 0 до 255	200
int	2 или 4 (16/32 бита)	как для short int или как для long int	2000
unsigned int (unsigned)	2 или 4 (16/32 бита)	как для unsigned short int или как для unsigned long int	OxFFFF
short int (short)	2	от -32768 до 32767	100
unsigned short int (unsigned short)	2	от 0 до 65535	OxFF
long int (long)	4	от-2147483648L до 2147483647L	OxFFFFL
unsigned long int (unsigned long)	4	от 0L до 4294967295L	123456L
float	4	от 3.4E-38 до 3.4E+38 и от -3.4E-38 до -3.4E+38	2.35 1.3e+l0
double	8	от 1.7E-308 до 1.7E+308 и от-1.7E-308 до-1.7E+308	12.35 1.3e+100
long double	10	от 3.4E-4932 до 3.4E+4932 и от-1.1E-4932 до-3.4E+4932	2.35 8.5 e -3000
void	0	отсутствие значения	

Ключевыми словами, используемыми при объявлении основных типов данных, являются: для целых типов (char, int, short, long, signed, unsigned); для плавающих типов (float, double, long double); для типа void (пустой).

Данные, имя типа которых включает модификатор signed, являются данными со знаком, а данные, имя типа которых содержит модификатор unsigned, – данными без знака. При отсутствии модификатора по умолчанию принимает-

ся модификатор signed. Имя типа, состоящее из одного модификатора, эквивалентно имени, содержащему ключевое слово int.

Тип void не ассоциируется с каким-либо набором данных. Его задача заключается в обеспечении синтаксических конструкций в специфических случаях, например при описании функций, не имеющих параметров или не возвращающих значений, для описания адресных указателей (тип void*) и т.д.

Указатель – это переменная, содержащая адрес некоторого объекта, например, другой переменной. Точнее – адрес первого байта этого объекта. Это дает возможность косвенного доступа к этому объекту через указатель. Пусть x – переменная типа int. Обозначим через px указатель. Унарная операция & выдает адрес объекта, так что оператор int *px = &x; присваивает переменной px адрес переменной x. Говорят, что px "указывает" на x. Операция & применима только к адресным выражениям, так что конструкции вида &(x-1) и &3 незаконны. Унарная операция * называется операцией разадресации или операцией разрешения адреса. Эта операция рассматривает свой операнд как адрес и обращается по этому адресу, чтобы извлечь объект, содержащийся по этому адресу. Заметим, что приоритет унарных операций * и & таков, что эти операции связаны со своими операндами более крепко, чем арифметические операции, так что выражение int y = *px + 2; берет то значение, на которое указывает px, прибавляет 2 и присваивает результат переменной y.

Ссылка в языке C++, является альтернативным именем и используется как другое имя уже существующего объекта. Основные достоинства ссылок проявляются при работе с функциями. Для описания ссылки используется символ &:

тип& имя_ссылки = выражение;

или

тип& имя_ссылки(выражение);

Инициализирующее ссылку выражение должно идентифицировать какой-либо объект, уже существующий в памяти. Примеры определения ссылок:

int i; char ch; int& refi = i; char& refch (ch);

После таких определений получаем две эквивалентные возможности обращения к переменным i и ch: i = 1; ch = 'A'; и refi = 1; refch = 'A';

Если в объявлении имени переменной присутствует модификатор **const**, то объект, с которым сопоставлено данное имя, рассматривается в области существования этого имени как константа. Например: const int i=50; Такие именованные константы в программе изменять нельзя.

Составные типы данных

Составными типами данных являются перечисления, массивы и классы.

Перечисление – это тип данных, который удобно использовать в случае применения в программе переменных и констант, принимающих значения из сравнительно небольшого множества целых чисел, причём таких, что обычно обращение к ним разумнее осуществлять по имени. Может быть и так, что их значение само по себе не важно. Перечислимый тип задается ключевым словом enum: enum имя_перечислимого_типа {список значений};

Примером такого множества констант могут служить названия цветов, названия дней недели или месяцев года, названия шахматных фигур или символические имена арифметических операций языка C++.

Члены перечислений являются константами типов unsigned char или int, в зависимости от их значений и режима компиляции. При использовании перечислителя в выражении его тип всегда преобразуется в int. Пример перечислений:

```
enum chess {king, queen, rook, bishop, knight, p};
enum month {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};
enum operator_CPP{plus='+', minus='-', mult='*', div='/', rem='%'};
```

Если перечислителям явно не присвоено никакого значения, как в chess и mouth, то первый из них получает значение 0, второй - 1, и т.д. Вообще, любой перечислитель по умолчанию имеет значение, на 1 превышающее значение предыдущего, если умолчание не отменяется явной инициализацией.

Массивы состоят из компонентов одинакового типа и заданного количества. Для типа T T[size] является типом "массива из size элементов типа T". Элементы индексируются от 0 до size-1. Многомерные массивы представлены как массивы массивов. Например:

```
float v[3];   // массив из трех чисел с плавающей точкой: v[0], v[1], v[2]
int a[3][2];  // три массива, из двух целых каждый
int t[5][6][7]; // трехмерный массив
```

Массив a можно трактовать как двумерную таблицу (матрицу) с тремя строками и двумя столбцами. Двумерные таблицы хранятся по строкам. Обращения к элементам этой таблицы могут, например, иметь вид a[2][1]. Имя массива является указателем на нулевой элемент. Поскольку массивы часто образуются с помощью указателей, следует помнить, что если v – произвольный массив, то значение его i-го элемента v[i] трактуется как выражение *(v+i). Имея это в виду, легко понять, что выражение *(*(a+i)+j) является элементом a[i][j], а выражение *a – элементом a[0][0].

Структура – это совокупность элементов, каждый из которых может иметь любой тип. И в отличии от массива, который состоит из элементов одинакового типа, структура может состоять из элементов разных типов. Объявление структуры похоже на объявление перечисления, но начинается со слова struct:

struct имя_структуры {список_определений}описатель;

Один из компонентов объявления – либо имя_структуры, либо список_определений, либо описатель – могут быть пропущены. Примеры:

```
 struct                              struct kalendar
 {                                   {
     double x;                           int year;
     float y;                            char moth, day;
 } s1, s2, sm[9];                    } date1, date2;
```

Переменные s1 и s2 объявлены как структуры, каждая из которых состоит из двух компонент x и y. Переменная sm это массив из девяти элементов, каждый из которых является структурой. Каждая из двух переменных date1, date2 состоит из трех компонентов year, moth, day.

С помощью имени структуры kalendar можно объявить структуру такую же как date1 и date2 и указатель на такую структуру.

Использование имен структуры необходимо для описания рекурсивных структур. Ниже рассматривается использование рекурсивных структуры.

```
struct node
{
      int data;
      struct node * next;
} st1_node;
```

Имя структуры node действительно является рекурсивным, так как оно используется в своем собственном описании, т.е. в формализации указателя next. Структуры не могут быть прямо рекурсивными, т.е. структура node не может содержать компоненту, являющуюся структурой node, но любая структура может иметь компоненту, являющуюся указателем на свой тип, как и сделано в приведенном примере.

Доступ к компонентам структуры осуществляется с помощью указания имени структуры и следующего через точку имени выделенного компонента, например s1.x, s2.y, date3.day, sm[3].x. В последнем примере происходит обращение к компоненте x четвертого элемента массива sm.

Другим способом обращения к элементам структур является использования указателей на структуры. В этом случае вместо точки используются знаки –>, а вместо имени структуры указатель на структуру. Например adrdate–>day, adrdate–>moth. К элементам структур можно применять операции как к простым переменным соответствующего типа.

Значением переменной типа структура является совокупность значений всех ее элементов и такое значение может быть присвоено другой такой же структуре или передано функции как параметр.

Объединение подобно структуре, однако в каждый момент времени может использоваться только один из элементов объединения. Синтаксис объявления объединения полностью совпадает с синтаксисом объявления структуры за исключением первого слова. При объявлении объединения используется ключевое слово union:

union имя_объединения { список_определений }описатель;

Главной особенностью объединения является то, что для всех элементов списка_определений выделяется одна и та же область памяти, т.е. они перекрываются. Хотя доступ к этой области памяти возможен с использованием любого из элементов, элемент для этой цели должен выбираться так, чтобы полученный результат не был бессмысленным. Доступ к элементам объединения осуществляется тем же способом, что и к структурам. Пример:

```
union tu
{
        long a; int b[2]; char c[4];
} u1, *au, mu[5];
```

При объявлениях возможны обращения к элементам u1.a, u1.c[2], ua->b[1], mu[2].c[3]. Объединение применяется для следующих целей: инициализации используемого объекта памяти, если в каждый момент времени только один объект из многих является активным; интерпретации основного представления объекта одного типа, как если бы этому объекту был присвоен другой тип. Память, которая соответствует переменной типа объединения, определяется величиной, необходимой для размещения наиболее длинного элемента объединения. Когда используется элемент меньшей длины, то переменная типа объединения может содержать неиспользуемую память.

Элементом структуры может быть **битовое поле**, обеспечивающее доступ к отдельным битам памяти. Вне структур битовые поля объявлять нельзя. Нельзя также организовывать массивы битовых полей и нельзя применять к полям операцию определения адреса. В общем случае тип структуры с битовым полем задается в следующем виде:

```
struct
{
        unsigned идентификатор_1 : длинна_поля_1;
        unsigned идентификатор_2 : длинна_поля_2;
};
```

длинна – поля задается целым выражением или константой. Эта константа определяет число битов, отведенное соответствующему полю. Поле нулевой длинны обозначает выравнивание на границу следующего слова.

1.3 Инициализация данных

При объявлении любой переменной ей можно присвоить некоторое начальное значение, присоединяя инициализатор к описателю. Инициализатор начинается со знака "=", после которого следует константное выражение или список константных выражений в фигурных скобках.

Формат 1: = константное выражение;

Формат 2: = { список константных выражений };

Формат 1 используется при инициализации переменных основных типов и указателей, а формат 2 – при инициализации составных объектов: массивов, структур и более сложных объектов. Примеры:

```
char tol = 'N';
```

Переменная tol инициализируется символом 'N'.

```
const long megabute = (1024 * 1024);
```

Именованная константа megabute инициализируется константным выражением, после чего она не может быть изменена.

```
int b[2][2] = {1,2,3,4}; или int b[2][2] = {{1,2},{3,4}};
```

Инициализируется двухмерный массив b целых величин, элементам массива присваиваются значения из списка.

```
int b[][2] = {{1,2},{3,4}};
```

При инициализации массива можно опустить его первую размерность, в таком случае число элементов определяется по числу используемых инициализаторов.

```
int k[3][2] = {{1,2},{3,},};
```

Если при инициализации указано меньше значений чем элементов массива, то оставшиеся элементы инициализируются 0, т.е. при описании элементы первой строки получат значения 1 и 2, второй 3 и 0, третьей - 0 и 0.

При инициализации составных объектов, нужно быть внимательным к использованию скобок и списков инициализаторов. Пример:

```
struct complex
{
        double real,imag;
} comp [2][3] ={ { {1,1}, {2,3}, {4,5} },{ {6,7}, {8,9}, {10,11} } };
```

В данном примере инициализируется массив структур comp из двух строк и трех столбцов, где каждая структура состоит из двух элементов real и imag.

При инициализации объединения инициализируется первый элемент объединения в соответствии с его типом. Пример:

```
union tab
{
        unsigned char name[10];
        int tab1;
} pers = {'А','Б','В','Г','Д'};
```

Инициализируется переменная pers.name, и так как это массив, для его инициализации требуется список значений в фигурных скобках. Первые пять элементов массива инициализируются значениями из списка, остальные нулями.

Инициализацию массива символов можно выполнить путем использования строкового литерала.

```
char stroka[ ] = "привет";
```

Инициализируется массив символов из 7 элементов, последним элементом (седьмым) будет символ '\0', которым завершаются все строковые литералы.

Следущее объявление инициализирует переменную stroka строкой состоящий из семи элементов.

```
char stroka[5] = "привет";
```

В переменную stroka попадают первые пять элементов литерала, а символы 'т' и '\0' отбрасываются.

2 ВЫРАЖЕНИЯ

Выражение – это последовательность операндов, разделителей (круглых скобок) и знаков операций, задающая вычисление. Язык C/C++ в значительной степени является языком выражений. В общем случае операндом может быть выражение. Значение выражения зависит от расположения знаков операций и круглых скобок в выражении, а также от приоритета выполнения операций. Операции с более высоким приоритетом выполняются до операций с более низким приоритетом, если часть выражения заключена в скобки, то вначале вычисляется выражение внутри скобок.

Принципы вычисления значений выражений вытекают из семантики и приоритетов содержащихся в них операций, а также из видов использованных аргументов. Если аргумент находится в границах более одной операции с одинаковыми приоритетами, то операции являются связанными. Большинство операций связывают аргументы слева направо. Исключение составляют только операции с одним аргументом и операции присваивания, а также трехаргументная условная операция. Порядок применения операций к аргументам зависит от принятых в языке приоритетов и связей операций. Операции более высокого приоритета выполняются в первую очередь. Если аргумент касается двух операций одинакового приоритета, то очередность выполнения операций определяется по связыванию. Для изменения порядка выполнения операций используют круглые скобки. Таким образом, выражение x = y = z означает x = (y = z), а x + y + z означает (x + y) + z, согласно соответствующим правилам связывания.

Связывание и упорядочение операций в очередности уменьшения приоритетов представлены в таблице 3. Class_name означает имя класса, namespace-name – имя пространства имен, qualified-name – квалифицированное имя, name – какое-то имя, member – имя члена, object – выражение, дающее объект класса, pointer – выражение, дающее указатель (pointer-to-member – указатель на член), expr – некое выражение (expr-list – список выражений) и lvalue – выражение, обозначающее неконстантный объект, type – может быть полностью произвольным именем типа.

В каждом блоке расположены операции с одинаковым приоритетом. Операции в блоке, расположенном выше, имеют более высокий приоритет. Унарные операции и операции присваивания правоассоциативны, а все остальные левоассоциативны. Несколько грамматически правил нельзя выразить в терминах приоритетов (называемых также силой связывания) и ассоциативности. Например, a=b<c ? d=e: f=g означает a=((b<c)? (d=e): (f=g)).

Таблица 3 – Связывание и упорядочение операций в очередности уменьшения приоритетов

Операции	
разрешения области видимости	class_name :: member
разрешения области видимости	namespace-name :: member
глобально	:: name
глобально	:: qualified-name
выбор члена	object . member
выбор члена	pointer -> member
индексирование	pointer [expr]
вызов функции	expr (expr_list)
конструирование значения	type (expr_list)
постфиксный инкремент	lvalue ++
постфиксный декремент	lvalue --
идентификация типа	typeid (type)
идентификация типа во время выполнения	typeid (expr)
преобразование с проверкой во время выполнения	dynamic_cast< type >(expr)
преобразование с проверкой во время компиляции	static_cast< type >(expr)
преобразование без проверки	reinterpret_cast< type >(expr)
константное преобразование	const_cast< type >(expr)
размер объекта	sizeof expr
размер типа	sizeof (type)
префексный инкремент	++ lvalue
префексный декремент	-- lvalue
дополнение	~ lvalue
отрицание	! expr
унарный минус	- expr
унарный плюс	+ expr
адрес	& lvalue
разыменование	* expr
создать (выделить память)	new type
создать (выделить память и инициализировать)	new type (expr-list)
создать (разместить)	new (expr-list) type
создать (разместить и инициализировать)	new (expr-list) type (expr-list)
уничтожить (освободить память)	delete pointer
уничтожить массив	delete[] pointer
приведение (преобразование типа)	(type) expr
выбор члена	object .* pointer-to-member
выбор члена	pointer ->* pointer-to-member
умножение	expr * expr
деление	expr / expr
остаток от деления (деление по модулю)	expr % expr
сложение	expr + expr
вычитание	expr - expr
сдвиг влево	expr << expr
сдвиг вправо	expr >> expr

Продолжение таблицы 3

Операции	
меньше меньше или равно больше больше или равно	expr < expr expr <= expr expr > expr expr >= expr
равно не равно	expr == expr expr != expr
побитовое И (AND)	expr & expr
побитовое исключающее ИЛИ (XOR)	expr ^ expr
побитовое ИЛИ (OR)	expr | expr
логические И (AND)	expr && expr
логические ИЛИ (OR)	expr || expr
условное выражение	expr ? expr : expr
простое присваивание умножение и присваивание деление и присваивание остаток от присваивания сложение и присваивание вычитание и присваивание сдвиг влево и присваивание сдвиг вправо и присваивание И и присваивание ИЛИ и присваивание исключающее ИЛИ и присваивание	lvalue = expr lvalue *= expr lvalue /= expr lvalue %= expr lvalue += expr lvalue -= expr lvalue <<= expr lvalue >>= expr lvalue &= expr lvalue |= expr lvalue ^= expr
генерация исключения	throw expr
запятая (последовательность)	expr, expr

2.1 Арифметические операции

Арифметические операции служат для описания арифметических действий: замена знака (-), инкремент (++), декремент --), сложение (+), вычитание (-), умножение (*), деление (/), а также вычисление остатка от деления (%).

Операторы замены знака, инкремента и декремента – одноаргументные (унарные), а их использование приводит соответственно к изменению знака аргумента, добавлению к аргументу единицы и вычитанию из аргумента единицы. Если аргумент операции замены знака имеет тип unsigned, то замена знака реализуется путем вычитания аргумента из числа 2^n, где n – количество битов, используемых для представления данных типа int. Инкремент и декремент означают соответственно увеличение и уменьшение на единицу и относятся только к аргументу, имеющему вид lvalue. В случае преинкремента ++arg результатом операции является arg+1, в случае постинкремента arg++ результатом операции является arg. Аналогичные правила относятся к декременту.

Остальные арифметические операции являются двухаргументными и служат для выполнения основных арифметических действий, а также для вычисления остатка от деления первого аргумента на второй. Деление целых аргументов приводит к отбрасыванию дробной части результата. В случае вычисления остатка от деления знак результата совпадает со знаком делимого.

2.2 Операции сравнения

Операции сравнения служат для описания следующих действий: проверка на равенство (==), неравенство (!=), меньше (<), больше (>), меньше или равно (<=) и больше или равно (>=). Результат сравнения имеет тип int и принимает значение 0 при невыполнении условия сравнения (ложно) и 1 – при выполнении условия (истинно). Если какой-либо аргумент сравнения имеет тип char, то он подвергается преобразованию в тип int.

Если какой-либо аргумент является указателем, то он может сравниваться только с аргументом того же типа или с литералом 0 (NULL), представляющим пустой указатель. Например:

```
float arr[4];
float *ptr = arr+2;
int flag = ptr != arr;
```

Переменной flag будет присвоено начальное значение 1.

2.3 Логические операции

Логические операции служат для описания следующих действий: инверсия (!), конъюнкция (&&) и дизъюнкция (И)**.** Первый из перечисленных опера-

торов является одноаргументным, а два других – двухаргументными. Во всех случаях результат операции будет иметь значение 0 (условие ложно) или 1 (условие истинно). Результатом инверсии будет true, если аргумент имел значение false и наоборот. Результатом конъюнкции будет true, если аргументы имеют значения, отличные от нуля, или false – в противном случае. Результатом дизъюнкции является true, если, по крайней мере, один аргумент отличен от нуля, или false – в противном случае.

2.4 Битовые операции

К целочисленным операндам применяются следующие битовые операции: отрицание битов (~), конъюнкция битов (&), исключающее ИЛИ (^) и дизъюнкция (|) битов. Первая из перечисленных операций – одноаргументная, а остальные – двухаргументные. Результат операции также имеет целый тип. Результатом отрицания битов является единичное дополнение аргумента. Результаты конъюнкции, исключающего ИЛИ и дизъюнкции получаются путем поразрядных (побитовых) действий. Результатом конъюнкции является логическое произведение битов, результатом исключающего ИЛИ – сумма битов по модулю 2, а дизъюнкции – логическая сумма битов. Примеры:

а)отрицание битов:

```
int a = -2;          /* 1111...1110*/
int res = ~a;        /*0000...0001*/ переменной res присваивается начальное значение 1;
```

б) конъюнкция битов:

```
int a - 10;          /*0000...1010*/
int b = 12;          /*0000...1100*/
int res = a & b;  /*0000...1000*/ переменной res присваивается начальное значение 8;
```

в) исключающее ИЛИ:

```
int a = 10;          /*0000...1010*/
int b = 12;          /*0000...1100*/
int res = a ^ b;     /*0000...0110*/
```

переменной res присваивается начальное значение 6;

г)дизъюнкция битов:

```
int a = 10;          /*0000...1010*/
int b = 12;          /*0000...1100*/
int res = a | b;     /*0000...1110*/
```

переменной res присваивается начальное значение 14.

К целочисленным операндам применяются также двухаргументные операции сдвига: для сдвига битов аргумента влево <<) или вправо (>>) на задан-

ное число позиций. Левый аргумент подлежит сдвигу, а правый – определяет количество битов, на которое необходимо сдвинуть. Оба аргумента должны быть целыми, а результат – того же типа, что и левый аргумент.

Результат операции не определен, если правый аргумент является отрицательным или превышает количество битов, представляющих левый аргумент. Во время сдвига влево происходит дополнение нулями справа. Во время сдвига вправо дополнение нулями происходит только в том случае, когда левый аргумент имеет тип unsigned. В остальных случаях определение вида сдвига – логический (дополнение нулями) или арифметический (дополнение битом знака) – производит система.

Операции сдвига C/C++ компилируются в процессорные команды арифметического сдвига. Примеры:

а) сдвиг вправо:

```
int a = 15;          /*0000...1111 */
int res = a >> 2;  /*0000...0011*/
```

переменной res будет присвоено начальное значение 3;

б) сдвиг влево:

```
int a = -2;              /*1111...1110*/
int res = a << -a; /*1111...1000*/
```

переменной res будет присвоено начальное значение -8.

2.5 Операции присваивания

Операция присваивания может быть простой и составной. Простая – двухаргументная операция вида lvalue = expr;

В этой операции левая сторона является lvalue-выражением, а правая – expr-выражением, тип которого всегда преобразуется в тип левой стороны. Выполнение операции присваивания приводит к присваиванию представленной lvalue-выражением переменной значения expr-выражения, стоящего с правой стороны оператора присваивания. Результатом операции является значение правой стороны оператора, что позволяет составлять цепочки присваиваний:

```
x = y = z;
```

Кроме простого присваивания, в языке C/C++ есть составное присваивание вида el op= e2; где el и e2 – выражения, а op= – один из следующих двухаргументных операторов: +=, -=, *=, /=, %=, »=, «=, &=, ^=, |=. Составное присваивание выполняется как присваивание (el) = ((el) op (e2));

Примеры:

```
i += i;   // равносильно  i =     i      4      i;
i /= 4;   // равносильно  i =     i      /      4;
i  *= 4; // равносильно  i =     i      *      4;
```

i -= i; // равносильно i = i - i;

2.6 Тернарная условная операция

В языке C/C++ имеется одна тернарная операция – условная операция, которая имеет следующий формат: el ? e2 : e3 где el, e2 и e3 – выражения.

Выполнение условной операции начинается с анализа значения выражения el. Если оно отлично от нуля (условие истинно), то вычисляется выражение e2, значение которого и становится результатом условной операции. Если значение условия (выражение el) равно нулю (ложно), то в качестве значения всего условного выражения вычисляется e3. Всегда вычисляется только одно из двух выражений, разделенных двоеточием.

Поскольку результат условной операции не представляет собой lvalue-выражение, то такая операция не может выступать в качестве левой стороны оператора присваивания. Пример: max = (d<=b) ? b : d; Переменной max присваивается максимальное значение переменных d и b.

2.7 Операция последовательного вычисления

Операция последовательного вычисления обозначается запятой (,) и вычисляет свои операнды слева направо. При выполнении операции последовательного вычисления, преобразование типов не производится. Операнды могут быть любых типов. Результат операции имеет значение и тип второго операнда. Эту операцию можно использовать для вычисления двух и более выражений там, где по синтаксису допустимо только одно выражение.

2.8 Операция получения размера

Операция sizeof вычисляет размер памяти занимаемой операндом. Операция sizeof имеет следующий формат: sizeof (выражение).

В качестве выражения может быть использован любой идентификатор, либо имя типа. Нельзя использовать тип void, а идентификатор не может быть именем битового поля или функции. Если в качестве выражения указанно имя массива, то результатом является размер всего массива (т.е. произведение числа элементов на длину типа), а не размер указателя, соответствующего идентификатору массива.

Когда sizeof применяются к имени типа структуры или объединения или к идентификатору, имеющему тип структуры или объединения, то результатом является фактический размер структуры или объединения, который может включать участки памяти, используемые для выравнивания элементов структуры или объединения. Таким образом, этот результат может не соответствовать размеру, получаемому путем сложения размеров элементов структуры.

2.9 Операция разрешения области видимости

Операция разрешения области видимости (::) имеет две формы: унарную и бинарную. Унарная форма позволяет обратиться к глобальной переменной, если видима локальная переменная с тем же именем ::name, бинарная форма применяется для доступа к компоненту класса class-name::member.

2.10 Операции для работы с динамической памятью

Операции new, new[], delete и delete[] введены в язык C++ с целью решения одной из задач управления памятью, а именно ее динамического распределения. Две операции new предназначены для выделения участков свободной памяти и размещения в них переменных, а в случае new[] – массивов. Продолжительность существования созданных таким образом (динамических) переменных – от точки создания до конца программы или до явного освобождения соответствующего участка памяти применением операций delete (delete[] – для удаления массивов). Время жизни обычных переменных жестко связано со структурой кода программы. Таким образом, механизм динамического распределения памяти предоставляет альтернативный способ создания, использования и удаления переменных.

Выражение, содержащее операцию new, имеет следующий вид:

указатель_на_тип = new имя_типа (инициализатор);

Инициализатор – это необязательное инициализирующее выражение, которое может использоваться для всех типов, кроме массивов. При выполнении оператора int *ip = new int; создаются 2 объекта: динамический безымянный объект и указатель на него с именем ip, значением которого является адрес динамического объекта. Можно создать и другой указатель на тот же динамический объект: int *other = ip; как показано на рисунке 2.

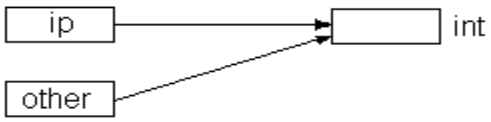


Рисунок 2 – Создание указателя на динамический объект

Если указателю ip присвоить другое значение, то можно потерять доступ к динамическому объекту, что схематично приведено на рисунке 3:

```
int *ip = new int;
int i = 0;
ip = &i;
```

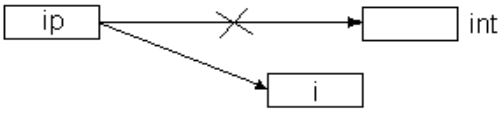


Рисунок 3 – Потеря доступа к динамическому объекту

В результате динамический объект по-прежнему будет существовать, но обратиться к нему уже нельзя. Такие объекты называются мусором. При выделении памяти объект можно инициализировать: int *ip = new int(3);

Можно динамически распределить память и под массив, что показано на рисунке 4: double *mas = new double [50];

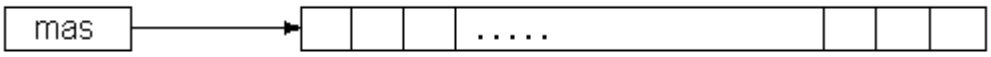


Рисунок 4 – Распределение памяти под массив

В случае успешного завершения операция new возвращает указатель со значением, отличным от нуля. Результат операции, равный 0, т.е. нулевому указателю NULL, говорит о том, что не найден непрерывный свободный фрагмент памяти нужного размера.

Операция delete освобождает для дальнейшего использования в программе участок памяти, ранее выделенной операцией new:

```
delete ip; // Удаляет динамический объект типа int, если было ip = new int;
delete[] mas; // удаляет динамический массив длиной 50, если было
              // double *mas = new double[50];
```

Совершенно безопасно применять операцию к указателю NULL. Результат же повторного применения операции delete к одному и тому же указателю не определен.

2.11 Приведение типов

Лучшая практика по приведению типов: не делать этого. Потому что, если в программе потребовалось приведение типов, значит в этой программе с большой долей вероятности что-то неладно. Для довольно редких ситуаций, когда это все-таки действительно нужно, есть четыре способа приведения типов. Старый, оставшийся со времен C, но все еще работающий, лучше не использовать вовсе. Хотя бы потому, что конструкцию вида (type) expr очень сложно обнаружить при чтении кода программы.

Для приведения типов в C++ используются четыре операции:

- const_cast – самое простое приведение типов, убирает модификаторы const и volatile. volatile встречается не очень часто, так что более известно как приведение типов, предназначенное для убирания const. Если приведение типов не удалось, выдается ошибка на этапе компиляции. Если снимать const с переменной, которая изначально была const, то дальнейшее её использование приведёт к неопределенному поведению. Пример:

```
int i;
const int * pi = &i; // *pi имеет тип const int,
                     // но pi указывает на int, который константным не является
```

int* j = const_cast<int *> (pi);

- static_cast – может быть использован для приведения одного типа к другому. Если это встроенные типы, то будут использованы встроенные в C++ правила их приведения. Если это типы, определенные программистом, то будут использованы правила приведения, определенные программистом. static_cast между указателями корректно, только если один из указателей – это указатель на void или если это приведение между объектами классов, где один класс является наследником другого. То есть для приведения к какому-либо типу от void*, который возвращает malloc, следует использовать static_cast. Пример:

int *p = static_cast<int*>(malloc(100));

- dynamic_cast – безопасное приведение по иерархии наследования, в том числе и для виртуального наследования, имеет смысл только для классов. dynamic_cast<derv_class *>(base_class_ptr_expr) Используется RTTI (Runtime Type Information), чтобы привести один указатель на объект класса к другому указателю на объект класса. Классы должны быть полиморфными, то есть в базовом классе должна быть хотя бы одна виртуальная функция. Если эти условие не соблюдено, ошибка возникнет на этапе компиляции. Если приведение невозможно, то об этом станет ясно только на этапе выполнения программы и будет возвращен NULL.

dynamic_cast<derv_class&>(base_class_ref_expr) Работа со ссылками происходит почти как с указателями, но в случае ошибки во время исполнения будет выброшено исключение bad_cast.

- reinterpret_cast – самое опасное приведение типов. Результат может быть некорректным, никаких проверок не делается. Считается, что вы лучше компилятора знаете как на самом деле обстоят дела, а он тихо подчиняется. Не может быть приведено одно значение к другому значению. Обычно используется, чтобы привести указатель к указателю, указатель к целому, целое к указателю. Умеет также работать со ссылками. Пример:

reinterpret_cast<whatever *>(some *)

reinterpret_cast<integer_expression>(some *)

reinterpret_cast<whatever *>(integer_expression)

Чтобы использовать reinterpret_cast, нужны веские причины. Используется, например, при приведении указателей на функции.

Что делает приведение типов в стиле C: пытается использовать static_cast, если не получается, использует reinterpret_cast. Далее, если нужно, использует const_cast.

3 ОПЕРАТОРЫ

Выполняемая часть программы на любом алгоритмическом языке состоит из операторов, которые определяют элементарные шаги и логику реализованного алгоритма. Операторы языка C++ делятся на следующие группы:

- условные операторы, к которым относятся оператор условия if и оператор выбора switch;
- операторы цикла (for,while,do while);
- операторы перехода (break, continue, return, goto);
- другие операторы (оператор выражение, пустой оператор).

Операторы в программе могут объединяться в составные операторы с помощью фигурных скобок {}. Любой оператор в программе может быть помечен меткой, состоящей из имени и следующего за ним двоеточия.

Все операторы языка, кроме составных операторов, заканчиваются точкой с запятой (;).

3.1 Оператор выражение

Любое выражение, которое заканчивается точкой с запятой, является оператором выражением. Выполнение оператора выражения заключается в вычислении выражения. Полученное значение выражения никак не используется, поэтому, как правило, такие выражения вызывают побочные эффекты. Заметим, что вызов функции не возвращающей значения можно осуществить только при помощи оператора выражения. Пример:

```
clrscr(); /* вызов функции */
a=b+c;  /* выражения */
i++;      /* с присваиванием */
```

3.2 Оператор безусловного перехода

Использование оператора безусловного перехода goto в практике программирования на языке C++ настоятельно не рекомендуется, так как он затрудняет понимание программ и возможность их модификаций. Формат этого оператора следующий: goto метка;

Оператор goto передает управление на оператор, помеченный меткой метка. Помеченный оператор должен находиться в той же функции, что и оператор goto, а метка должна быть уникальной, т.е. нельзя помечать одной меткой более одного оператора.

Любой оператор в составном операторе может иметь свою метку. Используя оператор goto, можно передавать управление внутрь составного оператора, условного оператора, операторов цикла и оператора переключателя . Но в

таких случаях следует быть осторожным так как при таком входе инициализация объявленных переменных не выполняется и их значения будут не определены, неопределенными также останутся и переменные определяемые в заголовке цикла for.

3.3 Условный оператор

Условный оператор if позволяет разветвить вычислительный процесс на два варианта в зависимости от значения некоторого условия. В состав этого оператора могут входить один или два любых других оператора. Формат оператора: if (выражение) оператор_1; [else оператор_2;]

Выполнение оператора if начинается с вычисления выражения. Далее выполнение осуществляется по следующей схеме:

- если выражение истинно (т.е. отлично от 0), то выполняется оператор_1;
- если выражение ложно (т.е. равно 0),то выполняется оператор_2;
- если выражение ложно и отсутствует конструкция со словом else (в квадратные скобки заключена необязательная конструкция), то выполнение оператора if завершается.

После выполнения оператора if значение передается на следующий оператор программы, если последовательность выполнения операторов программы не будет принудительно нарушена использованием операторов перехода. Пример:

```
if ( i<j ) { j = 0; i--; }
else { j = i-3; i++; }
```

Этот пример иллюстрирует также и тот факт, что на месте оператора_1, так же как и на месте оператора_2 могут находиться сложные конструкции.

Допускается использование вложенных операторов if. Оператор if может быть включен в конструкцию if или в конструкцию else другого оператора if. Чтобы сделать программу более читабельной, рекомендуется группировать операторы и конструкции во вложенных операторах if, используя фигурные скобки. Если же фигурные скобки опущены, то компилятор связывает каждое ключевое слово else с наиболее близким if, для которого нет else.

```
int t=2, b=7, r=3;
if (t>b){
    if (b<r) r=b;
} else r=t;
```

Пример вложенных операторов if:

```
char ZNAC;
int x,y,z;
/*…*/
```

```
if (ZNAC == '-') x = y - z;
else if (ZNAC == '+') x = y + z;
else if (ZNAC == '*') x = y * z;
else if (ZNAC == '/') x = y / z;
```

Конструкции, использующие вложенные операторы if, являются довольно громоздкими и не всегда достаточно надежными. Другим способом организации выбора из множества различных вариантов является использование специального оператора выбора switch.

3.4 Пустой и составной операторы

Составной оператор имеет вид последовательности операторов, заключенной в фигурные скобки: { оператор1; оператор2; . . . }

Наличие в языке составного оператора значительно расширяет возможности синтаксических конструкций: везде, где синтаксис позволяет использовать оператор, допустимо использование и составного оператора.

Операторы, содержащиеся в списке, выполняются один раз в порядке их следования. В любом месте составного оператора могут находиться описания переменных. На этапе выполнения переменным присваиваются начальные значения с помощью инициирующих выражений. Область действия этих описаний ограничивается составным оператором, в котором они локализованы. «Время жизни» таких (автоматических) переменных – от точки описания до конца составного оператора.

Специальным случаем оператора является пустой оператор. Он обозначается символом «точка с запятой», перед которым нет никакого выражения или незавершенного разделителем (;) оператора. Пустой оператор не предусматривает выполнения каких-либо действий. Он используется там, где синтаксис языка требует присутствия оператора, а по смыслу программы никакие действия не должны выполняться (например, в качестве тела цикла, когда все циклически выполняемые действия определены в заголовке оператора цикла). Другое его применение связано с необходимостью в ряде случаев помечать пустое действие, например для реализации перехода к концу блока операторов:

```
{
    if ( errorcode > 0 ) goto exit;
    /*операторы*/
    exit:;
}
```

3.5 Оператор-переключатель

Оператор переключатель switch предназначен для организации выбора из множества различных вариантов. Формат оператора следующий:

```
switch ( выражение )
{
        [ case константное_выражение_1]: [ список_операторов_1]
        [ case константное_выражение_2]: [ список_операторов_2]
        [ default: [ список_операторов_default ] ]
}
```

Выражение, следующее за ключевым словом switch в круглых скобках, может быть любым выражением, допустимыми в языке C++, значение которого должно быть целым. Можно использовать явное приведение к целому типу, однако необходимо помнить о тех ограничениях и рекомендациях о которых говорилось выше.

Значение этого выражения является ключевым для выбора из нескольких вариантов. Тело оператора switch состоит из нескольких операторов, помеченных ключевым словом case с последующим константным выражением. Использование целого константного выражения является существенным недостатком присущим рассмотренному оператору.

Так как константное выражение вычисляется во время трансляции, оно не может содержать переменные или вызовы функций. Обычно в качестве константного выражения используются целые или символьные константы.

Все константные выражения в операторе switch должны быть уникальны. Кроме операторов, помеченных ключевым словом case, один из фрагментов может быть помечен ключевым словом default. Списки операторов, следующих за двоеточиями, могут быть пустыми, либо содержать один или более операторов. Причем эти списки не требуется заключать в фигурные скобки.

Схема выполнения оператора switch следующая:

- вычисляется выражение в круглых скобках;
- вычисленные значения последовательно сравниваются с константными_выражениями следующими за ключевыми словами case;
- если одно из константных_выражений совпадает со значением выражения, то управление передается на оператор, помеченный соответствующим ключевым словом case;
- если ни одно из константных_выражений не равно выражению, то управление передается на оператор, помеченный ключевым словом default, а в случае его отсутствия управление передается на следующий после switch оператор.

Особенность использования оператора switch: конструкция со словом default может быть не последней в теле оператора switch. Ключевые слова case

и default в теле оператора switch существенны только при начальной проверке, когда определяется начальная точка выполнения тела оператора switch. Все операторы, между начальным оператором и концом тела, выполняются вне зависимости от ключевых слов, если только какой-то из операторов не передаст управления из тела оператора switch. Таким образом, программист должен сам позаботиться о выходе из case, если это необходимо. Чаще всего для этого используется оператор break.

Для того чтобы выполнить одни и те же действия для различных значений выражения, можно пометить один и тот же оператор несколькими ключевыми словами case. Пример:

```
int i=2;
switch (i)
{
    case 1: i += 2;
    case 2: i *= 3;
    case 0: i /= 2;
    case 4: i -= 5;
    default: ;
}
```

Выполнение оператора switch начинается с оператора, помеченного case 2. Таким образом, переменная i получает значение, равное 6, далее выполняется оператор, помеченный ключевым словом case 0, а затем case 4, переменная i примет значение 3, а затем значение – 2. Оператор, помеченный ключевым словом default, не изменяет значения переменной.

Пример, иллюстрирующий использование вложенных операторов if, можно теперь записать с использованием оператора switch.

```
char ZNAC;
int x,y,z;
switch (ZNAC)
{
      case '+': x = y + z; break;
      case '-': x = y - z; break;
      case '*': x = y * z; break;
      case '/': x = u / z; break;
      default : ;
}
```

Использование оператора break позволяет в необходимый момент прервать последовательность выполняемых операторов в теле оператора switch и передать управление оператору, следующему за switch.

3.6 Операторы цикла

Существуют три формы оператора цикла: for, while, do while.

Оператор **for** это наиболее общий способ организации цикла. Он имеет следующий формат: for (выражение_1 ; выражение_2 ; выражение_3) тело;

Выражение_1 обычно используется для установления начального значения переменных, управляющих циклом. Выражение_2 это выражение, определяющее условие, при котором тело цикла будет выполняться. Выражение_3 определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла. В качестве тела может быть использован любой оператор, в том числе пустой или составной.

Любое из выражений, а также все сразу, могут быть опущены, при этом разделяющие их символы ; пропускать нельзя. Схема выполнения оператора for:

1) вычисляется выражение_1;
2) вычисляется выражение_2;
3) если значения выражения_2 отлично от нуля (истина), выполняется тело цикла, вычисляется выражение_3 и осуществляется переход к пункту 2, если выражение_2 равно нулю (ложь), выполнение оператора for завершается и управление передается на оператор, следующий за оператором for. При отсутствии выражения_2 оно подразумевается истинным.

Существенно то, что проверка условия всегда выполняется в начале цикла. Это значит, что тело цикла может ни разу не выполниться, если условие выполнения сразу будет ложным. Пример:

```
int main ( )
{
      int i,b;
      for ( i=1; i<10; i++) b=i*i;
      return 0;
}
```

В этом примере вычисляются квадраты чисел от 1 до 9. Некоторые варианты использования оператора for повышают его гибкость за счет возможности использования нескольких переменных, управляющих циклом. Пример:

```
int main ( )
{
      int top, bot;
      char string[100], temp;
      for ( top=0, bot=98 ; top<bot ; top++, bot--){
            temp = string[top];
            string[top] = string[bot];
            string[bot] = temp;
```

```
    }
    return 0;
}
```

В этом примере, реализующем запись строки символов в обратном порядке, для управления циклом используются две переменные top и bot, значения которых движутся навстречу друг другу. На месте выражения_1 и выражения_3 здесь используются несколько выражений, записанных через запятую, и выполняемых последовательно.

Оператор цикла **while** называется циклом с предусловием и имеет следующий формат: while (выражение) тело;

В качестве выражения допускается использовать любое выражение языка C++, а в качестве тела любой оператор, в том числе пустой или составной. Схема выполнения оператора while следующая:

1) вычисляется выражение;

2) если выражение ложно, то выполнение оператора while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполняется тело оператора while;

3) процесс повторяется с пункта 1.

Оператор цикла вида: for(выражение-1; выражение-2; выражение-3) тело; может быть заменен оператором while следующим образом:

```
выражение-1;
while ( выражение-2 )
{
    операторы;
    выражение-3;
}
```

Так же как и при выполнении оператора for, в операторе while вначале происходит проверка условия. Поэтому оператор while удобно использовать в ситуациях, когда тело оператора не всегда нужно выполнять.

Оператор цикла **do while** называется оператором цикла с постусловием и используется в тех случаях, когда необходимо выполнить тело цикла хотя бы один раз. Формат оператора имеет следующий вид: do тело; while (выражение);

Схема выполнения оператора do while :

1) выполняется тело цикла (которое может быть составным оператором).

2) вычисляется выражение.

3) если выражение ложно, то выполнение оператора do while заканчивается и выполняется следующий по порядку оператор. Если выражение истинно, то выполнение оператора продолжается с пункта 1.

Чтобы прервать выполнение цикла до того, как условие станет ложным, можно использовать оператор break. Операторы while и do while могут быть

вложенными. Пример:

```
int i = 0, j = 0, k = 0, a[30];
do {
        i++; j--;
        while ( a[k]<i ) k++;
} while ( i<30 && j<-30 );
```

3.7 Оператор продолжения

Этот оператор имеет вид: **continue;** и может употребляться только в операторах цикла. Выполнение этого оператора приводит к прерыванию текущей итерации и переходу к проверке условия выхода из цикла. Рассмотрим следующие три схемы, использующие различные операторы цикла:

```
while (выражение)
{оператор_1; continue; оператор_2;}

do{ оператор_1; continue; оператор_2;}
while (выражение);

for (выражение_1; выражение_2; выражение_3)
{оператор_1; continue; оператор_2;}
```

Если среди операторов тела цикла есть оператор continue, то он передает управление на очередную итерацию цикла, не выполняя операторы следующие за ним. Пример использования оператора continue демонстрирует расчет суммы положительных элементов массива:

```
for (s=0.0, i=0; i<n; i++) {
        if(x[i]<=0) continue;
        s += x[i];
}
```

3.8 Оператор завершения

Оператор **break** может быть использован только внутри операторов switch, for, while или do while и обеспечивает прекращение выполнения самого внутреннего из содержащих его операторов. После выполнения оператора break управление передается оператору следующему за прерванным. Формат оператора: break; Оператор break нельзя использовать для выхода из нескольких вложенных циклов.

Необходимость использования оператора break; в теле цикла возникает, когда условие выхода из цикла нужно проверять не в начале или в конце итера-

ции, как предусмотрено операторами цикла, а в середине тела цикла. В этом случае тело цикла может иметь следующую структуру:

{оператор_1; if (выражение) break; оператор_2;}

Что касается оператора-переключателя, то применение внутри него break; является правилом, чем исключением.

3.9 Оператор возврата из функции

Оператор предназначен для завершения выполнения блока операторов функции и возврата управления в точку вызова функции. Две возможные формы оператора имеют следующий вид: return выражение; или return;

Первая из них используется внутри блока функции, возвращающей значение. Выражение (значение которого возвращается) может быть только скалярным, причем если его тип не идентичен описанному или принятому по умолчанию типу идентификатора функции, то выражение будет преобразовано в данный тип. Вторая форма используется в функциях, не предназначенных для возврата каких-либо значений (описанных с типом void для возвращаемого значения). При необходимости операторов return в блоке функции может быть несколько.

4 ФУНКЦИИ

Функция − это поименованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо. Синтаксис языка C++ рассматривает вызов функции как одну из стандартных операций. Каждая программа на языке C++ – это совокупность функций. Мощность языка C++ во многом определяется легкостью и гибкостью в определении и использовании функций. В отличие от других языков программирования высокого уровня в языке C++ нет деления на процедуры, подпрограммы и функции, здесь вся программа строится только из функций.

4.1 Определение и вызов функции

Программа C++ состоит из одной или нескольких функций. Функции разбивают большие задачи на маленькие подзадачи. Имя одной из функций – main, которая обязательно должна присутствовать в любой программе, является зарезервированным. Функция main необязательно должна быть первой, хотя именно с нее начинается выполнение программы. Функция не может быть определена в другой функции.

С использованием функции связаны 3 понятия – определение функции, объявление функции и вызов функции.

Определение функции имеет вид:

тип имя_функции (список описаний аргументов){ операторы }

Здесь тип – тип возвращаемого функцией значения;

список описаний аргументов – формальные параметры;

операторы в фигурных скобках { } тело функции.

Например, функция, находящая и возвращающая максимальное значение из двух целых величин a и b определяется так:

```
int max(int a, int b){ return(a>=b)? a:b; }
```

Это определение говорит о том, что функция с именем max имеет два целых аргумента и возвращает целое значение. Если функция действительно должна возвращать значение какого-либо типа, то в ее теле обязательно должен присутствовать оператор return выражение; при выполнении этого оператора выполнение функции прекращается, управление передается в функцию, вызывающую данную функцию, а значением функции будет значение выражения.

Если у функции нет формальных параметров, то она определяется, например, так:

```
double f(void){тело функции};
```

или, эквивалентно,

```
double f( ) {тело функции};
```

Обращаются в программе к этой функции, например, так: a = b*f( ) + c;

Функция может и не возвращать никакого значения. В этом случае ее определение таково: void имя (список описаний аргументов){ операторы }. Вызов такой функции имеет вид: имя (список фактических аргументов); Выполнение функции, не возвращающей никакого значения, прекращается оператором return без следующего за ним выражения. Выполнение такой функции и возврат из нее в вызывающую функцию происходит также и в случае, если при выполнении тела функции произошел переход на самую последнюю закрывающую фигурную скобку этой функции. Пример функции копирующей одну строку в другую:

```
void copy (char* to, char* from)
{
    while(* to ++ = *from ++ );
}
```

Особым способом вызова является вызов функции но указателю. Указатель на функцию определим следующим образом:

тип_функции (*имя_указателя) (список параметров);

Например, int (*fptr) (double);

Здесь определяется fptr как указатель на функцию с одним аргументом типа double, которая возвращает значение int. Имя функции без следующих за ним ( ) – это указатель на функцию, который содержит адрес начала кода этой функции. Пример:

```
void f1(void){
        cout<<"\n Выполняется f1().";
}
void f2(void){
        cout<<"\n Выполняется f2().";
}
void main(){
        void (*ptr)(void);
        ptr = f2;
        (*ptr)();       //вызов функции f2();
        ptr = f1;
        (*ptr)();       //вызов f1();
        ptr();          //альтернативная запись!
}
```

4.2 Параметры функции

В языке C++ связь формального параметра с аргументом (фактическим параметром) может быть осуществлена и по значению, и по ссылке. По умолчанию параметры передаются по значению. Это означает, что при вызове создается внешний блок, в котором происходит создание локальных переменных с именами формальных параметров, и им присваиваются значения фактических параметров (аргументов). Таким образом, никакие операции над формальными параметрами функции не изменяют значений фактических параметров. Однако существуют, по меньшей мере, два механизма, позволяющие изменять значения объектов вызывающей программы действиями в вызванной функции: параметры-указатели и параметры-ссылки.

Передача аргументов по значению.

Например, вариант функции, возводящей целое x в целую степень n.

```
int power (int x, int n)
{
    for (int p = 1; n > 0; n--) p* = x;
    return p;
}
```

Аргумент n используется как временная переменная. Что бы ни происходило с n внутри функции power, это никак не влияет на фактический аргумент, с которым первоначально обратились к этой функции в вызываемой функции.

Рассмотрим процесс вызова функции более подробно. При вызове функции:

– в стеке резервируется место для формальных параметров, в которые записываются значения фактических параметров. Обычно это производится в порядке, обратном их следованию в списке;

– при вызове функции в стек записывается точка возврата – адрес той части программы, где находится вызов функции;

– в начале тела функции в стеке резервируется место для локальных (автоматических) переменных.

Передача указателей в качестве аргументов функции.

Рассмотрим пример описания и использования функции с параметром-указателем:

```
void positive(int *m) { *m = *m > 0 ? *m: -*m; }
void main() { int k = -3; positive(&k); }
```

Тип формального параметра – указатель типа int*, поэтому в качестве аргумента при вызове функции необходимо использовать операцию раскрытия указателя, вычисляющую адрес переменной, описанной в вызывающей функ-

ции main. Именно по значению этого адреса обеспечивается доступ к переменной из тела вызываемой функции positive с применением операции нахождения (*).

Некоторую особенность имеет использование массивов в качестве аргументов. Эта особенность заключается в том, что имя массива преобразуется к указателю на его первый элемент, т.е. при передаче массива происходит передача указателя. По этой причине вызываемая функция не может отличить, относится ли передаваемый ей указатель к началу массива или к одному единственному объекту.

```
int summa (int array[ ], int size)
{
    int res=0;
    for (int i = 0; i < size; i++) res+ = array[i];
    return res;
}
```

В заголовке int array[] можно заменить на int* array, а выражение в теле функции array[i] заменить на *(array+i), или даже на *array ++.

Передача ссылок в качестве аргументов функции.

Использование параметров-ссылок рассмотрим на другом варианте функции positive:

```
void positive(int &m) { m = m > 0 ? m: -m; }
void main() { int k = -3; positive(k); }
```

Параметр-ссылка обеспечивает доступ из тела функции к соответствующему фактическому параметру и в этом смысле обладает теми же возможностями, что и параметр-указатель. Более того, текст программы упростился: в теле функции для параметра-ссылки не нужно применять операцию разыменования (*), а фактическим параметром должен быть не адрес (как для параметра-указателя), а просто переменная. Логика при интерпретации параметров-ссылок следующая: поскольку ссылка в языке С++ определяется как еще одно имя (псевдоним) объекта, то формальный параметр-ссылка при вызове функции становится синонимом фактической переменной, на которую автоматически переносятся все действия над формальным параметром.

Возможность использования вызова функции в качестве lvalue-выражения предоставляют ссылки, формируемые как результат выполнения функции. Например, следующая функция, возвращающая ссылку на минимальный элемент массива:

```
float& max(int n, float *x)
{
        int m=0;
```

```
    for (int i=1; i<n; i++)
        m = x[m]<x[i]? m: i;      // Индекс наименьшего элемента
    return x[m];
}
```

позволит не только найти значение максимального элемента конкретного массива:

```
float x = max(sizeof(z)/sizeof(z[0]), z);
```

но и записать в этот элемент новое значение:

```
max(sizeof(z)/sizeof(z[0]), z) = 0.0;
```

Логика компиляции такого оператора следующая: вызов функции, возвращающей ссылку на элемент массива, является псевдонимом этого элемента, позволяющим обращаться к нему любым способом.

4.3 Массивы в качестве параметров функции

C++ позволяет определять массивы в качестве параметров функции. Размер массива можно указать при объявлении параметра или объявить параметр с пустыми скобками, что будет означать использование массива неопределенного (на этапе компиляции) размера. В последнем случае добавляется дополнительный параметр, через который передается количество элементов массива. Например, функции, вычисляющей скалярное произведение трехмерных векторов, можно явным образом указать размеры массивов-параметров:

```
float multMass(float v1[3], float v2[3])
{
    float s = 0;
    for(int i = 0; i < 3; i++)
        s += v1[i]*v2[i];
    return s;
}
```

В случае, когда размер массива неизвестен, необходимо передавать его размер отдельным параметром:

```
float multMass(int len, float v1[], float v2[])
{
    float s = 0;
    for(int i = 0; i < len; i++)
        s += v1[i]*v2[i];
    return s;
}
```

Альтернативным способом передачи функциям массивов в качестве аргументов является использование параметров-указателей. Например, следующая функ-

ция вычисляет среднее значение элементов массива, указатель на начало которого передан в качестве параметра:

```
double average(double *a, int len)
{
        float s = 0.0f;
        for (int i = 0; i < len; i++) s += a[i];
        return s/len;
}
```

Особым случаем можно считать функции для обработки строк – символьных массивов с нулевым конечным элементом (в частности, такими массивами являются строковые константы – последовательности символов, заключенные в кавычки). При этом нет необходимости передавать функции длину массива отдельным аргументом. Рассмотрим пример функции для определения длины строки, переданной в качестве параметра:

```
int strLength(char str[])
{
        int l =0;
        while(str[l++]);
        return l – 1; // нулевой символ отбрасываем
}
```

Проблемы неполного использования памяти возникают при реализации универсальных функций для обработки многомерных массивов. Описание соответствующего параметра должно в явном виде указывать все размерности, кроме старшей. Например, допустимым описанием является float a[][4][2], а описание float b[][][] – недопустимо в качестве параметра функции. Данное ограничение приводит к следующим неудобствам. Во-первых, при описании функции необходимы дополнительные договоренности относительно максимальных размерностей обрабатываемых массивов. Во-вторых, требование соответствия типов матрицы-аргумента и матрицы-параметра принуждает для реализации вызова функции резервировать память, достаточную для размещения всех элементов матрицы этой максимальной размерности (даже если конкретная матрица меньших размеров). Пример функции для транспонирования матрицы m x n (причем m – не больше 10):

```
void Transp(float a[][10], int m, int n)
{
        float w;
        int 1 = m>n? m: n;
        for (int j=0; j<1-1; j++)
                for (int i=j+l; i<l; i++)
                {
```

```
                w=a[j][i];
                a[j][i]=a[i][j];
                a[i][j]=w;
            }
}
```

Вызов этой функции в любом случае потребует описания матрицы с длиной строки, равной 10. Например, обработка матрицы размером 4x3 потребует описания массива с запасом памяти, достаточным для размещения 10x4 элементов (дополнительная 4-я строка необходима для сохранения результата транспонирования – матрицы 3x4).

Наиболее эффективным способом передачи массива в качестве параметра функции является передача указателя на массив. Пример создания и передачи в качестве параметра функции трехмерного массива:

```
void printMass(float***matrix, int n, int m, int l)
{
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            for(int k =0; k < l; k++)
                cout << matrix[i][j][k] << '\t';
// вывод на экран всех элементов массива через табуляцию
}

int main()
{
    int i, j, k, n = 5, m = 10, l = 3;

    // создание массива
    float ***mass = new float**[n];
    for(i = 0; i < n; i++)
        mass[i] = new float*[m];
    for(i = 0; i < n; i++)
        for(j = 0; j < m; j++)
            mass[i][j] = new float[l];

    // инициализация элементов массива

    // вывод на экран элементов массива
    printMass(mass, n, m, l);
```

```
    // освобождение динамической памяти занимаемой массивом
    for(i = 0; i < n; i++)
        for(j = 0; j < m; j++)
            delete[] mass[i][j];
    for(i = 0; i < n; i++)
        delete[] mass[i];
    delete[] mass;

    return 0;
}
```

В примере создается трехмерный массив в динамической памяти, управление которой полностью ложится на программиста, поэтому необходимо явно выполнять операции по освобождению памяти.

4.4 Аргументы функции по умолчанию

Удобным свойством C++ является наличие предопределённых инициализаторов аргументов. Значения аргументов по умолчанию можно задать в объявлении функции, при этом они подставляются автоматически в вызов функции, содержащий меньшее число аргументов, чем объявлено. Например, следующая функция объявлена с тремя аргументами, два из которых инициализированы: void error (char* msg, int level = 0, int kill = 0);

Эта функция может быть вызвана с одним, двумя или тремя аргументами:

```
error("Ошибка!");       // вызывается error ("ошибка", 0, 0);
error("Ошибка!",1);     // вызывается error ("ошибка", 1, 0);
error("Ошибка!",3,1); // аргументы по умолчанию не используются.
```

Все аргументы по умолчанию должны быть последними аргументами в списке – ни один явный аргумент не может находиться правее их. Если аргумент по умолчанию уже определён в одном объявлении, он не может быть переопределён в другом. Аргументы по умолчанию должны быть объявлены при первом объявлении имени функции. Если инициализация аргументов произведена в объявлении функции, в определении функции задавать инициализацию аргументов не надо.

4.5 Функция с переменным количеством параметров

Список формальных параметров, последним элементом которого является многоточие, делает возможным описание функций, которые могут быть вызваны с переменным количеством аргументов. Для таких функций во время

компиляции фактическое количество параметров, а также их типы неизвестны. Таким образом, сама функция должна иметь механизм определения (на этапе выполнения) количества и типов параметров. Что касается количества параметров, то принципиально различных подходов к созданию этого механизма всего два: либо передавать это количество вызываемой функции явным образом через обязательный аргумент, либо условиться о специальном признаке конца списка аргументов (например, о добавлении в конец списка аргумента с уникальным значением). В обоих подходах доступ к аргументам осуществляется с использованием указателей. Пример реализации первого подхода:

```
int multiSumCol(int n, ...)      // n – количество суммируемых аргументов
{
      int s = 0, *p = &n;   // Указатель начала списка аргументов
            while (n--)  s += *++p;
      return s;
}
```

Перебор суммируемых аргументов осуществляется при помощи указателя p, значение которого инициализируется адресом обязательного первого аргумента n и затем в цикле наращивается для получения доступа к следующему аргументу.

Второй пример использует аргумент с нулевым значением как индикатор конца списка аргументов:

```
float multiSumTerm(float x,   ...)
{
      float s = 0.0,  *p = &x;
      while(*p) s += *p++;
      return s;
}
```

Цикл суммирования будет перебирать аргументы до тех пор, пока не встретит нулевой аргумент, который должен быть обязательно использован при вызове функции. Принципиальным ограничением описанных здесь функций является невозможность их работы с аргументами различных типов: обе они предназначены для обработки аргументов вполне определенных типов (int, float). Это ограничение можно обойти, используя макросы: va_start, va_list, va_arg, va_end.

4.6 Указатель на функцию как параметр

Указатели на функции могут передаваться в качестве аргументов другим функциям, если последние реализуют какой-либо метод обработки функций, не зависящий от вида конкретной обрабатываемой функции. Эта конкретная (фак-

тическая) функция в этом случае вызывается из тела обрабатывающей функции по переданному ей указателю.

Пример функции вызывающей внешнюю функцию для определения арифметического действия:

```
double proxy(double (*operation) (double x, double y) , double a, double b)
{
        return (*operation)(a, b) + operation(b, a);
}
```

Представлены два возможных варианта вызова функции через указатель на функцию. В C++ идентификатор функции одновременно обозначает се адрес, поэтому вызов функции proxy может иметь следующий вид:

```
double result = proxy(add, 5.5, 76.2);
```

4.7 Рекурсивные функции

Ситуацию, когда функция тем или иным образом вызывает саму себя, называют **рекурсией**. Рекурсия, когда функция обращается сама к себе непосредственно, называется прямой; в противном случае она называется косвенной. Все функции языка C++ (кроме функции main) могут быть использованы для построения рекурсии. В рекурсивной функции обязательно должно присутствовать хотя бы одно условие, при выполнении которого последовательность рекурсивных вызовов должна быть прекращена.

Обработка вызова рекурсивной функции ничем не отличается от вызова функции обычной: перед вызовом функции в стек помещаются её аргументы, затем адрес точки возврата, затем, уже при выполнении функции – автоматические переменные, локальные относительно этой функции. Но если при вызове обычных функций число обращений к ним невелико, то для рекурсивных функций число вызовов и, следовательно, количество данных, размещаемых в стеке, определяется глубиной рекурсии. Поэтому при рекурсии может возникнуть ситуация переполнения стека.

Если попытаться отследить по тексту программы процесс выполнения рекурсивной функции, то будет следующая ситуации: войдя в рекурсивную функцию, мы “движемся” по ее тексту до тех пор, пока не встретим ее вызова, после чего мы опять начнем выполнять ту же самую функцию сначала. При этом следует отметить самое важное свойство рекурсивной функции – ее первый вызов еще не закончился. Чисто внешне создается впечатление, что текст функции воспроизводится (копируется) всякий раз, когда функция сама себя вызывает. На самом деле этот эффект воспроизводится в компьютере. Однако копируется при этом не весь текст функции (не вся функция), а только ее части, связанные с данными (формальные, фактические параметры, локальные пере-

менные и точка возврата). Алгоритм (операторы, выражения) рекурсивной функции не меняется, поэтому он присутствует в памяти компьютера в единственном экземпляре. Пример вычислить n!:

```
long factorial (int n){
    if ( n<1 ) return 1;
    else return n*fact(n-1);
}
```

Пример последовательности вызовов для factorial (3) приведен на рисунке 5.

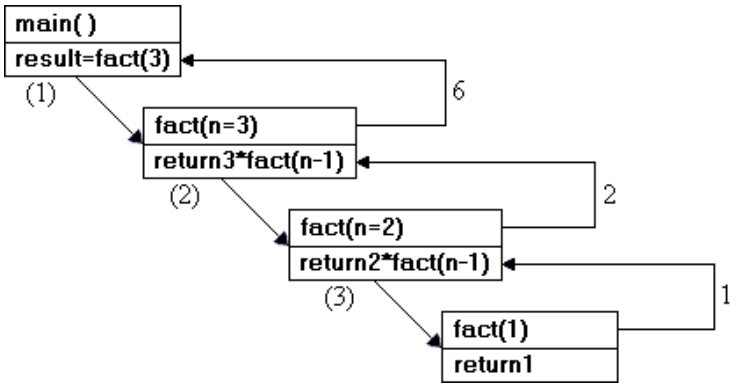


Рисунок 5 – Пример последовательности вызовов для factorial (3)

При косвенной рекурсии осуществляется перекрёстный вызов функциями друг друга. Хотя бы в одной из них должно быть условие, вызывающее прекращение рекурсии.

4.8 Встраиваемые функции

В C++ можно задать функцию, которая на самом деле не вызывается, а ее тело встраивается в программу в месте ее вызова. Она действует почти так же, как макроопределение с параметрами в C. Преимуществом встраиваемых *(inline)* функций является то, что они не связаны с механизмом вызова функций и возврата ими своего значения. Это значит, что встраиваемые функции могут выполняться гораздо быстрее обычных. (Выполнение машинных команд, которые генерируют вызов функции и возвращение функцией своего значения, занимает определенное время. Если функция имеет параметры, то ее вызов занимает еще большее время.)

Недостатком встраиваемых функций является то, что если они слишком большие и вызываются слишком часто, объем ваших программ сильно возрастает. Из-за этого применение встраиваемых функций обычно ограничивается короткими функциями. Для объявления встраиваемой функции используется спецификатор inline перед определением функции. Например, в этой короткой программе показано, как объявить встраиваемую функцию:

```
inline int even(int x)
{
    return !(x%2);
}
int main()
{
    if (even(10)) cout << "10 является четным\n";
    if (even(ll)) cout << "11 является четным\n";
    return 0;
}
```

В этом примере функция even()**,** которая возвращает истину при четном аргументе, объявлена встраиваемой. Это означает, что строка

```
if (even(10)) cout << "10 является четным\n";
```

функционально идентична строке

```
if (!(10%2)) cout << "10 является четным\n";
```

Этот пример указывает также на другую важную особенность использования встраиваемой функции: она должна быть задана до ее первого вызова.

4.9 Перегрузка функций

Перегрузка – особенность языка C++, которая позволяет определять функции с одинаковым именем, но разными списками параметров. Перегрузка функций не только обеспечивает механизм, посредством которого в C++ достигается один из типов полиморфизма, она также формирует то ядро, вокруг которого развивается вся среда программирования на C++. Если две или более функции имеют одинаковое имя, говорят, что они перегружены.

Перегруженные функции позволяют упростить программы, допуская, обращение к одному имени для выполнения близких по смыслу действий. Перегрузить функцию очень легко: просто объявите и определите все требуемые варианты. Компилятор автоматически выберет правильный вариант вызова на основании числа и/или типа используемых в функции аргументов.

Одно из основных применений перегрузки функций – это достижение полиморфизма при компиляции программ, который воплощает в себе философию – один интерфейс, множество методов. При программировании на C необходимо иметь определенное число близких по назначению функций, отличающихся только типом данных, с которыми они работают. Классический пример этой ситуации дает набор библиотечных функций C. Функции abs(), labs() и fabs() возвращают абсолютное значение, соответственно, целого, длинного целого и числа с плавающей точкой. Однако из-за того, что для трех типов данных требуется три типа функции, ситуация выглядит более сложной, чем это необходимо. Во всех трех случаях возвращается абсолютная величина числа, отличие

только в типе данных. В то же время, программируя на C++ можно исправить эту ситуацию путем перегрузки одного имени для трех типов данных так, как показано в следующем примере:

```
int abs(int n) // absO для целых
{
        cout << "В целом abs()\n";
        return n<0 ? -n: n;
}
long abs (long n)  // absO для длинных целых
{
        cout << "В длинном целом abs()\n" ;
        return n<0 ? -n: n;
}
double abs (double n) // abs() для вещественных двойной точности
{
        cout << "В вещественном abs() двойной точности\n" ;
        return n<0 ? -n: n;
}
int main()
{
        cout << "Абсолютная величина -10:" << abs (-10) << '\n';
        cout << "Абсолютная величина -10L:" << abs(-l0L) << '\n';
        cout << "Абсолютная величина -10.01:" << abs(-l0.01) << '\n';
        return 0;
}
```

Поскольку одно имя используется для описания основного набора действий, искусственная сложность, вызванная тремя слабо различающимися именами, в данном случае abs(), labs() и fabs(), устраняется. Теперь необходимо помнить только одно имя – то, которое описывает общее действие. На компилятор возлагается задача выбора соответствующей конкретной версии вызываемой функции (а значит и метода обработки данных).

4.10 Шаблоны функций

Цель введения шаблонов функций – автоматизация создания функций, которые могут обрабатывать разнотипные данные. В определении шаблонов семейства функций используется служебное слово template, за которым в угловых скобках следует список параметров шаблона. Каждый формальный параметр шаблона обозначается служебным словом class (можно использовать typename), за которым следует имя параметра:

template <typename/class тип>

возвр_значение имя_функции (список параметров) { тело функции }

Здесь вместо «тип» указывается тип используемых функцией данных. Это имя можно указывать внутри определения функции. Однако это всего лишь фиктивное имя, которое компилятор автоматически заменит именем реального типа данных при создании конкретной версии функции.

В следующей программе создается шаблонная функция, которая меняет местами значения двух переменных, передаваемых ей в качестве параметров. Поскольку в своей основе процесс обмена двух значений не зависит от типа переменных, этот процесс удачно реализуется с помощью шаблонной функции.

```
template <class X> void swapargs (X &a, X &b)
{
    X temp;
    temp = a;
    a = b;
    b = temp;
}
int main()
{
    int i = 10, j = 20;
    float x = 10.1, y = 23.3;
    cout << "Исходные значения i, j равны: " << i << ' ' << j << '\n';
    cout << "Исходные значения x, y равны: " << x << ' ' << y << '\n';
    swapargs (i, j ) ; // обмен целых
    swapargs (x, y) ; // обмен действительных
    cout << "Новые значения i, j равны: " << i << ' ' << j << '\n';
    cout << "Новые значения x, y равны: " << x << ' ' << y << '\n';
    return 0;
}
```

Когда создается шаблонная функция, то по существу предписывается компилятору генерировать столько разных версий этой функции, сколько нужно для обработки всех способов вызова этой функции в программе.

5 ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

C++ обеспечивает встроенный механизм обработки ошибок, называемый обработкой исключительных ситуаций (exception handling). Благодаря обработке исключительных ситуаций можно упростить управление и реакцию на ошибки во время выполнения программ. Обработка исключительных ситуаций в C++ организуется с помощью трех ключевых слов: **try, catch** и **throw.** В самых общих словах, инструкции программы, во время выполнения которых вы хотите обеспечить обработку исключительных ситуаций, располагаются в блоке **try**. Если исключительная ситуация (т. е. ошибка) имеет место внутри блока **try**, она возбуждается (ключевое слово **throw),** перехватывается (ключевое слово **catch)** и обрабатывается. Ниже поясняется приведенное здесь общее описание.

Любая инструкция, которая возбуждает исключительную ситуацию, должна выполняться внутри блока try. (Функции, которые вызываются из блока try также могут возбуждать исключительную ситуацию.) Любая исключительная ситуация должна перехватываться инструкцией catch**,** которая располагается непосредственно за блоком try, возбуждающем исключительную ситуацию. Далее представлена основная форма инструкций try и catch:

```
try {
// блох возбуждения исключительной ситуации
}
catch (тип_1 arg) {
// блок перехвата исключительной ситуации
}
catch (тип_2 arg) {
// блок перехвата исключительной ситуации
}
catch (…) {
// обработка всех исключительных ситуаций
}
```

Блок try должен содержать ту часть программы, в который необходимо отслеживать ошибки. Это могут быть как несколько инструкций внутри одной функции, так и все инструкции внутри функции main() (что естественно ведет к отслеживанию ошибок во всей программе). После того как исключительная ситуация возбуждена, она перехватывается соответствующей этой конкретной исключительной ситуации инструкцией catch**,** которая ее обрабатывает. С блоком try может быть связано более одной инструкции catch. То, какая именно инструкция catch используется, зависит от типа исключительной ситуации. То

есть, если тип данных, указанный в инструкции catch, соответствует типу исключительной ситуации, выполняется данная инструкция catch. При этом все оставшиеся инструкции блока try игнорируются (т. е. сразу после того, как какая-то инструкция в блоке try вызвала появление исключительной ситуации, управление передается соответствующей инструкции catch, минуя оставшиеся инструкции блока try). Если исключительная ситуация перехвачена, аргумент arg получает ее значение. Если не нужен доступ к самой исключительной ситуации, можно в инструкции catch указать только ее тип, аргумент arg указывать не обязательно. В некоторых случаях необходимо настроить систему так, чтобы перехватывать все исключительные ситуации, независимо от их типа, для этого используется указание в инструкции catch(…). Данная инструкция должна стоять в списке последней иначе она будет перехватывать все исключения. Можно перехватывать любые типы данных, включая и типы классов. Фактически в качестве исключительных ситуаций часто используются именно типы классов. Далее представлена основная форма инструкции **throw:**

throw исключительная_ситуация;

Инструкция throw должна выполняться либо внутри блока try, либо в любой функции, которую этот блок вызывает (прямо или косвенно). Здесь исключительная_ситуация – это возбуждаемая инструкцией throw исключительная ситуация. Если возбуждается исключительная ситуация, для которой нет соответствующей инструкции catch, может произойти ненормальное завершение программы. Если ваш компилятор функционирует в соответствии со стандартом C++, то возбуждение необрабатываемой исключительной ситуации приводит к вызову стандартной библиотечной функции terminate(). По умолчанию для завершения программы функция terminate() вызывает функцию abort(), однако при желании можно задать собственную процедуру завершения программы. В следующем очень простом примере показано, как в C++ функционирует система обработки исключительных ситуаций:

```
int main(}
{
    cout << "начало\n";
    try
    { // начало блока try
        cout << "Внутри блока try\n";
        throw 10; // возбуждение ошибки
        cout << "Эта инструкция выполнена не будет";
    }
    catch (int i)
    { // перехват ошибки
        cout << "перехвачена ошибка номер: " << i << "\n";
```

```
    }
    cout << "конец";
    return 0;
}
```

Здесь имеется блок try**,** содержащий три инструкции, и инструкция catch (int i), обрабатывающая исключительную ситуацию целого типа. Внутри блока try будут выполнены только две из трех инструкций – инструкции cout и throw. После того как исключительная ситуация возбуждена, управление передается выражению catch и выполнение блока try завершается. Таким образом, инструкция catch вызывается не явно, управление выполнением программы просто передается этой инструкции. Следовательно, следующая за инструкцией throw инструкция cout не будет выполнена никогда.

6 КОМПИЛЯЦИЯ, КОМПОНОВКА И ОБЛАСТИ СУЩЕСТВОВАНИЯ ИМЕН

Текст программы можно разместить в одном файле, а можно и в нескольких различных файлах, каждый из которых содержит целиком одну или несколько функций. Для объединения в одну программу эти файлы компилируются совместно. Компилятор для каждого исходного файла создаёт объектный код (файл с расширением .obj). Затем все объектные файлы (вместе с библиотечными) объединяются компоновщиком в исполняемый, или загрузочный модуль, с расширением .exe. Область существования имени нужна компилятору для того, чтобы сгенерировать верный машинный код.

Схема подготовки программы к выполнению показана на рисунке 6.

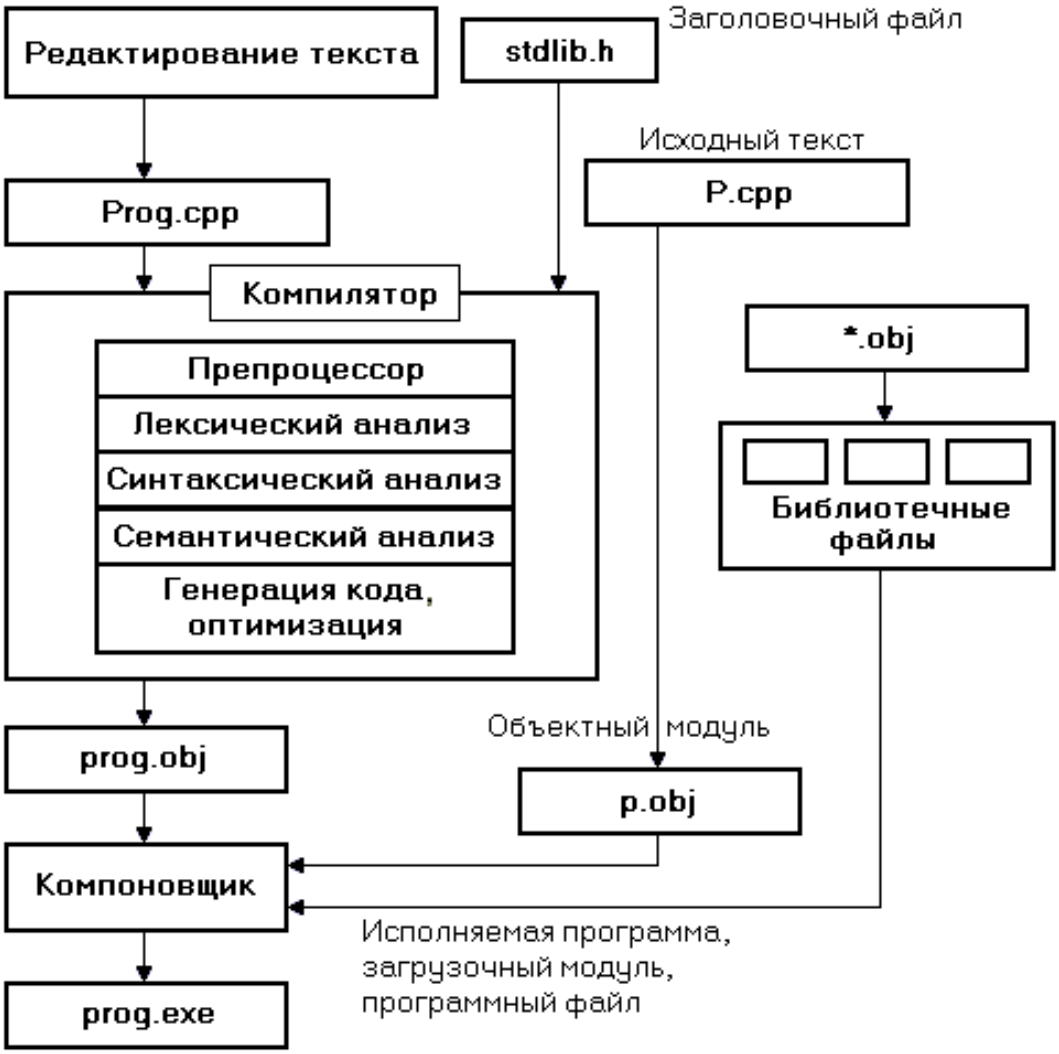


Рисунок 6 – Схема подготовки программы к выполнению

Подготовка программы начинается с редактирования файла, содержащего текст этой программы, который имеет стандартное расширение ".cpp". Затем

выполняется его компиляция, которая включает в себя несколько фаз: препроцессор, лексический, синтаксический, семантический анализ, генерация кода и его оптимизация. В результате компиляции получается объектный модуль - некий "полуфабрикат" готовой программы, который потом участвует в ее сборке. Файл объектного модуля имеет стандартное расширение ".obj". Компоновка (сборка) программы заключается в объединении одного или нескольких объектных модулей программы и объектных модулей, взятых из библиотечных файлов и содержащих стандартные функции и другие полезные вещи. В результате получается исполняемая программа в виде отдельного файла (загрузочный модуль, программный файл) со стандартным расширением -".exe", который затем загружается в память и выполняется.

Можно выделит 5 видов областей существования имени.

Область существования БЛОК или составной оператор. Блок – это фрагмент программы, заключённый в фигурные скобки { }, например

```
if (a != 5) {
    int j=0;
    double k = 3.5;
    a++;
    //...
}
```

Заметим, что тело любой функции является блоком.

Имя, объявленное в блоке, может быть использовано от точки, где находится его объявление, и до конца блока. Такую же область существования имеют и имена в определении функции:

```
int f1 (int i){ return i; }
```

Имя i имеет область существования "блок". Область существования "блок" распространяется и на вложенные блоки.

Область существования ФУНКЦИЯ. Эту область существования имеют только имена меток перехода, используемые оператором goto:

```
void f ()
{
    goto lab;
    //...
    {//...
         lab:;
     //...
     }
    //...
}
```

Область существования ПРОТОТИП ФУНКЦИИ. Прототип функции есть

объявление функции, не являющееся её определением и имеющий, например, вид: int F(int a, double b, char* str);
Область существования "прототип" заключена между открывающей и закрывающей круглыми скобками. Иначе говоря, имена a, b, str в примере определены только внутри круглых скобок. Из этого следует, что в прототипах можно использовать для аргументов любые имена или не использовать их совсем: int F(int, double, char*);

Область существования ФАЙЛ. Область существования "файл" имеют имена, объявленные вне любого блока и класса. Такие имена называют глобальными. Глобальные имена определены от точки их объявления и до конца файла, где встретилось их объявление. Примером таких имён являются имена функций:

```
#include <iostream.h>
int a, b, c[40];        // Глобальные имена
int f1()                // глобальное имя f1
{
      int i;            // локальное имя
}
int count;              // глобальное имя
void f2() { ... }       // глобальное имя f2
```

Область существования КЛАСС. Такую область существования имеют имена, объявленные в классах. Эти имена определены во всем классе, в котором они объявлены, независимо от точки их объявления.

Структура программы на C++ состоит из исходных файлов на языке C++ (*.cpp) и заголовочных файлов (*.h).

Заголовочный файл, как правило, не содержит исполняемых инструкций и поэтому не может компилироваться отдельно. Он предназначен лишь для включения в другие файлы C++ с помощью директивы препроцессора

```
#include <имя_файла.h>
```

или

```
#include “имя_файла.h”
```

для предоставления доступа к содержащимся в нем декларациям из включающих его файлов. Угловые скобки (<>) в директиве #include используются, если заголовочный файл размещен в системной директории операционной системы или среды разработки. Эти директории обычно прописаны в переменной окружения PATH или INCLUDE. Двойные кавычки (""), в свою очередь, позволяют указать непосредственно путь к файлу, причем допускаются стандартные для многих операционных систем обозначения: по умолчанию для текущей директории – это точка (./), а для родительской директории – двойная точка (../). В

#include всегда можно использовать UNIX-разделитель поддиректорий – прямой слеш (/), даже если вы программируете в Windows.

Желательно использовать следующую структуру заголовочного файла:

```
#ifndef _КОНСТАНТА_H
#define _КОНСТАНТА_H
#include “необходимыйе .h файлы”
// глобальные декларации
#endif
```

Директивы #ifndef, #define и #endif заставляют препроцессор включать в компиляцию этот файл только один раз, поскольку директива #include для этого заголовочного файла может появиться в нескольких различных единицах компиляции в одном проекте. Пара #ifndef-#endif определяет условный блок препроцессора, проверяющий не определена ли уже константа _КОНСТАНТА_H, которая может иметь любое имя, но обычно выбирается совпадающее с именем файла и добавляется несколько символов подчеркивания в различных местах, чтобы избежать случайного совпадения имени константы с какой-либо системной константой. Если указанная константа еще не была определена, то обрабатывается содержимое этого блока, и первым делом эта константа определяется директивой #define.

Если глобальные декларации или объявляемый класс использует имена или функции (в случае inline-методов), объявленные в других заголовочных файлах, то наш h-файл может содержать директивы #include, а также, при необходимости, любые другие директивы препроцессора.

Файл реализации, декларированного в заголовочном файле, содержит инструкции определения функций. В начале файла ставится директива #include для включения заголовочного файла. Для каждого модуля можно создать отдельный файл реализации и компилировать его независимо от остальной части программы. Это облегчает отладку и позволяет распространять модули (или библиотеку), включая в поставку только исходные тексты заголовочных файлов и объектные (двоичные) файлы реализации. Таким образом, можно скрыть алгоритмы, использованные при реализации функций, защищая авторские права.

7 КЛАССЫ

Класс инкапсулирует данные и функции, определяя структуру и поведение представителей класса, которыми являются объекты. Элементы-данные объектов называются полями, а элементы-функции – методами.

Для правильного понимания термина «класс» необходимо научиться выделять в задаче реальные объекты, которые могут ассоциироваться с некоторыми данными (какого-либо типа), но, кроме того, определяются своим поведением (способностью изменять данные по определенным алгоритмам) в задаче. Совокупность объектов с одинаковым поведением и составляет класс (или иерархию классов). Формально в языке C++ класс определяется как новый тип, а объекты декларируются как переменные этого типа. Главное отличие объекта от обычной переменной заключается в том, что с ним уже связаны не только определенные значения данных, но и методы, т.е. функции класса, определяющие поведение объектов класса. Фактически, каждый объект может решать определенные задачи, манипулируя своими данными с помощью методов класса.

7.1 Определение класса

Определение класса в общем случае может иметь следующий вид:

```
class Class_Name : public Class_PublicBaseClass,
private Class_PrivateBaseClass, protected Class_ProtectedClass {
private:
// сокрытые элементы класса
protected:
// защищенные элементы класса
public:
// открытые элементы класса
};
```

Определение начинается с ключевого слова class, за которым следует имя класса (Class_Name), которое выбирается произвольно, как и любой идентификатор. Если класс является производным (при использовании наследования) от одного или нескольких базовых классов, то после имени класса следует двоеточие и список имен базовых классов, разделенных запятыми, с указанием (необязательно) типа наследования (private, protected или public). Если наследование не используется, то двоеточие отсутствует, а тело класса начинается открывающей фигурной скобкой и заканчивается закрывающей фигурной скобкой и точкой с запятой.

Тело класса содержит элементы, сгруппированные по типу доступа к ним (private, protected или public, указанные перед соответствующей группой с двоеточием), обеспечивая инкапсуляцию, т.е. контроль доступа к элементам класса. Элементами класса могут быть поля данных или методы, разделенные точкой с запятой.

Таким образом, класс в C++ рассматривается как расширение понятия блока данных путем включения в него, помимо полей данных, методов их обработки, что уже на уровне определений закрепляет их неразрывную связь друг с другом.

7.2 Поля данных

Поле данных объявляется своим типом и именем (как обычная переменная в C/C++), это может быть и объект другого класса (при этом класс оказывается агрегатным). Поля данных содержат описания переменных и (или) объектов классов, являющихся полями каждого объекта данного класса, если только поле не содержит спецификации static. Таким образом, при декларации класса не резервируется память для полей данных. Память, соответствующая типу каждого поля, резервируется для каждого объекта, декларированного где-либо в программе. Только поля static относятся к классу вообще и доступны всем объектам данного класса одновременно, как глобальная переменная. Например:

```
class Sample
{
    static double d;
    int i;
    char ch;
};
```

Итак, сколько бы объектов данного класса ни использовалось, каждый из них будет обладать собственными полями, декларированными в классе. Исключение составляют элементы класса, декларированные с ключевым словом static, использующиеся для совместного доступа и обмена информацией между всеми объектами класса. Областью видимости объекта является блок, в котором он декларирован, а областью видимости элементов класса является сам класс, вследствие инкапсуляции. Инкапсуляция позволяет ограничить доступ к элементам класса, так что вне класса, например в функции main (), нет доступа к полям данных объектов a, b, c, которые по умолчанию являются сокрытыми (private) в классе. Такое ограничение доступа контролируется классом.

7.3 Методы класса

Методы декларируются в классе прототипами соответствующих функций, хотя допускается и полное определение функции-метода непосредственно в классе, которая тогда будет рассматриваться как встраиваемая (inline). Такой метод будет компилироваться в соответствии с правилами для обычных подставляемых (inline) функций. Однако, как правило, реализация соответствующего прототипу метода задается в отдельном C++ файле реализации (имя_класса. cpp), где каждый метод описывается отдельно в соответствии со следующим синтаксисом:

```
тип_возвращаемого_значения
Имя_Класса::имя_метода(список_формальных_параметров)
{
    // описания и операторы функции
}
```

Существенным отличием от декларации обычной функции является только наличие спецификации принадлежности классу Имя_Класса:: перед именем функции с использованием операции разрешения области видимости (::).

Поскольку прототипы всех методов уже заданы в декларации класса, то порядок следования реализаций методов в файле не имеет значения. Методы одного класса могут вызывать друг друга и обращаться к любым полям данных этого класса.

Аналогично полям данных, классы могут содержать методы static, которые принадлежат классу, а не отдельным объектам. Для вызова методов static или для обращения к полям данных static можно пользоваться обычными операциями (точка или ->) с учетом ограничений доступа, заданных в каждом классе. Однако чаще всего для этих целей используется операция разрешения области видимости (::), подчеркивающая существование соответствующих полей и методов «в одном экземпляре» независимо от количества работающих в данный момент объектов класса. Более того, к полям и методам static можно обращаться даже при отсутствии объектов класса, и в этом случае операция разрешения видимости – единственная возможность обращения.

Помимо static, существует возможность объявить методы, которым запрещено изменять значения полей класса, спецификатором const. Рассмотрим пример:

```
// Файл Sample.h
#ifndef _Sample_H
#define _Sample_H
class Sample
{
```

```
        static int common; // глобальное сокрытое поле класса
        int b;  // сокрытое поле объекта
    public:
        void set (int);       // открытый метод, изменяющий объект
        int get () const;    // открытый метод, не изменяющий объект
        static void pass(int);       // инициализатор глобального поля
        void show();       // показать поля
    );
```

Реализация методов класса не отличается для методов const и остальных:

```
    // Файл Sample.cpp
    #include "Sample.h"
    #include <iostream.h>
    // инициализатор поля объекта:
    void Sample::set(int b)   {this->b=b;}
    int Sample::get() const {return b;}
    // инициализатор глобального поля:
    void Sample::pass(int n)   {common=n;}
    void Sample::show()
    {
        cout << b <<  '\t' << common << endl;
    }
```

Пример использования класса:

```
    // Файл main.cpp
    #include "Sample.h"
    int Sample::common; // повторное объявление в модуле компиляции
    void main()
    {
        Sample s1;
        s1.set(5);
        Sample::pass(s1.get()); }
```

7.4 Управление доступом к классу

Для ограничения уровня доступа к данным и функциям-членам класса в C++ существуют три ключевых слова private: (частный), protected: (защищенный), public: (общедоступный), задающие разделы поступа в классе. Каждый раздел в классе начинается с одного из приведенных слов. Если ни одно из ключевых слов не использовалось, то все объявления в классе считаются частными. Разделы с разными привилегиями доступа могут появлятся в любом по-

рядке и в любом колличестве. Рассмотрим пример:

```
class Example {
        int x1;       // частные по умолчанию
        int f1(void);
protected:
        int x2;       // защищенные
        int f2(void);
private:
        int x3;       // опять частные
        int f3(void);
public:
        int x4;       // общедоступные
        inf f4(void);
};
```

Частные (private) члены класса имеют наиболее ограниченный доступ. К частным членам класса имеют доступ только функции-члены данного класса или классы и функции объявленные как дружественные (friend) к данному классу, например:

```
class TPrivateClass {
        int value;
        int GetValue(void);
};
int TPrivateClass::GetValue(void)
{
        return value; //доступ разрешен
}
void main(void)
{
        TPrivateClass cl; //создание объекта

        int i = cl.value; //ошибка! Нет доступа
        i = cl.GetValue();//ошибка! Нет доступа
}
```

Защищенные члены класса (protected) и функции доступны только для функций производных классов. Обычно в этом возникает необходимость тогда, когда разрабатываемый класс является базовым классом для других классов. В этом случае он ограничивает доступ к данным внешним пользователям и разрешает доступ для классов наследников. Забегая вперед, рассмотрим простой пример иерархии объектов:

```
class A
```

```
{
protected:
    int val;
};
class B: public A
{ //наследуется от A
public:
    void fb(){ val = 0; } //доступ разрешен
};
class C: public B
{ //наследуется от B
public:
    void fc(){val = 10;} //доступ разрешен
};
```

В данном примере приведена иерархия классов A->B->C. Свойство защищенности распространяется вниз по иерархии до тех пор пока производный класс объявляет свой базовый общедоступным (public). При этом любые функции-члены в классах C и B имеют доступ к члену данных val базового класса. Если функция-член производного класса в качестве входного параметра имеет указатель или ссылку на объект базового класса, то правила становятся другими. Модифицируем класс C следующим образом:

```
class C: public B
{
public:
    void fc(A&)  //Входной параметр ссылка на базовый класс
    {
        val = 10; //доступ разрешен
        a.val = 10; //ошибка! нарушение прав доступа
    }
};
```

С общедоступными (public) членами класса все обстоит намного проще. Доступ к общедоступным членам класса разрешен функциям-членам самого класса, производных классов и обычным пользователям класса.

7.5 Указатель this

Каждый вызов метода устанавливает указатель на объект, по отношению к которому осуществляется вызов. Ссылаться на этот указатель можно посредством ключевого слова this (указатель на самого себя), предоставляющего

функциям-членам класса возможность доступа к данным объекта. Пример использования указателя this:

```
class Test
{
      int a;
public:
      void setA(int a) {this->a = a;}
}
```

В данном примере указатель this использовался для устранения неоднозначности, т.к. параметр метода класса и член класса имели одинаковые идентификаторы.

7.6 Конструкторы и деструкторы

Каждый класс имеет один или несколько конструкторов, которые вызываются автоматически при создании нового объекта и поэтому не являются методами. Если ни один конструктор не определен явно, то компилятор автоматически генерирует «конструктор по умолчанию», который просто резервирует память для объекта. Однако конструкторы могут быть переопределены в классе.

Для конструктора всегда выполняются следующие правила:

– конструктор никогда не вызывается явно, а по своему назначению используется в основном для инициализации полей данных при создании объекта, хотя в принципе может выполнять и другие действия (например, автоматическое преобразование типов);

– конструктор автоматически вызывается при создании объекта класса;

– имя конструктора совпадает с именем класса;

– конструктор не может иметь возвращаемого значения – результатом выполнения конструктора является объект;

– конструктор может иметь параметры, а следовательно, в классе можно определить несколько конструкторов;

– если в классе не определен ни один конструктор, то компилятором автоматически включается «конструктор по умолчанию», не имеющий параметров;

– если среди определенных конструкторов не задан конструктор без параметров, то «конструктор по умолчанию» компилятором не создается.

Для любого класса существует также деструктор, который может быть определен в классе явно следующим образом:

– имеет имя, построенное из имени класса с тильдой (~) перед ним;

– не имеет возвращаемого значения и параметров;

– не может быть переопределен, поскольку не может иметь параметров;
– в определенных случаях вызывается явно.

Деструктор вызывается автоматически для каждого объекта при выходе из области видимости объекта. Если деструктор явно не определен в классе, то он создастся компилятором автоматически. Такой деструктор по умолчанию освобождает память, занимаемую объектом класса. Определяя деструктор в классе явным образом, можно выполнить, кроме того, и любые другие действия.

Конструктор обычно используется при создании объектов класса для начальной инициализации их полей. Деструктор, наоборот, – для уничтожения объектов и освобождения занимаемой ими памяти (в том числе динамической).
Пример конструкторов и деструктора:

```
class Sample
{
    int n;
public:
    Sample() {n = 0;}
    Sample(int n) {this->n = n;}
    ~Sample() {cout << “Уничтожение объекта ” << n <<’\n’;}
};
```

Первый конструктор – «конструктор но умолчанию» – может, например, использоваться для начального обнуления поля n каждого объекта класса.
Второй конструктор позволит использовать параметр для задания начального значения поля n. Тогда объекты класса можно объявить двумя способами, например: Sample s1, s2(10);
Для инициализации полей класса в конструкторе допускается использование списка инициализации вместо операторов присваивания. Тогда приведенные конструкторы можно переписать так:

```
Sample::Sample(): n(0) {}
Sample::Sample(int n): n(n) {}
```

Список инициализации – единственный способ инициализации константного поля класса и параметров ссылок.

7.7 Дружественные функции и классы

Возможны ситуации, когда для получения доступа к закрытым членам класса понадобится функция, не являющаяся членом этого класса. Для достижения этой цели в C++ поддерживаются дружественные функции (friend functions). Дружественные функции не являются членами класса, но тем не менее имеют доступ к его закрытым элементам.

Дружественная функция задается так же, как обычная, не являющаяся членом класса, функция. Однако в объявление класса, для которого функция будет дружественной, необходимо включить ее прототип, перед которым ставится ключевое слово **friend.** Чтобы понять, как работает дружественная функция, рассмотрим следующую короткую программу:

```
class myclass {
        int n, d;
public:
        myclass (int i, int j) { n = i; d = j; }
        // объявление дружественной функции для класса myclass
        friend int isfactor (myclass ob) ;
};
int isfactor (myclass ob)
{
        if(!(ob.n % ob.d)) return 1;
        else return 0;
}
int main()
{
        myclass obi (10. 2), ob2(13. 3) ;
        if (isfactor (obi) cout << "10 без остатка делится на 2\n";
        else cout << "10 без остатка не делится на 2\n";
        if (isfactor (ob2) cout << "13 без остатка делится на 3\n";
        else cout << "13 без остатка не делится на 3\n";
        return 0;
}
```

В этом примере в объявлении класса myclass объявляются конструктор и дружественная функция isfactor (). Поскольку функция isfactor() дружественна для класса myclass, то функция isfactor() имеет доступ к его закрытой части. Поэтому внутри функции isfactorQ можно непосредственно ссылаться на объекты ob.n и ob.d. Важно понимать, что дружественная функция не является членом класса, для которого она дружественна. Поэтому невозможно вызвать дружественную функцию, используя имя объекта и оператор доступа к члену класса (точку или стрелку). Например, по отношению к предыдущему примеру эта инструкция неправильна:

```
obi.isfactor () ; // неправильно, isfactor() – это не функция-член
```

Дружественная функция должна вызываться точно так же, как и обычная функция. Хотя дружественная функция "знает" о закрытых элементах класса, для которого она является дружественной, доступ к ним она может получить только через объект этого класса. Таким образом, в отличие от функции-члена

myclass, в котором можно непосредственно упоминать переменные n и d, дружественная функция может иметь доступ к этим переменным только через объект, который объявлен внутри функции или передан ей.

Объявление дружественного класса позволяет всем его методам получить доступ ко всем переменным и методам другого класса. Пример:

```
class A
{
public:
      int Fx(B &b) {cout << b.a <<'\n';}
}

class B {public:
      friend class A;
private:
      int a;
};
```

Дружественные классы не наследуются, и их дружественность не является транзитивной. Пример:

```
class A {int Fx();}
class B {friend class A;}
class C {friend class B;}
   // Класс A не является
   // дружественным классу C
class D : public B {}
   // Класс A не является
   // дружественным классу D
```

7.8 Перегрузка операторов

Методы в классе, как и обычные функции, могут быть переопределены (перегружены). При переопределении методы с одинаковым именем должны различаться хотя бы по одному из следующих параметров:

- по количеству формальных параметров;
- по типу формальных параметров;
- по порядку следования параметров разных типов.

Тип возвращаемого значения в переопределенных методах может быть одинаковым или различным, но не является определяющим при переопределении метода (это правило действует и при перегрузке обычных функций).

Перегрузка операторов напоминает перегрузку функций. Более того, перегрузка операторов является фактически одним из видов перегрузки функций.

Однако при этом вводятся некоторые дополнительные правила. Например, оператор всегда перегружается относительно определенного пользователем типа данных, такого, как класс. Другие отличия будут обсуждаться ниже по мере необходимости. Когда оператор перегружается, то ничего из его исходного значения не теряется. Наоборот, он приобретает дополнительное значение, связанное с классом, для которого оператор был определен. Для перегрузки оператора создается оператор-функция (operator function). Чаще всего, оператор-функция является членом класса или дружественной классу, для которого она определена.

Здесь представлена основная форма оператор-функции – члена класса:

возвращаемый_тип имя_класса :: operator# (список_аргументов)

{ // выполняемая операция }

Часто типом возвращаемого значения оператор-функции является класс, для которого она определена. (Хотя оператор-функция может возвращать данные любого типа.) В представленной общей форме оператор-функции вместо знака # нужно подставить перегружаемый оператор. Например, если перегружается оператор +, то у функции должно быть имя operator+**.** Содержание списка список-аргументов зависит от реализации оператор-функции и от типа перегружаемого оператора. Следует запомнить два важных ограничения на перегрузку операторов. Во-первых, нельзя менять приоритет операторов. Во-вторых, нельзя менять число операндов оператора. Например, нельзя перегрузить оператор / так, чтобы в нем использовался только один операнд. Большинство операторов C++ можно перегружать. Ниже представлены те несколько операторов, которые перегружать нельзя: . :: .* ?

Нельзя перегружать операторы препроцессора. В C++ понятие оператора трактуется очень широко: в это понятие входят оператор индексирования [], оператор вызова функции (), операторы new и delete**,** операторы . (точка) и -> (стрелка). Оператор-функции, за исключением оператора =, наследуются производным классом. Тем не менее для производного класса тоже можно перегрузить любой выбранный оператор (включая операторы, уже перегруженные в базовом классе).

Допустимо иметь оператор-функцию для реализации любого действия – связанного или нет с традиционным употреблением оператора – лучше, если действия перегружаемых операторов остаются в сфере их традиционного использования. При создании перегружаемых операторов, для которых этот принцип не поддерживается, имеется риск существенного снижения читабельности программ. Оператор-функции не могут иметь параметров по умолчанию.

Когда оператор-функция – член класса перегружает бинарный оператор, у функции будет только один параметр. Этот параметр получит тот объект, который расположен справа от оператора. Объект слева генерирует вызов опера-

тор-функции и передается неявно, с помощью указателя this. Пример перегруженного оператора +:

```
class coord
{
        int x,y; // значения координат
public:
        coord() { x = 0; y= 0; }
        coord (int i, int j) { x = i ; y = j ; }
        void get_xy(int &i, int &j) { i = x; j = y; }
        coord operator+ (coord ob2);
};
// Перегрузка оператора + относительно класса coord
coord coord: :operator+ (coord ob2)
{
        coord temp;
        temp.x = x + ob2.x;
        temp.y = y + ob2.y;
        return temp;
}
int main()
{
        coord ol(10, 10), o2(5, 3), o3;
        o3 = ol + o2; // сложение двух объектов
        // вызов функции operator+()
}
```

Перегрузка унарных операторов аналогична перегрузке бинарных, за исключением того, что мы имеем дело не с двумя, а с одним операндом. При перегрузке унарного оператора с использованием функции-члена у функции нет параметров. Поскольку имеется только один операнд, он и генерирует вызов оператор-функции. Другие параметры не нужны. Пример перегрузки оператора инкремента ++:

```
class coord
{
        int x, y; // значения координат
        public:
        coord () { x = 0; y = 0; }
        coord (int i, int j) { x = i; y = j; }
        void get_xy(int &i, int &j) { i = x; j = y; }
        coord operator++ ( ) ;
};
```

```
// Перегрузка оператора ++ для класса coord
coord coord: :operator++()
{
        x++;
        y++;
        return *this;
}
int main()
{
        coord ol (10, 10) ;
        ++ol; // инкремент объекта
}
```

Важное значение имеет оператор присваивания. По умолчанию, если оператор присваивания применяется к объекту, то происходит поразрядное копирование объекта, стоящего справа от оператора, в объект, стоящий слева от оператора. Если это то, что вам нужно, нет смысла создавать собственную функцию operator=(). Однако бывают случаи, когда точное поразрядное копирование нежелательно. Например, если класс содержит указатели и недопустимо копирование, то необходимо использовать перегруженный оператор присваивания:

```
class strtype
{
      char *p;
      int len;
public:
      strtype (char *s) ;
      ~strtype ()
      {
            cout << "Освобождение памяти по адресу "<< (unsigned) p<< '\n';
            delete[]p;
      }
      char *get() { return p; }
      strtype &operator=( strtype &ob) ;
};
strtype: : strtype (char *s)
{
      int l;
      l = strlen(s) + 1;
      p = new char [1];
      if(!p) {
```

```
            cout << "Ошибка выделения памяти \n";
            exit(l);
        }
        len = 1;
        strcpy (p, s) ;
}
// Присваивание. объекта
strtype& strtype::operator=( strtype &ob)
{
        // выяснение необходимости дополнительной памяти
        if (len < ob.len) { // требуется выделение дополнительной памяти
            delete[] p;
            p = new char [ob.len];
            if(!p) {
                cout << "Ошибка выделения памяти \n";
                exit(l);
            }
        }
        len = ob.len;
        strcpy (p, ob.p) ;
        return *this; }
```

8 НАСЛЕДОВАНИЕ

В C++ наследование – это механизм, посредством которого один класс может наследовать свойства другого. Наследование позволяет строить иерархию классов, переходя от более общих к более специальным. Когда один класс наследуется другим, класс, который наследуется, называют базовым классом (base class). Наследующий класс называют производным классом (derived class). Обычно процесс наследования начинается с задания базового класса. Базовый класс определяет все те качества, которые будут общими для всех производных от него классов. В сущности, базовый класс представляет собой наиболее общее описание ряда характерных черт. Производный класс наследует эти общие черты и добавляет свойства, характерные только для него.

Дерево будет иметь обычный вид B-дерева, если используется только однократное наследование, когда каждый класс-потомок имеет только один родительский класс ближайшего уровня (который, однако, может иметь множество потомков, но лишь одного родителя).

Язык C++ допускает множественное наследование, когда несколько базовых классов используются для создания нового класса-потомка, наследующего свойства всех своих родительских классов. При использовании множественного наследования дерево классов может превратиться в сложный граф.

8.1 Однократное наследование

Чтобы понять, как один класс может наследовать другой, рассмотрим пример, который, несмотря на свою простоту, иллюстрирует несколько ключевых положений наследования.

Объявление базового класса:

```
class B {
    int i ;
public:
    void set_i(int n)  {i = n;}
    int get_i() {return i;}
}
```

Объявление производного класс, наследующий этот базовый:

```
class D: public B {
    int j;
public:
    void set_j (int n) {j = n;}
    int mul() {return j * get_i();}
}
```

```
int main()
{
    D ob;
    ob.set_i(10); // загрузка i в базовый класс
    ob.set_j(4);  // загрузка j в производный класс
    cout << ob.mul(); // вывод числа 40
}
```

После имени класса D имеется двоеточие, за которым следует ключевое слово public и имя класса B. Для компилятора это указание на то, что класс D будет наследовать все компоненты класса B. Само ключевое слово public нформирует компилятор о том, что, поскольку класс B будет наследоваться, значит, все открытые элементы базового класса будут также открытыми элементами производного класса. Однако все закрытые элементы базового класса останутся закрытыми и к ним не будет прямого доступа из производного класса. Функция get_i(), которая является членом базового класса B, а не производного D, вызывается внутри класса D без всякой связи с каким бы то ни было объектом. Это возможно потому, что открытые члены класса B становятся откры тыми членами класса D. В функции mul() вместо прямого доступа к i, необходимо вызывать функцию get_i(), поскольку закрытые члены базового класса (в данном случае i) остаются закрытыми для нее и недоступными из любого производного класса. Причина, по которой закрытые члены класса становятся недоступными для производных классов – поддержка инкапсуляции. Если бы закрытые члены класса становились открытыми просто посредством наследования этого класса, инкапсуляция была бы совершенно несостоятельна.

8.2 Управление доступом к базовому классу

Когда один класс наследуется другим, используется следующая основная форма записи:

class имя_производного_класса : сп_доступа имя_базового_класса {}

Здесь сп_доступа – это одно из трех ключевых слов: public, private или protected. Спецификатор доступа (access specifier) определяет то, как элементы базового класса (base class) наследуются производным классом (derived class). Если спецификатором доступа наследуемого базового класса является ключевое слово public, то все открытые члены базового класса остаются открытыми и в производном. Если спецификатором доступа наследуемого базового класса является ключевое слово private, то все открытые члены базового класса в производном классе становятся закрытыми. В обоих случаях все закрытые члены базового класса в производном классе остаются закрытыми и недоступными. Если спецификатором доступа является ключевое слово private, то хотя откры-

тые члены базового класса становятся закрытыми в производном, они остаются доступными для функций – членов производного класса.

Технически спецификатор доступа не обязателен. Если спецификатор доступа не указан и производный класс определен с ключевым словом class, то базовый класс по умолчанию наследуется как закрытый. Если спецификатор доступа не указан и производный класс определен с ключевым словом struct, то базовый класс по умолчанию наследуется как открытый. Тем не менее, для ясности большинство программистов предпочитают явное задание спецификатора доступа.

Наследование производным классом базового как открытого совсем не означает, что для производного класса станут доступными закрытые члены базового. Пример:

```
class base {
        int x;
public:
        void setx(int n) { x = n; }
        void showx() { cout << x << '\n'; }
};
// Класс наследуется как открытый
class derived: public base {
        int y;
public:
        void setyfint n) { y = n; }
/* Закрытые члены базового класса недоступны, x — это закрытый
член базового класса и поэтому внутри производного класса он
недоступен */
        void show_sum() { cout << x+y << '\n'; } // Ошибка!!!
        void showy () { cout << y << ' \n ' ; }
};
```

Здесь в производном классе derived сделана попытка доступа к переменной x, которая является закрытым членом базового класса base. Это неверно, поскольку закрытые члены базового класса остаются закрытыми, независимо от того, как он наследуется. Если производному классу необходим доступ к некоторым членам базового, то эти члены должны быть открытыми. Однако возможна ситуация, когда необходимо, чтобы члены базового класса, оставаясь закрытыми, были доступны для производного класса. Для реализации этой идеи в C++ включен спецификатор доступа protected (защищенный).

Спецификатор доступа protected эквивалентен спецификатору private с единственным исключением: защищенные члены базового класса доступны для

членов всех производных классов этого базового класса. Вне базового или производных классов защищенные члены недоступны.

Спецификатор доступа protected может находиться в любом месте объявления класса, хотя обычно его располагают после объявления закрытых членов (задаваемых по умолчанию) и перед объявлением открытых членов. Ниже показана полная основная форма объявления класса:

```
class имя_класса (
// закрытые члены класса
protected:
// защищенные члены класса
public :
// открытые члены класса
};
```

Когда базовый класс наследуется производным классом как открытый (public), защищенный член базового класса становится защищенным членом производного класса. Когда базовый класс наследуется как закрытый (private), то защищенный член базового класса становится закрытым членом производного класса. Базовый класс может также наследоваться производным классом как защищенный (protected). В этом случае открытые и защищенные члены базового класса становятся защищенными членами производного класса. (Естественно, что закрытые члены базового класса остаются закрытыми, и
они не доступны для производного класса.) Спецификатор доступа protected можно также использовать со структурами. Пример защищенного наследования:

```
class base {
protected: // закрытые члены класса base,
    int a,b; // но для производного класса они доступны
public:
    void setab(int n, int m) { a = n; b = m; }
};
class derived: protected base { // класс base наследуется
// как защищенный
    int c;
public:
    void setc(int n) { c = n; }
// эта функция имеет доступ к переменным a и b класса base
    void showabc () { cout << a << ' ' << b << ' ' << c << '\n'; }
int main()
{
    derived ob;
```

```
    // ОШИБКА:  функция setab()
    // является защищенным членом класса base
    ob.setab(l, 2); // функция setab() здесь недоступна
    ob.setc(3);
    ob . showabc();
    return 0;
}
```

Поскольку класс base наследуется как защищенный, его открытые и защищенные элементы становятся защищенными членами производного класса derived и следовательно внутри функции main() они недоступны.

8.3 Конструкторы, деструкторы и наследование

Базовый класс, производный класс или оба класса вместе могут иметь конструкторы и/или деструкторы. Если у базового и у производного классов имеются конструкторы и деструкторы, то конструкторы выполняются в порядке наследования, а деструкторы – в обратном порядке. Таким образом, конструктор базового класса выполняется раньше конструктора производного класса. Для деструкторов правилен обратный порядок: деструктор производного класса выполняется раньше деструктора базового класса. Последовательность выполнения конструкторов и деструкторов достаточно очевидна. Поскольку базовый класс "не знает" о существовании производного, любая инициализация выполняется в нем независимо от производного класса и возможно становится основой для любой инициализации, выполняемой в производном классе. Поэтому инициализация в базовом классе должна выполняться первой. С другой стороны, деструктор производного класса должен выполняться раньше деструктора базового класса потому, что базовый класс лежит в основе производного. Если бы деструктор базового класса выполнялся первым, это бы разрушило производный класс. Таким образом, деструктор производного класса должен вызываться до того, как объект прекратит свое существование.

Когда инициализация проводится только в производном классе, аргументы передаются обычным образом. Однако при необходимости передать аргумент конструктору базового класса ситуация несколько усложняется. Во-первых, все необходимые аргументы базового и производного классов передаются конструктору производного класса. Затем, используя расширенную форму объявления конструктора производного класса, соответствующие аргументы передаются дальше в базовый класс. Синтаксис передачи аргументов из производного в базовый класс показан ниже:

```
конструктор_произв_класса (список-арг) : базов_класс (спясок_арг)
{// тело конструктора производного класса}
```

Для базового и производного классов допустимо использовать одни и те же аргументы. Кроме этого, для производного класса допустимо игнорирование всех аргументов и передача их напрямую в базовый класс.

```
class base {
    int i;
public:
    base(int n) : i(n) {cout << "Работа конструктора базового класса \n";}
    ~base () { cout << "Работа деструктора базового класса \n"; }
    void showi() { cout << i << '\n'; }
};
class derived: public base {
int j;
public:
derived (int n, int m) : base (m), j(n) { // передача аргумента в базовый класс
cout << "Работа конструктора производного класса \n"; }
~derived() { cout << "Работа деструктора производного класса \n";}
void showj () { cout << j << '\n'; }
};
int main ()
{
derived o (10, 20);
return 0;
}
```

После выполнения программы на экран выводится следующее:

Работа конструктора базового класса
Работа конструктора производного класса
Работа деструктора производного класса
Работа деструктора базового класса

Как видно, конструкторы выполняются в порядке наследования, а деструкторы – в обратном порядке.

8.4 Множественное наследование

Имеются два способа, посредством которых производный класс может наследовать более одного базового класса. Во-первых, производный класс может использоваться в качестве базового для другого производного класса, создавая многоуровневую иерархию классов. В этом случае говорят, что исходный базовый класс является косвенным (indirect) базовым классом для второго производного класса. (Любой класс – независимо от того, как он создан – может использоваться в качестве базового класса.) Во-вторых, производный класс может прямо наследовать более одного базового класса. В такой ситуации соз-

данию производного класса помогает комбинация двух или более базовых классов. Ниже исследуются результаты, к которым приводит наследование нескольких базовых классов. Когда класс используется как базовый для производного, который, в свою очередь, является базовым для другого производного класса, конструкторы всех трех классов вызываются в порядке наследования. (Это положение является обобщением ранее исследованного принципа.) Деструкторы вызываются в обратном порядке. Таким образом, если класс B1 наследуется классом Dl, а D1 – классом D2**,** то конструктор класса B1 вызывается первым, за ним конструктор класса D1**,** за которым, в свою очередь, конструктор класса D2**.** Деструкторы вызываются в обратном порядке. Если производный класс напрямую наследует несколько базовых классов, используется такое расширенное объявление:

```
class имя_проязводяого_класса: сп_доступа имя_базового_класса1,
                               сп_досчупа. имя_базового_класса2,
                               ..., сп_досчупа имя_базового_классаN
{/* . . . тело класса */}
```

Здесь имя_базового_класса1 ... имя_базового_классаN – имена базовых классов, сп_доступа – спецификатор доступа, который может быть разным у разных базовых классов. Когда наследуется несколько базовых классов, конструкторы выполняются слева направо в том порядке, который задан в объявлении производного класса. Деструкторы выполняются в обратном порядке.

Когда класс наследует несколько базовых классов, конструкторам которых необходимы аргументы, производный класс передает эти аргументы. Если производный класс наследует иерархию классов, каждый производный класс должен передавать предшествующему в цепочке базовому классу все необходимые аргументы.

Традиционно программисты C++ изображают отношения наследования в виде прямых графов, стрелки которых направлены от производного к базовому классу. Пример множественного наследования, соответствующего графу показано на риснке 7.

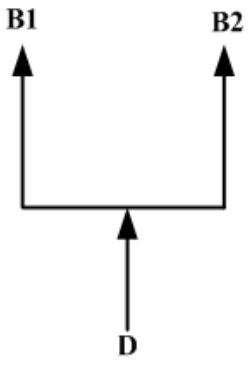


Рисунок 7 – Пример множественного наследования

```
class Bl {
public:
        Bl() { cout << "Работа конструктора класса B1\n"; }
        ~B1() { cout << "Работа деструктора класса B1\n"; }
};
class B2 {
public:
        B2() { cout << "Работа конструктора класса B2\n"; }
        ~B2() { cout << "Работа деструктора класса B2\n"; }
// Наследование двух базовых классов
};
class D: public Bl, public B2 {
public:
        D() { cout << "Работа конструктора класса D\n"; }
        ~D() { cout << "Работа деструктора класса D\n"; }
};
int main()
{
        D ob;
        return 0 ;
}
```

Эта программа выводит на экран следующее:

```
Работа конструктора класса B1
Работа конструктора класса B2
Работа конструктора класса D
Работа деструктора класса D
Работа деструктора класса B2
Работа деструктора класса B1
```

Когда прямо наследуются несколько базовых классов, конструкторы вызываются слева направо в порядке, задаваемом списком. Деструкторы вызываются в обратном порядке.

8.5 Виртуальные базовые классы

При многократном прямом наследовании производным классом одного и того же базового класса может возникнуть проблема. Чтобы понять, что это за проблема, рассмотрим следующую иерархию классов, представленную на рисунке 8.

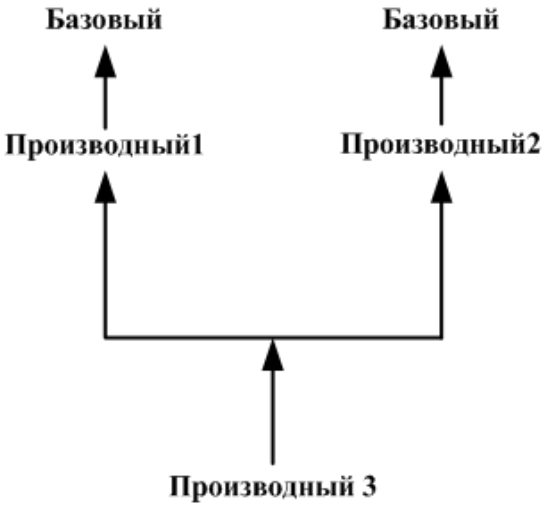


Рисунок 8 – Иерархия классов

Здесь базовый класс Базовый наследуется производными классами Производный1 и Производный2. Производный класс Производный3 прямо наследует производные классы Производный1 и Производный2. Однако это подразумевает, что класс Базовый фактически наследуется классом Производный3 дважды – первый раз через класс Производный1, а второй через класс Производный2. Однако, если член класса Базовый будет использоваться в классе Производный3, это вызовет неоднозначность. Поскольку в классе Производный3 имеется две копии класса Базовый, то будет ли ссылка на элемент класса Базовый относиться к классу Базовый, наследуемому через класс Производный1, или к классу Базовый, наследуемому через класс Производный2? Для преодоления этой неоднозначности в C++ включен механизм, благодаря которому в классе Производный3 будет включена только одна копия класса Базовый. Класс, поддерживающий этот механизм, называется виртуальным базовым классом (virtual base class).

В таких ситуациях, когда производный класс более одного раза косвенно наследует один и тот же базовый класс, появление двух копий базового класса в объекте производного класса можно предотвратить, если базовый класс наследуется как виртуальный для всех производных классов. Такое наследование не дает появиться двум (или более) копиям базового класса в любом следующем производном классе, косвенно наследующем базовый класс. В этом случае перед спецификатором доступа базового класса необходимо поставить ключевое слово virtual. Пример использования виртуального наследования:

```
class base {
public:
      int i;
};
// Наследование класса base как виртуального
```

```
class derivedl: virtual public base {
public:
    int j;
};
// Здесь класс base тоже наследуется как виртуальный
class derived2: virtual public base {
public:
    int k;
};
/* Здесь класс derived3 наследует как класс derivedl, так и класс
derived2. Однако в классе derived3 создается только одна копия
класса base
*/
class derived3: public derivedl, public derived2 {
public:
    int product() { return i * j * k; }
};
int main()
{
    derived3 ob;
    // Здесь нет неоднозначности, поскольку
    // представлена только одна копия класса base
    ob.i = 10;
    ob.j = 3;
    ob.k = 5;
    cout << "Результат равен " << ob. product () << '\n';
    return 0;
}
```

Если бы классы derived1 и derived2 наследовали класс base не как виртуальный, тогда инструкция ob.i = 10; вызывала бы неоднозначность и при компиляции возникла бы ошибка.

Даже если базовый класс наследуется производным как виртуальный, то копия этого базового класса все равно существует внутри производного. Например, по отношению к предыдущей программе этот фрагмент совершенно правилен:

```
berived1 ob;
ob.i = 100;
```

Отличие между обычным и виртуальным базовыми классами проявляется только тогда, когда объект наследует базовый класс более одного раза. Если используются виртуальные базовые классы, то в каждом конкретном объекте

присутствует копия только одного из них. В противном случае (при обычном наследовании) там было бы несколько копий.

8.6 Виртуальные функции

Виртуальные функции имеют важное значение в C++ потому, что они используются для поддержки одного из принципов ООП динамического полиморфизма (run-time polymorphism). В C++ полиморфизм поддерживается двумя способами. Во-первых, при компиляции он поддерживается посредством перегрузки операторов и функций. Во-вторых, во время выполнения программы он поддерживается посредством виртуальных функций. Основой виртуальных функций и динамического полиморфизма являются указатели на производные классы.

Указатель, объявленный в качестве указателя на базовый класс, также может использоваться, как указатель на любой класс, производный от этого базового. В такой ситуации представленные ниже инструкции являются правильными:

```
base *p; // указатель базового класса
base base_ob; // объект базового класса
derived derived ob; // объект производного класса
// Естественно, что указатель p может указывать
/ / на объект базового класса
p = &base_ob; // указатель p для объекта базового класса
// Кроме базового класса указатель p может указывать
/ / на объект производного класса
p = &derived_ob; // указатель p для объекта производного класса
```

Как отмечено в комментариях, указатель базового класса может указывать на объект любого класса, производного от этого базового и при этом ошибка несоответствия типов генерироваться не будет. Для указания на объект производного класса можно воспользоваться указателем базового класса, при этом доступ может быть обеспечен только к тем объектам производного класса, которые были унаследованы от базового. Объясняется это тем, что базовый указатель "знает" только о базовом классе и ничего не знает о новых членах, добавленных в производном классе. Указатель базового класса можно использовать для указания на объект производного класса, но обратный порядок недействителен. Указатель производного класса нельзя использовать для доступа к объектам базового класса. Чтобы обойти это ограничение, можно использовать приведение типов.

Виртуальная функция (virtual function) является членом класса. Она объявляется внутри базового класса и переопределяется в производном классе. Для того, чтобы функция стала виртуальной, перед объявлением функции ставится

ключевое слово virtual. Если класс, содержащий виртуальную функцию, наследуется, то в производном классе виртуальная функция переопределяется. По существу, виртуальная функция реализует идею "один интерфейс, множество методов", которая лежит в основе полиморфизма. Виртуальная функция внутри базового класса определяет вид интерфейса этой функции. Каждое переопределение виртуальной функции в производном классе определяет ее реализацию, связанную со спецификой производного класса. Таким образом, переопределение создает конкретный метод. При переопределении виртуальной функции в производном классе, ключевое слово virtual не требуется.

Виртуальная функция может вызываться так же, как и любая другая функция-член. Однако наиболее интересен вызов виртуальной функции через указатель, благодаря чему поддерживается динамический полиморфизм. Если указатель базового класса ссылается на объект производного класса, который содержит виртуальную функцию и для которого виртуальная функция вызывается через этот указатель, то компилятор определяет, какую версию виртуальной функции вызвать, основываясь при этом на типе объекта, на который ссылается указатель. При этом определение конкретной версии виртуальной функции имеет место не в процессе компиляции, а в процессе выполнения программы. Другими словами, тип объекта, на который ссылается указатель, и определяет ту версию виртуальной функции, которая будет выполняться. Поэтому, если два или более различных класса являются производными от базового, содержащего виртуальную функцию, то, если указатель базового класса ссылается на разные объекты этих производных классов, выполняются различные версии виртуальной функции. Этот процесс является реализацией принципа динамического полиморфизма. Фактически, о классе, содержащем виртуальную функцию, говорят как о полиморфном классе (polymorphic class). Пример использования виртуальных функций:

```
class base {
public:
    int i;
    base (int x) { i = x; }
    virtual void func() {
        cout << "Выполнение функции func() базового класса: ";
        cout << i << '\n';
    }
};
class derivedl : public base {
public:
    derivedl (int x) : base (x) { }
    void f unc ( ) {
```

```
            cout << "Выполнение функции func() класса derivedl: ";
            cout << i * i << '\n' ;
        }
};
class derived2: public base {
public: .
        derived2 (int x) : base(x) { }
        void f unc ( ){
            cout << "Выполнение функции funcO класса derived2: ";
            cout << i + i << ' \n ' ;
        }
};
int main ( )
{
        base *p;
        base ob(10);
        derivedl d_obl(10);
        derived2 d_ob2(10);
        p = &ob;
        p->func(); // функция funcO класса base
        p = &d_obl;
        p->func(); // функция func() производного класса derivedl
        p = &d_ob2;
        p->func(); // функция func() производного класса derived2
        return 0;
}
```

После выполнения программы на экране появится следующее:

```
Выполнение функции fun() базового класса: 10
Выполнение функции func() класса derivedl: 100
Выполнение функции func() класса derived2: 20
```

Ключевым для понимания примера является тот факт, что, во-первых, тип адресуемого через указатель объекта определяет вызов той или иной версии подменяемой виртуальной функции, во-вторых, выбор конкретной версии происходит уже в процессе выполнения программы.

Переопределение виртуальной функции внутри производного класса может показаться похожим на перегрузку функций. Однако эти два процесса совершенно различны. Во-первых, перегружаемая функция должна отличаться типом и/или числом параметров, а переопределяемая виртуальная функция должна иметь точно такой же тип параметров, то же их число, и такой же тип возвращаемого значения. (Если при переопределении виртуальной функции

изменяется число или тип параметров, она просто становится перегружаемой функцией и ее виртуальная природа теряется.) Далее, виртуальная функция должна быть членом класса. Это не относится к перегружаемым функциям. Кроме этого, если деструкторы могут быть виртуальными, то конструкторы нет. Чтобы подчеркнуть разницу между перегружаемыми функциями и переопределяемыми виртуальными функциями, для описания переопределения виртуальной функции используется термин подмена (overriding).

8.7 Чистые виртуальные функции и абстрактный класс

Если в виртуальной функции базового класса отсутствует значимое действие, в любом классе, производном, от этого базового, такая функция обязательно должна быть переопределена. Для реализации этого положения в C++ поддерживаются так называемые чистые виртуальные функции (pure virtual function). Чистые виртуальные функции не определяются в базовом классе. Туда включаются только прототипы этих функций. Для чистой виртуальной функции используется такая основная форма:

virtual тип имя_функции (саписок_параметров) = 0;

Ключевой частью этого объявления является приравнивание функции нулю. Это сообщает компилятору, что в базовом классе не существует тела функции. Если функция задается как чистая виртуальная, это предполагает, что она обязательно должна подменяться в каждом производном классе. Если этого нет, то при компиляции возникнет ошибка. Таким образом, создание чистых виртуальных функций – это путь, гарантирующий, что производные классы обеспечат их переопределение. Если класс содержит хотя бы одну чистую виртуальную функцию, то о нем говорят как об **абстрактном классе** (abstract class). Поскольку в абстрактном классе содержится, по крайней мере, одна функция, у которой отсутствует тело функции, технически такой класс неполон, и ни одного объекта этого класса создать нельзя. Таким образом, абстрактные классы могут быть только наследуемыми. Однако по-прежнему можно создавать указатели абстрактного класса, благодаря которым достигается динамический полиморфизм. Пример абстрактного класса:

```
class area {
        double diml, dim2; // размеры фигуры
public:
        void setarea (double dl, double d2) {diml = dl; dim2 = d2;}
        void getdim (double &dl, double &d2) {dl = diml; d2 = dim2;}
        virtual double getarea() = 0; // чистая виртуальная функция
};
class rectangle: public area {
public:
```

```
        double getarea()
        {
            double dl, d2;
            getdim (dl, d2);
            return dl * d2;
        }
};
int main()
{
        area *p;
        rectangle r;
        r.setarea(3.3, 4.5);
        p = &r;
        cout << "Площадь прямоугольника: " << p->getarea() << '\n';
}
```

То, что функция getarea() является чистой виртуальной, гарантирует ее обязательную подмену в каждом производном классе.

8.8 Применение полиморфизма

Полиморфизм является процессом, благодаря которому общий интерфейс применяется к двум или более схожим (но технически разным) ситуациям, т. е. реализуется философия " один интерфейс, множество методов". Полиморфизм важен потому, что может сильно упростить сложные системы. Один хорошо определенный интерфейс годится для доступа к некоторому числу разных, но связанных по смыслу действий, и таким образом устраняется искусственная сложность. Если связанные действия реализуются через общий интерфейс, нужно гораздо меньше помнить. Имеются два термина, которые часто ассоциируются с ООП вообще и с C++ в частности. Этими терминами являются **раннее связывание** (early binding) и **позднее связывание** (late binding). Важно понимать, что означают указанные термины. Раннее связывание относится к событиям, о которых можно узнать в процессе компиляции. Особенно это касается вызовов функций, которые настраиваются при компиляции. Функции раннего связывания – это "нормальные" функции, перегружаемые функции, невиртуальные функции-члены и дружественные функции. При компиляции функций этих типов известна вся необходимая для их вызова адресная информация. Главным преимуществом раннего связывания (и доводом в пользу его широкого использования) является то, что оно обеспечивает высокое быстродействие программ. Определение нужной версии вызываемой функции во время компиляции программы – это самый быстрый метод вызова функций. Главный недостаток – потеря гибкости. Позднее связывание относится к событиям, которые

происходят в процессе выполнения программы. Вызов функции позднего связывания – это вызов, при котором адрес вызываемой функции до запуска программы неизвестен. В C++ виртуальная функция является объектом позднего связывания. Если доступ к виртуальной функции осуществляется через указатель базового класса, то в процессе работы программа должна определить, на какой тип объекта он ссылается, а затем выбрать, какую версию подменяемой функции выполнить. Главным преимуществом позднего связывания является гибкость во время работы программы. Ваша программа может легко реагировать на случайные события. Его основным недостатком является то, что требуется больше действий для вызова функции. Это обычно делает такие вызовы медленнее, чем вызовы функций раннего связывания. В зависимости от нужной эффективности, следует принимать решение, когда лучше использовать раннее связывание, а когда – позднее.

9 ШАБЛОНЫ КЛАССОВ

Шаблоны классы полезны, когда класс содержит общую логику работы. Например, алгоритм, который реализует очередь целых, будет также работать и с очередью символов. С помощью шаблона класса можно создать класс, реализующий очередь, связанный список и т. д. для любых типов данных. Компилятор будет автоматически генерировать правильный тип объекта на основе типа, заданного при создании объекта. Ниже представлена основная форма объявления шаблона класса:

template <typename/class тип> class имя_класса {}

Здесь тип – это фиктивное имя типа, который будет задан при создании экземпляра класса. При необходимости можно определить более одного шаблона типа данных, разделяя их запятыми. После создания шаблонного класса с помощью представленной ниже формы можно создать конкретный экземпляр этого класса:

имя_класса <тип> объект;

Здесь тип – это имя типа данных, с которым будет оперировать класс. Функции-члены шаблонного класса сами автоматически становятся шаблонными. Для них не обязательно явно задавать ключевое слово template.

В C++ имеется встроенная библиотека классов-шаблонов, которая называется библиотекой стандартных шаблонов (Standard Template Library, STL). Эта библиотека предоставляет шаблоны классов для наиболее часто используемых алгоритмов и структур данных. Пример шаблона односвязного списка:

```
template <class data_t> class list {
        data_t data;
        list *next;
public:
        list (data_t d) ;
        void add (list *node) { node->next = this; next =0; }
        list *getnext() { return next; }
        data_t getdata() { return data; }
};
template <class data_t> list<data_t>: : list (data_t d)
{
        data = d;
        next = 0;
};
int main ( )
{
```

```
        list<char> start ('a');
        list<char> *p, *last;
        int i;
        // создание списка
        last = &start;
        for(i=l; i<26; i++) {
                p = new list<char> ('a' + i) ;
                p->add(last) ;
                last = p;
        }
        // вывод списка
        p = &start;
        while (p) {
                cout << p->getdata ();
                p = p->getnext ();
        }
        return 0;
}
```

Объявление шаблонного класса похоже на объявление родовой функции. Тип данных, хранящихся в списке, становится шаблонным в объявлении класса. Но он не проявляется, пока не объявлен объект, который и задает реальный тип данных. В данном примере объекты и указатели создаются внутри функции main(), где указывается, что типом хранящихся в списке данных является тип char. В зависимости от типа можно создавать списки для хранения любых данных: list<double> start_double_list(12.2); list<string> start_string_list(“абв”); и.т.д.

10 СИСТЕМА ВВОДА/ВЫВОДА С++

Система ввода/вывода С++, так же, как система ввода/вывода С, действует через потоки (streams). Поток ввода/вывода – это логическое устройство, которое выдает и принимает пользовательскую информацию. Поток связан с физическим устройством с помощью системы ввода/вывода С++. Поскольку все потоки ввода/вывода действуют одинаково, то, несмотря на то, что программисту приходится работать с совершенно разными по характеристикам устройствами, система ввода/вывода предоставляет для этого единый удобный интерфейс. Например, функция, которая используется для записи информации на экран монитора, вполне подойдет как для записи в файл, так и для вывода на принтер.

Когда запускается программа на С++, автоматически открываются четыре потока, представленные в таблице 4.

Таблица 4 – Потоки призапуске программы на С++

Поток	Значение	Устройство по умолчанию
cin	Стандартный ввод	Клавиатура
cout	Стандартный вывод	Экран
cerr	Стандартная ошибка	Экран
clog	Буферизуемая версия cerr	Экран

По умолчанию, стандартные потоки используются для связи с клавиатурой и экраном. Однако в среде, в которой поддерживается переопределение ввода/вывода, эти потоки могут быть перенаправлены на другие устройства.

В С++ ввод/вывод обеспечивается подключением к программе заголовочного файла #include<iostream.h>**.** В этом файле определены сложные наборы иерархий классов, поддерживающие операции ввода/вывода. Система ввода/вывода С++ строится на двух связанных, но различных иерархиях классов-шаблонов. Первая является производной от класса нижнего уровня basic_streambuf. Этот класс предоставляет базу для операций нижнего уровня по вводу и выводу, а также обеспечивает надлежащую поддержку всей системы ввода/вывода С++. Иерархия классов реализующих базовые функции является производной от класса basic_ios. Это класс ввода/вывода верхнего уровня, который обеспечивает форматирование, контроль ошибок и информацию о состоянии потока ввода/вывода. Класс basic_ios является базовым для нескольких производных классов, среди которых классы basic_istream, basic_ostream и basic_iostream. Эти классы используются соответственно для создания потоков ввода, вывода и ввода/вывода.

10.1 Форматируемый ввод/вывод

Каждый поток ввода/вывода связан с набором флагов формата (format flags), которые управляют способом форматирования информации и представляют собой битовые маски (bitmasks). Эти маски объявлены в классе ios как данные перечислимого типа fmtflags, в котором определены следующие значения: adjustfield, basefield, boolalpha, dec, fixed, floatfield, hex, internal, left, oct, right, scientific, showbase, showpoint, showpos, skipws, unitbuf, uppercase. Эти значения определены в классе ios и необходимы для установки или сброса флагов формата.

Когда при вводе информации в поток установлен флаг skipws, начальные невидимые символы (пробелы, табуляции и символы новой строки) отбрасываются. Когда флаг skipws сброшен, невидимые символы не отбрасываются.

Когда установлен флаг left, происходит выравнивание вывода по левому краю. Когда установлен флаг right, происходит выравнивание вывода по правому краю. Когда установлен флаг internal, для заполнения поля вывода происходит вставка пробелов между всеми цифрами и знаками числа. Если все эти флаги не установлены, то по умолчанию используется выравнивание по правому краю.

По умолчанию числовые значения выводятся в десятичной системе счисления. Однако основание системы счисления можно поменять. Установка флага oct ведет к тому, что вывод будет осуществляться в восьмеричной системе счисления, а установка флага hex – в шестнадцатеричной. Чтобы вернуться к десятичной системе счисления, установите флаг dec.

Установка флага showbase ведет к выводу основания системы счисления. Например, шестнадцатеричное значение 1F с этим флагом будет выводиться как 0x1F.

По умолчанию при выводе значений в научной нотации символ "e" выводится в нижнем регистре. Кроме этого, при выводе шестнадцатеричного значения символ "x" тоже выводится в нижнем регистре. При установке флага uppercase, эти символы выводятся в верхнем регистре.

Установка флага showpos приводит к выводу знака + перед положительными значениями.

Установка флага showpoint ведет к появлению десятичной точки и последующих нулей при выводе любых значений с плавающей точкой.

При установке флага scientific числа с плавающей точкой выводятся в научной нотации.

При установке флага fixed числа с плавающей точкой выводятся в обычной нотации. Если ни один из этих флагов не установлен, компилятор сам выбирает подходящий способ вывода.

Если установлен флаг unitbuf, то буфер очищается (flush) после каждой операции вставки (insertion operation). При установленном флаге boolalpha значения булева типа выводятся в виде ключевых слов true и false.

Одновременно на все поля, определенные с флагами oct, dec и hex, можно сослаться с помощью флага basefield. Аналогично на поля, определенные с флагами left, right и internal, можно сослаться с помощью флага adjustfield. И наконец, на поля с флагами scientific и fixed можно сослаться с помощью флага floatfield.

Для установки флага формата используется функция setf(). Эта функция является членом класса ios. Здесь показана ее основная форма:

fmtflags setf(fmtflags флаги);

Эта функция возвращает предыдущие установки флагов формата и устанавливает новые, заданные значением флаги. (Значения всех остальных флагов не изменяются.) Например, для установки флага showpos можно воспользоваться следующей инструкцией:

поток_ввода/вавода.setf(ios::showpos);

Каждый поток ввода/вывода поддерживает собственную информацию о состоянии формата. Вместо повторных вызовов функции setf() в одном вызове можно установить сразу несколько флагов. Для объединения необходимых флагов используется оператор OR. Например, в следующем вызове функции setf() для потока cout устанавливаются флаги showbase и hex:

cout.setf(ios::showbase | ios::hex);

Дополнением setf() является функция unsetf(). Эта функция-член класса ios сбрасывает один или несколько флагов формата.

void unsetf(fmtflags флаги);

Флаги, заданные параметром флаги, сбрасываются. (Все остальные флаги остаются без изменений.)

Пример установки нескольких флагов формата:

```
int main()
{
    cout << 123.23 << " привет " << 100 << '\n';
    cout << 10 << ' ' << -10 << '\n';
    cout << 100.0 << '\n';
    cout.unsetf(ios::dec);
    cout.setf(ios::hex | ios::scientific);
    cout << 123.23 << " привет " << 100 << '\n';
    cout.setf(ios::showpos);
    cout << 10 << ' ' << -10 << '\n';
    cout.setf(ios::showpoint | ios::fixed);
    cout << 100.0;
```

```
        return 0;
    }
```

После выполнения программы на экран выводится следующее:

```
123.23 привет 100
10 -10
100
1.232300e+02 привет 64
a fffffff6
+100.000000
```

Кроме флагов формата в классе ios определены три функции-члена. Эти функции устанавливают следующие параметры формата: ширину поля, точность и символ заполнения. Этими функциями являются соответственно функции width(), precision() и fill(). По умолчанию при выводе любого значения оно занимает столько позиций, сколько символов выводится. Однако с помощью функции width() можно задать минимальную ширину поля. Ниже показан прототип этой функции:

```
streamsize width(strearnsize w);
```

Ширина поля задается параметром w, а функция возвращает предыдущую ширину поля. Тип данных streamsize определен в заголовочном файле <iostream.h> как одна из форм целого. В некоторых компиляторах при выполнении каждой операции вывода значение ширины поля возвращается к своему состоянию по умолчанию, поэтому перед каждой инструкцией вывода может понадобиться устанавливать минимальную ширину поля. После установки минимальной ширины поля, если выводимое значение требует поле, меньшее заданной ширины, остаток поля заполняется текущим символом заполнения (по умолчанию пробелом) так, чтобы была занята вся ширина поля. Если размер выводимого значения превосходит минимальную ширину поля, будет занято столько символов, сколько нужно. Выводимое значение не усекается.

По умолчанию при выводе значений с плавающей точкой точность равна шести цифрам. Однако с помощью функции precision() это число можно изменить. Ниже показан прототип функции precision():

```
streamsize precision(streamsize p);
```

Точность (число выводимых цифр после запятой) задается параметром p, а возвращает функция прежнее значение точности.

По умолчанию, если требуется заполнить свободные поля, используются пробелы. Однако с помощью функции fill() можно изменить символ заполнения. Ниже показан прототип функции fill():

```
char fill(char ch);
```

После вызова функции fill() символ ch становится новым символом заполнения, а функция возвращает прежнее значение символа заполнения. Пример работы функций форматирования:

```
int main()
{
    cout.width(10); // установка минимальной ширины поля
    ccmt << "Привет" << '\n'; // по умолчанию выравнивание вправо
    cout.fill('%'); // установка символа заполнения
    cout.width(10); // установка ширины поля
    cout << "Привет" << '\n'; // по умолчанию выравнивание вправо
    cout.setf (ios::left) ; // выравнивание влево
    cout.width(10}; // установка ширины поля
    cout << "Привет" << '\n'; // выравнивание влево
    cout. width (10); // установка ширины поля
    cout. precision (10) ; // установка точности в 10 цифр
    cout << 123.234567 <<  '\n';
    cout .width (10) ; // установка ширины поля
    cout .precision (6) ; // установка точности в 6 цифр
    cout << 123.234567 << '\n';
    return 0;
}
```

После выполнения программы на экран выводится следующее:

```
Привет
%%%%Привет
Привет%%%%
123.234567
123.235%%%
```

Ширина поля устанавливается перед каждой инструкцией вывода.

10.2 Манипуляторы ввода/вывода

В системе ввода/вывода C++ имеется еще один способ форматирования информации. Этот способ подразумевает использование специальных функций – манипуляторов ввода/вывода (I/O manipulators). Манипуляторы ввода/вывода являются, в некоторых ситуациях, более удобными, чем флаги и функции формата класса ios.

Манипуляторы ввода/вывода являются специальными функциями формата ввода/вывода, которые, в отличие от функций – членов класса ios, могут располагаться внутри инструкций ввода/вывода. Например:

```
cout << oct << 100 << hex << 100;
```

cout << setw(10) << 100;

Первая инструкция сообщает потоку cout о необходимости вывода целых в восьмеричной системе счисления и выводит число 100 в восьмеричной системе счисления. Затем она сообщает потоку ввода/вывода о необходимости вывода целых в шестнадцатеричной системе счисления и далее осуществляется вывод числа 100 уже в шестнадцатеричном формате. Во второй инструкции устанавливается ширина поля равная 10, и затем снова выводится 100 в шестнадцатеричном формате.

Список основных манипуляторов ввода/выводы приведен в таблице 5.

Манипулятор ввода/вывода влияет только на поток, частью которого является выражение ввода/вывода, содержащего манипулятор. Манипуляторы ввода/вывода не влияют на все, открытые в данный момент, потоки.

Таблица 5 – Основные манипуляторы ввода/выводы

Манипулятор	Назначение	Ввод/Вывод
dec	Установка флага dec	Ввод/Вывод
endl	Вывод символа новой строки и очистка потока	Вывод
ends	Вывод значения NULL	Вывод
flush	Очистка потока	Вывод
hex	Установка флага hex	Ввод/Вывод
left	Установка флага left	Вывод
oct	Установка флага oct	Ввод/Вывод
right	Установка флага right	Вывод
setw(int w)	Задание ширины поля равным w позиций	Вывод
scientific	Установка флага scientific	Вывод
showpos	Установка флага showpos	Вывод
setprecision(int p)	Задание числа цифр точности равным p	Вывод

10.3 Основы файлового ввода/вывода

Файловый и консольный ввод/вывод очень близко связаны. Фактически файловый ввод/вывод поддерживается той же иерархией классов, что и консольный ввод/вывод.

Для реализации файлового ввода/вывода, необходимо включить в программу заголовок <fstream>. В нем определено несколько классов, включая классы ifstream, ofstream и fstream. Эти классы являются производными от классов istream и ostream. Вспомните, что классы istrcam и ostream, в свою очередь, являются производными от класса ios, поэтому классы ifstream, ofstream и fstream также имеют доступ ко всем операциям, определяемым классом ios.

В C++ файл открывается посредством его связывания с потоком. Имеется три типа потоков: ввода, вывода и ввода/вывода. Перед тем как открыть файл, нужно, во-первых, создать поток. Для создания потока ввода необходимо объя-

вить объект типа ifstream. Для создания потока вывода – объект типа ofstream. Потоки, которые реализуют одновременно ввод и вывод, должны объявляться как объекты типа fstream. Например, в следующем фрагменте создается один поток для ввода, один поток для вывода и еще один поток одновременно для ввода и для вывода:

```
ifstream in; // ввод
ofstream out; // вывод
fstream io; // ввод и вывод
```

После создания потока, одним из способов связать его с файлом является функция open(). Эта функция является членом каждого из трех потоковых классов. Здесь показаны ее прототипы для каждого класса:

```
void ifstream::open(const char *имя_файла,
                    openmode режим = ios::in);
void ofstream::open(const char *имя_файла,
                    openmode режим = ios::out | ios::trunc);
void fstream::open(const char *имя_файла,
                    openmode режим = ios::in | ios::out);
```

Здесь имя_файла – имя файла, в которое может входить и спецификатор пути. Значение “режим” задает режим открытия файла. Оно должно быть значением типа openmode, которое является перечислением, определенным в классе ios. Значение режим может быть одним из следующих: ios::app, ios::ate, ios::binary, ios::in, ios::out, ios::trunk. Можно объединить два или более этих значения с помощью оператора OR.

Значение ios::app вызывает открытие файла в режиме добавления в конец файла. Это значение может применяться только к файлам, открываемым для вывода. Значение ios::ate задает режим поиска конца файла при его открытии. Хотя значение ios::ate вызывает поиск конца файла, тем не менее, операции ввода/вывода могут быть выполнены в любом месте файла.

Значение ios::in задает режим открытия файла для ввода. Значение ios::out задает режим открытия файла для вывода.

Значение ios::binary вызывает открытие файла в двоичном режиме. По умолчанию все файлы открываются в текстовом режиме. В текстовом режиме имеет место преобразование некоторых символов. Любой файл, независимо от того, что в нем содержится – отформатированный текст или необработанные данные – может быть открыт как в текстовом, так и в двоичном режиме.

Значение ios::trunc приводит к удалению содержимого ранее существовавшего файла с тем же названием и усечению его до нулевой длины. При создании потока вывода с помощью ключевого слова ofstream любой ранее существовавший файл с тем же именем автоматически усекается до нулевой длины. В следующем фрагменте для вывода открывается файл test:

```
ofstream mystream;
mystream.open("test");
```

В этом примере параметр режим функции open() по умолчанию устанавливается в значение, соответствующее типу открываемого потока, поэтому нет необходимости указывать его явно.

Если выполнение функции open() завершилось с ошибкой, в булевом выражении поток будет равен значению false. Этот факт можно использовать для проверки правильности открытия файла с помощью, например, такой инструкции:

```
if(!mystream) cout << "Файл открыть невозможно \n";
```

Как правило, перед тем как пытаться получить доступ к файлу, следует проверить результат выполнения функции open().

Проверить правильность открытия файла можно также с помощью функции is_open(), являющейся членом классов ifstream, ofstream и fstream. Ниже показан прототип этой функции:

```
bool is_open();
```

Функция возвращает истину, если поток удалось связать с открытым файлом, в противном случае функция возвращает ложь. Например, в следующем фрагменте проверяется, открыт ли файл, связанный с потоком mystream:

```
if (!mystream.is_open())  cout << "Файл не открыт \n";
```

Классы ifstream, ofstream и fstream имеют конструкторы, которые открывают файл автоматически. Конструкторы имеют те же параметры, в том числе и задаваемые по умолчанию, что и функция open(). Поэтому чаще используется такой способ открытия файла:

```
ifstream mystream ("myfile") ; // открытие файла для ввода
```

Если по каким-то причинам файл не открывается, переменная, соответствующая потоку, в условной инструкции будет равна значению false. Поэтому, независимо от того, используете ли вы конструктор или явно вызываете функцию open(), вам потребуется убедиться в успешном открытии файла путем проверки значения потока.

Для закрытия файла используйте функцию-член close(). Например, чтобы закрыть файл, связанный с потоком mystream, необходима следующая инструкция:

```
mystream.close();
```

Функция close() не имеет параметров и возвращаемого значения.

С помощью функции eof(), являющейся членом класса ios, можно определить, был ли достигнут конец файла ввода. Ниже показан прототип этой функции:

```
bool eof ();
```

Функция возвращает истину, если был достигнут конец файла; в противном случае функция возвращает ложь.

После того как файл открыт, очень легко считать из него или записать в него текстовые данные. Для этого используются операторы << и >> так же, как это делалось для консольного ввода/вывода, только меняется поток cin или cout тем потоком, который связан с файлом. Так же, как и операторы << и >> для чтения из файла и записи в файл годятся функции C – fprintf() и fscanf(). Вся информация в файле хранится в том же формате, как если бы она находилась на экране. Следовательно, файл, созданный с помощью оператора <<, представляет из себя файл с отформатированным текстом, и наоборот, любой файл, содержимое которого считывается с помощью оператора <<, должен быть файлом с отформатированным текстом. То есть, как правило, файлы с отформатированным текстом, следует открывать в текстовом, а не в двоичном режиме. Пример использования файлового ввода/вывода:

```
int main()
{
    ofstream fout ("test") ; // создание файла для вывода
    if (!fout)
    {
        cout << "Файл открыть невозможно\n";
        return 1;
    }
    fout << "Привет! \n";
    fout << 100 << ' ' << hex << 100 << endl;
    fout. close () ;
    ifstream fin("test"); // открытие файла для ввода
    if(!fin)
    {
        cout << "Файл открыть невозможно\n";
        return 1;
    }
    char str[80];
    int i;
    fin >> str >> i;
    cout << str << ' ' << i << endl;
    fin.close();
    return 0;
}
```

Текстовые файлы (т. е. файлы, информация в которых представлена в кодах ASCII) полезны во многих ситуациях, но у них нет гибкости неформатированных двоичных файлов. Неформатированные файлы содержат те самые исходные или "сырые" двоичные данные, которые непосредственно используются

программой, а не удобный для восприятия человека текст, данные для которого транслируются операторами << и >>. Поэтому о неформатируемом вводе/выводе иногда говорят как о "сыром" (raw) вводе/выводе. В C++ для двоичных файлов поддерживается широкий диапазон функций ввода/вывода. Эти функции дают возможность точно контролировать процессы считывания из файлов и записи в файлы.

На нижнем уровне двоичного ввода/вывода находятся функции get() и put(). **С** помощью функции-члена put() можно записать байт; а с помощью функции-члена get() – считать. Эти функции являются членами всех потоковых классов соответственно для ввода и для вывода. Функции get() и put() имеют множество форм. Ниже приведены их наиболее часто встречающиеся версии:

```
istream &get(char &символ);
ostream &put(char символ);
```

Функция get() считывает один символ из связанного с ней потока и передает его значение аргументу символ. Ее возвращаемым значением является ссылка на поток. При считывании символа конца файла функция возвратит вызывающему потоку значение false. Функция put() записывает символ в поток и возвращает ссылку на поток. Для считывания и записи блоков двоичных данных используются функции read() и write(), которые также являются членами потоковых классов соответственно для ввода и для вывода. Здесь показаны их прототипы:

```
istream &read(char *буфер, streamsize число_байт);
ostream Swrite(const char *буфер, streamsize число_байт);
```

Функция read() считывает из вызывающего потока столько байтов, сколько задано в аргументе число_байт и передает их в буфер, определенный указателем буфер. Функция write() записывает в соответствующий поток из буфера, который определен указателем буфер, заданное в аргументе число_байт число байтов. Значения типа streamsize представляют собой некоторую форму целого. Если конец файла достигнут до того, как было считано число_байт символов, выполнение функции read() просто прекращается, а в буфере оказывается столько символов, сколько их было в файле. Узнать, сколько символов было считано, можнос помощью другой функции-члена gcount(), прототип которой приведен ниже:

```
streamsize gcount();
```

Функция возвращает количество символов, считанных во время последней операции двоичного ввода. Пример использования неформатированного двоичного ввода/вывода

```
int main(int argc, char *argv[])
{
    char ch;
    if(argc!=2)
```

```
{
        cout << "Запись: <имя_файла>\n";
        return 1;
}
ofstream out (argv[l] , ios::out | ios::binary)
if (!out)
{
        cout << "Файл открыть невозможно\n";
        return 1;
}
cout << "Для остановки введите символ $ \n";
do {
        cout << ":";
        cin. get (ch);
        out. put (ch);
} while (ch != '$');
out. close();
return 0;
}
```

Использование функции get() предотвращает игнорирование начальных пробелов.

11 КОНТРОЛЬНЫЕ РАБОТЫ

В учебном пособии составлены 4 контрольные работы. Студенту необходимо представить отчёт о выполненных контрольных работах в распечатанном виде и в электронном виде на любом носителе информации.

Отчёт должен включать: титульный лист, условие задачи, алгоритм решения (при необходимости), программу и результаты. Во время защиты контрольных работ студент должен подтвердить работоспособность программной реализации заданий. На титульном листе отчёта о выполнении контрольных работ необходимо указать фамилию, имя и отчество студента, номер учебной группы, номер контрольной работы, номер варианта.

Номер варианта соответствует номеру первой буквы фамилии студента согласно таблице 6.

Таблица 6 – Распределение вариантов заданий

Первая буква фамилии студента	Номер варианта	Первая буква фамилии студента	Номер варианта
А	1	П	15
Б	2	Р	16
В	3	С	17
Г	4	Т	18
Д	5	У	19
Е, Ё	6	Ф	20
Ж	7	Х	21
З	8	Ц	22
И, Й	9	Ч	23
К	10	Ш, Щ	24
Л	11	Э	25
М	12	Ю	26
Н	13	Я	27
О	14	Пример решения	28

11.1 КОНТРОЛЬНАЯ РАБОТА №1. Теоретические вопросы по программированию на языке высокого уровня С++

11.1.1 Задания контрольной работы №1

Вариант №1

1) Что такое программирование и язык программирования? Перечислите основные языки программирования высокого уровня и их особенности.
2) Для чего применяется указатель на функцию, особенности использования?
3) Чем отличается конструктор копирования от обычного конструктора?

Вариант №2

1) Что такое парадигма программирования? Какие парадигмы программирования существуют?
2) В чем отличие передачи параметров по значению и по ссылке? В чем особенность использования указателя и ссылки в качестве параметров функции?
3) В каких случаях необходима перегрузка оператора присваивания?

Вариант №3

1)Что такое императивное программирование, какие парадигмы программирования к нему относятся?
2) Что такое исключения? Какие преимущества имеет механизм исключений относительно способа обработки ошибок в С?
3) Какой порядок вызовов конструкторов при наследовании?

Вариант №4

1) Что такое декларативное программирование, какие парадигмы к нему относятся?
2) Что такое функция? Как объявляются функции?
3) Для чего используется множественное наследование? Как объявляется множественное наследование?

Вариант №5

1) Что такое массив? Как вычисляется размер массива? Какая связь массива с указателем?
2) Что такое класс (class), определение класса?
3) Для чего необходимы виртуальные базовые классы?

Вариант №6

1) Что такое структура (struct) с точки зрения структурного программирования? Какая существует связь структуры и класса (class)?
2) Для чего используются статические (static) и константные (const) поля классов?
3) Какой порядок вызовов деструкторов при наследовании?

Вариант №7

1) Что такое тип данных, какие существуют типы данных?
2) Для чего необходимы оператор выражения и составной оператор?
3) Чем абстрактный класс отличается от обычного класса?

Вариант №8

1) В чем отличия явной инициализации данных от неявной?
2) Для чего необходимы условный оператор и оператор безусловного перехода?
3) Какие существуют области видимости переменных?

Вариант №9

1) Что такое объединение (union), особенности его использования, вычисление размера объединения?
2) Для чего необходим оператор-переключатель?
3) Для чего используются чистые виртуальные функции?

Вариант №10

1) Какие скалярные типы данных существуют? Укажите размер и диапазон значений скалярных типов данных для вашей машины?
2) Что такое методы класса и как они объявляются?
3) Каким образом обеспечивается однократное включение заголовочных файлов?

Вариант №11

1) Что такое выражение, приоритеты, ассоциативность? Как можно изменить порядок вычисления выражений?
2) Что такое закрытые (private) члены класса, особенности их использования?
3) Зачем применяются виртуальные деструкторы?

Вариант №12

1) Какие существуют операции присваивания? Назовите их приоритет и ассоциативность?

2) Что такое открытые (public) члены класса, и в чём заключаются особенности их использования?
3) Какие классы используются для файлового ввода/вывода?

Вариант №13

1) Перечислите составные типы данных.
2) Какие существуют операторы цикла? Укажите особенности использования каждого оператора цикла?
3) Для чего используются виртуальные функции?

Вариант №14

1) Перечислите арифметические операции. Укажите их приоритет и ассоциативность.
2) Опишите оператор завершения и продолжения.
3) Как реализуется однократное наследование?

Вариант №15

1) Перечислите логические операции. Укажите их приоритет и ассоциативность.
2) Для чего используются и как объявляются шаблоны функций?
3) Какие основные функции используются для неформатированного доступа к файлу?

Вариант №16

1) Перечислите операции сравнения. Укажите их приоритет и ассоциативность.
2) Что такое защищенные (protected) члены класса? В чем заключаются особенности их использования?
3) Как реализуется открытое наследование? В чем его особенность?

Вариант №17

1) Перечислите битовые операции. Укажите их приоритет и ассоциативность.
2) По каким параметрам возможна перегрузка функций?
3) Как создаются многомерные динамические массивы с применением операций (new[], delete[])?

Вариант №18

1) Что такое указатель? В чем особенности его использования?
2) Укажите отличие встраиваемых функции от обычных.
3) Какие существуют классы памяти?

Вариант №19

1) Опишите операции для работы с динамической памятью (new, delete).
2) Что такое конструктор класса, для чего он используется?
3) Как реализуется защищенное наследование? Укажите его особенности.

Вариант №20

1) Опишите тернарную условную операцию, ее приоритет и ассоциативность.
2) Что означает ключевое слово this и для чего оно используется?
3) Как объявляется шаблон класса?

Вариант №21

1) Что такое ссылка? В чем заключаются особенности ее использования?
2) Опишите особенности функций с параметрами по умолчанию и функций с переменным количеством параметров?
3) Для чего используется ключевое слово void?

Вариант №22

1) В чем заключается концепция объектно-ориентированного программирования? Что такое полиморфизм?
2) Опишите рекурсивные функции, особенности их применения. Укажите альтернативный вариант реализации рекурсивных вычислений без использования рекурсивных функций.
3) Как реализуется закрытое наследование и в чём его особенности?

Вариант №23

1) Для чего используется приведение типов? Укажите особенности const_cast и reinterperet_cast.
2) Что такое деструктор класса и для чего он используется?
3) Как создаются битовые поля? Для чего используется безымянное битовое поле?

Вариант №24

1) Опишите перечисление (enum), особенности его использования. Какие преимущества имеет перечисление перед целочисленными константами?
2) Что такое дружественная функция и для чего она используется?
3) Для чего используются манипуляторы ввода/вывода?

Вариант №25

1) Опишите явное и неявное приведение типов. Укажите особенности static_cast, dynamic_cast.

2) Что такое дружественный класс, для чего он используется?
3) Для чего используется специализации шаблонов классов?

Вариант №26

1) В чем заключается концепция объектно-ориентированного программирования? Что такое инкапсуляция?
2) В чем особенность использования массивов в качестве параметров функции?
3) Для чего используются функции форматированного ввода/вывода: width, fill, precision?

Вариант №27

1) В чем заключается концепция объектно-ориентированного программирования? Что такое наследование?
2) Опишите операцию получения размера (sizeof), а также её использование применительно к скалярным типам данных, массивам и объектам?
3) Что такое перегрузка операторов? Какие операции нельзя перегружать?

Вариант №28

1) Что такое оператор языка программирования C++? Перечислите операторы, существующие в C++.
2) Для чего используется спецификатор volatile?
3) Для чего используется спецификатор mutable?

11.1.2 Пример выполнения контрольной работы № 1

Вариант № 28

1) Оператор языка программирования С++ – это мельчайшая независимая часть программы.

Основные операторы С++ приведены в таблице 7.

Таблица 7 – Основные операторы С++

;	Пустой оператор, при выполнении не имеет никакого эффекта;
e;	Оператор-выражение, вычисляет выражение e;
{}	Составной оператор или блок, выполняет операторы в блоке один за другим. Переменные, определенные в блоке, разрушаются по окончанию блока;
if(*условие*) *оператор1* else *оператор2*	Вычисляет *условие* и выполняет элемент *оператор1*, если *условие* имеет значение true, в противном случае выполняет *оператор2*;
while(*условие*) *оператор*	Проверяет *условие* и выполняет *оператор* до тех пор, пока условие сохраняет значение true;
do оператор while (условие);	Выполняет *оператор*, а затем проверяет *условие.* Выполняет *оператор* до тех пор, пока *условие* не примет значение false;
for(*оператор1*; *условие*; *выражение*) *оператор2*	Выполняет *оператор1* один раз при входе в цикл, а затем проверяет *условие*. Если *условие* принимает значение true, выполняет *оператор2*, а затем вычисляет элемент *выражение*.
switch (*выражение*) *оператор*	Элемент *оператор* почти всегда представляет собой блок, который включает операторы с метками следующего вида case *значение:* При выполнении switch-оператора вычисляется *выражение* и управление передается той case-метки, значение которой совпадает с результатом вычисления элемента *выражение.*
break;	Передает управление *оператору*, непосредственно следующему за окончанием ближайшего оператора while, for, do или switch, который включает инструкцию break;
continue;	Передает управление назад к началу следующей итерации в операторах for, while или do, которые включают оператор continue;
goto *метка*;	Передает управление *оператору*, помеченному *меткой*. *Метка* должна находиться внутри текущей функции;
try{ *оператор* } catch (*параметр*) { *оператор* }	Выполняет код представленный элементом *оператор,* который может сгенерировать исключение (посредством оператора throw). Это исключение должно быть отработано оператором catch. Оператор catch обрабатывает исключения (генерируемые значения которых должны иметь подобный тип, как и тип элемента *параметр*) посредством выполнения *оператора*;
throw *выражение*;	Прекращает выполнение программы или передает управление catch-ветви текущего *оператора* try. Передает *выражение*, тип которого определяет, в какой именно catch-ветви будет обработано это исключение.

Оператор соответствует предложению естественного языка, но завершается точкой с запятой (;). Выражение C++ (например, ival + 5) становится простым оператором, если после него поставить точку с запятой. Составной оператор – это последовательность простых операторов, заключенная в фигурные скобки. По умолчанию операторы выполняются в порядке записи. Операторы языка C++ делятся на три группы: операторы-выражения, получающиеся из произвольных выражений добавлением точки с запятой; пустые операторы и блоки; операторы, начинающиеся с ключевого слова.

2) Спецификатор volatile отмечает, что в процессе выполнения программы значения объекта может изменяться в промежутке между явными обращениями к нему. Например, для многопоточного приложения, на объект может повлиять один из потоков. Поэтому компилятор не должен помещать его в регистровую память и не должен делать никаких предположений о постоянстве объекта в те моменты, когда в программе нет явных операций, изменяющих значения объекта.

3) Спецификатор mutable используется только для членов данных класса и позволяет этим членам данных быть модифицированными даже в случае, если они являются членами const-объектов. Спецификатор mutable необходим в тех случаях, когда строгое придерживание константности неудобно. Объект может оставаться логически константным ("logically const"), но при этом его физическая константность ("physically const") может быть нарушена.

11.2 КОНТРОЛЬНАЯ РАБОТА №2. Основы программирования на языке высокого уровня С++

11.2.1 Задания контрольной работы №2

1) Вычисление простого выражения. Разработать программу для вычисления арифметического выражения, представленного в таблице 8, и вывода полученного результата. Исходные данные ввести с клавиатуры.

Таблица 8 - Арифметические выражения для выполнения задания

Номер варианта	Выражение	Исходные данные
1	$a = ln(y^{-\sqrt{\lvert x \rvert}}) \cdot (\sin(x) + e^{(x+y)})$	*x, y*
2	$b = \sqrt{c(\sqrt{y} + x^2)} \cdot (\cos(x) - \lvert c - y \rvert)$	*c, x, y*
3	$c = arctg(x) - \frac{3}{5} e^{xy} + 0.5 \frac{\lvert x + y \rvert}{(x+y)^b}$	*b, x, y*
4	$d = \frac{e^{\lvert x-y \rvert} \cdot tg(z)}{arctg(y) + \sqrt{x}} + ln(x)$	*x, y, z*
5	$e = \frac{(\cos(x) - \sin(y))^3}{\sqrt{tg(z)}} + ln^2(x \cdot y \cdot z)$	*x, y, z*
6	$f = y^x + \sqrt{\lvert x \rvert + e^y} - \frac{z^3 \cdot \sin^2(y)}{y + z^2 / (y - x)}$	*x, y, z*
7	$g = \frac{1 + \cos(x+y)}{\lvert e^x - 2y / (1 + x^2 \cdot y^2) \rvert} \cdot x^3 + arcsin(y)$	*x, y*
8	$h = 2 + \frac{x^2}{\sqrt{2}} + \frac{\lvert y^3 \rvert}{\sqrt{2}} + \frac{z^4 \cdot (ln(x) + 1) \cdot \sqrt{2}}{\sqrt{3}}$	*x, y, z*
9	$j = \left((1 + y) \cdot \sqrt{\sin(3z)} - \frac{\lvert y - x \rvert}{5} \right)^3$	*x, y, z*
10	$k = ln \left\lvert (y - \sqrt{\lvert x \rvert}) \cdot \left(x - \frac{y}{z + x^2 / 4} \right) \right\rvert$	*x, y, z*
11	$l = 0.5x^5 + 3 \cdot \cos(x + y) + e^{-0.1 \cdot y \cdot z} - \sqrt{\lvert x \cdot y \rvert}$	*x, y, z*
12	$m = \sqrt{\left\lvert -3 \cdot tg(x) \cdot lg(x^4 + y) / e^{-x} + 1 \right\rvert}$	*x, y*
13	$n = \sqrt{e^x + tg(x) + 1} \cdot (lg(y) + \cos(x \cdot y) + \sqrt[3]{x})$	*x, y*

Продолжение таблицы 8

Номер варианта	Выражение	Исходные данные
14	$p = \dfrac{lg(x) - e^{x+y}}{\sqrt{2} + y^2 + \lvert x^3 - ln(y)\rvert}$	*x, y*
15	$q = \sqrt{12x^4 - 3x^3 + 4x^2 - 5x + 6} - lg^2(z)$	*x, y, z*
16	$r = lg\lvert 1 - 2x + 3x^2 - 4x^3\rvert + \sqrt{\lvert x\rvert} / z$	*x, y, z*
17	$s = \dfrac{2 \cdot \cos(x - 1/6)}{1/2 + \sin^2(y)} - \dfrac{1}{\lvert x^2 / (y + x^3)\rvert}$	*x, y*
18	$t = \dfrac{x \cdot y \cdot z - y \cdot \lvert x + \sqrt{z}\rvert}{10^7 + \sqrt[4]{lg(4)}}$	*x, y, z*
19	$u = \dfrac{(\lvert x + y\rvert - z)^3 - (x - y + x)^2 + \sqrt{\lvert x + y + z\rvert}}{log_2(tg(2))}$	*x, y, z*
20	$w = \dfrac{(x / y) \cdot (z + x) \cdot e^{\lvert x - y\rvert} + ln(1 + e)}{\sin^2(y) - (\sin(x) \cdot \sin(y))^2}$	*x, y, z*
21	$a = \dfrac{\sqrt{\lvert x - 1\rvert} - \sqrt[3]{\lvert y\rvert}}{1 + \dfrac{x^2}{2} + \dfrac{y^2}{4}} + tg^2(xy)$	*x, y*
22	$b = \dfrac{3 + e^{y-1}}{1 + x^2 \cdot \lvert y - tg(z)\rvert}$	*x, y, z*
23	$c = (1 + y) \cdot \dfrac{x + y / (x^2 + 4)}{e^{-x-2} + 1 / \lvert x^2 + 4\rvert}$	*x, y*
24	$d = \dfrac{2\cos(x - \pi / 6)}{1/2 + \sin^2(y)} + \dfrac{\lvert y - x\rvert}{3}$	*x, y*
25	$e = \dfrac{1 + \sin^2(x + y)}{2 + \lvert x - 2x / (1 + x^2 y^2)\rvert} + x$	*x, y*
26	$f = ln\left\lvert \left(y - \sqrt{\lvert x\rvert}\right) \cdot \left(x - \dfrac{y}{z + x^2 / 4}\right)\right\rvert$	*x, y, z*
27	$h = \lvert x\rvert - \dfrac{y \cdot \left(arctg(z) + e^{-(x+3)}\right)}{x^2 + 2y^3}$	*x, y, z*
28	$k = 1 + \lvert y - x\rvert + \dfrac{(y - x)^2}{\sqrt{arctg(xy)}}$	*x, y*

Рекомендуемые теоретические разделы для ознакомления: операция и выражение присваивания; арифметические операции.

2) Вычисление условного выражения. Разработать программу для вычисления выражения и вывода полученного результата. Соответствующие исходные данные ввести с клавиатуры.

Таблица 9 - Условные выражения для выполнения задания

Номер варианта	Выражение	Исходные данные
1	$a = \begin{cases} (x+y)^2 - \sqrt{x \cdot y}, x \cdot y > 0 \\ (x+y)^2 + \sqrt{\lvert x \cdot y \rvert}, x \cdot y < 0 \\ (x+y)^2 + 1, x \cdot y = 0 \end{cases}$	*x, y*
2	$b = \begin{cases} ln(x/y) + (x^2+y)^3, x/y > 0 \\ ln \lvert x/y \rvert + (x^2+y)^3, x/y < 0 \\ (x^2+y)^3, y \neq 0, x = 0 \\ 0, y = 0 \end{cases}$	*x, y*
3	$c = \begin{cases} x^2 + y^2 + \sin(x), x - y = 0 \\ (x-y)^2 + \cos(x), x - y > 0 \\ (y-x)^2 + tg(x), x - y < 0 \end{cases}$	*x, y*
4	$d = \begin{cases} (x-y)^3 + arctg(x), x > y \\ (y-x)^3 + arctg(x), y > x \\ (y+x)^3 + 0.5, y = x \end{cases}$	*x, y*
5	$e = \begin{cases} i \cdot \sqrt{a}, i - \text{нечетное}, a > 0 \\ i/2 \cdot \sqrt{\lvert a \rvert}, i - \text{четное}, a < 0 \\ \sqrt{\lvert i \cdot a \rvert}, \text{иначе} \end{cases}$	*i, a*
6	$g = \begin{cases} e^{\lvert a \rvert - \lvert b \rvert}, 0.5 < a \cdot b < 10 \\ \sqrt{\lvert a+b \rvert}, 0.1 < a \cdot b < 0.5 \\ 2 \cdot x^2, \text{иначе} \end{cases}$	*a, b, x*
7	$h = \begin{cases} arctg(x + \lvert y \rvert), x < y \\ arctg(\lvert x \rvert + y), x > y \\ (x+y)^2, x = y \end{cases}$	*x, y*

Продолжение таблицы 9

Номер варианта	Выражение	Исходные данные
8	$j = \begin{cases} \sin(5 \cdot k + 3 \cdot m \cdot \lvert k \rvert), k < m \\ \cos(5 \cdot k + 3 \cdot m \cdot \lvert k \rvert), k > m \\ k^3, k = m \end{cases}$	*k, m*
9	$l = \begin{cases} 3 \cdot k^3 + 3 \cdot p^2, k > \lvert p \rvert \\ \lvert k - p \rvert, 3 < k < \lvert p \rvert \\ (k - p)^2, k = \lvert p \rvert \\ 0, \text{иначе} \end{cases}$	*k, p*
10	$k = \begin{cases} \ln(\lvert f \rvert + \lvert q \rvert), \lvert f \cdot q \rvert > 10 \\ e^{f+q}, \lvert f \cdot q \rvert < 10 \\ f + a, \lvert f \cdot q \rvert = 10 \end{cases}$	*f, q*
11	$m = \begin{cases} z + \sqrt{x^2 + y^2} \cdot \sin(xy), x < -3 \\ 2x \cdot \sqrt{x^2 + 2 \cdot \lvert yz \rvert}, -3 \le x \le 1 \\ \lvert x^2 + 2y^2 + z \rvert, x > 1 \end{cases}$	*x, y, z*
12	$p = \begin{cases} \arcsin\left(\frac{x+y}{z}\right), \left\lvert \frac{x+y}{z} \right\rvert \le 1 \\ 2x^2 + y + tg(z), \left\lvert \frac{x+y}{z} \right\rvert > 1 \end{cases}$	*x, y, z*
13	$q = \begin{cases} xyz - \lvert y - z \rvert, xy < -1 \\ x^2 - y^2 + z, -1 \le xy \le 1 \\ \sqrt{x^2 + y^2 + 2\sin(2z)}, xy > 1 \end{cases}$	*x, y, z*
14	$r = \begin{cases} \sqrt{x^2 + z^2} - \sqrt{x^2 + y^2}, x < -2 \\ \lvert 3xz - 4y \rvert, -2 \le x \le 2 \\ \sin\left(\frac{2x^2 + y^2}{z^2}\right), x > 2 \end{cases}$	*x, y, z*
15	$s = \begin{cases} 3x + 2y, x < 0 \\ x^2 + 4 \cdot \sqrt{y^2 + 1}, 0 \le x \le 1 \\ 3 \cdot \ln\left\lvert \arcsin\left(\frac{x}{y}\right) \right\rvert, x > 1 \end{cases}$	*x, y*

Продолжение таблицы 9

Номер варианта	Выражение	Исходные данные
16	$t=\begin{cases}\dfrac{3x^2+2y}{x^2+y^2}, x<-2\\ \lvert x-3y\rvert, -2\le x\le 2\\ ln\left\lvert\dfrac{x-y}{2+3y}\right\rvert, x>2\end{cases}$	*x, y*
17	$z=\begin{cases}\dfrac{a\cdot b}{\sqrt{a^2+\lvert 2\cdot b\cdot \sin(a+b)\rvert}}, a+b<-2\\ 3\cdot a\cdot b+2\cdot b^2+c, -2\le a+b\le 1\\ \dfrac{1}{a\cdot b}+a\cdot c-\sqrt{\lvert a-b\rvert}, a+b>1\end{cases}$	*a, b, c*
18	$a=\begin{cases}2\cdot x^2-y\cdot x\cdot\sqrt{x+y}+y, x\cdot y>0\\ 3\cdot y^2+y\cdot x\cdot\sqrt{\lvert x-y\rvert}, x\cdot y<0\\ 2\cdot x\cdot y+3, x\cdot y=0\end{cases}$	*x, y*
19	$b=\begin{cases}\sin(x+y)+2\cdot(x+y)^2, x-y>0\\ \sin(x-y)+(x-y)^3, x-y<0\\ \lvert x^2+\sqrt{y}\rvert, y\ne 0, x=0\\ 0, y=0\end{cases}$	*x, y*
20	$c=\begin{cases}\lvert x^2-y^2\rvert+\sqrt{x/y}, -2<x\cdot y<0\\ (x-y)^2+\sqrt{x\cdot y+2\cdot x/y}, 0<x\cdot y<2\\ 1/(x+y), \text{иначе}\end{cases}$	*x, y*
21	$d=\begin{cases}(2\cdot x-y)^2+\sin(x), x>y\\ (y-2\cdot x)^2+\cos(x), y>x\\ x^2-2\cdot y, y=x\end{cases}$	*x, y*
22	$e=\begin{cases}x^{\sqrt{\lvert a+b\rvert}}, 0.5<a\cdot b<10\\ \sqrt{\lvert a\rvert-\lvert b\rvert}+2\cdot x, 0.1<a\cdot b<0.5\\ a\cdot b/x, \text{иначе}\end{cases}$	*a, b, x*

Продолжение таблицы 9

Номер варианта	Выражение	Исходные данные
23	$g=\begin{cases}\sqrt{(k^2+m)}+3\cdot\lvert k\rvert, k<m\\ \sqrt{(k+m^3)}-0.5\cdot\lvert m-k\rvert, k>m\\ 2\cdot m^2-0.5, k=m\end{cases}$	*k, m*
24	$h=\begin{cases}\sqrt{k+p}+2\cdot p^2, k>\lvert p\rvert\\ 0.5\cdot p+(p/k), k<\lvert p\rvert\\ 3\cdot k-2, k=\lvert p\rvert\end{cases}$	*k, p*
25	$k=\begin{cases}\sqrt{(\lvert x\rvert+\lvert y\rvert)}+2\cdot x-3\cdot y, \lvert x\cdot y\rvert>5\\ 2^{x\cdot y}-(x+y), \lvert x\cdot y\rvert<5\\ x-y, \lvert x\cdot y\rvert=5\end{cases}$	*x, y*
26	$m=\begin{cases}\sin(x)+0.5\cdot\sqrt{x+\lvert yz\rvert}, y<-3\\ 2y\cdot\sqrt{x^2+z}, -3\le y\le 1\\ 3\cdot x^3-2\cdot y^2+z, y>1\end{cases}$	*x, y, z*
27	$n=\begin{cases}\sin(a\cdot b+c), \left\lvert\frac{a-b}{c}\right\rvert\le 0.5\\ 2a\cdot\sqrt{\lvert b+c\rvert}, \left\lvert\frac{a-b}{c}\right\rvert>0.5\end{cases}$	*a, b, c*
28	$p=\begin{cases}\sqrt{\lvert a\cdot b\rvert}+2\cdot c, a\cdot b<-2\\ a^3+b^2-c^2, -2\le a\cdot b\le 2\\ a^c-b, a\cdot b>2\end{cases}$	*a, b, c*

Рекомендуемые теоретические разделы для ознакомления: операция и выражение присваивания; арифметические операции; условный оператор.

3) Табулирование функции. Вычислить и вывести на экран таблицу функции *y=f(x)* в интервале [*a, b*] с шагом *h*. Результаты представить в виде таблицы.

x	*y*
—	—
—	—
—	—
—	—
—	—
...	...

Таблицу выровнять с помощью функций форматирования cout.width(), cout.precision().

Таблица 10 - Функции *y=f(x)* в интервале [*a, b*] с шагом *h* для выполнения задания

Номер варианта	Функция *y=f(x)*
1	$y=\begin{cases}\dfrac{1}{(x+1)\sqrt{x^2+1}}, & x>-1\\ \dfrac{-(\ln\lvert x\rvert)^3+3(\ln\lvert x\rvert)^2/2+3(\ln\lvert x\rvert)/2+\frac{3}{4}}{2x^2}, & x\leqslant -1\end{cases}$ $x\in[-3,0],\ h=0.1$
2	$y=\begin{cases}\dfrac{e^x(1+\sin x)}{1+\cos x}, & x<0\\ e^x\cdot \operatorname{tg}\dfrac{x}{2}, & x\geqslant 0\end{cases}$ $x\in\left[-\frac{\pi}{4},\frac{\pi}{4}\right],\ h=\frac{\pi}{20}$
3	$y=\begin{cases}\sin x\cdot\ln(\lvert\operatorname{tg}x\rvert), & x<0\\ 0, & x=0\\ \ln(\lvert\operatorname{tg}\frac{x}{2}\rvert)-\cos x\cdot\ln(\lvert\operatorname{tg}x\rvert), & x>0\end{cases}$ $x\in\left[-\frac{\pi}{8},\frac{\pi}{8}\right],\ h=\frac{\pi}{40}$
4	$y=\begin{cases}(x\cdot\ln x)^2, & x>0\\ 0, & x=0\\ \frac{x^3}{27}(9\ln^2\lvert x\rvert-6\ln\lvert x\rvert+2), & x<0\end{cases}$ $x\in[-0.6,0.6],\ h=0.12$
5	$y=\begin{cases}\arccos\sqrt{\frac{x}{1+x}}, & x\geqslant 0\\ 1-x^2-2\sin^2 x, & x<0\end{cases}$ $x\in\left[-\frac{\pi}{2},\frac{\pi}{2}\right],\ h=\frac{\pi}{20}$
6	$y=\begin{cases}\frac{2}{3}, & x=0\\ \frac{2}{3}\cdot\frac{\sin x}{x}+\frac{1}{x^2}\sin\frac{1}{x}, & \textit{иначе}\end{cases}$ $x\in[0,1],\ h=0.1$

Продолжение таблицы 10

Номер варианта	Функция $y=f(x)$
7	$y=\begin{cases}\sqrt{e^x-1}, & x\geq 0\\ -x^2, & x<0\end{cases}$ $x\in[-0.5, 0.5],\ h=0.1$
8	$y=\begin{cases}\mathrm{tg}^2 x, & x>0\\ x^2\cdot\sin x, & x\leq 0\end{cases}$ $x\in[-1,1],\ h=0.2$
9	$y=\begin{cases}\frac{\ln^2 x}{x}, & x>0\\ 0, & x=0\\ \frac{3}{5}\cdot\frac{\sin 2x}{x}, & x<0\end{cases}$ $x\in\left[-\frac{\pi}{2}, \frac{\pi}{2}\right],\ h=\frac{\pi}{20}$
10	$y=\begin{cases}x\,\mathrm{sh}2x, & x>0\\ 0, & x=0\\ -x^3\cdot e^x, & x<0\end{cases}$ $x\in[-0.5, 0.5],\ h=0.1$
11	$y=\begin{cases}\frac{\frac{3}{7}}{\sqrt{9+x^2}}, & x<4\\ \frac{3}{35}(x-4)^3, & \textit{иначе}\end{cases}$ $x\in[3,5],\ h=0.1$
12	$y=\begin{cases}2e^{\sqrt{x}}, & x\geq 0\\ 2x^3\cdot\cos(x^2+1), & x<0\end{cases}$ $x\in[-1,1],\ h=0.2$
13	$y=\begin{cases}\frac{x}{x^4+3x^2+2}, & x\leq 0\\ \frac{1}{2}\ln\frac{x^2+1}{x^2+2}-\frac{1}{2}\ln\frac{2x}{3}, & \textit{иначе}\end{cases}$ $x\in[-2,2],\ h=0.2$

Продолжение таблицы 10

Номер варианта	Функция $y=f(x)$
14	$y=\begin{cases}\frac{x^3}{3+x}, & x\leqslant -3\\ 0, & x=0\\ 9x-27\ln(3+x), & \textit{иначе}\end{cases}$ $x\in[-4,-2],\ h=0.1$
15	$y=\begin{cases}\left(\frac{\ln x}{3}\right)^3, & x>0\\ \frac{1}{\sqrt{2}}\ln\frac{1+\sqrt{2}(x-1)+\sqrt{2x^2+1}}{2}, & \textit{иначе}\end{cases}$ $x\in[-0.5,0.5],\ h=0.1$
16	$y=\begin{cases}\frac{1}{(3\sin x+2\cos x)^2}, & x<-1\\ \frac{3}{26}-\frac{3\cos x-2\sin x}{13(2\cos x+3\sin x)}, & x\geqslant -1\end{cases}$ $x\in[-1.5,-0.5],\ h=0.1$
17	$y=\begin{cases}2\sin^2 2x+3\cos^2 3x, & x<0\\ 3e^{\sin x+2\cos x}, & x\geqslant 0\end{cases}$ $x\in\left[-\frac{\pi}{2},\frac{\pi}{2}\right],\ h=\frac{\pi}{10}$
18	$y=\begin{cases}xe^x(\sin x-\cos x)/2, & x<-1\\ (e^x\cos x-1)/2, & -1\leqslant x\leqslant 0\\ x^2\sin 2x, & x>0\end{cases}$ $x\in[-2,2],\ h=0.2$
19	$y=\begin{cases}\frac{\ln^2 x}{x}, & x>0\\ 0, & x=0\\ \frac{1}{x^2\sin 2x}, & x<0\end{cases}$ $x\in[-2,2],\ h=0.2$
20	$y=\begin{cases}x\,\mathrm{sh}x, & x>0\\ \frac{1-\cos x}{1+2\sin 3x}, & x\leqslant 0\end{cases}$ $x\in[-1,3],\ h=0.2$

Продолжение таблицы 10

Номер варианта	Функция $y=f(x)$
21	$y=\begin{cases}0, x=0\\ x\cdot\sin\left(\frac{1}{x}\right), x>0\\ \frac{1}{x}\cdot\cos\left(\frac{2}{x}\right), x<0\end{cases}$ $x\in\left[-\frac{2}{\pi},\frac{2}{\pi}\right], h=\frac{1}{10\pi}$
22	$y=\begin{cases}\frac{\sin(x-1)}{x-1}, x>1\\ 1, x=1\\ e^{-x}\cdot\cos(\pi x), x<1\end{cases}$ $x\in[0,2], h=0.1$
23	$y=\begin{cases}\frac{\sin(2x)}{x}, x>0\\ 2, x=0\\ 2\cdot e^{-x}\cdot\cos(x), x<0\end{cases}$ $x\in[-1,1], h=0.1$
24	$y=\begin{cases}\sqrt{x+0.5}, x=0\\ \ln(x+2), x>0\\ \frac{1}{x}+0.2\cdot x, x<0\end{cases}$ $x\in[-0.5,1], h=0.1$
25	$y=\begin{cases}x/2-x, x=0.5\\ \sin(x)-2x+0.5x, x>0.5\\ \frac{\sqrt{x^2+2}}{x-1}, x<0.5\end{cases}$ $x\in[0,1], h=0.1$

Продолжение таблицы 10

Номер варианта	Функция $y=f(x)$
26	$y=\begin{cases}\frac{\pi x}{2}, x>0\\ \pi/x+\sin(\pi/x), x<0\\ 0, x=0\end{cases}$ $x\in[-1,2], h=0.2$
27	$y=\begin{cases}e^{-2x}, x>0\\ e^{x}+2\cdot x, x<0\\ -2, x=0\end{cases}$ $x\in[-1,1], h=0.1$
28	$y=\begin{cases}\sin(x/2), x>0.5\\ 2x, x=0.5\\ \cos\left(\frac{\lvert 2x\rvert}{0.5\pi}\right), x<0.5\end{cases}$ $x\in[0,1.5], h=0.1$

Рекомендуемые теоретические разделы для ознакомления: оператор цикла while; цикл с постусловием do-while; оператор for.

4) Работа с битами

Вариант №1

Ввести 8 символов. В символе с наибольшим кодом заменить 3-й бит нулем, а в символе с наименьшим кодом 4-й бит – единицей. Вывести исходную последовательность, ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №2

Ввести последовательность из 8 символов. В каждом из символов в их двоичном представлении заменить: для нечетных (по порядку) символов 3-й бит единицей; для четных символов 4-й бит нулем. Вывести исходную последовательность, ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №3

Ввести последовательность из 8 символов. В их двоичном представлении заменить: если младший бит 1, заменить его на 0; если младший бит 0, заменить его и 2-й бит единицами. Вывести исходную последовательность и ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №4

Ввести последовательность из 8 символов. Если символ – буква, то заменить в нем 3-й бит нулем, иначе – заменить 2-й бит единицей. Вывести исходную последовательность и ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №5

Ввести последовательность из 8 символов. Если символ – цифра, то заменить в нем 4-й бит единицей, иначе – 2-й бит нулем. Вывести исходную последовательность и ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №6

Ввести последовательность из 8 целых чисел. Если число четное, то заменить его младший байт нулями, если нечетное, то заменить в его младшем байте 3-й и 4-й бит единицами. Вывести исходную последовательность в десятичной и восьмеричной форме; преобразованную последовательность в десятичной и восьмеричной форме.

Вариант №7

Ввести последовательность из 8 целых чисел. В каждом втором числе заменить (i–1)-й бит единицей, где i-номер члена последовательности. Вывести исходную последовательность в десятичной и восьмеричной формах; преобразованную последовательность в десятичной и восьмеричной формах.

Вариант №8

Ввести последовательность из 8 символов. Если символ – русская гласная буква, то заменить в нем младший бит единицей, иначе – заменить 2-й и 3-й биты нулями. Вывести исходную и преобразованную последовательности в символьной форме и в восьмеричных кодах.

Вариант №9

Ввести последовательность из 8 символов. Если символ – восьмеричная цифра, то заменить в нем бит, номер которого совпадает с этой цифрой, нулем; иначе –

заменить младший бит единицей. Вывести исходную и преобразованную последовательности в символьной и восьмеричной формах.

Вариант №10

Ввести последовательность из 8 символов. Если код символа четный, то заменить в нем младший бит единицей, иначе – заменить два младших бита нулями. Вывести исходную и преобразованную последовательности в символьной и восьмеричной формах.

Вариант №11

Ввести последовательность из 8 символов. Если символ есть + – * / % , то заменить в нем четыре младших бита единицами, иначе – заменить 5-й бит нулем. Вывести исходную и преобразованную последовательности в символьной и восьмеричной формах.

Вариант №12

Ввести последовательность из 8 символов. Если символ – большая латинская буква, то заменить в нем 3-й бит нулем, иначе – заменить младший бит единицей. Вывести исходную и преобразованную последовательности в символьной и восьмеричной формах.

Вариант №13

Ввести последовательность из 8 целых чисел. В каждом нечетном числе заменить (i–1)-й бит нулем (i-номер члена последовательности). Вывести исходную последовательность в десятичной и восьмеричной формах; преобразованную последовательность в десятичной и восьмеричной формах.

Вариант №14

Ввести последовательность из 8 символов. Если символ – латинская согласная буква, то заменить в нем младший бит единицей, иначе – заменить 3-й и 5-й биты нулями. Вывести исходную и преобразованную последовательности в символьной форме и в восьмеричных кодах.

Вариант №15

Реализовать алгоритм инвертирования n-разрядов целого числа без знака, начинающихся с p-ой позиции. Оставшиеся разряды остаются без изменения. Значения переменной, подлежащей преобразованию, а также значения n и p вводятся с клавиатуры. Результат вывести на экран в восьмеричном виде.

Вариант №16

Реализовать алгоритм, выполняющий зеркальное отображение битов значения целого числа без знака. Значение переменной, подлежащей преобразованию, вводится с клавиатуры. Результат вывести на экран в восьмеричном виде.

Вариант №17

Реализовать алгоритм поиска первой пары несовпадающих разрядов в двух переменных типа unsigned. Значения сравниваемых переменных вводятся с клавиатуры, результат выводится на экран.

Вариант №18

Ввести последовательность из 8 символов. Сравнить 5-й и 6-й биты каждого символа. Если они не равны, то сделать их равными младшему биту, иначе – старшему. Вывести исходную последовательность, ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №19

Ввести 8 символов. В символе с наибольшим кодом заменить 5-й бит единицей, а в символе с наименьшим кодом 6-й бит – нулем. Вывести исходную последовательность, ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №20

Ввести последовательность из 8 символов. В их двоичном представлении заменить: если старший бит 1, заменить его на 0; если старший бит 0, заменить его и младший бит единицами. Вывести исходную последовательность и ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №21

Реализовать алгоритм зеркального отображения тетрад битов значения целого числа без знака. Значение переменной, подлежащей преобразованию, вводится с клавиатуры. Результат вывести на экран в восьмеричном виде.

Вариант №22

Ввести последовательность из 8 целых чисел. Если код символа нечетный, то заменить в нем старший бит нулем, иначе – заменить два младших бита единицами. Вывести исходную последовательность в десятичной и восьмеричной формах; преобразованную последовательность в десятичной и восьмеричной формах.

Вариант №23

Ввести последовательность из 8 символов. В каждом из символов в их двоичном представлении заменить все четные биты единицами. Вывести исходную последовательность, ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №24

Ввести последовательность из 8 символов. Если сумма единиц в представлении символа нечетная, то заменить 2 старших бита нулями, иначе – единицами. Вывести исходную последовательность, ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №25

Ввести последовательность из 8 символов. Если сумма трех старших бит в символе равна единице, то заменить их единицами, иначе – нулями. Вывести исходную последовательность, ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №26

Ввести последовательность из 8 символов. Сравнить их младший и старший биты. Если они равны, то заменить старший нулем, младший – единицей, иначе заменить старший бит единицей, младший – нулем. Вывести исходную последовательность, ее восьмеричные коды; преобразованную последовательность и ее восьмеричные коды.

Вариант №27

Ввести последовательность из 8 символов. Если символ – цифра, то заменить в нем 3 младших бита единицами, иначе – первый и последний нулями. Вывести исходную и преобразованную последовательности в символьной форме и в восьмеричных кодах.

Вариант №28

Ввести последовательность из 8 символов. Если символ – латинская гласная буква, то заменить в нем 2 младших бита нулем, иначе – 2-й и 4-й единицами. Вывести исходную и преобразованную последовательности в символьной форме и в восьмеричных кодах.

Рекомендуемый теоретический раздел для ознакомления: побитовые операции.

5) Рекуррентные последовательности

Вариант №1

Найти произведение

$$P = \prod_{i=1}^{30} Z_i$$

$$Z_1 = 0.45;$$

$$Z_2 = 0.17;$$

$$Z_K = 0.5 \sin 2Z_{K-1} - 0.9 \cos 3Z_{K-2}$$

Массивом не пользоваться.

Вариант №2

Вычислить сумму

$$S = \sum_{i=1}^{150} a_i$$

$$a_1 = 3.14;$$

$$a_2 = 1.57;$$

$$a_i = 2 \sin(ka_{i-1}) + 3 \cos(ka_{i-2})$$

$$k = \sqrt{a_1^2 + a_2^2}$$

Массивом не пользоваться.

Вариант №3

Задана последовательность $\{r_k\}$

$$r_1 = 2.2;$$

$$r_2 = 3.3;$$

$$r_3 = r_1 \cdot r_2;$$

$$r_K = (\sqrt{r_{k-1}^2 + 2r_{k-2}^2}) \cdot \sin(r_{k-1} \cdot r_{k-3})$$

Найти и напечатать наименьший элемент этой последовательности.

Массивом не пользоваться.

Вариант №4

Вычислить сумму

$$S = \sum_{i=1}^{100} x_i$$

$$x_1 = 0.327;$$

$$x_2 = 0.3;$$

$$x_i = i + 2 \sin(x_{i-1}) - x_{i-2}$$

Массивом не пользоваться.

Вариант №5

Найти минимальный член последовательности $\{U_i\}$ и его номер

$$U_1 = 0.5;$$
$$U_2 = 0.27;$$
$$U_i = 2\sin(U_{i-1}) - 3\cos(U_{i-2})$$

Массивом не пользоваться.

Вариант №6

Найти максимальный член последовательности $\{x_i\}$ и его номер

$$x_1 = 0.15;$$
$$x_2 = 0.19;$$
$$x_i = \cos(ix_{i-1}) - 3\sin(x_{i-2})$$

Массивом не пользоваться.

Вариант №7

Вычислить произведение

$$P = \prod_{i=1}^{100} U_i,$$

где последовательность $\{U_i\}$ задается так:

$$U_1 = 0.4;$$
$$U_2 = 0.5;$$
$$U_3 = 0.93;$$
$$U_i = \sin(U_{i-1}) + \cos(U_{i-3}),\ i = 4, 5, ..., 100.$$

Массивом не пользоваться.

Вариант №8

Напечатать значения очередной пары x_i, y_i последовательностей:

$$x_1 = 1;$$
$$y_1 = 0.14;$$
$$x_i = x_{i-1} + y_{i-1};$$
$$y_i = y_{i-1} + \sin(x_{i-1}) - 0.5x_i$$

Найти и напечатать maxx_i, miny_i.

Массивом не пользоваться.

Вариант №9

Напечатать значения очередных пар x_k, y_k последовательностей. Найти $\min\{x_k\}$

$$\{x_k\},\{y_k\}:$$

$$x_1 = 0.5;\ \ x_2 = 0.6;\ \ y_1 = y_2 = 0.67;$$

$$x_k = y_{k-1} + y_{k-2};$$

$$y_k = x_k^{\,2} + x_{k-2} - y_{k-1} - 0.3x_k \qquad , \text{k=3, 4, … 10.}$$

Массивом не пользоваться.

Вариант №10

Напечатать значения очередных пар последовательностей $\{z_i\}$, $\{t_i\}$

$$z_1 = 0.14;\ \ t_1 = 0.53;$$

$$z_i = \sin z_{i-1} + 2 \cdot t_{i-1};$$

$$t_i = t_{i-1} + z_{i-1} \cdot \sin z_{i-1}$$

i = 2, 3, … 30.

Массивом не пользоваться.

Вариант №11

Напечатать значения очередных пар последовательностей и произведение

$$P = \prod_{i=1}^{100} y_i$$

$$\{x_i\},\{y_i\}:$$

$$x_1 = 1;\ \ x_2 = 1.5;\ \ y_1 = -1;\ \ y_2 = 0.6;$$

$$x_i = 5\sin(y_{i-1}) + [y_{i-2}];$$

$$y_i = 2\ x_{i-1} + 3\,x_{i-2} + 5\,y_{i-1}$$

i = 3, 4, …, 100. [] – обозначение целой части.

Массивом не пользоваться.

Вариант №12

Вычислить сумму

$$S = \sum_{i=1}^{100} U_i$$

$$U_1 = 2,\ \ U_2 = 2.5,\ \ U_3 = 0.15;$$

$$U_i = \cos[U_{i-1}] + \left|0.7 \sin U_{i-3}\right|,$$

i = 4, 5, …, 100. [] – обозначение целой части.

Массивом не пользоваться.

Вариант №13

Вычислить произведение

$$P = \prod_{i=1}^{150} x_i$$

$$x_1 = 2.3;\ \ x_2 = 1.5;\ \ x_3 = -2.1;$$

$$x_i = \log_7 \left| \frac{1 + x_{i-1}}{2} \right| + 2\sin(x_{i-3})$$

$i = 4, 5, \ldots, 150$.

Массивом не пользоваться.

Вариант №14

Вычислить сумму

$$S = \sum_{i=1}^{1000} r_i$$

$$r_1 = 0.27;\ \ r_2 = -0.5;$$

$$r_k = (\sin r_{k-1}^2 + \cos r_{k-1}) \sqrt{r_{k-1}^2 + r_{k-2}^2}$$

Массивом не пользоваться.

Вариант №15

Найти произведение

$$P = \prod_{i=1}^{20} A_i$$

$$A_1 = 0.8$$

$$A_2 = 0.75$$

$$A_3 = 0.3$$

$$A_i = 0.25 \cdot \sin A_{i-1} + 0.35 \cdot \cos A_{i-3},\ i = 4, 5, \ldots 20.$$

Массивом не пользоваться.

Вариант №16

Вычислить сумму

$$S = \prod_{i=1}^{100} R_i$$

$$R_1 = 1.01;$$

$$R_2 = 0.75;$$

$$R_i = 0.25i + 2\sin R_{i-1} - 0.35\cos R_{i-2}$$

$i = 3, \ldots, 100$.

Массивом не пользоваться.

Вариант №17

Задана последовательность $\{r_i\}$ следующим образом:

$$r_1 = 1.55;$$
$$r_2 = 1.77;$$
$$r_i = |r_{i-1} - r_{i-2}| + \cos r_{i-2}$$

Найти и напечатать наименьший элемент этой последовательности. Массивом не пользоваться.

Вариант №18

Вычислить сумму

$$S = \prod_{i=1}^{200} U_i$$

$$U_1 = 0.5;$$
$$U_2 = 1.2;$$
$$U_k = U_{k-1} - \cos(0.3k + U_{k-2}), \quad k = 3,4,\ldots,200.$$

Массивом не пользоваться.

Вариант №19

Найти минимальный член последовательности $\{U_k\}$ и его номер:

$$U_1 = 2;$$
$$U_2 = 0.8;$$
$$U_k = \sin(1.5k + U_{k-1}) + 0.25U_{k-2}$$

$i = 3, 4, \ldots, 150$.

Массивом не пользоваться.

Вариант №20

Найти максимальный член последовательности $\{U_k\}$ и его номер:

$$U_1 = 0.9;$$
$$U_2 = 1;$$
$$U_k = \sin k U_{k-1} - 0.77 \cos U_{k-2}$$

$k = 3, 4, \ldots, 100$.

Массивом не пользоваться.

Вариант №21

Вычислить произведение

$$P = \prod_{i=1}^{50} W_i$$

$$W_1 = 0.3;$$
$$W_2 = 0.5;$$
$$W_i = i + \sin(W_{i-1}) - \cos(W_{i-2})$$

Массивом не пользоваться.

Вариант №22

Напечатать значения очередной пары x_i, y_i последовательностей:

$$x_1 = 0.7;\ x_2 = 0.65;$$
$$y_1 = 0.4;\ y_2 = 0.5;$$
$$x_i = x_{i-1} + x_{i-2} - 0.6\,y_{i-1};$$
$$y_i = y_{i-1} + 0.2\,x_{i-2}$$

i = 3, 4, …, 15. Найти и напечатать max x_i, min y_i.

Массивом не пользоваться.

Вариант №23

Напечатать значения очередных пар x_k, y_k последовательностей.

Найти

$$\sum_{k=1}^{50} x_k$$

$\{x_k\},\{y_k\}$:

$$x_1 = 0.35;\quad x_2 = 0.8;$$
$$y_1 = 0.7;\quad y_2 = 0.3;$$
$$x_k = y_{k-1} - 0.7\,x_{k-2};$$
$$y_k = x_k + x_{k-1} - 0.5\,y_{k-1}$$

k=3, 4, …, 100.

Массивом не пользоваться.

Вариант №24

Напечатать значения очередных пар последовательностей $\{u_i\}$, $\{v_i\}$

$$u_1 = 0.33;$$
$$v_1 = 0.77;$$
$$u_i = 0.8\,u_{i-1} + 1.2\,v_{i-1};$$
$$v_i = \sqrt{i} + u_i - 0.7\,v_{i-1}$$

i = 2, 3, …, 35.

Массивом не пользоваться.

Вариант №25

Напечатать значения очередных пар последовательностей и произведение

$$P=\prod_{i=1}^{50} U_i$$

$\{u_i\},\{v_i\}$:

$u_1 = -0.8;\ u_2 = 0.3;$

$v_1 = 0.23;\ \ v_2 = 0.44;$

$u_i = 0.3\, v_{i-1} + 0.5\, v_{i-2} - u_{i-1};$

$v_i = 0.4\sqrt{(|u_i - v_{i-2}|)} + 0.6\, v_{i-1}$, i = 3, 4,..., 100.

Массивом не пользоваться.

Вариант №26

Вычислить сумму

$$S = \sum_{i=1}^{150} A_i$$

$A_1 = 1.5;\ A_2 = 2;$

$A_i = 2.5\sin(A_{i-1} - A_{i-2}) + 3\cos(A_{i-1} + A_{i-2}),$

i = 3, 4, …, 150.

Массивом не пользоваться.

Вариант №27

Вычислить произведение

$$P = \prod_{i=1}^{100} z_i$$

$z_1 = 1.33;\quad z_2 = -1.44;\quad z_3 = 1.55;$

$z_i = \sin(i) + 2.2\cos([z_{i-3} + z_{i-1}]) - 1.1\, z_{i-2}$

i = 4, 5, …, 100.

[] – обозначение целой части.

Массивом не пользоваться.

Вариант №28

Вычислить сумму

$$S = \sum_{i=1}^{1000} r_i$$

$$r_1 = 1.23;\ r_2 = 0.65;\ r_3 = -0.25;$$

$$r_k = \frac{1.2\sin(r_{k-1} + r_{k-2}) + 0.8 r_{k-3}}{1 - \cos(r_{k-1})} - [0.55 r_{k-2}]$$

$k = 4, 5, \ldots, 1000$.
[] – обозначение целой части.
Массивом не пользоваться.

Рекомендуемые теоретические разделы для ознакомления: оператор цикла while; цикл с постусловием do-while.

6) Одномерные массивы

Вариант №1
Даны действительные числа $a_1, \ldots, a_{15}$.
Получить

$$\tilde{a} = \frac{1}{15}\sum_{i=1}^{15} a_i, \quad s = \sqrt{\frac{\sum_{i=1}^{15}(a_i - \tilde{a})^2}{14}}.$$

Вариант №2
Получены экспериментальные данные по влажности материала $a_1, a_2, \ldots, a_{50}$ в различные моменты времени. Вычислить среднее значение влажности и отклонение от среднего для каждого значения.

Вариант №3
Система из 25 материальных точек в пространстве задана с помощью последовательности действительных чисел $x_1, y_1, z_1, p_1; x_2, y_2, z_2, p_2; \ldots; x_{25}, y_{25}, z_{25}, p_{25}$, где x_i, y_i, z_i – координаты i-й точки, а p_i – ее вес (i=1, 2, …, 25). Получить координаты центра тяжести системы, а также расстояние от центра тяжести до всех точек системы.

Вариант №4
Даны действительные числа $a_1, \ldots, a_{20}$. Получить числа $b_1, \ldots, b_{20}$, где b_i – среднее арифметическое всех членов последовательности $a_1, \ldots, a_{20}$, кроме a_i (i = 1, 2, …, 20).

Вариант №5
Построить последовательность целых чисел $a_1, \ldots, a_{30}$, где a_1=1, a_2=1, $a_i = a_{i-1} + a_{i-2}$ (i = 3, …, 30).

Вариант №6

Даны действительные числа a_1, …, a_{30}. Получить a_{30}, a_{29}, …, a_1.

Вариант №7

Даны натуральные числа n_1, …, n_{20}, действительные числа x_1, …, x_{20}.
Вычислить

$$\frac{n_1 x_1 + \ldots + n_{20} x_{20}}{n_1 + \ldots + n_{20}}.$$

Вариант №8

Даны действительные числа a_1, …, a_{20}, b_1, …, b_{20}. Вычислить $(a_1+b_{20})\cdot(a_2+b_{19})\cdot\ldots\cdot(a_{20}+b_1)$.

Вариант №9

Даны действительные числа a_1, …, a_{28}, b_1, …, b_{28}. Члены последовательности c_1, …, c_{29} связаны с членами данных последовательностей соотношениями

$$c_{29} = 0,$$

$$c_{29-i} = \frac{a_{29-i}}{b_{29-i} - c_{29-i+1}}.$$

$i = 1, \ldots, 28$.
Получить c_1, …, c_{29}.

Вариант №10

Даны действительные числа a_1, …, a_{30}. Если в результате замены отрицательных членов последовательности a_1, …, a_{30} их квадратами члены будут образовывать неубывающую последовательность, то получить сумму членов исходной последовательности; в противном случае получить их произведение.

Вариант №11

Даны целые числа a_1, …, a_{30}. Все члены последовательности с четными номерами, предшествующие первому по порядку члену со значением $\max(a_1, \ldots, a_{30})$, домножить на $\max(a_1, \ldots, a_{30})$.

Вариант №12

Даны натуральное число m, действительные числа a_1, …, a_{30} (числа a_1, …, a_{30} попарно различны, $m \le 30$). В последовательности a_1, …, a_{30} поменять местами наибольший член и член с номером m.

Вариант №13

Даны действительные числа $x_1, \ldots, x_{101}, y_1, \ldots, y_{101}$. Получить действительные

$$x'_1,\ldots,x'_{101},$$
$$y'_1,\ldots,y'_{101},$$

преобразовав для этого члены x_i, y_i по правилу: если они оба отрицательны, то каждый из них увеличить на 0.5; если отрицательно только одно число, то отрицательное число заменить его квадратом; если оба числа неотрицательны, то каждое из них заменить на среднее арифметическое исходных значений.

Вариант №14

Даны действительные числа $a_1, \ldots, a_{30}$. Получить:

а) $\max(a_1 + a_{30}, a_2 + a_{29}, \ldots, a_{15} + a_{16})$;

б) $\min(a_1 \cdot a_{16}, a_2 \cdot a_{17}, \ldots, a_{15} \cdot a_{30})$.

Вариант №15

Даны действительные числа $a_1, \ldots, a_{20}$. Преобразовать эту последовательность по правилу: большее из a_i и a_{10+i} ($i = 1, \ldots, 10$) принять в качестве нового значения a_i, а меньшее – в качестве нового значения a_{10+i}.

Вариант №16

Даны целые числа $a_1, \ldots, a_{30}$. Если в данной последовательности ни одно четное число не расположено после нечетного, то распечатать все отрицательные члены последовательности, иначе – все положительные. Порядок следования чисел в обоих случаях заменяется на обратный.

Вариант №17

Даны действительные числа $r_1, \ldots, r_{17}$, среди которых заведомо есть как отрицательные, так и неотрицательные. Получить $x_1y_1 + \ldots + x_sy_s$, где $x_1, \ldots, x_p$ – отрицательные члены последовательности $r_1, \ldots, r_{17}$, взятые в порядке их следования, $y_1, \ldots, y_q$ – неотрицательные члены, взятые в обратном порядке, $s=\min(p,q)$.

Вариант №18

Даны целые числа $a_1, \ldots, a_{20}$. Наименьший член этой последовательности заменить целой частью среднего арифметического всех членов, остальные члены оставить без изменения. Если в последовательности несколько членов со значением $\min(a_1, \ldots, a_{20})$, то заменить последний по порядку.

Вариант №19

Даны действительные числа a_1, …, a_{20} (все числа попарно различны). Поменять в этой последовательности местами:

а) наибольший и наименьший члены;

б) наибольший и последний члены.

Вариант №20

Даны целые числа a_1, …, a_{100}. Получить новую последовательность из 100 целых чисел, заменяя a_i нулями, если $|a_i|$ не равно $\max(a_1, \ldots, a_{100})$, и заменяя a_i единицей в противном случае ($i = 1, \ldots, 100$).

Вариант №21

Даны целые числа a_1, …, a_{25}, b_1, …, b_{25}. Преобразовать последовательность b_1, …, b_{25} по правилу: если $a_i \le 0$, то b_i увеличить в 10 раз, иначе b_i заменить нулем (i=1, …, 25).

Вариант №22

Даны действительные числа a_1, …, a_{26}. Требуется домножить все члены этой последовательности на квадрат ее наименьшего члена, если $a_1 \ge 0$, и на квадрат ее наибольшего члена, если $a_1 < 0$.

Вариант №23

Даны натуральное число n, действительные числа a_1, …, a_{30}. Получить b_1, …, b_{10}, где b_i равно сумме тех членов последовательности a_1, …, a_{30}, которые принадлежат полуинтервалу $(i - 1, i]$ ($i = 1, \ldots, 10$). Если полуинтервал не содержит членов последовательности, то соответствующее b_i положить равным нулю.

Вариант №24

В массиве A[30] найти наибольшее число подряд идущих одинаковых элементов (например {1,5,3,6,6,6,6,6,3,4,4,5,5,5} = 5).

Вариант №25

В массиве B[30] найти и вывести значение наиболее часто встречающегося элемента.

Вариант №26

Интервал между минимальным и максимальным значениями элементов массива C[25] разбить пополам и относительно этого значения разбить массив на две части, каждую из которых записать в новый массив (части не сортировать).

Вариант №27

Найти в массиве $D[30]$ элемент, наиболее близкий к среднему арифметическому его элементов.

Вариант №28

Задана последовательность вещественных чисел $b_1, \ldots, b_{30}$. Сформировать одномерный массив A такой, что:

$$a_i = 1/(b_i - b_{i-1}),\ i = 2,3,\ldots,30;$$
$$a_1 = 1/(b_1 - b_{30}).$$

Последовательность $b_1, \ldots, b_{30}$ ввести с клавиатуры.

Рекомендуемый теоретический раздел для ознакомления: указатели и массивы.

7) Двумерные массивы

Вариант №1

Задана матрица $Z(5,4)$. Найти в каждой строке, если там есть отрицательный элемент, среднее арифметическое всех элементов, исключая нулевые и записать эти значения в массив B. Вывести исходную матрицу Z и массив B.

Вариант №2

В матрице $X(4,5)$ в каждой строке найти максимальный элемент и заменить им первый элемент строки. Предварительно первый элемент строки вывести в массив, если он не равен нулю. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №3

В матрице $Z(4,5)$ сдвинуть все элементы влево (циклически) в тех строках, которые начинаются с положительного элемента. Сдвинутые элементы вывести в массив. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №4

В матрице $Z(5,5)$ найти сумму элементов в тех строках, в которых элемент на главной диагонали равен нулю. Этой суммой заменить элемент на главной диагонали. Вывести исходную и преобразованную матрицы.

Вариант №5

Каждую строку матрицы $Z(5,4)$ преобразовать по правилу: если максимальный элемент не первый, то поменять его местом с первым. Вывести количество таких строк, исходную и преобразованную матрицы.

Вариант №6

В каждой строке матрицы *Z*(5,6) сдвинуть все элементы вправо на один разряд (циклически). Если при этом в последнем столбце оказался нуль, то заменить его числом *P*, введенным с клавиатуры. Элементы последнего столбца вывести в массив. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №7

Задана матрица *Z*(4,5). В каждой строке найти произведение элементов, расположенных до первого нулевого и их количество. Этим количеством заменить первый нулевой, а произведение записать в массив *B*. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №8

В матрице *Z*(4,5) переписать в обратном порядке элементы в тех строках, которые начинаются с нуля. Все отрицательные элементы вывести в массив B. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №9

Дана действительная квадратная матрица порядка 7. Если в *i*-ой строке матрицы элемент, принадлежащий главной диагонали, отрицателен, то заменить этот элемент суммой элементов *i*-той строки, предшествующих первому отрицательному элементу; в противном случае – суммой последних элементов *i*-той строки, начиная с первого отрицательного элемента. Элементы главной диагонали (неизмененные) вывести в массив. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №10

В матрице *Z*(5,6) первый отрицательный элемент каждого столбца заменить суммой оставшихся. Отрицательные элементы до замены вывести в массив *B*. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №11

Задана матрица *Z*(5,6). Выбрать строку с наибольшей суммой элементов и вывести элементы этой строки в массив, затем каждый отрицательный элемент умножить на номер столбца. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №12

Дана действительная квадратная матрица порядка 7. Вычислить сумму тех ее элементов, расположенных на главной диагонали и выше нее, которые превос-

ходят по величине все элементы, расположенные ниже главной диагонали. Заменить этой суммой элемент на главной диагонали соответствующего столбца. Если на главной диагонали и выше нее нет элементов с указанным свойством, то элемент на главной диагонали оставить без изменения. Элементы главной диагонали (неизмененные) вывести в массив. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №13

В матрице *Z*(5,6) поделить элементы нечетных столбцов на свой номер, если в остатке не нуль, то заменить этот элемент полученным значением. Вывести количество таких элементов, исходную и преобразованную матрицы.

Вариант №14

В матрице *Z*(5,5) найти номер строки, в которой содержится наибольшее количество отрицательных элементов. Количеством элементов (отрицательных) в каждой строке заменить соответствующий элемент главной диагонали. Вывести исходную и преобразованную матрицы.

Вариант №15

Даны две действительные квадратные матрицы порядка 6. Получить новую матрицу умножением элементов каждой строки первой матрицы на наибольшее из значений элементов соответствующей строки второй матрицы. Вывести исходные и полученную матрицы.

Вариант №16

В матрице *Z*(5,4) в каждой нечетной строке выполнить следующее преобразование: сложить все отрицательные элементы и заменить этой суммой элемент первого столбца, а все элементы первого столбца вывести в вектор. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №17

В матрице *Z*(4,6) поменять местами первый элемент в каждой строке с последним, второй – с предпоследним и т.д., если ни один из этих элементов не равен нулю. Вывести исходную и преобразованную матрицы.

Вариант №18

Даны две действительные квадратные матрицы порядка 6. Получить новую матрицу прибавлением к элементам каждого столбца первой матрицы произведения элементов соответствующих строк второй матрицы. Вывести исходные и полученную матрицы.

Вариант №19

В матрице *Z*(5,5) найти сумму элементов в тех строках, в которых элемент на главной диагонали равен нулю. Найти и вывести количество таких строк. Элемент на главной диагонали, равный нулю, заменить найденной суммой элементов. Вывести исходную и преобразованную матрицы.

Вариант №20

Дана матрица *Z*(5,5). В каждой четной строке просуммировать отрицательные элементы и заменить этой суммой первый элемент строки, в каждой нечетной строке найти количество отрицательных элементов. Вывести исходную и преобразованную матрицы.

Вариант №21

Дана целочисленная квадратная матрица порядка 6. Найти матрицу, получающуюся из данной перестановкой строк – первой с последней, второй – с предпоследней и т.д. Перестановка осуществляется при условии, что элемент главной диагонали обоих строк не равен нулю. Вывести исходную и преобразованную матрицы.

Вариант №22

Задана матрица *Z*(5,4). В каждой строке найти первый отрицательный элемент и заменить его произведением этого элемента на его номер. Первоначальное значение запомнить в массиве. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №23

Дана матрица *Z*(5,4). Найти номера строк, которые содержат не более двух отрицательных элементов. Эти элементы возвести в квадрат. Вывести номера строк, исходную и полученную матрицы.

Вариант №24

В матрице *Z*(4,6) в каждой строке элементы, стоящие на нечетных местах, заменить суммой, на четных – произведением соответствующей пары. Элементы главной диагонали вывести в массив. Распечатать результат в виде исходной и преобразованной матрицы, полученного массива.

Вариант №25

В массиве *B*[30] найти и вывести значение наиболее часто встречающегося элемента.

Вариант №26

Задана матрица *Z*(6,6). Элемент главной диагонали на каждой строке заменить суммой элементов, расположенных за ним (если элемент на главной диагонали не равен нулю). Элементы главной диагонали вывести в массив *B*. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №27

Дана матрица *Z*(4,5). Составить новую, заменив нулями элементы строки и столбца, где находится максимальный элемент. Элементы из строки и столбца переписать в вектор С. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №28

Дана матрица *Z*(4,6). Определить и вывести в массив *B* все элементы, которые в своей строке больше предыдущего и меньше последующего. Вывести исходную матрицу и полученный массив.

Рекомендуемый теоретический раздел для ознакомления: многомерные массивы.

8) Динамическое распределение памяти

Вариант №1

Разработать программу перемножения двух матриц *A, B* размерности *n*x*n*. Все матрицы размещаются в оперативной памяти динамически с помощью операции new, а значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходные и результирующую матрицы.

Вариант №2

Разработать программу нормирования матрицы размерности *m*x*n*, которое заключается в том, что каждый элемент в этой матрице вычисляется на основании исходной матрицы, как отношение суммы всех других элементов в его строке к сумме всех других элементов в его столбце. Матрица должна размещаться в оперативной памяти динамически с помощью операции new, значения *m* и *n* вводятся с клавиатуры по запросу. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

Вариант №3

Разработать программу, которая в матрице размерности *n*x*n* меняет местами строку, содержащую элемент с минимальным значением со столбцом, содержащим элемент с максимальным значением. Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

Вариант №4

Дана действительная квадратная матрица порядка *n*, все элементы которой различны. Найти наибольший элемент среди стоящих на главной и побочной диагоналях и поменять его местами с элементом, стоящим на пересечении этих диагоналей. Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

Вариант №5

Построить квадратную матрицу порядка 2*n*:

$$\begin{matrix} n\{ \\ n\{ \\ \\ \end{matrix} \begin{bmatrix} 11\ldots1 & 22\ldots2 \\ 11\ldots1 & 22\ldots2 \\ \ldots\ldots & \ldots\ldots \\ 11\ldots1 & 22\ldots2 \\ 33\ldots3 & 44\ldots4 \\ 33\ldots3 & \ldots4 \\ \ldots\ldots & \ldots\ldots \\ \underbrace{33\ldots3}_{n} & \underbrace{44\ldots}_{n} \end{bmatrix}$$

Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №6

Дано действительное число *x*. Получить квадратную матрицу порядка *n*<10:

$$\begin{bmatrix} 1 & x & \dots & x^{n-2} & x^{n-1} \\ x & 0 & \dots & 0 & x^{n-2} \\ \vdots & & & & \\ x^{n-2} & 0 & \dots & 0 & x \\ x^{n-1} & x^{n-2} & \dots & x & 1 \end{bmatrix}$$

Матрица размещается в памяти динамически с помощью операции new, значения x и n вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №7

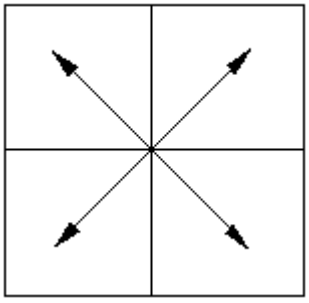
Даны действительные числа a_1, …, a_n. Получить квадратную матрицу порядка n:

$$\begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_{n-2} & a_{n-1} & a_n \\ a_2 & a_3 & a_4 & \dots & a_{n-1} & a_n & a_1 \\ a_3 & a_4 & a_5 & \dots & a_n & a_1 & a_2 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_n & a_1 & a_2 & \dots & a_{n-3} & a_{n-2} & a_{n-1} \end{bmatrix}$$

Матрица размещается в памяти динамически с помощью операции new, значение n и числа a_1, …, a_n вводятся по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №8

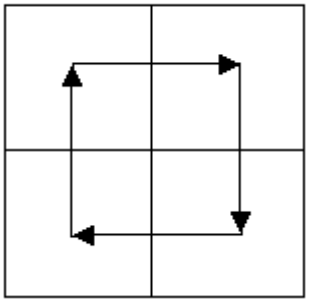
Дана действительная квадратная матрица порядка $2n$. Получить новую матрицу, переставляя ее блоки размера nxn:



Матрица размещается в памяти динамически с помощью операции new, значение n вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

Вариант №9

Дана действительная квадратная матрица порядка 2n. Получить новую матрицу, переставляя ее блоки размера *n*x*n*:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

Вариант №10

Получить квадратную матрицу порядка *n*:

$$\begin{bmatrix} n & & \\ n-1 & n & 0 \\ n-2 & n-1 & n \\ \cdots & \cdots & \cdots \\ 1 & 2 & 3 \ldots n \end{bmatrix}$$

Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №11

Получить квадратную матрицу порядка *n*:

$$\begin{bmatrix} 1 & 2 & 3 & \ldots & n-1 & n \\ 2 & 1 & 2 & \ldots & n-2 & n-1 \\ 3 & 2 & 1 & & n-3 & n-2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ n-1 & n-2 & n-3 & \ldots & 1 & 2 \\ n & n-1 & n-2 & \ldots & 2 & 1 \end{bmatrix}$$

Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №12

Дана действительная квадратная матрица порядка n. Преобразовать матрицу по правилу: строку с номером n сделать столбцом с номером n, а столбец с номером n сделать строкой с номером n. Матрица размещается в памяти динамически с помощью операции new, значение n вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

Вариант №13

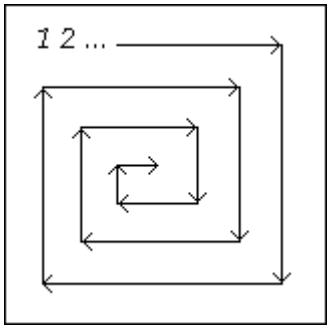
Даны две действительные квадратные матрицы порядка n. Получить новую матрицу умножением элементов каждой строки первой матрицы на наибольшее из значений элементов соответствующей строки второй матрицы. Матрицы размещаются в памяти динамически с помощью операции new, значение n вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходные и результирующую матрицы.

Вариант №14

Даны две действительные квадратные матрицы порядка n. Получить новую матрицу прибавлением к элементам каждого столбца первой матрицы произведения элементов соответствующих строк второй матрицы. Матрицы размещаются в памяти динамически с помощью операции new, значение n вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходные и результирующую матрицы.

Вариант №15

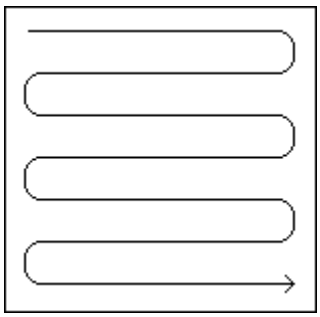
Получить целочисленную квадратную матрицу порядка $n<8$, элементами которой являются числа 1, 2, ..., n^2, расположенные в ней по спирали:



Матрица размещается в памяти динамически с помощью операции new, значение n вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №16

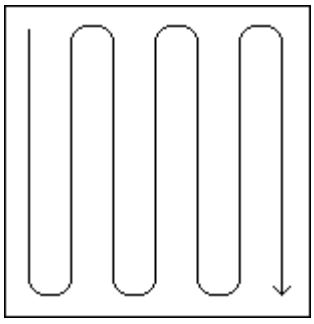
Даны действительные числа $a_1, a_2, ..., a_{n^2}$. Получить действительную квадратную матрицу порядка n<8, элементами которой являются числа $a_1, a_2, ..., a_{n^2}$, расположенные в ней по схеме:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №17

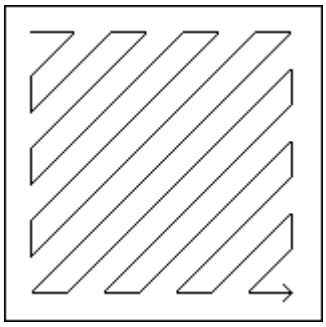
Даны действительные числа $a_1, a_2, ..., a_{n^2}$. Получить действительную квадратную матрицу порядка n<8, элементами которой являются числа $a_1, a_2, ..., a_{n^2}$, расположенные в ней по схеме:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №18

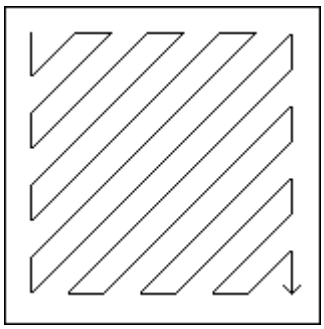
Даны действительные числа $a_1, a_2, ..., a_{n^2}$. Получить действительную квадратную матрицу порядка n<8, элементами которой являются числа $a_1, a_2, ..., a_{n^2}$, расположенные в ней по схеме:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №19

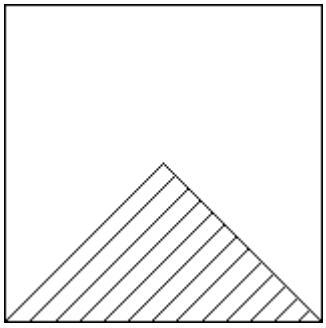
Даны действительные числа $a_1, a_2, ..., a_{n^2}$. Получить действительную квадратную матрицу порядка $n<8$, элементами которой являются числа $a_1, a_2, ..., a_{n^2}$, расположенные в ней по схеме:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести полученную матрицу.

Вариант №20

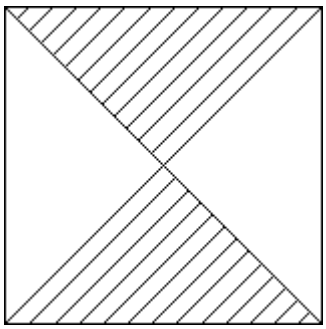
Дана действительная квадратная матрица порядка *n*. Найти и вывести наибольшее из значений элементов, расположенных в заштрихованной части матрицы:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную матрицу.

Вариант №21

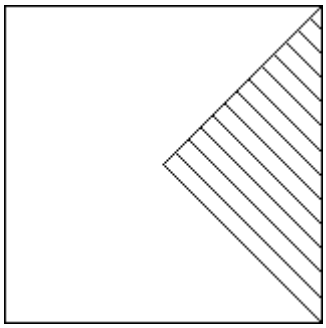
Дана действительная квадратная матрица порядка *n*. Найти и вывести наибольшее из значений элементов, расположенных в заштрихованной части матрицы:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную матрицу.

Вариант №22

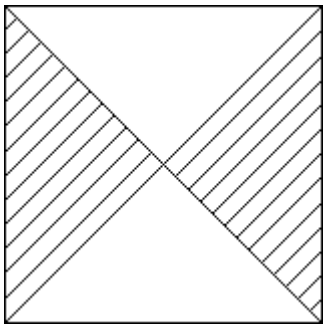
Дана действительная квадратная матрица порядка *n*. Найти и вывести наибольшее из значений элементов, расположенных в заштрихованной части матрицы:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную матрицу.

Вариант №23

Дана действительная квадратная матрица порядка *n*. Найти и вывести наибольшее из значений элементов, расположенных в заштрихованной части матрицы:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную матрицу.

Вариант №24

Дана целочисленная матрица размера *m*x*n*. Найти матрицу, получающуюся из данной перестановкой столбцов – первого с последним, второго с предпоследним и т.д. Матрица размещается в памяти динамически с помощью операции new, значения *m* и *n* вводятся по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

Вариант №25

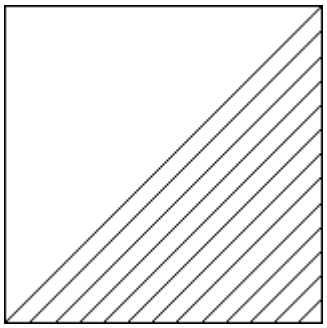
Дана целочисленная матрица размера размера *m*x*n*. Найти матрицу, получающуюся из данной перестановкой строк – первой с последней, второй с предпоследней и т.д. Матрица размещается в памяти динамически с помощью операции new, значения *m* и *n* вводятся по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

Вариант №26

Задана матрица *Z*(6,6). Элемент главной диагонали на каждой строке заменить суммой элементов, расположенных за ним (если элемент на главной диагонали не равен нулю). Элементы главной диагонали вывести в массив *B*. Вывести исходную и преобразованную матрицы, полученный массив.

Вариант №27

Дана действительная квадратная матрица порядка *n*. Найти и вывести наибольшее из значений элементов, расположенных в заштрихованной части матрицы:



Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную матрицу.

Вариант №28

Дана действительная квадратная матрица порядка *n*, все элементы которой различны. В этой матрице в каждой строке элементы, стоящие на нечетных мес-

тах, заменить суммой, на четных – произведением соответствующей пары. Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

Рекомендуемые теоретические разделы для ознакомления: операция выделения памяти new; операция освобождения памяти delete.

9) Работа с символьными данными

Вариант №1
Прочитать из файла строку символов. Преобразовать данную строку, удалив из нее каждую пару символов >> и повторив (вставив еще раз) каждую пару символов <<. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №2
Прочитать из файла строку символов. Расположить символы в этой строке так, чтобы сначала шли символы, больше введенного с клавиатуры, а затем меньше. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №3
Прочитать из файла строку символов. Данная строка состоит из нулей, единиц и пробелов. Группы нулей и единиц, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Требуется подсчитать и вывести количество слов в данной строке. Рассматривая слова как числа, определить и вывести количество слов, делящихся на 5 без остатка. Новую строку не создавать. Вывести исходную строку.

Вариант №4
Прочитать из файла строку символов. Написать функцию, которая изменяет данную строку. Если она находит в строке число, то она заменяет его на соответствующую по счету букву из этой строки (например, "*aaabb1bbcc5cc*" – "*aaabbabbccbcc*"). Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №5
Прочитать из файла строку символов. Преобразовать данную строку, содержащую выражение на Си с операциями (=, ==, !=, *a*+=, *a*–=), в строку, содержа-

щую эти же операции с синтаксисом языка Паскаль (:=, =, <>, *a*=*a*+, *a*=*a*–). Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №6

Прочитать из файла строку символов. Удалить из этой строки комментарии вида "/* ... */". Игнорировать вложенные комментарии. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №7

Прочитать из файла строку символов. Заменить в этой строке последовательности одинаковых символов (не пробелов) на число, соответствующее их количеству и один этот символ (т.е. "*abcdaaaaa xyznnnnnnn*" на "*abcd*5*a xyz*7*n*"). Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №8

Прочитать из файла строку символов. Подсчитать в этой строке наибольшее количество пробелов, идущих подряд. Новую строку не создавать. Вывести исходную строку и найденное число.

Вариант №9

Прочитать из файла строку символов. Из этой строки получить символы между первым и вторым двоеточием. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №10

Прочитать из файла строку символов. Найти во входной строке самую внутреннюю пару скобок {...} и переписать в выходную строку содержащиеся между ними символы. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №11

Прочитать из файла строку символов. Заменить в этой строке все целые числа соответствующим повторением следующего за ними символа (например "*abc5xacb15y*" – "*abcxxxxxacbyyyyyyyyyyyyyyy*"). Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №12

Прочитать из файла строку символов. Удалить в этой строке каждый символ * и повторить каждый символ, отличный от *. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №13

Прочитать из файла строку символов. Выяснить, имеется ли в этой строке пара одинаковых соседних букв и вывести их количество. Вывести исходную строку.

Вариант №14

Прочитать из файла строку символов. Выяснить, имеется ли в этой строке пара соседствующих букв *он* или *но* и вывести их количество. Вывести исходную строку.

Вариант №15

Прочитать из файла строку символов. Выяснить, верно ли, что среди символов имеются все буквы, входящие в слово *диаметр*. Вывести исходную строку и результат проверки.

Вариант №16

Прочитать из файла строку символов. Изменить данную строку так, чтобы каждый символ не цифра был заменен цифрой. Цифры при замене брать последовательно, начиная с 9 и заканчивая 0. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №17

Прочитать из файла строку символов. Если в ней встречается символ-цифра, то заменить его символом !, причем количество замен должно соответствовать значению цифры. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №18

Прочитать из файла строку символов. Изменить ее, заменив две подряд встречающиеся буквы *CH* символом *S*. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №19

Прочитать из файла строку символов. В этой строке подсчитать, сколько символов стоят перед символом ?, все последующие заменить символами !. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №20

Прочитать из файла строку символов. В данной строке каждый символ ? удвоить. Каждый символ, стоящий перед !, заменить символом, введенным с клавиатуры. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №21

Прочитать из файла строку символов. Зашифровать текст из этой строки так, чтобы каждый символ-цифра был заменен его порядковым номером. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №22

Прочитать из файла строку символов. В данной строке каждый символ-цифру заменить символом, введенным с клавиатуры, если символы равны, то заменить его следующим по алфавиту. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №23

Прочитать из файлов две строки символов. В первой оставить только те символы, которых нет во второй строке. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №24

Прочитать из файла строку символов. Исключить из данной строки все символы, идущие подряд, равные символу, введенному с клавиатуры. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №25

Прочитать из файла строку символов. Поместить в ее начале все латинские заглавные буквы, которые в ней присутствуют. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №26

Прочитать из файла строку символов. Изменить данную строку так, чтобы каждый символ не цифра был заменен цифрой. Цифры при замене брать последовательно, начиная с нуля. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №27

Прочитать из файла строку символов. Заменить в ней символ, равный символу, введенному с клавиатуры, символом ?, а все следующие – порядковыми номерами предыдущего символа. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №28

Прочитать из файла строку символов. Заменить в ней символ, равный символу, введенному с клавиатуры, его порядковым номером. Новую строку не создавать. Вывести исходную и преобразованную строки.

Рекомендуемый теоретический раздел для ознакомления: символьные массивы и строки.

11.2.2 Пример выполнения контрольной работы № 2

Для написания программ использовалась IDE Visual Studio, программы представляют консольные приложения.

Вариант № 28

1) Расчет арифметического выражение:

$$k = 1 + |y - x| + \frac{(y - x)^2}{\sqrt{arctg(xy)}}$$

```
#include <iostream>
#include <math.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
	double x,y;
	cout << "Введите положительные значения параметров x, y" << endl;
	cout << "x = ";
	cin >> x;
	cout << "y = ";
	cin >> y;
	if (x < 0 || y < 0)
	{
		cerr << "Параметры x,y должны быть положительными!";
		return 1;
	}
	double k = 1 + fabs(y - x) + pow(y - x, 2) / (sqrt(atan(x * y)));
	cout << "Результат вычисления k = " << k;
	return 0;
}
```

Результаты работы программы по вычислению арифметического выражения приведены в таблице 11.

Таблица 11 – Результаты расчета арифметического выражения

x	y	Результат
1	1	Результат вычисления k = 1
5	7	Результат вычисления k = 6,20779
-1	3	Параметры x,y должны быть положительными!

2) Вычисление условного выражения:

$$p = \begin{cases} \sqrt{|a \cdot b|} + 2 \cdot c, a \cdot b < -2 \\ a^3 + b^2 - c^2, -2 \le a \cdot b \le 2 \\ a^c - b, a \cdot b > 2 \end{cases}$$

```
#include <iostream>
#include <math.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
	double a, b, c, p;
	cout << "Введите значения параметров a, b, c" << endl;
	cout << "a = ";
	cin >> a;
	cout << "b = ";
	cin >> b;
	cout << "c = ";
	cin >> c;
	if (a * b > 2.0)
	{
		p = pow(a, c) - b;
	}
	else if (a * b < -2.0)
	{
		p = sqrt(fabs(a * b)) + 2 * c;
	}
	else
	{
		p = a * a * a + b * b - c * c;
	}
	cout << "Результат условного выражения p = " << p;
	return 0;
}
```

Результаты работы программы по вычислению условного выражения приведены в таблице 12.

Таблица 12 – Результаты расчета условного выражения

a	b	c	p
2	1	4	-7
-1	5	1	4.23607
3	2	-1	-1.6667

3) Вычислить функцию на интервале и вывести таблицу:

$$y = \begin{cases} \sin(x/2), x > 0.5 \\ 2x, x = 0.5 \\ \cos\left(\frac{|2x|}{0.5\pi}\right), x < 0.5 \end{cases}$$
$$x \in [0,1.5], h = 0.1$$

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <math.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
	double y;
	cout << "    x     |     y    " << endl;
	cout << "----------|----------" << left << endl;
	for(double x = 0.0; x < 1.6; x += 0.1)
	{
		if (x < 0.5)
		{
			y = cos(fabs(2 * x)/(0.5 * M_PI));
		}
		else if (x > 0.5)
		{
			y = sin(x / 2.0);
		}
		else
		{
			y = 2.0 * x;
		}
		cout.width(10);
		cout.precision(10);
		cout << x << "|" << y << endl;
	}
	return 0;
}
```

Результаты работы программы по вычислению функции на заданном интервале приведены в таблице 13.

Таблица 13 – Результаты работы программы по вычислению функции на заданном интервале

x	y
0	1
0.1	0.9919052498
0.2	0.9677520491
0.3	0.9279314261
0.4	0.8730880569
0.5	1
0.6	0.2955202067
0.7	0.3428978075
0.8	0.3894183423
0.9	0.4349655341
1	0.4794255386
1.1	0.5226872289
1.2	0.5646424734
1.3	0.6051864057
1.4	0.6442176872
1.5	0.68163876

4) Ввести последовательность из 8 символов. Если символ – латинская гласная буква, то заменить в нем 2 младших бита нулем, иначе – 2-й и 4-й единицами. Вывести исходную и преобразованную последовательности в символьной форме и в восьмеричных кодах.

```
#include <iostream>
#include <conio.h>
#include <ctype.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
	int c;
	char symbol[8], ch;
	const char *vowel = "eyuioaEYUIOA";
	cout << "Введите 8 символов латинского алфавита: ";
const int size_mass = 8;
	for (int count = 0; count < size_mass;)
	{
		c = getch();
		if(isalpha(c))
		{
			ch = static_cast<char>(c);
			symbol[count++] = ch;
			cout << ch << ' ';
		}
	}
	cout << endl << oct;
	for(int i = 0; i < size_mass; i++)
	{
		if (strchr(vowel, symbol[i]))
			ch = static_cast<char>(symbol[i] & 0xFC);
		else
```

```
                ch = static_cast<char>(symbol[i] | 0x0A);
            cout << symbol[i] << '\t' << static_cast<int>(symbol[i])
                << "   |   " << ch << '\t' << static_cast<int>(ch) << endl;
      }
      return 0;
}
```

q 161 | { 173
a 141 | ` 140
z 172 | z 172
w 167 | ⌂ 177
s 163 | { 173
x 170 | z 172
e 145 | d 144
d 144 | n 156

5) Вычислить сумму

$$S = \sum_{i=1}^{1000} r_i$$

$$r_1 = 1.23;\ r_2 = 0.65;\ r_3 = -0.25;$$

$$r_k = \frac{1.2\sin(r_{k-1} + r_{k-2}) + 0.8\, r_{k-3}}{1 - \cos(r_{k-1})} - [0.55\, r_{k-2}]$$

$k = 4, 5, \ldots, 1000$.

[] – обозначение целой части.

Массивом не пользоваться.

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

double calculate(double r1, double r2, double r3)
{
      return (1.2 * sin(r3 + r2) + 0.8 * r1) /
(1 - cos(r3)) - floor(0.55 * r2);
}
int _tmain(int argc, _TCHAR* argv[])
{
      double r1 = 1.23, r2 = 0.65, r3 = -0.25, result;
      for(int i = 3; i < 1000; i++)
      {
            result = calculate(r1, r2, r3);
            r1 = r2; r2 = r3; r3 = result;
      }
      cout << "Результат вычисления рекуррентного выражения: " << result;
      return 0;
}
```

Результат вычисления рекуррентного выражения: 5.60909*e+205.

6) Задана последовательность вещественных чисел $b_1, \ldots, b_{30}$. Сформировать одномерный массив A такой, что:

$$a_i = 1/(b_i - b_{i-1}),\ i = 2,3,\ldots,30;$$
$$a_1 = 1/(b_1 - b_{30}).$$

Последовательность $b_1, \ldots, b_{30}$ ввести с клавиатуры.

```
#include <iostream>
#include <math.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
      int i;
      const int size_mass = 30;
      double a[size_mass], b[size_mass];
      cout << "Введите последовательность вещественных чисел:" << endl;
      for(i = 0; i < size_mass; i++)
      {
            cout << "b" << i + 1 << " = ";
            cin >> b[i];
      }
      a[0] = 1.0 / (b[0] - b[size_mass - 1]);
      for(i = 1; i < size_mass; i++)
            a[i] = 1.0 / (b[i] - b[i - 1]);
      for(i = 0; i < size_mass; i++)
            cout << "b" << i + 1 << " = " << b[i] << "   |   " << "a" << i + 1
<< " = " << a[i] << endl;
      return 0;}
```

7) Дана матрица $Z(4,6)$. Определить и вывести в массив B все элементы, которые в своей строке больше предыдущего и меньше последующего. Вывести исходную матрицу и полученный массив.

```
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
      int i, j, elements = 0;
      const int row_size = 4, col_size = 6;
      double matrix[row_size][col_size] =
                                    {{1.1, 4.7, 7.2, 5.0, 2.7, 8.3},
                                    {5.6, 6.0, 77.1, 99.0, 1.2, 3.4},
                                    {7.2, 5.4, 7.8, 7.9, 8.9, 10.0},
                                    {5.5, 2.3, 4.6, 6.0, 7.8, 11.2}};
      double vec[row_size * col_size];
      for(i = 0; i < row_size; i++)
            for(j = 0; j < col_size; j++)
            {
                  if (j > 0 && j < col_size - 1 &&
                        (matrix[i][j - 1] < matrix[i][j]
&& matrix[i][j] < matrix[i][j + 1]))
                  {
                        vec[elements++] = matrix[i][j];
                  }
            }
      cout << "Исходная матрица: " << endl;
      for(i = 0; i < row_size; i++)
      {
            for(j = 0; j < col_size; j++)
            {
                  cout << matrix[i][j] << "    ";
            }
```

```
            cout << endl;
      }
      cout << endl << "Вектор результатов: " << endl;
      for (i = 0; i < elements; i++)
            cout << vec[i] << "   ";
      return 0;
}
```

Результирующий вектор:

4.7	6	77.1	7.8	7.9	8.9	4.6	6	7.8

8) Дана действительная квадратная матрица порядка *n*, все элементы которой различны. В этой матрице в каждой строке элементы, стоящие на нечетных местах, заменить суммой, на четных – произведением соответствующей пары. Матрица размещается в памяти динамически с помощью операции new, значение *n* вводится по запросу с клавиатуры. В конце работы программы освободить выделенную память. Вывести исходную и результирующую матрицы.

```
include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
      int n, i, j, temp;
      cout << "Введите размер квадратной матрицы n = ";
      cin >> n;
      int** matrix = new int*[n];
      for(i = 0; i < n; i++)
            matrix[i] = new int[n];
      srand(static_cast<unsigned>(time(NULL)));
      cout << "Исходная матрица:" << endl;
      for(i = 0; i < n; i++)
      {
            for(j = 0; j < n; j++)
            {
                  temp = rand();
                  matrix[i][j] = temp;
                  cout << temp << "   ";
            }
            cout << endl;
      }
      cout << "Результирующая матрица:" << endl;
      for(i = 0; i < n; i++)
      {
            for(j = 0; j < n; j++)
            {
                  matrix[i][j] = (j + 1) % 2 ? i + j + 2: (i + 1) * (j + 1);
                  cout << matrix[i][j] << "   ";
            }
            cout << endl;
      }
      for(i = 0; i < n; i++)
            delete[]matrix[i];
      delete[]matrix;
      return 0;}
```

Результаты работы программы по вычислению матриц приведены в таблице 14.

Таблица 14 – Результаты работы программы по вычислению матриц

Исходная матрица при n = 3:			Результирующая матрица при n = 3:		
14568	15923	20173	2	2	4
4418	11273	14447	3	4	5
13180	32328	21502	4	6	6

9) Прочитать из файла строку символов. Заменить в ней символ, равный символу, введенному с клавиатуры, его порядковым номером. Новую строку не создавать. Вывести исходную и преобразованную строки.

```
#include <iostream>
#include <fstream>
using namespace std;
char* getStringFromFile(const char* filename) throw (char*)
{
	int len;
	ifstream in(filename, ios::binary | ios::in);
	if(!in) throw "Не возможно открыть файл";
	if(!(len = in.seekg(0, ios::end).tellg())) throw "Пустой файл";
	in.seekg(0, ios::beg);
	char *buffer = new char[len + 1];
	in.read(buffer, len);
	buffer[len] = 0;
	in.close();
	return buffer;
}
int _tmain(int argc, _TCHAR* argv[])
{
	char *str, ch;
	const char *filename = "test.txt";
	try
	{
		str = getStringFromFile(filename);
		cout << "Введите искомый символ: ";
		cin >> ch;
		cout << "Исходная строка: " << str << endl;
		cout << "Результирующая строка: ";
		for(int i = 0; i < strlen(str); i++)
		{
			if(str[i] == ch) cout << i + 1;
			else cout << str[i];
		}
		delete[]str;
	}
	catch (char* s)
	{
		cerr << s << endl;
	}
	return 0;}
```

Введите искомый символ: l

Исходная строка: Hello world!

Результирующая строка: He34o wor10d!

11.3 КОНТРОЛЬНАЯ РАБОТА №3. Объектно-ориентированное программирование

11.3.1 Задания контрольной работы №3

1) Классы. Протокол класса. Конструкторы и деструкторы

Вариант №1

Создать абстрактный тип данных – класс вектор, который имеет указатель на int, число элементов и переменную состояния. Определить конструктор без параметров, конструктор с параметром, конструктор с двумя параметрами. Конструктор без параметров выделяет место для одного элемента и инициализирует его в ноль. Конструктор с одним параметром, – размер вектора, – выделяет место и инициализирует номером в массиве, конструктор с двумя параметрами выделяет место (первый аргумент) и инициализирует вторым аргументом. Деструктор освобождает память. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавливать код ошибки, когда не хватает памяти, выходит за пределы массива. Определить функцию печати. Определить функции сложения, умножения, вычитания, которые производят эти арифметические операции с данными этого класса и встроенного int. Определить методы сравнения: больше, меньше или равно. Предусмотреть возможность подсчета числа объектов данного типа. Проверить работу этого класса.

Вариант №2

Создать класс матрица. Данный класс содержит указатель на int, размер строк и столбцов и состояние ошибки. Определить конструктор без параметров, конструктор с одним параметром и конструктор с двумя параметрами, деструктор. Определить методы доступа: возвращать значение элемента (*i, j*) и адрес этого элемента. Определить функцию печати. Определить функции сложения и вычитания (матрицы с матрицей), умножение матрицы на матрицу. Определить умножение матрицы на число. Проверить работу этого класса. В случае нехватки памяти, несоответствия размерностей, выхода за пределы устанавливать код ошибки.

Вариант №3

Создать класс типа – дата с полями: день (1-31), месяц (1-12), год (целое число). Класс имеет конструктор. Функции-члены установки дня, месяца и года, функ-

ции-члены получения дня, месяца и года, а также две функции-члены печати: печать по шаблону: «5 января 1997 года» и «05.01.1997». Функции-члены установки полей класса должны проверять корректность задаваемых параметров.

Вариант №4

Создать абстрактный тип данных – класс вектор, который имеет указатель на float, число элементов и переменную состояния. Определить конструктор без параметров, конструктор с параметром, конструктор с двумя параметрами. Конструктор без параметров выделяет место для одного элемента и инициализирует его в ноль. Конструктор с одним параметром, – размер вектора, – выделяет место и инициализирует номером в массиве, конструктор с двумя параметрами выделяет место (первый аргумент) и инициализирует вторым аргументом. Деструктор освобождает память. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавливать код ошибки, когда не хватает памяти, выходит за пределы массива. Определить функцию печати. Определить функции сложения, умножения, вычитания, которые производят эти арифметические операции с данными этого класса и встроенного float. Определить методы сравнения: больше, меньше или равно. Предусмотреть возможность подсчета числа объектов данного типа. Проверить работу этого класса.

Вариант №5

Создать класс матрица Данный класс содержит указатель на float, размер строк и столбцов и состояние ошибки. Определить конструктор без параметров, конструктор с одним параметром и конструктор с двумя параметрами, деструктор. Определить методы доступа: возвращать значение элемента (*i, j*) и адрес этого элемента. Определить функцию печати. Определить функции сложения и вычитания (матрицы с матрицей), умножение матрицы на матрицу. Определить умножение матрицы на число. Проверить работу этого класса. В случае нехватки памяти, несоответствия размерностей, выхода за пределы устанавливать код ошибки.

Вариант №6

Создать класс типа – время с полями: час (0-23), минуты (0-59), секунды (0-59). Класс имеет конструктор. Функции-члены установки времени, функции-члены получения часа, минуты и секунды, а также две функции-члены печати: печать по шаблону: «16 часов 18 минут 3 секунды» и «4 p.m. 18 минут 3 секунды». Функции-члены установки полей класса должны проверять корректность задаваемых параметров.

Вариант №7

Создать абстрактный тип данных – класс вектор, который имеет указатель на double, число элементов и переменную состояния. Определить конструктор без параметров, конструктор с параметром, конструктор с двумя параметрами. Конструктор без параметров выделяет место для одного элемента и инициализирует его в ноль. Конструктор с одним параметром, - размер вектора, - выделяет место и инициализирует номером в массиве, конструктор с двумя параметрами выделяет место (первый аргумент) и инициализирует вторым аргументом. Деструктор освобождает память. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавливать код ошибки, когда не хватает памяти, выходит за пределы массива. Определить функцию печати. Определить функции сложения, умножения, вычитания, которые производят эти арифметические операции с данными этого класса и встроенного double. Определить методы сравнения: больше, меньше или равно. Предусмотреть возможность подсчета числа объектов данного типа. Проверить работу этого класса.

Вариант №8

Создать класс матрица Данный класс содержит указатель на double, размер строк и столбцов и состояние ошибки. Определить конструктор без параметров, конструктор с одним параметром и конструктор с двумя параметрами, деструктор. Определить методы доступа: возвращать значение элемента (*i, j*) и адрес этого элемента. Определить функцию печати. Определить функции сложения и вычитания (матрицы с матрицей), умножение матрицы на матрицу. Определить умножение матрицы на число. Проверить работу этого класса. В случае нехватки памяти, несоответствия размерностей, выхода за пределы устанавливать код ошибки.

Вариант №9

Создать класс типа – прямоугольник. Поля – высота и ширина. Функции-члены вычисляют площадь, периметр, устанавливают поля и возвращают значения. Функции-члены установки полей класса должны проверять корректность задаваемых параметров. Функция печати.

Вариант №10

Создать абстрактный тип данных – класс вектор, который имеет указатель на long, число элементов и переменную состояния. Определить конструктор без параметров, конструктор с параметром, конструктор с двумя параметрами.

Конструктор без параметров выделяет место для одного элемента и инициализирует его в ноль. Конструктор с одним параметром, – размер вектора, – выделяет место и инициализирует номером в массиве, конструктор с двумя параметрами выделяет место (первый аргумент) и инициализирует вторым аргументом. Деструктор освобождает память. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавливать код ошибки, когда не хватает памяти, выходит за пределы массива. Определить функцию печати. Определить функции сложения, умножения, вычитания, которые производят эти арифметические операции с данными этого класса и встроенного long. Определить методы сравнения: больше, меньше или равно. Предусмотреть возможность подсчета числа объектов данного типа. Проверить работу этого класса.

Вариант №11

Создать класс матрица Данный класс содержит указатель на long, размер строк и столбцов и состояние ошибки. Определить конструктор без параметров, конструктор с одним параметром и конструктор с двумя параметрами, деструктор. Определить методы доступа: возвращать значение элемента (*i, j*) и адрес этого элемента. Определить функцию печати. Определить функции сложения и вычитания (матрицы с матрицей), умножение матрицы на матрицу. Определить умножение матрицы на число. Проверить работу этого класса. В случае нехватки памяти, несоответствия размерностей, выхода за пределы устанавливать код ошибки.

Вариант №12

Создать класс типа – циклическая очередь. Функции-члены получают элемент и вставляют элемент.

Вариант №13

Создать абстрактный тип данных – класс вектор, который имеет указатель на int, число элементов и переменную состояния. Определить конструктор без параметров, конструктор с параметром, конструктор с двумя параметрами. Конструктор без параметров выделяет место для одного элемента и инициализирует его в ноль. Конструктор с одним параметром, – размер вектора, – выделяет место и инициализирует номером в массиве, конструктор с двумя параметрами выделяет место (первый аргумент) и инициализирует вторым аргументом. Деструктор освобождает память. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавли-

вать код ошибки, когда не хватает памяти, выходит за пределы массива. Определить функцию печати. Определить функции сложения, умножения, вычитания, которые производят эти арифметические операции с данными этого класса и встроенного int. Определить методы сравнения: больше, меньше или равно. Предусмотреть возможность подсчета числа объектов данного типа. Проверить работу этого класса.

Вариант №14

Создать класс матрица Данный класс содержит указатель на int, размер строк и столбцов и состояние ошибки. Определить конструктор без параметров, конструктор с одним параметром и конструктор с двумя параметрами, деструктор. Определить методы доступа: возвращать значение элемента (*i, j*) и адрес этого элемента. Определить функцию печати. Определить функции сложения и вычитания (матрицы с матрицей), умножение матрицы на матрицу. Определить умножение матрицы на число. Проверить работу этого класса. В случае нехватки памяти, несоответствия размерностей, выхода за пределы устанавливать код ошибки.

Вариант №15

Создать класс типа – двухсвязный список. Функции-члены добавляют элемент к списку, удаляют элемент из списка. Отображают элементы списка от начала и от конца. Найти элемент в списке.

Вариант №16

Создать абстрактный тип данных – класс вектор, который имеет указатель на float, число элементов и переменную состояния. Определить конструктор без параметров, конструктор с параметром, конструктор с двумя параметрами. Конструктор без параметров выделяет место для одного элемента и инициализирует его в ноль. Конструктор с одним параметром, – размер вектора, – выделяет место и инициализирует номером в массиве, конструктор с двумя параметрами выделяет место (первый аргумент) и инициализирует вторым аргументом. Деструктор освобождает память. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавливать код ошибки, когда не хватает памяти, выходит за пределы массива. Определить функцию печати. Определить функции сложения, умножения, вычитания, которые производят эти арифметические операции с данными этого класса и встроенного float. Определить методы сравнения: больше, меньше или равно. Предусмотреть возможность подсчета числа объектов данного типа. Проверить работу этого класса.

Вариант №17

Создать класс матрица Данный класс содержит указатель на float, размер строк и столбцов и состояние ошибки. Определить конструктор без параметров, конструктор с одним параметром и конструктор с двумя параметрами, деструктор. Определить методы доступа: возвращать значение элемента (*i, j*) и адрес этого элемента. Определить функцию печати. Определить функции сложения и вычитания (матрицы с матрицей), умножение матрицы на матрицу. Определить умножение матрицы на число. Проверить работу этого класса. В случае нехватки памяти, несоответствия размерностей, выхода за пределы устанавливать код ошибки.

Вариант №18

Создать абстрактный тип данных – классс вектор, который имеет указатель на double, число элементов и переменную состояния. Определить конструктор без параметров, конструктор с параметром, конструктор с двумя параметрами. Конструктор без параметров выделяет место для одного элемента и инициализирует его в ноль. Конструктор с одним параметром, – размер вектора, – выделяет место и инициализирует номером в массиве, конструктор с двумя параметрами выделяет место (первый аргумент) и инициализирует вторым аргументом. Деструктор освобождает память. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавливать код ошибки, когда не хватает памяти, выходит за пределы массива. Определить функцию печати. Определить функции сложения, умножения, вычитания, которые производят эти арифметические операции с данными этого класса и встроенного double. Определить методы сравнения: больше, меньше или равно. Предусмотреть возможность подсчета числа объектов данного типа. Проверить работу этого класса.

Вариант №19

Создать класс матрица Данный класс содержит указатель на double, размер строк и столбцов и состояние ошибки. Определить конструктор без параметров, конструктор с одним параметром и конструктор с двумя параметрами, деструктор. Определить методы доступа: возвращать значение элемента (*i, j*) и адрес этого элемента. Определить функцию печати. Определить функции сложения и вычитания (матрицы с матрицей), умножение матрицы на матрицу. Определить умножение матрицы на число. Проверить работу этого класса. В случае нехватки памяти, несоответствия размерностей, выхода за пределы устанавливать код ошибки.

Вариант №20

Создать класс типа – односвязный список. Функции-члены добавляют элемент к списку, удаляют элемент из списка. Отображают элементы списка от начала. Найти элемент в списке.

Вариант №21

Создать абстрактный тип данных – класс вектор, который имеет указатель на long, число элементов и переменную состояния. Определить конструктор без параметров, конструктор с параметром, конструктор с двумя параметрами. Конструктор без параметров выделяет место для одного элемента и инициализирует его в ноль. Конструктор с одним параметром, – размер вектора, – выделяет место и инициализирует номером в массиве, конструктор с двумя параметрами выделяет место (первый аргумент) и инициализирует вторым аргументом. Деструктор освобождает память. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавливать код ошибки, когда не хватает памяти, выходит за пределы массива. Определить функцию печати. Определить функции сложения, умножения, вычитания, которые производят эти арифметические операции с данными этого класса и встроенного long. Определить методы сравнения: больше, меньше или равно. Предусмотреть возможность подсчета числа объектов данного типа. Проверить работу этого класса.

Вариант №22

Создать класс матрица Данный класс содержит указатель на long, размер строк и столбцов и состояние ошибки. Определить конструктор без параметров, конструктор с одним параметром и конструктор с двумя параметрами, деструктор. Определить методы доступа: возвращать значение элемента (*i, j*) и адрес этого элемента. Определить функцию печати. Определить функции сложения и вычитания (матрицы с матрицей), умножение матрицы на матрицу. Определить умножение матрицы на число. Проверить работу этого класса. В случае нехватки памяти, несоответствия размерностей, выхода за пределы устанавливать код ошибки.

Вариант №23

Создать класс типа – окружность. Поля – радиус. Функции-члены вычисляют площадь, длину окружности, устанавливают поля и возвращают значения. Функции-члены установки полей класса должны проверять корректность задаваемых параметров. Функция печати.

Вариант №24

Создать абстрактный тип данных – класс вектор, который имеет указатель на int, число элементов и переменную состояния. Определить конструктор без параметров, конструктор с параметром, конструктор с двумя параметрами. Конструктор без параметров выделяет место для одного элемента и инициализирует его в ноль. Конструктор с одним параметром, – размер вектора, – выделяет место и инициализирует номером в массиве, конструктор с двумя параметрами выделяет место (первый аргумент) и инициализирует вторым аргументом. Деструктор освобождает память. Определить функцию, которая присваивает элементу массива некоторое значение (параметр по умолчанию), функцию, которая получает некоторый элемент массива. В переменную состояния устанавливать код ошибки, когда не хватает памяти, выходит за пределы массива. Определить функцию печати. Определить функции сложения, умножения, вычитания, которые производят эти арифметические операции с данными этого класса и встроенного int. Определить методы сравнения: больше, меньше или равно. Предусмотреть возможность подсчета числа объектов данного типа. Проверить работу этого класса.

Вариант №25

Прочитать из файла строку символов. Поместить в ее начале все латинские заглавные буквы, которые в ней присутствуют. Новую строку не создавать. Вывести исходную и преобразованную строки.

Вариант №26

Создать класс матрица Данный класс содержит указатель на int, размер строк и столбцов и состояние ошибки. Определить конструктор без параметров, конструктор с одним параметром и конструктор с двумя параметрами, деструктор. Определить методы доступа: возвращать значение элемента (*i, j*) и адрес этого элемента. Определить функцию печати. Определить функции сложения и вычитания (матрицы с матрицей), умножение матрицы на матрицу. Определить умножение матрицы на число. Проверить работу этого класса. В случае нехватки памяти, несоответствия размерностей, выхода за пределы устанавливать код ошибки.

Вариант №27

Создать класс типа – дата с полями: день (1-31), месяц (1-12), год (целое число). Класс имеет конструктор. Функции-члены установки дня, месяца и года, функции-члены получения дня, месяца и года, а также две функции-члены печати: печать по шаблону: «5 января 2008 года» и «05.01.2008». Функции-члены уста-

новки полей класса должны проверять корректность задаваемых параметров. Функция-член дает приращение на 1 день.

Вариант №28

Создайте класс, который реализует безопасный двухмерный массив целых чисел размерностью (2x3). Класс должен содержать методы для произвольного защищенного доступа к элементам массива, метод для записи элементов массива через возвращаемое значение ссылки.

2) Преобразование типов. Дружественные функции. Конструктор копирования

Вариант №1

Создать класс комплексных чисел, члены класса – реальная и мнимая части. Класс имеет конструктор по умолчанию, конструктор, преобразующий float в объект класса. Определить оператор преобразования объекта типа комплексных чисел в число типа float. Создать класс вещественных чисел. Определить взаимное преобразование с классом комплексных чисел.

Вариант №2

Создать класс комплексных чисел. Определить перегруженную функцию, возвращающую максимальный из двух аргументов. Функция не является членом класса комплексных чисел. Перегруженные функции имеют аргументы типа int, double, complex. Тело перегруженных функций должны быть одинаковыми.

Вариант №3

Создать два класса вектор (int *) и матрица (int **). Определить конструкторы по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицу на вектор как дружественную.

Вариант №4

Создать класс вещественных чисел. Класс имеет конструктор по умолчанию, конструктор – преобразующий float в объект класса. Определить оператор преобразования объекта типа вещественных чисел в число типа float. Создать класс целых чисел. Определить взаимное преобразование с классом вещественных чисел.

Вариант №5

Создать класс комплексных чисел. Определить перегруженную функцию, возвращающую минимальный из двух аргументов. Функция не является членом класса комплексных чисел. Перегруженные функции имеют аргументы типа int, double, complex. Тело перегруженных функций должны быть одинаковыми.

Вариант №6

Создать два класса: целые(Integer) и матрица (int *). Определить конструкторы по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицу на целое как дружественную.

Вариант №7

Создать класс целых чисел. Класс имеет конструктор по умолчанию, конструктор, преобразующий int в объект класса. Определить оператор преобразования объекта типа целых чисел в число типа int. Создать класс вещественных чисел. Определить взаимное преобразование с классом целых чисел.

Вариант №8

Создать класс целых чисел Integer. Определить перегруженную функцию, возвращающую максимальное из двух аргументов. Функция не является членом класса целых чисел. Перегруженные функции имеют аргументы типа int, double, Integer. Тело перегруженных функций должны быть одинаковыми.

Вариант №9

Создать два класса: вещественные (Float) и матрица (float **). Определить конструкторы по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицу на вещественное (Float) как дружественную.

Вариант №10

Создать класс целых чисел Integer. Определить перегруженную функцию, возвращающую минимальное из двух аргументов. Функция не является членом класса целых чисел. Перегруженные функции имеют аргументы типа int, double, Integer. Тело перегруженных функций должны быть одинаковыми.

Вариант №11

Создать класс вещественных чисел Double. Определить перегруженную функцию, возвращающую максимальное из двух аргументов. Функция не является

членом класса Double. Перегруженные функции имеют аргументы типа int, double, Double. Тело перегруженных функций должны быть одинаковыми.

Вариант №12

Создать два класса вектор (double *) и матрица (double **). Определить конструкторы по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицу на вектор как дружественную.

Вариант №13

Создать два класса вектор (float *) и матрица (float **). Определить конструкторы – по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицу на вектор как дружественную.

Вариант №14

Создать класс вещественных чисел Double. Определить перегруженную функцию, возвращающую максимальное из двух аргументов. Функция не является членом класса Double. Перегруженные функции имеют аргументы типа int, double, Double. Тело перегруженных функций должны быть одинаковыми.

Вариант №15

Создать два класса вектор (long *) и матрица (long **). Определить конструкторы по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицу на вектор как дружественную.

Вариант №16

Создать класс вещественных чисел Float. Определить перегруженную функцию, возвращающую минимальное из двух аргументов. Функция не является членом класса Float. Перегруженные функции имеют аргументы типа int, float, Float. Тело перегруженных функций должны быть одинаковыми.

Вариант №17

Создать класс комплексных чисел. Определить перегруженную функцию, возвращающую максимальный из двух аргументов. Функция не является членом класса комплексных чисел. Перегруженные функции имеют аргументы типа int, double, complex. Тело перегруженных функций должны быть одинаковыми.

Вариант №18

Создать два класса вектор (int *) и матрица (int **). Определить конструкторы по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицу на вектор как дружественную.

Вариант №19

Создать класс вещественных с двойной точностью чисел. Класс имеет конструктор по умолчанию, конструктор – преобразующий double в объект класса. Определить оператор преобразования объекта типа вещественных чисел с двойной точностью в число типа double. Создать класс целых чисел. Определить взаимное преобразование с классом вещественных чисел.

Вариант №20

Создать класс комплексных чисел. Определить перегруженную функцию, возвращающую максимальный из двух аргументов. Функция не является членом класса комплексных чисел. Перегруженные функции имеют аргументы типа int, double, complex. Тело перегруженных функций должны быть одинаковыми.

Вариант №21

Создать два класса вектор (float *) и матрица (float **). Определить конструкторы по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицу на вектор.

Вариант №22

Определить два класс, строку с преобразование из char * в строку и обратно и Целое Int с преобразованием из int и обратно, а также взаимное преобразование String и Int.

Вариант №23

Создать класс целых чисел Integer. Определить перегруженную функцию, возвращающую максимальное из двух аргументов. Функция не является членом класса целых чисел. Перегруженные функции имеют аргументы типа int, double, Integer. Тело перегруженных функций должны быть одинаковыми.

Вариант №24

Создать два класса вектор (long *) и матрица (long **). Определить конструкторы по умолчанию, с параметром, для класса матрица с двумя параметрами, копирования, деструкторы. Определить функцию умножения матрицу на вектор

Вариант №25

Создать класс коротких целых чисел. Класс имеет конструктор по умолчанию, конструктор, преобразующий short int в объект класса. Определить оператор преобразования объекта типа короткое целое в число типа short int. Создать класс вещественных чисел. Определить взаимное преобразование с классом коротких целых чисел.

Вариант №26

Создать класс вещественных чисел Double. Определить перегруженную функцию, возвращающую максимальное из двух аргументов. Функция не является членом класса Double. Перегруженные функции имеют аргументы типа int, double, Double. Тело перегруженных функций должны быть одинаковыми.

Вариант №27

Создать два класса: целые (Integer) и вектор (int *). Определить конструкторы по умолчанию, с параметром, для класса вектор с двумя параметрами, копирования, деструкторы. Определить функцию умножения вектора на целое как дружественную.

Вариант №28

Создать класс длинных целых чисел. Класс имеет конструктор по умолчанию, конструктор, преобразующий long в объект класса. Определить оператор преобразования объекта типа длинных целых чисел в число типа long. Создать класс целых чисел. Определить взаимное преобразование с классом длинных целых чисел.

3) Перегрузка операторов

Вариант №1

Создать класс целых чисел. Определить оператор ++, как функцию-член и -- как дружественную функцию.

Вариант №2

Создать класс целых чисел. Определить оператор +, как функцию-член и – как дружественную функцию.

Вариант №3

Создать класс целых чисел. Определить оператор --, как функцию-член и ++ как дружественную функцию.

Вариант №4

Создать класс координат. Определить оператор +, как функцию-член и – как дружественную функцию. Сложить и вычесть координаты с друг другом и с числом. Присвоить координаты, сравнить координаты (==, !=).

Вариант №5

Создать класс вещественных чисел. Определить оператор ++, как функцию-член и -- как дружественную функцию.

Вариант №6

Создать класс целых чисел. Определить оператор –, как функцию-член и + как дружественную функцию.

Вариант №7

Создать объект типа очередь. Перегрузить оператор ++ как функцию член и -- как дружественную функцию. (Как постфиксными так префиксными). ++ добавляет элемент в очередь (пустой элемент, например int i=0), -- вытаскивает элемент из очереди. Оператор ! проверяет очередь на пустоту.

Вариант №8

Создать объект типа стек. Перегрузить оператор + как функцию член и * как дружественную функцию. + складывает элемент в новый стек, * умножает верхушку стека на параметр. Стеки можно присваивать, проверять на равенство == или !=, вводить и выводить в поток, добавлять += элемент в стек.

Вариант №9

Создать объект типа стек. Перегрузить оператор ++ как функцию член и -- как дружественную функцию. (Как постфиксными так префиксными). ++ добавляет элемент новый в стек, -- удаляет верхушку стека. Оператор ! проверяет стек на пустоту.

Вариант №10

Создать объект типа очередь. Перегрузить оператор + как функцию член и * как дружественную функцию. + добавляет элемент в очередь, * умножает элемент в очереди. Вытаскивает элемент из очереди --. Очереди можно присваивать, проверять на равенство == или !=, вводить и выводить в поток, добавлять += элемент в очередь.

Вариант №11

Создать объект – связный двунаправленный список, с перегруженными унарными операциями ++, --, как движение по списку. (Как постфиксными так префиксными).

Вариант №12

Создать объект динамический стек. Перегрузить операции +, +=, –= (с извлечением элемента).

Вариант №13

Создать объект стек, перегрузив ++ и --. (Как постфиксными так префиксными). ++ Добавляет элемент в стек. -- извлекает элемент из стека.

Вариант №14

Создать объект очередь с перегруженными +, +=, добавление элемента в очередь и сложение очередей, -- для извлечения из очереди, - для вычитания очередей.

Вариант №15

Создать класс – координаты с унарным ++ и --, -. ++ и -- постфиксная и префиксная. - меняет знак у обеих координат. ++ как функция-член, -- как дружественная функция.

Вариант №16

Создать класс целых чисел (long). Определить оператор -, как функцию-член и + как дружественную функцию. Оператор присвоения, и сравнений.

Вариант №17

Создать класс вещественных чисел (double). Определить оператор ++, как функцию-член и -- как дружественную функцию.

Вариант №18

Создать класс целых чисел (long). Определить оператор +, как функцию-член и - как дружественную функцию.

Вариант №19

Создать класс вещественных чисел (double). Определить оператор --, как функцию-член и ++ как дружественную функцию.

Вариант №20
Создать класс вещественных чисел (double). Определить оператор -, как функцию-член и + как дружественную функцию.

Вариант №21
Создать класс целых чисел (long). Определить оператор ++, как функцию-член и -- как дружественную функцию.

Вариант №22
Создать класс вещественных чисел (double). Определить оператор +, как функцию-член и - как дружественную функцию.

Вариант №23
Создать класс целых чисел (long). Определить оператор --, как функцию-член и ++ как дружественную функцию.

Вариант №24
Создать класс вещественных чисел. Определить оператор -, как функцию-член и + как дружественную функцию.

Вариант №25
Создать объект – очередь с перегруженными операциями ++ как функциями-членами, -- как дружественными функциями. (Как постфиксными так префиксными).

Вариант №26
Создать объект – однонаправленный список, в котором определены операции, + - добавляет в конец списка, += добавляет в этот же список в конец списка. - удаляет указанный элемент из списка (номер элемента через параметр), = - присвоение списков, сравнение списков ==, !=, >, <, >=, <=, [] получение элемента списка, ++ - устанавливает указатель на следующий элемент. () выдать подсписок от первого до второго элемента.

Вариант №27
Создать объект - однонаправленный список, в котором определены операции, ++ - добавляет в конец списка, -- удаляет элемент из списка. (Как постфиксными так префиксными).

Вариант №28

Создать класс тип строка с перегруженным оператором присваивания = и индексирования массива [] для работы со строкой как с массивом.

4) Наследование. Иерархия и контейнерные классы

Вариант №1

Создать класс химический элемент, имеющий наименование (указатель на строку), группу и период (на основании периодической системы элементов). Определить конструкторы, деструктор и функцию печати. Создать public-производный класс – свойство химического элемента. Определить конструкторы по умолчанию и с разным числом параметров, деструкторы, функцию печати. Определить функции переназначения свойства и группы химического элемента.

Вариант №2

Создать класс цех, имеющего площадь. Определить конструктор и метод доступа. Создать класс цехов по производству минеральных удобрений, содержащий площадь, этаж. Определить конструкторы, методы доступа. Определить public-производный класс цехов по производству минеральных удобрений разных предприятий (дополнительный параметр – название предприятия). Определить конструкторы, деструктор и функцию печати.

Вариант №3

Создать класс сорбентов, имеющий классификацию (указатель на строку), диаметр гранул, механическую прочность, удельную поверхность и др. Определить конструкторы, деструктор и функцию печати. Создать public-производный класс – сорбенты для газоочистки. Определить конструкторы по умолчанию и с разным числом параметров, деструкторы, функцию печати. Определить функции переназначения названия марки сорбента и значения диаметра гранул.

Вариант №4

Создать класс брак полимерной пленки (царапина, имеющая длину). Определить конструкторы и метод доступа. Создать класс полимерных пленок, содержащий класс брак. Дополнительно есть цвет (указатель на строку), толщина. Определить конструкторы и деструктор. Определить public- производный класс полимерная пленка, имеющая дополнительно марку (указатель на строку). Определить конструкторы, деструкторы и функцию печати.

Вариант №5

Создать класс технологическая печь, имеющая марку (указатель на строку), максимальную рабочую температуру, мощность. Определить конструкторы, деструктор и функцию печати. Создать public-производный класс – трубчатая печь, имеющая время нагрева. Определить конструкторы по умолчанию и с разным числом параметров, деструкторы, функцию печати. Определить функции переназначения марки и времени нагрева печи.

Вариант №6

Создать класс двигатель, имеющий мощность. Определить конструкторы и метод доступа. Создать класс технологического оборудования, содержащего класс двигатель. Дополнительно есть марка (указатель на строку), цена. Определить конструкторы и деструктор. Определить public- производный класс химический насос, имеющий дополнительно КПД. Определить конструкторы, деструкторы и функцию печати.

Вариант №7

Создать иерархию классов химических реакторов. Переопределить вывод в поток и ввод из потока, конструктор копирования, оператор присваивания через соответствующие функции базового класса.

Вариант №8

Создать иерархию классов химическая реакция и экзотермическая реакция. Переопределить вывод в поток и ввод из потока, конструктор копирования, оператор присваивания через соответствующие функции базового класса.

Вариант №9

Создать класс химических реакций и производный от него – класс химических реакций замещения. Определить конструкторы и деструкторы, переопределить вывод и ввод в поток. Перегрузить оператор присваивания и конструктор копирования в базовом и производном классе.

Вариант №10

Создайте класс точка, которая имеет координаты. Класс эллипсов, и класс окружностей. Определить иерархию типов. Определить функции печати, конструкторы, деструкторы, вычисление площади.

Вариант №11

Используя иерархию и композицию классов, создать бинарное дерево. У бинарного дерева есть корневой узел. Мы можем вставлять узел. Мы можем обходить в ширину и обратный обход. Узел может быть помещен в дерево двоичного поиска только в качестве концевого узла. Если дерево является пустым, то создается новый экземпляр класса узел дерева и узел помещается в дерево. Если дерево не является пустым, то программа сравнивает вставляемое в дерево значение со значением в корневом узле и если меньше, то помещает в левые поддеревья, а если больше, то в правые. Если значения равны, то выводится сообщение, что повтор и не вставляется.

Вариант №12

Создать класс химических реакций соединения, разложения, обмена и замещени. Создать из них иерархию. Определить функции печати, конструкторы и деструкторы, вычисление количества реагентов и продуктов химической реакции.

Вариант №13

Создать иерархию классов вектор и безопасный вектор с проверкой выхода за пределы. Безопасный вектор определяет переменные нижний и верхний предел. Переопределить вывод в поток и ввод из потока, конструктор копирования, оператор присваивания через соответствующие функции базового класса.

Вариант №14

Создать класс оборудование для гранулирования, одношнековый экструдер, планетарный гранулятор, тарельчатый гранулятор, шнековый пресс. Создать из них иерархию. Определить функции печати, конструкторы и деструкторы.

Вариант №15

Создать классы сырьё, полупродукт и продукция. Составить из них иерархию или композицию.

Вариант №16

Создать иерархию классов исходные материалы и химическая продукция. Переопределить вывод в поток и ввод из потока, конструктор копирования, оператор присваивания через соответствующие функции базового класса.

Вариант №17

Создать класс жидкость, имеющий название (указатель на строку), плотность. Определить конструкторы, деструктор и функцию печати. Создать public-производный класс – жидкость, имеющая вредные вещества. Определить кон-

структоры по умолчанию и с разным числом параметров, деструкторы, функцию печати. Определить функции переназначения плотности и вредных веществ.

Вариант №18

Создать иерархию классов химическое вещество и опасное химическое вещество. Переопределить вывод в поток и ввод из потока, конструктор копирования, оператор присваивания через соответствующие функции базового класса.

Вариант №19

Создать класс технологическое оборудование, имеющий имя (указатель на строку), производительность, вес. Определить конструкторы, деструктор и функцию печати. Создать public-производный класс – высокотемпературное технологическое оборудование (напр., промышленные высокотемпературные термошкафы), имеющий объем рабочей камеры. Определить конструкторы по умолчанию и с разным числом параметров, деструкторы, функцию печати. Определить функции переназначения производительности и объема рабочей камеры.

Вариант №20

Создать класс жесткий диск, имеющий объем (Мбайт). Определить конструкторы и метод доступа. Создать класс компьютер, содержащий класс жесткий диск. Дополнительно есть марка (указатель на строку), цена. Определить конструкторы и деструктор. Определить private-, public- производный класс компьютеров с монитором, имеющий дополнительно размер монитора. Определит конструкторы, деструкторы и функцию печати.

Вариант №21

Создать иерархию классов четырехугольник и квадрат. Переопределить вывод в поток и ввод из потока, конструктор копирования, оператор присваивания через соответствующие функции базового класса.

Вариант №22

Создать класс окно, имеющий координаты верхнего левого и нижнего правого угла, цвет фона (указатель на строку). Определить конструкторы, деструктор и функцию печати. Создать public-производный класс – окно с меню, имеющий строку меню. Определить конструкторы по умолчанию и с разным числом параметров, деструкторы, функцию печати. Определить функции переназначения цвета фона и строки меню.

Вариант №23

Создать класс процессор, имеющий мощность (МГц). Определить конструкторы и метод доступа. Создать класс компьютер, содержащий класс процессор. Дополнительно есть марка (указатель на строку), цена. Определить конструкторы и деструктор. Определить private-, public- производный класс компьютеров с монитором, имеющий дополнительно размер монитора. Определить конструкторы, деструкторы и функцию печати.

Вариант №24

Создать класс точка, имеющая координаты. Определить конструкторы, деструктор и функцию печати. Создать public-производный класс – цветная точка, имеющий цвет точки. Определить конструкторы по умолчанию и с разным числом параметров, деструкторы, функцию печати. Определить функции переназначения цвета и координат точки, вывода точки на экран.

Вариант №25

Создать класс термошкаф, имеющий объем рабочей камеры. Определить конструкторы и метод доступа. Создать класс промышленного высокотмпературного оборудования, содержащий класс термошкаф. Дополнительно есть марка (указатель на строку), цена. Определить конструкторы и деструктор. Определить public- производный класс промышленный высокотемпературный термошкаф для драгоценных металлов, имеющий дополнительно мощность. Определить конструкторы, деструкторы и функцию печати.

Вариант №26

Создать иерархию классов окно и окно с заголовком. Переопределить вывод в поток и ввод из потока, конструктор копирования, оператор присваивания через соответствующие функции базового класса.

Вариант №27

Создать класс точка и производные от него – окружность и эллипс. Определить конструкторы, деструктор и функцию печати. Определить функции переустановки центра окружности и эллипса.

Вариант №28

Создать иерархию классов химическое вещество с дочерними классами: тип вещества (неорганическое, органическое) и число элементов. Создать класс кислоты, который является наследником классов тип вещества и число элементов. В каждом классе создать метод отображения особенностей кислот. Продемонстрировать виртуальное наследование.

5) Виртуальные функции

Вариант №1

Создать абстрактный базовый класс с виртуальной функцией – площадь. Создать производные классы: прямоугольник, круг, прямоугольный треугольник, трапеция со своими функциями площади. Для проверки определить массив ссылок на абстрактный класс, которым присваиваются адреса различных объектов. Площадь трапеции: $S=(a+b)\cdot h/2$.

Вариант №2

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №3

Создать абстрактный класс с виртуальной функцией: норма. Создать производные классы: комплексные числа, вектор из 10 элементов, матрица (2x2). Определить функцию нормы – для комплексных чисел – модуль в квадрате, для вектора – корень квадратный из суммы элементов по модулю, для матрицы – максимальное значение по модулю.

Вариант №4

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №5

Создать абстрактный класс (кривые) вычисления координаты y для некоторой x. Создать производные классы: прямая, эллипс, гипербола со своими функциями вычисления y в зависимости от входного параметра x.

Вариант №6

Уравнение прямой: $y=ax+b$, эллипса: $x^2/a^2+y^2/b^2=1$, гиперболы: $x^2/a^2-y^2/b^2=1$

Вариант №7

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомога-

тельные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №8

Создать абстрактный базовый класс с виртуальной функцией – сумма прогрессии. Создать производные классы: арифметическая прогрессия и геометрическая прогрессия. Каждый класс имеет два поля типа double. Первое – первый член прогрессии, второе (double) – постоянная разность (для арифметической) и постоянное отношение (для геометрической). Определить функцию вычисления суммы, где параметром является количество элементов прогрессии.

Арифметическая прогрессия $a_j=a_0+jd$, $j=0,1,2,\ldots$

Сумма арифметической прогрессии: $s_n=(n+1)(a_0+a_n)/2$

Геометрическая прогрессия: $a_j=a_0r^j$, $j=0,1,2,\ldots$

Сумма геометрической прогрессии: $s_n=(a_0-a_nr)/(1-r)$

Вариант №9

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №10

Создать базовый класс список. Реализовать на базе списка стек и очередь с виртуальными функциями вставки и вытаскивания.

Вариант №11

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №12

Создать базовый класс – фигура, и производные класс – круг, прямоугольник, трапеция. Определить виртуальные функции площадь, периметр и вывод на печать.

Вариант №13

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомога-

тельные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №14

Создать базовый класс – работник и производные классы – служащий с почасовой оплатой, служащий в штате и служащий с процентной ставкой. Определить функцию начисления зарплаты.

Вариант №15

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №16

Создать абстрактный базовый класс с виртуальной функцией – площадь поверхности. Создать производные классы: параллелепипед, тетраэдр, шар со своими функциями площади поверхности. Для проверки определить массив ссылок на абстрактный класс, которым присваиваются адреса различных объектов.

Площадь поверхности параллелепипеда: $S=6xy$. Площадь поверхности шара: $S=4\pi r^2$. Площадь поверхности тетраэдра: $S=\sqrt{3}a^2$.

Вариант №17

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №18

Создать класс вещество, производные от которого органические вещества и неорганические. Определить виртуальную функцию на результат взаимодействия с водой.

Вариант №19

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №20

Создать абстрактный базовый класс с виртуальной функцией – объем. Создать производные классы: параллелепипед, пирамида, тетраэдр, шар со своими функциями объема. Для проверки определить массив ссылок на абстрактный класс, которым присваиваются адреса различных объектов.

Объем параллелепипеда $V=xyz$, пирамиды $V=xyh$, тетраэдра $V = \sqrt{2}/12 \cdot a^3$, шара $V=4/3 \cdot \pi \cdot r^3$.

Вариант №21

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №22

Создать абстрактный класс – агрегатное состояние вещества. Определить производные классы – твердое тело, жидкость, газообразное состояние, плазма. У твердого тела определить производные классы аморфное и кристаллическое, у жидкости определить производные классы атомарная, ассоциированная и т.д. Определить виртуальные функции описания твердого тела, жидкости, газообразного состояния, плазмы.

Вариант №23

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №24

Создать базовый класс – химическая реакция определить виртуальную функцию печати. Создать производный класс классификация (по фазовому составу реагирующей системы; по изменению степеней окисления реагентов; по тепловому эффекту реакции; по типу превращений реагирующих частиц). Создать производный класс от последнего класса:

по фазовому составу реагирующей системы (гомогенные гомофазные реакции, гетерогенные гетерофазные реакции, гетерогенные гомофазные реакции, гомогенные гетерофазные реакции);

по изменению степеней окисления реагентов (окислительно-восстановительные реакции, не окислительно-восстановительные реакции);

по тепловому эффекту реакции (эндотермические, экзотермические);

по типу превращений реагирующих частиц (соединения, разложения, замещения, обмена). Написать функцию печати названия класса.

Вариант №25

Создать класс – данные – абстрактный базовый класс. Создать производные классы – данные типа сигнал, данные типа результат обработки и вспомогательные данные. Все данные имеют функции отображения, сохранения и обработки.

Вариант №26

Создать абстрактный базовый класс с виртуальной функцией – корни уравнения. Создать производные классы: класс линейных уравнений и класс квадратных уравнений. Определить функцию вычисления корней уравнений.

Вариант №27

Создать абстрактный базовый класс с виртуальной функцией – процессы химической технологии. Создать производные классы: гидромеханические; тепловые; массообменные (или диффузионные) процессы; химические процессы; механические процессы. Написать функцию печати.

Вариант №28

Создать класс – связанный список – абстрактный базовый класс. Интерфейс списка определяется с помощью чистых виртуальных функций store() и retrieve(). Для хранения значения в списке вызывается функция store(). Для выборки значения из списка вызывается функция retrieve(). В базовом классе list для выполнения этих действий никакого встроенного метода не задается. Необходимо реализовать списки двух типов: очередь и стек.

6) Применение основ программирования на языке высокого уровня в области химической технологии

Создать класс «Скорость химической реакции».

Создать процедуру ввода параметров химической реакции (таблица 15):

- стехиометрические коэффициенты α_i;
- число исходных реагентов m;
- константа скорости химической реакции k;
- концентрации исходных реагентов во времени $C_i(t)$.

Создать функцию расчёта скорости химической реакции.

Для вывода уравнения скорости химической реакции используется закон действия масс (основной постулат химической кинетики).

Если в реагирующей системе протекает реакция в соответствии со следующим стехиометрическим уравнением:

$\alpha_1 A_1 + \alpha_2 A_2 + \ldots + \alpha_m A_m \xrightarrow{k} \alpha_{m+1} A_{m+1} + \alpha_{m+2} A_{m+2} + \ldots + \alpha_n A_n$,

то в соответствии с законом действия масс скорость элементарной химической реакции пропорциональна **концентрациям реагирующих веществ** в степенях, равных их стехиометрическим коэффициентам:

$$w = k \prod_{i=1}^{m} C_i^{|\alpha_i|},$$

где k – константа скорости реакции (размерность зависит от порядка реакции); m – число реагентов в реакции; C_i – концентрация i-го компонента, моль/м^3; α_i – стехиометрический коэффициент i-го компонента.

Создать функцию вывода результатов расчёта скорости реакции в каждый момент времени.

Примеры химических реакций для выполнения задания по вариантам приведены в талице 15.

Таблица 15 – Примеры химических реакций для выполнения задания

Номер варианта	Химическая реакция	Номер варианта	Химическая реакция
1	$NaOH + HCl \rightarrow NaCl + H_2O$	15	$Zn + 2HCl = H_2 + ZnCl_2$
2	$CO_3 + 2HCl \rightarrow Cl_2 + CO_2 + H_2O$	16	$2NaBr + Cl_2 = 2NaCl + Br_2$
3	$NH_3 + HCl \rightarrow NH_4Cl$	17	$Zn + 2HCl = H_2 + ZnCl_2$
4	$N_2 + 3H_2 \rightarrow 2\,NH_3$	18	$CH_4 + 2O_2 \rightarrow CO_2 + 2H_2O$
5	$2H_2 + O_2 = 2H_2O$	19	$H_2 + CuO \rightarrow Cu + H_2O$
6	$NH_4NO_3 = N_2O + 2H_2O$	20	$2NaBr + Cl_2 \rightarrow 2NaCl + Br_2$
7	$2Cu + O_2 \rightarrow 2CuO$	21	$Ca_3(PO_4)_2 + 3SiO_2 = 3CaSiO_3 + P_2O_5$
8	$2HgO \rightarrow 2Hg+O_2$	22	$2AgCl + SnCl_2 = 2Ag + SnCl_4$
9	$Fe + CuSO_4 \rightarrow FeSO_4 + Cu$	23	$Fe + 5CO = [Fe(CO)_5]$
10	$H_2S + 2NaOH \rightarrow Na_2S + 2H_2O$	24	$2KClO_3 \rightarrow 2KCl + 3O_2$
11	$CaO + H_2O = Ca(OH)_2$	25	$C_6H_{12}O_6 + 6O_2 \rightarrow 6CO_2 + 6H_2O$
12	$H_2 + CuO = Cu + H_2O$	26	$FeS_2 + 8HNO_3 = Fe(NO_3)_3 + 5NO + 2H_2SO_4 + 2H_2O$
13	$CaBr_2 + 2HF = CaF_2 + 2HBr$	27	$2AgNO_3 = 2Ag + 2NO_2 + O_2$
14	$CaCl_2 + Na_2CO_3 = CaCO_3 + 2NaCl$	28	$2A + 4B + C = D + 2E$

11.3.2 Пример выполнения контрольной работы № 3

Для написания программ использовалась IDE Visual Studio, программы представляют консольные приложения.

Вариант №28

1) Создайте класс, который реализует безопасный двухмерный массив целых чисел размерностью (2x3). Класс должен содержать методы для произвольного защищенного доступа к элементам массива, метод для записи элементов массива через возвращаемое значение ссылки.

```
#include <iostream>
#include <cstdlib>
using namespace std;
class array
{
      int isize, jsize;
      int *p;
public:
      array(int i, int j);
      int&put(int i, int j);
      int get(int i, int j);
};
array::array(int i, int j)
{
      p = new int[i * j];
      isize = i;
      jsize = j;
}
int & array::put(int i, int j)
{
      if (i < 0 || i >= isize || j < 0 || j >= jsize)
      {
            cerr << "Ошибка, нарушены границы массива!!!!" << endl;
            exit(1);
      }
      return p[i* jsize + j];
}
int array::get(int i, int j)
{
      if (i < 0 || i >= isize || j < 0 || j >= jsize)
      {
            cerr << "Ошибка, нарушены границы массива!!!!" << endl;
            exit(1);
      }
      return p[i* jsize + j];
}
int _tmain(int argc, _TCHAR* argv[])
{
      array a(2, 3);
      int i, j;
      for(i = 0; i < 2; i++)
            for(j = 0; j < 3; j++)
                  a.put(i, j) = i + j;
      for(i = 0; i < 2; i++)
            for(j = 0; j < 3; j++)
```

```
                cout << a.get(i, j) << ' ';
      a.put(10, 10);
      return 0;
}
```

Результат работы: 0 1 2 1 2 3
Ошибка, нарушены границы массива!!!!

2) Создать класс длинных целых чисел. Класс имеет конструктор по умолчанию, конструктор, преобразующий long в объект класса. Определить оператор преобразования объекта типа длинных целых чисел в число типа long. Создать класс целых чисел. Определить взаимное преобразование с классом длинных целых чисел.

```
class Long
{
      long value;
public:
      Long(): value(0){}
      Long(long l):value(l) {}
      operator long() {return value;}
      operator int() {return value;}
};
class Int
{
      int value;
public:
      Int(): value(0){}
      Int(int i):value(i) {}
      Int(Long l)
      {
            value = static_cast<int>(l);
      }
      operator int() {return value;}
      operator Long() {return Long(value);}
};
int _tmain(int argc, _TCHAR* argv[])
{
      Long l1(100L), l2(50L);
      Int i1(10);
      long l = 1L + static_cast<long>(l2);
      Long ll = i1;
      return 0;
}
```

3) Создать класс тип строка с перегруженным оператором присваивания = и индексирования массива [] для работы со строкой как с массивом.

```
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;
class strtype
{
      char *p;
      int len;
public:
```

```
      strtype(char*s);
      ~strtype(){delete p;}
      char *get() {return p;}
      strtype &operator=(strtype &ob);
      char operator[](int i);
};
strtype::strtype(char*s)
{
      int l = strlen(s) + 1;
      p = new char[l];
      len = l;
      strcpy(p, s);
}
strtype &strtype::operator=(strtype &ob)
{
      if (len < ob.len)
      {
            delete[]p;
            p = new char[ob.len];
      }
      len = ob.len;
      strcpy(p, ob.p);
      return *this;
}
char strtype::operator[](int i)
{
      if (i > len)
      {
            cerr << "Ошибка, индекс не может быть больше количества символов в
строке";
            exit(1);
      }
      return p[i];
}
int _tmain(int argc, _TCHAR* argv[])
{
      strtype a("Hello!"), b("Hi!");
      cout << a.get() << endl;
      cout << b.get() << endl;
      for(int i = 0; i < strlen(a.get()); i++)
            cout << a[i] << "  ";
      cout << endl;
      a = b;
      cout << a.get() << endl;
      cout << b.get() << endl;
      return 0;
}
```

Результат работы:

Hello!

Hi!

H e l l o !

Hi!

Hi!

4) Создать иерархию классов химическое вещество с дочерними классами: тип вещества (неорганическое, органическое) и число элементов. Создать класс кислоты, который является наследником классов тип вещества и число элементов.

В каждом классе создать метод отображения особенностей кислот. Продемонстрировать виртуальное наследование.

```
#include <iostream>
using namespace std;
class acid
{
      int num_h;
      int conc;
public:
      acid(int w, int r): num_h(w), conc(r){}
      void showv()
      {
            cout << "Число атомов водорода " << num_h << endl;
            cout << "Концентрация % " << conc << endl;
      }
};
enum matter{organic, inanimate};
class chemcombination : public virtual acid
{
      enum matter mtr;
public:
      chemcombination(enum matter m, int w, int r): acid(w, r), mtr(m){}
      void showm()
      {
            cout << "Класс химического соединения:";
            switch (mtr)
            {
            case organic: cout << "Органические вещества" << endl;
                  break;
            case inanimate: cout << "Неорганические вещества" << endl;
                  break;
            }
      }
};
class r_use: public virtual acid
{
      int component;
public:
      r_use(int p, int w, int r): acid(w, r), component(p) {}
      void showr()
      {
            cout << "Число элементов " << component << endl;
      }
};
enum solubility {solvend, insoluble};
class substance: public chemcombination, public r_use
{
      enum solubility strng;
public:
      substance(enum solubility s, enum matter m, int w, int r, int p):
        chemcombination(m, w, r), r_use(p, w, r), acid(w, r), strng(s) {}
        void show ()
        {
              showv(); showr(); showm();
              cout << "Растворимость: ";
              switch (strng)
              {
              case solvend: cout << "Растворимые" << endl;
                    break;
              case insoluble: cout << "Нерастворимые" << endl;
                    break;
```

```
            }
        }
};
int _tmain(int argc, _TCHAR* argv[])
{
      substance c(solvend, inanimate, 2, 80, 3);
      c.show();
      return 0;
}
```

Результат работы:
Число атомов водорода: 2
Концентрация %: 80
Число элементов: 3
Класс химического соединения: Неорганические вещества
Растворимость: Растворимые

5) Создать класс – связанный список – абстрактный базовый класс. Интерфейс списка определяется с помощью чистых виртуальных функций store() и retrieve(). Для хранения значения в списке вызывается функция store(). Для выборки значения из списка вызывается функция retrieve(). В базовом классе list для выполнения этих действий никакого встроенного метода не задается. Необходимо реализовать списки двух типов: очередь и стек.

```
#include <iostream>
#include <cstdlib>
#include <cctype>
using namespace std;
class list
{
public:
      list *head;
      list *tail;
      list *next;
      int num;
      list () { head = tail = next = NULL; }
      virtual void store (int i) = 0;
      virtual int retrieve ( ) = 0;
};
class queue: public list
{
public:
      void store(int i);
      int retrieve();
};
void queue::store (int i)
{
      list *item = new queue;
      item->num = i;
      if (tail) tail->next = item;
      tail = item;
      item->next = NULL;
      if (!head) head = tail;
}
int queue::retrieve()
{
```

```
	int i;
	list *p;
	if(!head)
	{
		cout << "Список пуст" << endl;
		return 0;
	}
	i = head->num;
	p = head;
	head = head->next;
	delete p;
	return i;
}
class stack: public list {
public:
	void store(int i);
	int retrieve();
};
void stack::store(int i)
{
	list *item = new stack;
	item->num = i;
	if(head) item->next = head;
	head = item;
	if(!tail) tail = head;
}
int stack::retrieve()
{
	int i;
	list *p;
	if(!head)
	{
		cout << "Список пуст" << endl;
		return 0;
	}
	i = head->num;
	p = head;
	head = head->next;
	delete p;
	return i;
}
int _tmain(int argc, _TCHAR* argv[])
{
	list *p;
	queue q_ob;
	p = &q_ob;
	p->store(1);
	p->store(2);
	p->store(3);
	cout << "Очередь: ";
	cout << p->retrieve();
	cout << p->retrieve();
	cout << p->retrieve();
	cout << endl;
	stack s_ob;
	p = &s_ob;
	p->store(1);
	p->store(2);
	p->store(3);
	cout << "Стек: ";
	cout << p->retrieve();
	cout << p->retrieve();
	cout << p->retrieve();
	cout << endl;
```

```
	return 0;
}
```

Результат работы:

Очередь: 123

Стек: 321

6) Расчет скоростей химической реакции 2A + 4B + C = D + 2E на заданных временных интервалах.

```
#include <iostream>
#include <math.h>
using namespace std;
class ChemSpeed
{
	double k;
	double *a, *result;
	double **c;
	double time;
	int m;
	int t;
public:
	ChemSpeed(): a(NULL), result(NULL), c(NULL){}
	void inputData()
	{
		int i, j;
		cout << "Введите количество реагентов смеси m = ";
		cin >> m;
		cout << "Введите число замеров концентраций t = ";
		cin >> t;
		cout << "Введите шаг по времени time = ";
		cin >> time;
		cout << "Введите константу скорости реакции k = ";
		cin >> k;
		a = new double[m];
		c = new double*[m];
		for(i = 0; i < m; i++)
			c[i] = new double[t];
		cout << "Введите стехиометрические коэффициенты: " << endl;
		for(i = 0; i < m; i++)
		{
			cout << "a" << i + 1 << " = ";
			cin >> a[i];
		}
		for(i = 0; i < m; i++)
		{
cout << "Введите концентрации компонента №" << i + 1 << endl;
			for(j = 0; j < t; j++)
			{
				cout << "c[" << j << "] = ";
				cin >> c[i][j];
			}
		}
	}
	void calculate()
	{
		double temp;
		result = new double[t];
		for(int j = 0; j < t; j++)
		{
			temp = k;
```

```
                for(int i = 0; i < m; i++)
                        temp *= pow(c[i][j], fabs(a[i]));
                result[j] = temp;
        }
    }
    void printResult()
    {
        cout << "   time   |     w    " << endl;
        cout << "----------|----------" << left << endl;
        for(int i = 0; i < t; i++)
        {
            cout.width(10);
            cout.precision(5);
            cout << time * (i + 1) << "|" << result[i] << endl;
        }
    }
    ~ChemSpeed()
    {
        if(a) delete[]a;
        if(result) delete[]result;
        if(c)
        {
            for(int i = 0; i < m; i++)
                delete[]c[i];
            delete[]c;
        }
    }
};
int _tmain(int argc, _TCHAR* argv[])
{
    ChemSpeed speed;
    speed.inputData();
    speed.calculate();
    speed.printResult();
    return 0;
}
```

Входные данные: m = 3, t = 3, time = 5, k =1.5

Значения стехиометрических коэффициентов для химической реакции 2A + 4B + C = D + 2E приведены в таблице 16.

Таблица 16 – Значения стехиометрических коэффициентов

a1	a2	a3
2	4	1

Значения концентраций исходных компонентов для химической реакции 2A + 4B + C = D + 2E приведены в таблице 17.

Таблица 17 – Значения концентраций исходных компонентов.

Время	Концентрация A	Концентрация B	Концентрация C
5	0.9	0.75	0.55
10	0.8	0.62	0.50
15	0.5	0.20	0.36

Результат расчета скорости химической реакции на временных интервалах приведен в таблице 18.

Таблица 18 – Результат расчета скорости химической реакции на временных интервалах

Время	Скорость реакции
5	0.2114
10	0.0709
15	0.0002

11.4 КОНТРОЛЬНАЯ РАБОТА №4. Разработка программного комплекса для определения константы скорости химической реакции

Перечень подлежащих разработке вопросов:

1. Ознакомиться с теорией по синтезу и анализу математических моделей (ММ) кинетики химических реакций.
2. Выполнить постановку задачи по исследованию кинетики химической реакции.
3. Составить формализованное описание задачи. Разработать структуру входных (экспериментальных данных по изменению концентрации компонентов во времени) и выходных данных (порядок реакции, константа скорости реакции).
4. Уточнить методы решения математической задачи.
5. Разработать алгоритм (блок-схему) для определения порядка химической реакции и константы скорости химической реакции.
6. Спроектировать структуру программы.
7. Разработать пользовательский интерфейс.
8. Выполнить предварительную оценку кинетических констант с использованием метода регрессионного анализа.
9. Провести статистический анализ результатов.
10. Составить программу, реализующую поставленную задачу.
11. Протестировать работоспособность программного обеспечения.

Рекомендуемое программное средство для выполнения контрольной работы: среда объектно-ориентированного программирования C++Builder.

Вид химической реакции приведен в талице 19 и соответствует номеру варианта студента. Номер варианта соответствует номеру первой буквы фамилии студента.

11.4.1 Задания контрольной работы №4

Таблица 19 – Варианты заданий

Номер варианта	Химическая реакция
1	A = B + C + D
2	A = 2B + C + D + E
3	2A = 4B + C
4	A = 3B + 2C
5	2A = B + 2C
6	A = B + 2C
7	2A = 3B + 2C
8	2A = 2B + C
9	A = B + C + D + E
10	3A = 2B + C + 5D
11	4A = 4B + 2C + D
12	A = 4B + C + 2D + 2E
13	3A = 8B + 3C + 2D + 4E
14	3A = B + C + D + 2E
15	6A = 2B + C
16	2A = 5B + 2C + 4D + E
17	4A = 6B + 2C + 2D
18	3A = 8B + 5C + 18D
19	A = 10B + C
20	3A = 4B + C
21	2A = 9B + C + 7D
22	5A = 3B + 3C + 2D
23	3A = B + C + D
24	A = 2B + 2C + D
25	7A = 2B + 4C + D
26	2A = 9B + C + 7D
27	A = B + 3C + 3D
28	2A = 4B + C + D

11.4.2 МАТЕМАТИЧЕСКОЕ ОПИСАНИЕ ХИМИЧЕСКИХ РЕАКЦИЙ КАК ОБЪЕКТА ПРОГРАММИРОВАНИЯ

Изменение содержания вещества в химических реакторах происходит в результате его переноса за счет движения реакционной массы и химического реакций. Математическое описание химических реакций складывается из описания: стехиометрии; кинетики; порядка и молекулярности реакций.

Описание стехиометрии химических реакций и молекулярного состава веществ

Стехиометрическое уравнение химической реакции представляет собой краткое выражение материального баланса химических реакций.

В общем виде стехиометрическое уравнение простой химической реакции может быть записано в следующем виде:

$$\sum_{j=1}^{m} \alpha_j A_j \,. \qquad (1)$$

Этот способ записи стехиометрии можно распространить на сложные химические процессы с несколькими одновременно протекающими реакциями:

$$\sum_{j=1}^{m} \alpha_{ij} A_j = 0, \quad i = 1, 2, ..., r \,, \qquad (2)$$

где i – номер реакции, i = l, 2, ..., r; α_{ij} – стехиометрический коэффициент j-го компонента в i-й реакции; $\alpha_{ij} > 0$ – при образовании компонента; $\alpha_{ij} < 0$ – при расходовании компонента; r – число протекающих реакций; A_j – вектор-столбец реагирующих компонентов; m_i – число реагентов в i-той реакции.

Система уравнений химических реакций в ЭВМ может быть получена автоматически, путём умножения вектор-столбца символов веществ A_j на матрицу стехиометрических коэффициентов.

Задание механизма химических реакций с помощью матрицы стехиометрических коэффициентов обеспечивает возможность, при необходимости, гибкого её изменения.

Молекулярный состав веществ накладывает жёсткие ограничения на стехиометрические коэффициенты каждой химической реакции. Ограничения обусловлены тем, что должно сохраняться постоянным число атомов каждого элемента, участвующего в стадии, а в случае реакции с участием ионов должен сохраняться суммарный заряд.

Для отражения химического состава веществ используется молекулярная матрица |β|, элемент β_{jk} которой представляет число атомов *k*-го элемента, содержащихся в молекуле *j*-го реагента.

Размерность матрицы |β| - (*n*x*l*), где *n* – число реагентов в системе; *l* – число химических элементов.

Условие сохранения числа атомов (или единиц заряда) может быть записано следующим образом:

$$\sum_{j=1}^{n}\beta_{jk}\alpha_{ij}=0,\quad \begin{cases} i=1,2,...,r; \\ j=1,2,...,n; \\ k=1,2,...,l. \end{cases} \tag{3}$$

Размерность системы уравнений (3) определяется произведением числа химических элементов, входящих в состав молекул реагентов *l*.

Описание кинетики

Основным понятием химической кинетики является скорость химической реакции w_r, которая определяется как число молекул (или грамм-молекул), реагирующих в единицу времени, в единице объёма или на единице поверхности.

Скорость в единице объёма

$$w_r=-\frac{1}{V}\frac{dN_j}{dt}, \tag{4}$$

где w_r – скорость химической реакции, моль/(м^3·с); V – объем реагирующей системы, м^3; N_j – число молей *j*-го компонента (или грамм-моль).

Скорость на единице поверхности

$$w_r=-\frac{1}{S}\frac{dN_j}{dt}, \tag{5}$$

где S – поверхность реагирующей системы, м^2.

Для V=const можно записать скорость химической реакции через концентрации компонента:

$$w_r=-\frac{d(N_j/V)}{dt}=\frac{dC_j}{dt}, \tag{6}$$

где C_j – концентрация *j*-го компонента, моль/м^3.

Для вывода уравнения скорости химической реакции используется закон действия масс (основной постулат химической кинетики).

Если в реагирующей системе протекает реакция в соответствии со следующим стехиометрическим уравнением:

$$\alpha_1 A_1 + \alpha_2 A_2 + \ldots + \alpha_m A_m \xrightarrow{k} \alpha_{m+1} A_{m+1} + \alpha_{m+2} A_{m+2} + \ldots + \alpha_n A_n,$$

то в соответствии с законом действия масс скорость элементарной химической реакции пропорциональна концентрациям реагирующих веществ в степенях, равных их стехиометрическим коэффициентам, то есть в уравнении скорости реакции сомножителями являются вещества A_j, у которых стехиометрические коэффициенты отрицательны:

$$w_{ri} = k_i \prod_{j=1}^{m_i} C_{A_j}^{|\alpha_{ij}|}, \quad i = 1, 2, ..., r, \text{ для } \alpha_{ij} < 0, \tag{7}$$

где r – число протекающих реакций.

Если при моделировании химических реакций не удаётся получить уравнение элементарной химической реакции и приходится использовать суммарные стехиометрические уравнения нескольких элементарных реакций (формальные), то показатели степени в кинетическом уравнении определяются экспериментально, а не из стехиометрического уравнения:

$$w_{ri} = k_i \prod_{j=1}^{m_i} C_{A_j}^{p_{ij}}, \tag{8}$$

где p_{ij} – экспериментально определяемые показатели степени для j-го компонента в i-й реакции.

Порядок и молекулярность химической реакции

Для математического описания порядка вводятся два понятия – порядок реакции и частный порядок по компоненту.

Порядок реакции – это сумма показателей степеней кинетического уравнения скорости химической реакции.

Если скорость химической реакции определяется по уравнению

$$w_r = k_i \prod_{j=1}^{m_i} C_{A_j}^{|\alpha_{ij}|},$$

то порядок реакции

$$n = \sum_{j=1}^{m} |\alpha_j|. \tag{9}$$

Для нескольких одновременно протекающих реакций определяется порядок для каждой реакции:

$$n_i = \sum_{j=1}^{m_i} | \alpha_{ij} |, \; i = 1, 2, ..., r. \qquad (10)$$

Следует отметить, что порядок реакции при практическом моделировании определяется часто экспериментально, так как не всегда удаётся описать элементарные стадии химических реакций, для которых справедлив закон действия масс. Отсюда не удаётся вычислить порядок непосредственно из стехиометрического уравнения, но в любом случае порядок определяется как сумма показателей степеней кинетического уравнения:

$$w_{ri} = k_i \prod_{j=1}^{m_i} C_{A_j}^{p_{ij}} ,$$

$$n_i = \sum_{j=1}^{m_i} p_{ij} . \qquad (11)$$

Частный порядок p_{ij} определяется по отдельному исходному реагенту и равен показателю степени этого реагента в уравнении кинетики. При соблюдении закона действия масс: $p_{ij} = \alpha_{ij}$.

Порядок реакции определяет размерность константы скорости реакции:

$$w_{ri} = k_i \prod_{j=1}^{m_i} C_{A_j}^{p_{ij}} ,$$

$$[\text{моль/м}^3\cdot\text{с}]=[?] \prod_{j=1}^{m_i} [\text{моль/м}^3]^{\,p_{ij}} .$$

Например,

для n=1, [моль/м^3·с]=[?] [моль/м^3] ,

[k]=[1/с];

для n=2, [моль/м^3·с]=[?] [(моль/м^3)^2] ,

[k]=[м^3/моль·с]

Молекулярность химической реакции определяется как число молекул одновременно вступающих в элементарную стадию химических превращений. Для одной элементарной стадии молекулярность и порядок совпадают.

Имеются понятия мономолекулярной реакции, когда в реакцию вступает одна молекула какого-либо вещества; бимолекулярной – две молекулы; тримолекулярной – три молекулы. Реакции с молекуляриостыо больше трёх практически не встречаются, так как вероятность столкновения более трех молекул ничтожна. Молекулярность – только целое число.

Описание скоростей изменения концентраций компонентов

При описании скоростей сложных реакций с многокомпонентными смесями наряду с законом действия масс используется принцип независимости реакций, заключающийся в том, что скорость любой реакции не зависит от того, протекают ли в системе ещё какие-либо реакции.

Используя стехиометрические уравнения (2) и принцип независимости реакций, можно записать скорость образования или расходования любого компонента в системе:

$$\frac{dC_j}{dt} = w_j = \sum_{i=1}^{r} w_{r_i} \alpha_{ij} , \qquad (12)$$

или с учётом уравнения (8) скорости химической реакции:

$$w_j = \sum_{i=1}^{r} \alpha_{ij} \prod_{j=1}^{m_i} k_i C_j^{p_{ij}} \ , \ j = 1, 2, ..., n. \qquad (13)$$

где w_j – скорость образования или расходования j-того компонента в реакционной системе, моль/(м^3·с); α_{ij} – стехиометрический коэффициент j-го компонента в i-й реакции; C_j – концентрация j-го компонента, моль/м^3; k_i – константа скорости i-й реакции (размерность зависит от порядка реакции); p_{ij} – частный порядок j-того компонента в i-й реакции; n – число реагентов в системе; t – время.

Описание тепловых функций химической реакции

Обоснование зависимости скорости реакции от температуры было получено Аррениусом. Им была сформулирована теория активных столкновений между молекулами: всякая реакция протекает через промежуточную стадию, связанную с переходом состояния молекул от нормальной (средней) энергии к состоянию с повышенной энергией.

Каждая частица – молекула, радикал, ион – энергетически более или менее устойчивое состояние. Перестройка реагирующих частиц требует разрыва или ослабления отдельных химических связей, на что необходимо затратить энергию. Реагировать могут лишь молекулы с повышенной энергией (активные молекулы). Превращение исходных частиц в продукты реакции связано с преодолением потенциального барьера. Характеризует это сочетание энергия активации E. Энергия активации – то минимальное количество суммарной энергии сталкивающихся молекул, которое необходимо для вступления их в реакцию. В химическом превращении участвуют только такие частицы, энергия которых больше E, доля таких частиц равна $d = e^{-E/RT}$ (закон Больцмана).

Зависимость константы скорости реакции от температуры согласно закону Аррениуса имеет вид:

$$k = k_0 e^{-E/RT}, \tag{14}$$

где k_0 – предэкспоненциальный множитель, размерность его определяется размерностью константы скорости реакции; T – температура, К; R – универсальная газовая постоянная, кДж/(моль·К); E – энергия активации, кДж/моль.

Зависимость скорости реакции от температуры иногда выражают через температурный коэффициент скорости K_C реакции, характеризующий скорости при изменении температуры на 10 К:

$$K_C = \frac{k_{(T_1+10)}}{k_{(T_1)}} = \frac{k_0 \exp\left(-\dfrac{E}{R(T_1+10)}\right)}{k_0 \exp\left(-\dfrac{E}{RT_1}\right)} = \exp\left\{\frac{E}{R}\left(\frac{1}{T_1+10} - \frac{1}{T_1}\right)\right\}. \tag{15}$$

С увеличением энергии активации температурный коэффициент скорости увеличивается, с увеличением температуры – снижается.

В сложной реакционной системе реакции образования целевого продукта, как правило, имеют меньшую энергию активации, чем побочные реакции разложения.

Для выбора конструктивных характеристик реакторов, систем обогрева и охлаждения необходимо иметь численные значения тепловых эффектов химических реакций, т.е. количества энергии, выделяющейся или поглощённой на 1 моль превращенных веществ q, кДж/моль.

Математическое описание тепловых функций химической реакции позволяет оценить влияние температуры на скорость процесса, выход целевых и побочных продуктов; выбрать конструктивные и режимные характеристики процесса, обеспечивающие наилучшие значения оценок эффективности.

Описание кинетики каталитических реакций

Часто при проведении химических реакций используют вещества, которые ускоряют (катализаторы), либо замедляют (ингибиторы) химические превращения. Катализ – это изменение скорости химических реакций или их инициирование в результате воздействия веществ-катализаторов, которые участвуя в процессе, остаются по окончании его химически неизменными.

Сущность ускоряющего действия катализатора состоит в понижении энергии активации химических реакций в результате изменения реакционного

пути с участием катализатора или вследствие осуществления реакции по цепному механизму при инициирующем действии катализатора.

Для измерения ускоряющего действия катализатора рассчитывается активность катализатора.

Активность катализатора определяется как соотношение констант скоростей реакций с катализатором и без катализатора:

$$A_K = \frac{k_K}{k} = \frac{k_0 \exp\left(-\frac{E_K}{RT}\right)}{k_0 \exp\left(-\frac{E}{RT}\right)} = \exp\left(\frac{E - E_K}{RT}\right) = \exp\left(\frac{\Delta E}{RT}\right). \quad (16)$$

где A_K – активность катализатора; E_K, E – энергия активации с катализатором и без него, кДж/моль; ΔЕ – снижение энергии активации.

При математическом описании каталитических реакций следует учитывать избирательность катализатора, т.е. его способность изменять скорость только одной из одновременно протекающих реакций.

На практике в условиях промышленного катализа состав и свойства катализатора изменяются, происходит его дезактивация. Дезактивация катализатора может происходить по следующим причинам:

– отравление катализатора за счёт каталитических ядов (примесей в сырьё);

– блокировка катализатора за счёт физического покрытия поверхности катализатора пылью, мелкими порошками;

– спекание катализатора – необратимый физический процесс, приводящий к уменьшению активной поверхности;

– разбавление катализатора при щелочном или кислотном катализе за счёт выделяющейся в результате реакции воды и т.д.

При моделировании действующих промышленных аппаратов необходимо учитывать дезактивацию катализатора, которая приводит к изменению первоначально найденных кинетических констант (энергии активации и предэкспоненциального множителя). В большинстве случаев достаточно описать зависимость энергии активации во времени за счёт отравления, блокировки, спекания. С течением времени энергия активации увеличивается, с увеличением концентрации катализатора – снижается, например:

$$E_K(t) = a_0 + a_1 t;\ E_K(C_K) = a_0 + a_1 A_K. \quad (17)$$

Увеличение энергии активации приводит к снижению константы и скорости реакции. С течением времени необходима регенерация катализатора для обеспечения требуемой оценки эффективности.

Учёт активности катализатора при моделировании позволяет правильно выбрать время пребывания в аппаратах, цикл регенерации, конструктивные характеристики реакторов.

11.4.3 ПОСТАНОВКА ЗАДАЧИ ОПРЕДЕЛЕНИЯ КОНСТАНТ СКОРОСТЕЙ ХИМИЧЕСКОЙ РЕАКЦИИ

В учебном пособии рассмотрена следующая задача кинетики:

Известно:

экспериментальные данные по изменению концентрации компонентов или скоростей реакций во времени, при различных условиях эксперимента (температуры, активности катализатора):

$$C_{j\text{э}}=f(t) \text{ или } w_{j\text{э}}=f(C_{j\text{э}},\ t),$$
$$C_{j\text{э}}=f(t,T) \text{ или } w_{j\text{э}}=f(C_{j\text{э}},T,\ t),$$
$$C_{j\text{э}}=f(t,\ T,\ A_K) \text{ или } w_{j\text{э}}=f(C_{j\text{э}},\ t,\ T,\ A_K).$$

Требуется определить:

- константы скоростей реакций k_i;
- предэкспоненциальные множители k_{0i} и энергии активации E_i;
- предэкспоненциальные множители k_{0i} и коэффициенты уравнения a_0 и a_1, для $E_j=f(A_K)$.

11.4.4 ПРЕДВАРИТЕЛЬНАЯ ОЦЕНКА КИНЕТИЧЕСКИХ КОНСТАНТ С ИСПОЛЬЗОВАНИЕМ МЕТОДА РЕГРЕССИОННОГО АНАЛИЗА

В настоящее время термин «регрессия» означает статистическую связь между случайными величинами. Задача ставится таким образом: по данной выборке объема n найти уравнение приближенной регрессии и оценить допускаемою при этом ошибку. Эта задача решается методами регрессионного и корреляционного анализа. Уравнение приближенной регрессии существенно зависит от выбираемого метода приближения. В качестве такого метода обычно выбирают метод наименьших квадратов. Пусть задан некоторый класс функций $f(x)$, накладывающих на выборку одинаковое число связей l. Число связей l равно числу неопределенных коэффициентов, входящих в аналитическое выражение этой функции. Чаще всего используют многочлены различной степени.

Наилучшее уравнение приближенной регрессии дает та функция из рассматриваемого класса, для которой сумма квадратов имеет наименьшее значение.

$$\Phi = \sum_{i=1}^{n} \left[y_i - f(x_i) \right]^2 .$$

При нормальном распределении случайных величин метод наименьших квадратов обосновывается в теории вероятностей как частный случай принципа максимума правдоподобия.

По методу наименьших квадратов можно обрабатывать любые экспериментальные данные, однако оптимальность этой процедуры доказывается только для нормального распределения. В простейшем случае зависимость между измеряемыми величинами x и y носит линейный характер, поэтому экспериментальные точки группируются около некоторой прямой линии, как показано на рисунке 9.

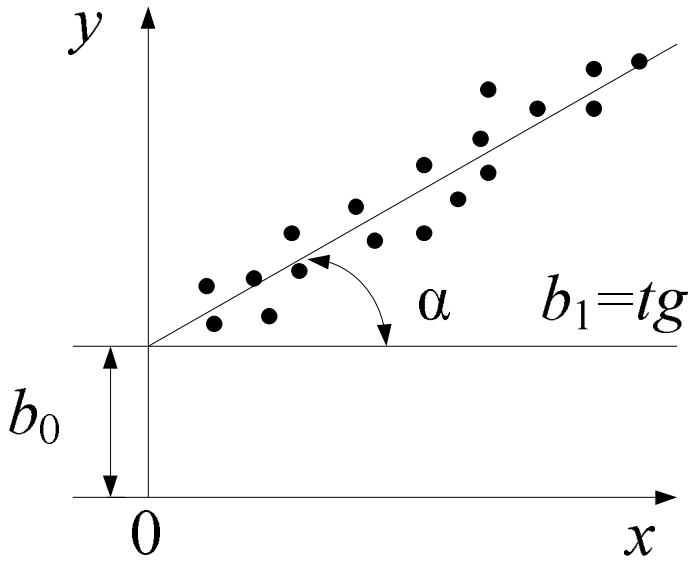


Рисунок 9 – Линейная зависимость между измеряемыми величинами

Уравнение прямой линии имеет вид:

$$y = b_0 + b_1 x , \qquad (18)$$

где b_0 – длина отрезка от начала координат до точки пересечения прямой с осью y; b_1 – тангенс угла наклона α к оси x.

Прямую линию стараются провести так, чтобы сумма квадратичных отклонений расчетных значений $y_р$ от экспериментальных значений $y_э$ была минимальной для всех n рассматриваемых опытов:

$$R = \sum_i (y_р - y_э)^2 \longrightarrow \min .$$

Уравнение $y = b_0 + b_1 x$, в которое подставлены значения коэффициентов, принято называть *уравнением линейной регрессии*.

Система уравнений для определения коэффициентов в общем виде записывается так:

$$\frac{\partial R}{\partial b_0} = 0; \quad \frac{\partial R}{\partial b_1} = 0. \tag{19}$$

Система нормальных уравнений при этом имеет вид:

$$\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} (b_0 + b_1 x_i) = 0, \; \sum_{i=1}^{n} y_i x_i - \sum_{i=1}^{n} (b_0 + b_1 x_i) x_i = 0,$$

или

$$n b_0 + b_1 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i, \; b_0 \sum_{i=1}^{n} x_i + b_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i. \tag{20}$$

Коэффициенты b_0 и b_1 легко найти при помощи определителей:

$$b_0 = \frac{\begin{vmatrix} \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}} = \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2} \tag{21}$$

$$b_1 = \frac{\begin{vmatrix} n & \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i y_i \end{vmatrix}}{\begin{vmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{vmatrix}} = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i \right)^2} = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}, \tag{22}$$

b_0 проще найти по известному b_1 из первого уравнения системы:

$$b_0 = \overline{y} - b_1 \overline{x}. \qquad (23)$$

Процедуры вычисления коэффициентов регрессии, корреляции и некоторых других величин, необходимых для их статистического оценивания можно упростить, если вычислить следующие промежуточные величины:

$$s_1 = n;\ s_2 = \sum_i x_i;\ s_3 = \sum_i y_i;\ s_4 = \sum_i x_i^2;\ s_5 = \sum_i x_i y_i;\ s_6 = \sum_i y_i^2.$$

Другими словами, n – это общее число опытов, в которых измерялись значения величин x и y; s_2 характеризует сумму всех измеренных значений x, а s_3 – сумму всех значений величины y. Остальные обозначения также понятны из формул.

Уравнение (23) показывает, что между коэффициентами b_0 и b_1 существует корреляционная зависимость. Для количественной оценки линейной корреляции используется коэффициент парной корреляции r_{xy}:

$$r_{xy} = \frac{s_1 s_5 - s_2 s_3}{\sqrt{(s_1 s_4 - s_2^2)(s_1 s_6 - s_3^2)}}. \qquad (24)$$

Этот коэффициент может принимать следующие значения:

1. $r_{xy}=0$ – это свидетельствует об отсутствии корреляционной связи между x и y;

2. $r_{xy}=1$ – в данном случае существует строгая положительная детерминистическая связь (т.е. описываемая строгой физико-химической формулой);

3. $r_{xy}=-1$ – между x и y существует строгая отрицательная детерминистическая связь;

4. $-1<r_{xy}<1$ – это наиболее распространенный случай: корреляционная связь может быть как положительной, так и отрицательной и характеризоваться различной степенью тесноты связи.

Чем ближе абсолютное значение коэффициента корреляции $|r_{xy}|$ к единице, тем сильнее линейная связь между x и y. Следует отметить, что r_{xy} одновременно отражает степень случайности и криволинейности связи между величинами x и y. Например, зависимость y от x может быть близкой к функциональной, но существенно нелинейной. В этом случае r_{xy} будет значительно меньше единицы.

После того как уравнение регрессии найдено, необходимо провести статистический анализ результатов. Этот анализ заключается в проверке значимости всех коэффициентов регрессии в сравнении с ошибкой воспроизводимости и адекватности уравнения. Такое исследование называется регрессионным анализом.

Оценка значимости коэффициентов производится по критерию Стьюдента:

$$t_j = \frac{|b_j|}{s_{bj}}. \qquad (25)$$

где b_j – коэффициент уравнения регрессии; s_{bj} – среднее квадратичное отклонение j-го коэффициента. Значения коэффициента Стьюдента при 10%-ном, 5%-ном и 2,5%-ном уровнях значимости приведены в приложении А.

Если t_j больше табличного $t_p(f)$ для выборочного уровня значимости p и числа степеней свободы $f=f_{\text{воспр}}$, то коэффициент b_j значимо отличается от нуля; s_{bj} определяется:

$$s_{b_0} = \sqrt{\frac{s^2_{\text{воспр}} \sum_{i=1}^{n} x_i^2}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}}, \quad s_{b_1} = \sqrt{\frac{s^2_{\text{воспр}} n}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}}.$$

Здесь $s^2_{\text{воспр}}$ – дисперсия воспроизводимости, характеризующая точность результатов измерений, полученных для m параллельных опытов по n измерений:

$$s^2_{\text{воспр}} = \frac{\sum_{i=1}^{n}\sum_{u=1}^{m}(y_{iu} - \bar{y}_i)}{n(m-1)}.$$

$f_{\text{воспр}}$ – число степеней свободы дисперсии воспроизводимости:

$$f_{\text{воспр}} = \sum_{i=1}^{n}(m_i - 1).$$

Незначимые коэффициенты исключаются из уравнения регрессии. Оставшиеся коэффициенты пересчитываются заново, поскольку коэффициенты закоррелированы друг с другом. Адекватность уравнения проверяется по критерию Фишера:

$$F = \frac{s^2_{\text{ад}}}{s^2_{\text{воспр}}}, \qquad (26)$$

где $s^2_{\text{ад}}$ – дисперсия адекватности

$$s^2_{\text{ад}} = \frac{SS_{\text{ад}}}{f_{\text{ад}}}.$$

$SS_{\text{ад}}$ – сумма квадратов адекватности $SS_{\text{ад}} = SS_{\text{ост}} - SS_{\text{воспр}}$,

$f_{\text{ад}}$ – число степеней свободы дисперсии адекватности

$$f_{\text{ад}} = f_{\text{ост}} - f_{\text{воспр}} = n - l, \qquad (27)$$

l – число коэффициентов в уравнении регрессии,

$SS_{\text{воспр}}$ – сумма квадратов, связанная с дисперсией воспроизводимости $s^2_{\text{воспр}}$

$$SS_{\text{воспр}} = \sum_{i=1}^{n}\sum_{u=1}^{m_i}(y_{iu} - \overline{y}_i)^2, \ \overline{y}_i = \frac{1}{m_i}\sum_{u=1}^{m_j} y_{iu},$$

$$s^2_{\text{воспр}} = \frac{SS_{\text{воспр}}}{f_{\text{воспр}}},$$

$SS_{\text{ост}}$ – остаточная сумма квадратов

$$SS_{\text{ост}} = \sum_{i=1}^{n}\sum_{u=1}^{m_i}(y_{iu} - \hat{y}_i)^2, \ f_{\text{ост}} = \sum_{i=1}^{n} m_i - l,$$

$f_{\text{ост}}$ – число степеней свободы остаточной дисперсии $s^2_{\text{ост}}$

$$s^2_{\text{ост}} = \frac{SS_{\text{ост}}}{f_{\text{ост}}} = \frac{\sum_{i=1}^{n}\sum_{u=1}^{m_i}(y_{iu} - \hat{y}_i)}{\sum_{i=1}^{n} m_i - l}.$$

Если F окажется меньше табличного значения $F_{1-p}(f_1, f_2)$ для уровня значимости p и числа степеней свободы $f_1 = f_{\text{ад}}$ и $f_2 = f_{\text{воспр}}$, уравнение адекватно эксперименту.

Для одинакового числа опытов $m_1 = m_2 = \ldots = m_i = \ldots = m_n = m$ вычисления упрощаются:

$$S^2_{\text{ад}} = \frac{m\sum_{i=1}^{n}(\overline{y}_i - \hat{y}_i)^2}{n - l},$$

Если опыты проведены без параллельных, а для получения дисперсии воспроизводимости проделана отдельная серия из m опытов, тогда:

$$S^2_{\text{ад}} = S^2_{\text{ост}} = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n-l},\ S^2_{\text{воспр}} = \frac{\sum_{u=1}^{m}(y_u^0 - \bar{y}^0)}{m-1},\ \bar{y}^0 = \frac{1}{m}\sum_{u=1}^{m} y_u^0.$$

При отсутствии параллельных опытов и дисперсии воспроизводимости можно оценить качество аппроксимации принятым уравнением, сравнив $s^2_{\text{ост}}$ и дисперсию относительно среднего s^2_y:

$$S^2_y = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n-1},$$

по критерию Фишера

$$F = \frac{s^2_y(f_1)}{s^2_{\text{ост}}(f_2)}. \qquad (28)$$

В этом случае критерий Фишера показывает, во сколько раз уменьшается рассеяние относительно полученного уравнения регрессии по сравнению с рассеянием относительно среднего. Чем больше значение F превышает табличное $F_{1-p}(f_1, f_2)$ для выборочного уровня значимости p и чисел степеней свободы $f_1=n-1$ и $f_2=n-l$, тем эффективнее уравнение регрессии.

Вычисление остаточной дисперсии имеет большое значение в теории статистических методов построения эмпирических зависимостей. В литературе можно встретить разные наименования этой величины: остаточная дисперсия, сумма квадратов остатков, остаточная сумма квадратов. Остаточная дисперсия представляет собой показатель ошибки предсказания уравнением регрессии результатов опытов. Качество предсказания определяют, сравнивая остаточную дисперсию с общей дисперсией s^2_y, таким образом, критерий Фишера показывает, во сколько раз уравнение регрессии предсказывает результаты опытов лучше, чем среднее y.

Значения критерия Фишера для уровней значимости 5% приведены в приложении Б.

11.4.5 ПРИМЕР ОПРЕДЕЛЕНИЯ КОНСТАНТ ДЛЯ РЕАКЦИИ РАЗЛОЖЕНИЯ

Для известного механизма химической реакции: $A \xrightarrow{k} B + C$ определим константу скорости k, если имеется таблица экспериментальной зависимости концентрации компонента C_A от времени: $C_A = f(t)$.

Скорость расходования компонент C_A в системе выражается уравнением:

$$\frac{dC}{dt} = w = kC^n, \tag{29}$$

где n – порядок реакции, w – скорость реакции.

Заменим производную концентрации компонента C по времени t на легко вычисляемую величину:

$$\frac{\Delta C}{\Delta t} = \frac{C_i - C_{i-1}}{t_i - t_{i-1}}, \tag{30}$$

C_i, C_{i-1} – значения концентрации компонента C соответственно в моменты времени t_i, t_{i-1}. Таким образом получаем формулу для вычисления расчетной концентрации компонента C:

$$C_i = C_{i-1} + kC^n(t_i - t_{i-1}). \tag{31}$$

Теперь необходимо привести уравнение (29) к такому виду, чтобы можно было воспользоваться линейной регрессией для решения поставленной задачи. Для этого прологарифмируем его и получим выражение:

$$\ln w = \ln k + n \ln C. \tag{32}$$

Эта зависимость линейная. Уравнение прямой в данном случае выражается формулой (23), где $y = \ln w$, $x = \ln C$, $b_1 = n$, $b_0 = \ln k$.

Зная экспериментальные значения концентрации C и времени t, мы можем определить y и x в выражении (26).

Необходимо подобрать коэффициенты b_0 и b_1 так, чтобы сумма квадратичных отклонений расчетных значений величины y от экспериментальных значений была минимальной для всех рассматриваемых опытов:

$$R = \sum_i (y_\text{р} - y_\text{э})^2 \longrightarrow \min. \tag{33}$$

Фактически необходимо минимизировать разницу между расчетными и экспериментальными значениями концентраций (рисунок 10).

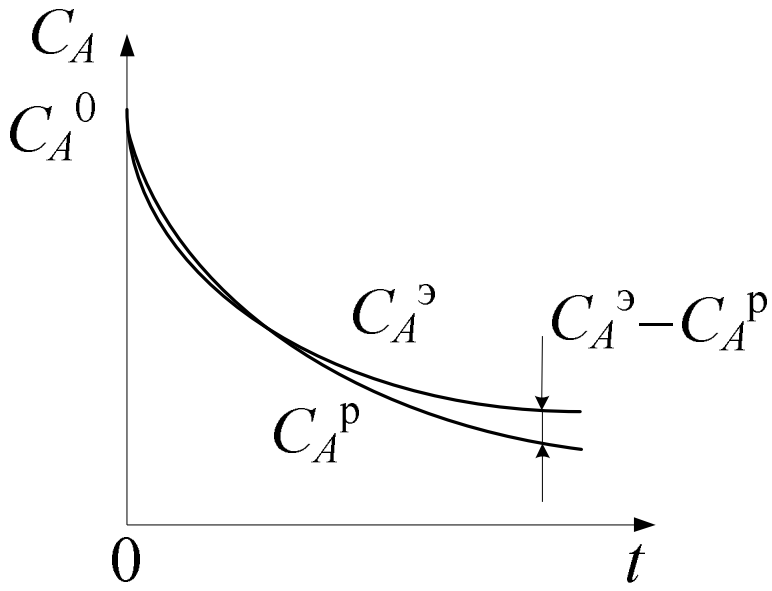


Рисунок 10 – Экспериментальные и расчетные концентрации компонента *A*

Определив по методу наименьших квадратов ln*k* и *n*, находим *k* =exp(ln*k*). Подставив полученное значение *k* в уравнение (29), мы сможем определить расчетные значения концентрации компонента *C*.

Если необходимо найти предэкспоненциальный множитель и энергию активации *E*, то можно снова обратиться к линейной регрессии. Для этого прологарифмируем выражение закона Аррениуса и получим выражение:

$$k = k_0 \exp(E/(RT)),\ \ln k = \ln k_0 - E/(RT). \qquad (34)$$

Введя обозначения (34):

$$y = \ln k,\ b_0 = \ln k_0,\ b_1 = -E/R,\ x = 1/T.$$

мы получили линеаризованное уравнение регрессии вида $y=a+bx$, и сможем рассчитать предэкспоненциальный множитель k_0 и энергию активации *E*.

Регрессионные модели относятся к классу стохастических моделей систем. Для применимости стохастического подхода должны выполняться следующие требования:

1) массовость проводимых экспериментов, то есть достаточно большое их число;

2) повторяемость условий экспериментов, оправдывающая сравнение результатов различных экспериментов;

3) статистическая устойчивость.

Поэтому для того, чтобы воспользоваться формулами для вычисления критерия Фишера, мы должны принять, что эти требования выполняются, иначе расчеты статистических критериев бессмысленны.

11.4.6 ОЦЕНКА АДЕКВАТНОСТИ МАТЕМАТИЧЕСКИХ МОДЕЛЕЙ КИНЕТИКИ

Свойство адекватности характеризует способность модели описывать свойства объекта с точностью не ниже заданной.

Для приближенного описания одних и тех же явлений могут быть предложены различные модели, и для оценки их качества необходимо выбрать критерий для сравнения расчетных и экспериментальных данных. Чаще всего рассчитываются функционалы, характеризующие степень расхождения между экспериментальными и расчетными данными.

Пусть для t значений начальных условий в q точках проведены измерений $r \le n$ компонент вектора переменных C. Если адекватность модели оценивается для заданного набора начальных условий, то функционал, характеризующий степень расхождения между экспериментальными и расчетными данными, будет иметь вид, представленный выражением:

$$F_1 = \sum_{j=1}^{r} \sum_{l=1}^{q} \sum_{k=1}^{t} m_j (C_{jlk}^{\text{р}} - C_{jlk}^{\text{э}})^2, \qquad (35)$$

где $C_{jlk}^{\text{р}}$, $C_{jlk}^{\text{э}}$ – расчетное и экспериментальное значение концентрации j-го компонента в 1-ый момент времени при k-ом начальном условии эксперимента; m_j – весовой коэффициент j-го компонента, который выбирается проектировщиком постоянным для всего времени исследования.

Критерий (35) может быть использован, если измеряется концентрация одного компонента, обычно целевого продукта, или нескольких из состава реакционной массы, имеющих близкий диапазон измерения. Если же концентрации измеряемых компонентов находятся в избытке и сильно отличаются друг от друга, можно использовать критерий вида (36):

$$F_2 = \sum_{j=1}^{r} \sum_{l=1}^{q} \sum_{k=1}^{t} m_j \left(\frac{(C_{jlk}^{\text{р}} - C_{jlk}^{\text{э}})^2}{(C_{jlk}^{\text{э}})^2} \right). \qquad (36)$$

В случае высокой конверсии отдельных компонентов, когда их содержание стремится к нулю, следует использовать критерий вида (37).

$$F_2 = \sum_{j=1}^{r} \sum_{l=1}^{q} \sum_{k=1}^{t} m_j \left(\frac{(C_{jlk}^{\text{р}} - C_{jlk}^{\text{э}})^2}{\Delta C_j^2} \right), \qquad (37)$$

где ΔC_j – диапазон изменения концентрации j-го компонента.

При анализе адекватности ММ должны учитываться ограничения на входные параметры, заданные в виде критериальных ограничений.

Выбор весовых коэффициентов для зависимостей (35)-(37) связан с точностью экспериментальных данных и их важностью с точки зрения мнения экспертов.

11.4.7 ХАРАКТЕРИСТИКА ПРОГРАММНОГО КОМПЛЕКСА

Таким образом, формализованное описание задачи определения констант скоростей химической реакции, как объекта программирования, позволяет создать программный комплекс, обладающий следующими характеристиками:

1. Операцио́нная систе́ма, **ОС** (англ. *operating system*) – базовый комплекс компьютерных программ, обеспечивающий управление аппаратными средствами компьютера, работу с файлами, ввод и вывод данных, а также выполнение прикладных программ и утилит: **Microsoft Windows.**

2. Язык программирования: C++ – компилируемый строго типизированный язык программирования общего назначения.

3. Количество разработанных классов (class): 2, а именно

TForm1 – класс ввода исходных данных (начальные концентрации конечных компонентов, распределение концентраций исходных компонентов во времени);

TForm2 – класс вывода результатов расчета (порядок реакции, константа скорости реакции, дисперсия, коэффициент корреляции).

4. Количество потоков выполнения команд (нитей, threads) – одна последовательность хода управления внутри программы **: 1.**

5. Распределенность приложения: локальное.

6. Инструментальная среда разработки: C++ Builder 6.

7. Количество исходных файлов программы: 5 (regres.cpp, MainInterpol.h, MainInterpol.cpp, RezultInterpol.h, RezultInterpol.cpp).

11.4.8 Пример выполнения Контрольной работы №4

Уравнение химической реакции дано в общем виде для реакции разложения: $A \xrightarrow{k} B+2C$.

Алгоритм для определения константы скорости реакции представлен на рисунке 11. Алгоритм позволяет на основании исходных данных определить k – константы скорости реакции, n – порядок реакции, r – коэффициент корреляции и d – дисперсии. Исходными данными являются:

- изменение концентрации исходного компонента $C_A(t)$ во времени,
- начальные концентрации компонентов C_B и C_C,
- количество экспериментальных точек N.

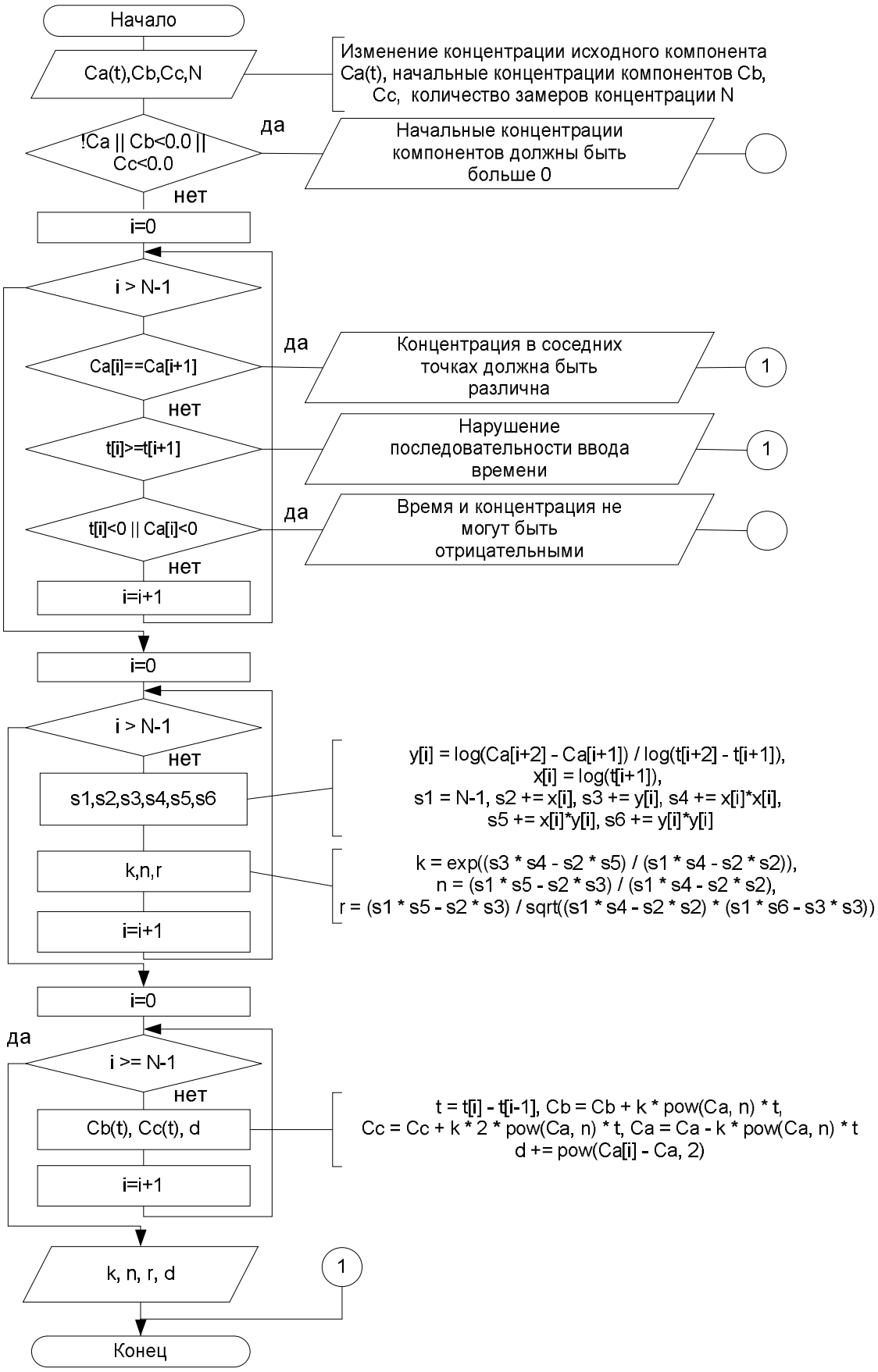


Рисунок 11 – Алгоритм определения константы скорости реакции

Для реализации алгоритма решения обратной задачи кинетики разработано программное обеспечение, включающее графический пользовательский интерфейс. В качестве интегрированной среды разработки программного обеспечения (*Integrated Development Environment, IDE*) используется C++ Builder. C++ Builder среда быстрой разработки (*rapid application development, RAD*), выпускаемая компанией CodeGear. Среда предназначена для написания программ на языке программирования C++. C++ Builder объединяет библиотеку визуальных компонентов и среду программирования.

Программа состоит из двух форм (окон), представленных на рисунках 12 и 13 соответственно и позволяющих вводить начальные данные и отображать результаты расчета. Программа включает пять исходных файлов программы: regres.cpp, MainInterpol.h, MainInterpol.cpp, RezultInterpol.h, RezultInterpol.cpp. Ниже приведен состав файлов с подробными комментариями.

Графический интерфейс включает следующие классы визуальных компонентов: формы TForm, изображения TImage, панели TPanel, группы элементов TGroupBox, метки TLabel, однострочный редактор текста с меткой TLabeledEdit, график TChart, кнопки TButton, компонент для ввода и изменения числовых значений TUpDown, таблица TStringGrid, и.т.д.

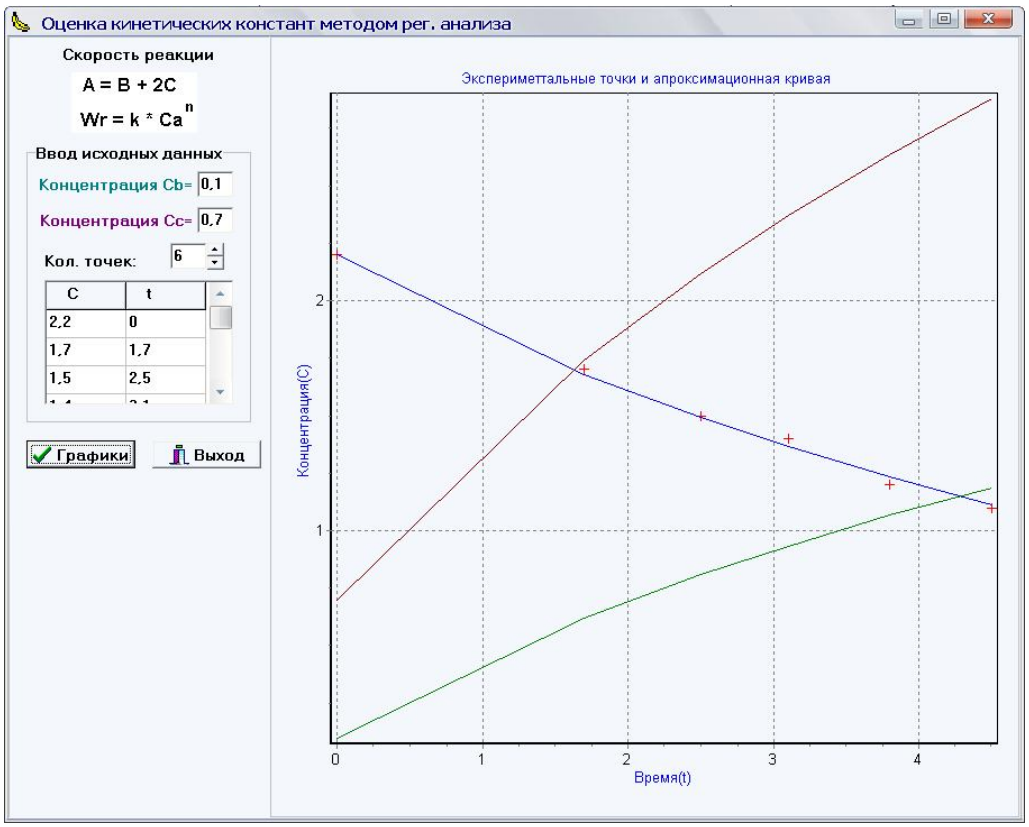


Рисунок 12 – Форма ввода исходных данных, отображения экспериментальных значений и расчетных кривых концентраций

Рисунок 13 – Форма отображения результатов расчета

Исходный код программы:

1) regres.cpp

```
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("MainInterpol.cpp", Form1);
USEFORM("RezultInterpol.cpp", Form2);
//Функция, с которой начинается выполнение программы
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
	try
	{
		Application->Initialize(); //Инициализация приложения
		Application->Title = "Regres"; //Задание заголовка
		Application->CreateForm(__classid(TForm1), &Form1); // Создание формы
		Application->Run();
		// Запуск обработчика событий и визуализация компонентов
	} // Обработчик исключений Exception
	catch (Exception &exception)
	{
		Application->ShowException(&exception);
	}
	catch (...) // Обработчик всех исключений
	{
		try
		{
			throw Exception("");
		}
		catch (Exception &exception)
		{
			Application->ShowException(&exception);
		}
	}
	return 0;
}
```

2) MainInterpol.h

```
//Задание стражей включения, обеспечивающих дублирования заголовочного файла
#ifndef MainInterpolH
#define MainInterpolH
//Подключение заголовочных файлов
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Chart.hpp>
#include <ExtCtrls.hpp>
#include <TeEngine.hpp>
#include <TeeProcs.hpp>
#include <Grids.hpp>
#include <Buttons.hpp>
#include <ComCtrls.hpp>
#include <Graphics.hpp>
#include <Series.hpp>
//Объявление класса формы
class TForm1 : public TForm
{
__published:   // Объявление визуальных компонентов на форме
        TPanel *Panel1;
        TPanel *Panel2;
        TChart *Chart1;
        TGroupBox *GroupBox1;
        TStringGrid *StringGrid1;
        TLabel *Label1;
        TImage *Image1;
        TBitBtn *BitBtn1;
        TBitBtn *BitBtn2;
        TLabel *Label2;
        TEdit *Edit1;
        TUpDown *UpDown1;
        TPointSeries *Series1;
        TLineSeries *Series2;
        TLabeledEdit *LabeledEdit1;
        TLabeledEdit *LabeledEdit2;
        TLineSeries *Series3;
        TLineSeries *Series4;
//Объявление обработчиков событий
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall UpDown1Click(TObject *Sender, TUDBtnType Button);
        void __fastcall BitBtn1Click(TObject *Sender);
private:          // User declarations
public:           // User declarations
        double k, n, d, r;
        __fastcall TForm1(TComponent* Owner);
};
extern PACKAGE TForm1 *Form1;
#endif
```

3) MainInterpol.cpp

```
#include <vcl.h>
#pragma hdrstop
#include "RezultInterpol.h"
#include "MainInterpol.h"
#include "math.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//Определение конструктора формы
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner), k(0.0), n(0.0), r(0.0), d(0.0)
{
}
//Определение инициализации формы
void __fastcall TForm1::FormCreate(TObject *Sender)
{
//Задание полей заголовка таблицы
  StringGrid1->Cells[0][0] = "    C";
  StringGrid1->Cells[1][0] = "    t";
}
//Определение обработчика нажатия на кнопку для изменения количества строк
//таблицы
void __fastcall TForm1::UpDown1Click(TObject *Sender, TUDBtnType Button)
{
  StringGrid1->RowCount = UpDown1->Position + 1;
}
//Определение обработчика кнопки расчета и построения графиков
void __fastcall TForm1::BitBtn1Click(TObject *Sender)
{
//Инициализация  переменных необходимых для решения задачи кинетики
  d = 0.0;
  int i, reg_up(0), reg_down(0);
  double *y, *x, s1(UpDown1->Position - 1),
  s2(0.0), s3(0.0), s4(0.0), s5(0.0), s6(0.0), ca, cb, cc, t;
  //Очистка графиков после предыдущего расчета
  Series1->Clear();
  Series2->Clear();
  Series3->Clear();
  Series4->Clear();
  try
   {
//Создание в динамической памяти массива для хранения экспериментальных данных
      y = new double[UpDown1->Position - 1];
      x = new double[UpDown1->Position - 1];
      if(!y || !x) throw "НЕТ свободной памяти";

      //Задание начальных значений концентраций
      ca = StringGrid1->Cells[0][1].ToDouble();
      cb = LabeledEdit1->Text.ToDouble();
```

```
    cc = LabeledEdit2->Text.ToDouble();
    //Проверка входных данных на допустимость значений
    if(!ca || cb < 0.0 || cc < 0.0)
throw "Начальные концентрации компонентов должны быть больше 0";
    for(i = 0; i < UpDown1->Position - 1; i++)
    {
      if(StringGrid1->Cells[0][i + 1].ToDouble() > StringGrid1->Cells[0][i + 2].ToDouble())
      reg_down++;
      else reg_up++;
      if(StringGrid1->Cells[0][i + 1].ToDouble() == StringGrid1->Cells[0][i + 2].ToDouble())
throw "Концентрация в соседних точках должна быть различна";
      if(StringGrid1->Cells[1][i + 1].ToDouble() >= StringGrid1->Cells[1][i + 2].ToDouble())
throw "Нарушение последовательности ввода времени";
      if(StringGrid1->Cells[1][i+1].ToDouble()< 0||StringGrid1->Cells[0][i + 1].ToDouble()<0)
throw "Время и концентрация не могут быть отрицательными";
    }
//Расчет значений локальных переменных, необходимых для решения задачи
    for(i = 0; i < UpDown1->Position - 1; i++)
    {
      y[i] = log(fabs(StringGrid1->Cells[0][i + 2].ToDouble() –
StringGrid1->Cells[0][i + 1].ToDouble()) /
(StringGrid1->Cells[1][i + 2].ToDouble() –
StringGrid1->Cells[1][i + 1].ToDouble()));
      x[i] = log(StringGrid1->Cells[0][i + 1].ToDouble());
      s2 += x[i]; s3 += y[i];
      s4 += x[i] * x[i];
      s5 += x[i] * y[i];
      s6 += y[i] * y[i];
    }
//Расчет константы скорости реакции, порядка реакции, коэффициента корреляции
    k = exp((s3 * s4 - s2 * s5) / (s1 * s4 - s2 * s2));
    n = (s1 * s5 - s2 * s3) / (s1 * s4 - s2 * s2);
    r = (s1 * s5 - s2 * s3) / sqrt((s1 * s4 - s2 * s2) * (s1 * s6 - s3 * s3));

//Построение графиков и расчет концентраций компонентов реакции
    for(int i = 1; i <= UpDown1->Position; i++)
    {
      Series1->AddXY(StringGrid1->Cells[1][i].ToDouble(),
  StringGrid1->Cells[0][i].ToDouble());
      if(ca < 0) throw "Модель невозможно описать линейной регрессией";
      if(i == 1)
      {
        Series2->AddXY(StringGrid1->Cells[1][i].ToDouble(), ca );
        Series3->AddXY(StringGrid1->Cells[1][i].ToDouble(), cb );
        Series4->AddXY(StringGrid1->Cells[1][i].ToDouble(), cc );

      }
      else
      {
        t = StringGrid1->Cells[1][i].ToDouble() - StringGrid1->Cells[1][i - 1].ToDouble();
        Series3->AddXY(StringGrid1->Cells[1][i].ToDouble(), cb = cb + k * pow(ca, n) * t);
        Series4->AddXY(StringGrid1->Cells[1][i].ToDouble(), cc = cc + k * 2*pow(ca, n)*t);
```

```
            if(reg_down > reg_up)
Series2->AddXY(StringGrid1->Cells[1][i].ToDouble(), ca =ca-k*pow(ca, n) * t);
            else
Series2->AddXY(StringGrid1->Cells[1][i].ToDouble(), ca =ca+k*pow(ca, n) * t);
        }
//расчет дисперсии
        d += pow(StringGrid1->Cells[0][i].ToDouble() - ca, 2);
    }
    //Создание формы отображения результатов
    Application->CreateForm(__classid(TForm2), &Form2);
    //Отображение формы
    Form2->ShowModal();

    delete[]y;
    delete[]x;
  }
  catch(char*str)
  {
    Series1->Clear();
    Series2->Clear();
    Series3->Clear();
    Series4->Clear();
   // Вывод сообщения об ошибке
    Application->MessageBox(str, "Ошибка", MB_ICONERROR | MB_OK);
  }
  catch(EConvertError&)
  {
// Вывод сообщения об ошибке
    Application->MessageBox("Не правильный тип данных", "Ошибка",
MB_ICONERROR | MB_OK);
    delete[]y;
    delete[]x;
  }
  catch(...)
  {
// Вывод сообщения об ошибке
    Application->MessageBox("НЕ известная ошибка", "Ошибка",
MB_ICONERROR | MB_OK);
  }
}
```

4) RezultInterpol.h

```
#ifndef RezultInterpolH
#define RezultInterpolH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <ExtCtrls.hpp>
```

```
//---------------------------------------------------------------------------
class TForm2 : public TForm
{
__published:   // IDE-managed Components
        TPanel *Panel1;
        TGroupBox *GroupBox2;
        TLabel *Label3;
        TLabel *Label4;
        TEdit *Edit2;
        TEdit *Edit3;
        TLabel *Label1;
        TEdit *Edit1;
        TLabel *Label2;
        TEdit *Edit4;
        TBitBtn *BitBtn1;
        void __fastcall FormShow(TObject *Sender);
private:        // User declarations
public:         // User declarations
        __fastcall TForm2(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm2 *Form2;
//---------------------------------------------------------------------------
#endif
```

5) RezultInterpol.cpp

```
#include <vcl.h>
#pragma hdrstop
#include "MainInterpol.h"
#include "RezultInterpol.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
//---------------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
        : TForm(Owner)
{
}
//Определение обработчика отображения формы
void __fastcall TForm2::FormShow(TObject *Sender)
{//Вывод результатов расчета
  Edit2->Text = FloatToStrF(Form1->n, ffGeneral, 4, 4);
  Edit3->Text = FloatToStrF(Form1->k, ffGeneral, 4, 4);
  Edit1->Text = FloatToStrF(Form1->d, ffGeneral, 4, 4);
  Edit4->Text = FloatToStrF(Form1->r, ffGeneral, 4, 4);
```

}Тестирование разработанного в среде C++ Builder программного обеспечения проводилось при следующих входных данных:

Начальная концентрация $C_B = 0{,}1$ моль/л; $C_C = 0{,}7$ моль/л.

Распределение концентрации $C_A(t)$ во времени t приведено в таблице 20.

Таблица 20 – Распределение концентрации компонента *A* во времени *t,* $C_A(t)$

Концентрация C_A, моль/л	**Время t, с**
2,2	0
1,7	1,7
1,5	2,5
1,4	3,1
1,2	3,8
1,1	4,5

Результаты расчета:

Порядок реакции $n = 1$

Константа скорости реакции $k = 0{,}14$ 1/с

Проверка адекватности модели:

Дисперсия $d = 0{,}003$ (моль / л)^2

Коэффициент корреляции $r = 0{,}69$

ЗАКЛЮЧЕНИЕ

Учебное пособие предназначено для студентов, обучающихся по направлению подготовки 230100 «Информатика и вычислительная техника». Приведенные в учебном пособии материалы могут быть использованы при выполнении учебных проектов и небольших программных продуктов.

В учебном пособии отражены основные принципы объектно-ориентированного программирования, основные структуры программирования, приведен пример программной реализации в среде C++ Builder для исследования кинетики химических реакций. Для уточнения сложных вопросов программирования необходимо использовать рекомендуемую в учебном пособии справочную литературу.

СПИСОК ЛИТЕРАТУРЫ

1 Архангельский, А.Я. Программирование в C++Builder 6 и 2006: [разработка программ для Windows: методические и справочные материалы по C++Builder] / А. Я. Архангельский, М. А. Тагин. – Москва : БИНОМ, 2007. – 1181 с.

2 Страуструп, Бьерн. Язык программирования C++ / Бьерн Страуструп; Пер. с англ. С. Анисимова, М. Кононова; Под ред. Ф. Андреева, А. Ушакова. – Спец. изд. – М.: Бином; СПб.: Нев. диалект, 2001. – 1098 с.

3 Подбельский, В.В. Язык СИ++ / В.В. Подбельский. – 5-е изд. – М.: Финансы и статистика, 2008. – 559 с.

4 Шилдт, Герберт. C++. Базовый курс / Герберт Шилдт; [пер. с англ. Н.М. Ручко]. – 3-е изд. – М.: Вильямс, 2008. – 620 с.

5 Элджер, Джефф. C++ / Джефф Элджер; [Пер. с англ. Е. Матвеев]. – СПб.: Питер: Питер Бук, 2001. – 320 с.

6 Эккель, Брюс. Философия C++: Практ. программирование / Брюс Эккель, Чак Эллисон; [Пер. с англ. Е. Матвеев]. – М.: Питер, 2004. – 608 с.

7 Липпман, Стенли Б. Язык программирования C++: вводный курс / Стенли Б. Липпман, Жози Лажойе, Барбара Му; [пер. с англ. и ред. В. А. Коваленко]. – 4-е изд. – М.: Вильямс, 2007. – 889 с.

8 Астахова, И.Ф. Язык C++ / И.Ф. Астахова, С.В. Власов, В.В. Фертиков, А.В. Ларин. – Минск: Новое знание, 2003. – 200 с.

9 Иванова, Г.С. Основы программирования / Г.С. Иванова. – Изд. 4-е, стер. – М.: Изд-во МГТУ, 2007. – 415 с.

10Иванова, Г.С. Объектно-ориентированное программирование / Г.С. Иванова, Т.Н. Ничушкина, Е.К. Пугачев; под ред. Г.С. Ивановой. – Изд. 3-е, стер. – М.: Изд-во МГТУ, 2007. – 366 с.

11Гартман, Т.Н. Основы компьютерного моделирования химико-технологических процессов : учебное пособие / Т.Н. Гартман, Д.В. Клушин. – Москва : Академкнига, 2008. – 415 с.

ПРИЛОЖЕНИЕ А
(справочное)

Значения критерия Стьюдента

Значения критерия Стьюдента приведены в таблице А1.

Таблица А.1 – Значения критерия Стьюдента

f	**Процентные точки распределения Стьюдента**		
	10%	**5%**	**2,5%**
	t	***t***	***t***
1	3,08	6,31	12,74
2	1,89	2,92	4,30
3	1,64	2,35	3,18
4	1,53	2,13	2,78
5	1,48	2,01	2,57
6	1,44	1,94	2,45
7	1,41	1,89	2,36
8	1,40	1,86	2,31
9	1,38	1,83	2,26
10	1,37	1,81	2,23
11	1,36	1,80	2,20
12	1,35	1,78	2,18
13	1,35	1,77	2,16
14	1,35	1,76	2,14
15	1,34	1,75	2,13
16	1,34	1,75	2,12
17	1,33	1,74	2,11
18	1,33	1,73	2,10
19	1,33	1,73	2,09
20	1,33	1,73	2,09

ПРИЛОЖЕНИЕ Б

(справочное)

Значения критерия Фишера (*p*=5%)

Таблица Б.1 – Значения критерия Фишера

f_2 \ f_1	1	2	3	4	5	6	7	8	9	10	12	15	20	24	30	40	60	120	∞
1	161,45	199,50	215,71	224,58	230,16	233,99	236,77	238,88	240,54	241,88	243,91	245,95	248,01	249,05	250,09	251,14	252,20	253,25	254,32
2	18,513	19,000	19,164	19,247	19,296	19,330	19,353	19,371	19,385	19,396	19,413	19,429	19,446	19,454	19,462	19,471	19,479	19,487	19,496
3	10,128	9,5521	9,2766	9,1172	9,0135	8,9406	8,8868	8,8452	8,8123	8,7855	8,7446	8,7029	8,6602	8,6385	8,6166	8,5944	8,5720	8,5494	8,5265
4	7,7086	6,9443	6,5914	6,3883	6,2560	6,1631	6,0942	6,0410	5,9988	5,9644	5,9117	5,8578	5,8025	5,7744	5,7459	5,7170	5,6878	5,6581	5,6281
5	6,6079	5,7861	5,4095	5,1922	5,0503	4,9503	4,8759	4,8183	4,7725	4,7351	4,6777	4,6188	4,5581	4,5272	4,4957	4,4638	4,4314	4,3984	4,3650
6	5,9874	5,1433	4,7571	4,5337	4,3874	4,2839	4,2066	4,1468	4,0990	4,0600	3,9999	3,9381	3,8742	3,8415	3,8082	3,7743	3,7398	3,7047	3,6688
7	5,5914	4,7374	4,3468	4,1203	3,9715	3,8660	3,7870	3,7257	3,6767	3,6365	3,5747	3,5108	3,4445	3,4105	3,3758	3,3404	3,3043	3,2674	3,2298
8	5,3177	4,4590	4,0662	3,8378	3,6875	3,5806	3,5005	3,4381	3,3881	3,3472	3,2840	3,2184	3,1503	3,1152	3,0794	3,0428	3,0053	2,9669	2,9276
9	5,1174	4,2565	3,8626	3,6331	3,4817	3,3738	3,2927	3,2296	3,1789	3,1373	3,0729	3,0061	2,9365	2,9005	2,8637	2,8259	2,7872	2,7475	2,7067
10	4,9646	4,1028	3,7083	3,4780	3,3258	3,2172	3,1355	3,0717	3,0204	2,9782	2,9130	2,8450	2,7740	2,7372	2,6996	2,6609	2,6211	2,5801	2,5379
11	4,8443	3,9823	3,5874	3,3567	3,2039	3,0946	3,0123	2,9480	2,8962	2,8536	2,7876	2,7186	2,6464	2,6090	2,5705	2,5309	2,4901	2,4480	2,4045
12	4,7472	3,8853	3,4903	3,2592	3,1059	2,9961	2,9134	2,8486	2,7964	2,7534	2,6866	2,6169	2,5436	2,5055	2,4663	2,4259	2,3842	2,3410	2,2962
13	4,6672	3,8056	3,4105	3,1791	3,0254	2,9153	2,8321	2,7669	2,7144	2,6710	2,6037	2,5331	2,4589	2,4202	2,3803	2,3392	2,2966	2,2524	2,2064
14	4,6001	3,7389	3,3439	3,1122	2,9582	2,8477	2,7642	2,6987	2,6458;	2,6021	2,5342	2,4630	2,3879	2,3487	2,3082	2,2664	2,2230	2,1778	2,1307
15	4,5431	3,6823	3,2874	3,0556	2,9013	2,7905	2,7066	2,6408	2,5876	2,5437	2,4753	2,4035	2,3275	2,2878	2,2468	2,2043	2,1601	2,1141	2,0658
16	4,4940	3,6337	3,2389	3,0069	2,8524	2,7413	2,6572	2,5911	2,5377	2,4935	2,4247	2,3522	2,2756	2,2354	2,1938	2,1507	2,1058	2,0589	2,0096
17	4,4513	3,5915	3,1968	2,9647	2,8100	2,6987	2,6143	2,5480	2,4943	2,4499	2,3807	2,3077	2,2304	2,1898	2,1477	2,1040	2,0584	2,0107	1,9604
18	4,4139	3,5546	3,1599	2,9277	2,7729	2,6613	2,5767	2,5102	2,4563	2,4117	2,3421	2,2686	2,1906	2,1497	2,1071	2,0629	2,0166	1,9681	1,9168
19	4,3808	3,5219	3,1274	2,8951	2,7401	2,6283	2,5435	2,4768	2,4227	2,3779	2,3080	2,2341	2,1555	2,1141	2,0712	2,0264	1,9796	1,9302	1,8780
20	4,3513	3,4928	3,0984	2,8661	2,7109	2,5990	2,5140	2,4471	2,3928	2,3479	2,2776	2,2033	2,1242	2,0825	2,0391	1,9938	1,9464	1,8963	1,8432
21	4,3248	3,4668	3,0725	2,8401	2,6848	2,5727	2,4876	2,4205	2,3661	2,3210	2,2504	2,1757	2,0960	2,0540	2,0102	1,9645	1,9165	1,8657	1,8117
22	4,3009	3,4434	3,0491	2,8167	2,6613	2,5491	2,4638	2,3965	2,3441	2,2967	2,2258	2,1508	2,0707	2,0283	1,9842	1,9380	1,8895	1,8380	1,7831
23	4,2793	3,4221	3,0280	2,7955	2,6400	2,5277	2,4422	2,3748	2,3201	2,2747	2,2036	2,1282	2,0476	2,0050	1,9605	1,9139	1,8649	1,8128	1,7570
24	4,2597	3,4028	3,0088	2,7763	2,6207	2,5082	2,4226	2,3551	2,3001	2,2547	2,1834	2,1077	2,0267	1,9838	1,9390	1,8920	1,8424	1,7897	1,7331
25	4,2417	3,3852	2,9912	2,7587	2,6030	2,4904	2,4047	2,3371	2,2821	2,2365	2,1649	2,0889	2,0075	1,9643	1,9192	1,8718	1,8217	1,7684	1,7110
26	4,2252	3,3690	2,9751	2,7426	2,5868	2,4741	2,3883	2,3205	2,2655	2,2197	2,1479	2,0716	1,9898	1,9464	1,9010	1,8533	1,8027	1,7488	1,6906
27	4,21,00	3,2541	2,9604	2,7278	2,5719	2,4591	2,3732	2,3053	2,2601	2,2043	2,1323	2,0558	1,9736	1,9299	1,8842	1,8361	1,7851	1,7307	1,6717
28	4,1960	3,3404	2,9467	2,7141	2,5581	2,4453	2,3593	2,2913	2,2360	2,1900	2,1179	2,0411	1,9586	1,9147	1,8687	1,8203	1,7689	1,7138	1,6541
29	4,1830	3,3277	2,9340	2,7014	2,5454	2,4324	2,3463	2,2782	2,2229	2,1768	2,1045	2,0275	1,9446	1,9005	1,8543	1,8055	1,7537	1,6981	1,6377
30	4,1709	3,3158	2,9223	2,6896	2,5336	2,4205	2,3343	2,2662	2,2107	2,1646	2,0921	2,0148	1,9317	1,8874	1,8409	1,7918	1,7396	1,6835	1,6223
40	4,0848	3,2317	2,8387	2,6060	2,4495	2,3359	2,2490	2,1802	2,1240	2,0772	2,0035	1,9245	1,8389	1,7929	1,7444	1,6928	1,6373	1,5766	1,5089
60	4,0012	3,1504	2,7581	2,5252	2,3683	2,2540	2,1665	2,0970	2,0401	1,9926	1,9174	1,8364	1,7480	1,7001	1,6491	1,5943	1,5343	1,4673	1,3893
120	3,9201	3,0718	2,6802	2,4472	2,2900	2,1750	2,0867	2,0164	1,9588	1,9105	1,8337	1,7505	1,6587	1,6084	1,5543	1,4952	1,4290	1,3519	1,2539
∞	3,8415	2,9957	2,6049	2,3719	2,2141	2,0986	2,0096	1,9384	1,8799	1,8307	1,7522	1,6664	1,5705	1,5173	1,4591	1,3940	1,3180	1,2214	1,0000

Кафедра систем автоматизированного проектирования и управления

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ
на примере объектов химической технологии

Учебное пособие

Чистякова Тамара Балабековна
Новожилова Инна Васильевна
Антипин Роман Васильевич

Отпечатано с оригинал макета. Формат 60x90 $^{1}/_{16}$
Печ. л. 14,5. Тираж 100 экз. Заказ № 16

федеральное государственное бюджетное образовательное,
учреждение высшего профессионального образования
«Санкт-Петербургский государственный технологический институт
(технический университет)»,
Издательство СПбГТИ(ТУ)

190013, г. Санкт-Петербург, Московский пр., д. 26