Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

сборник учебно-методических материалов

для специальности 09.02.07 Информационные системы и программирование

Благовещенск

2020

Печатается по решению
редакционно-издательского совета
факультета математики и информатики
Амурского государственного
университета

Составитель: Галаган Т.А.

Основы алгоритмизации и программирования: сборник учебно-методических материалов для специальности 09.02.07 – Благовещенск: Амурский гос. ун-т, 2020

КРАТКОЕ ИЗЛОЖЕНИЕ ЛЕКЦИОННОГО МАТЕРИАЛА

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

Компьютер выполняет инструкции встроенной системы элементарных команд – машинного языка, которой состоит из команд в двоичном представлении.

Языки программирования высокого уровня позволяют создавать компьютерные программы на языке, близком к естественному языку. Специальная программа, называемая *компилятором*, переводит программу, написанную на языке высокого уровня в машинный код.

Программа, написанная на языке высокого уровня, называется *исходной программой* и может быть выполнена на любом компьютере в соответствии с используемым компилятором. Это возможно, поскольку большинство языков высокого уровня стандартизировано.

Компилятор транслирует исходную программу в программу на машинном языке, которая называется *объектной программой*. Если программа содержит ошибки, компилятор создает *листинг* программы – текст с сообщениями об ошибках и другой полезной информацией.

Объектная программа отправляется на *выполнение*.

Язык С++ поддерживает различные парадигмы программирования – полный набор структурного и объектно-ориентированного программирования: модульность, блочную структуру программ, раздельную компиляцию, характернее для языков высокого уровня. С другой стороны, он имеет ряд низкоуровневых черт, в частности, операции над битами.

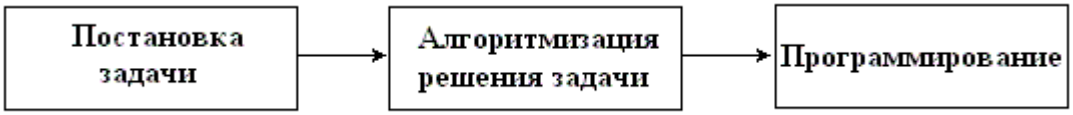
Он хорошо подходит как для начинающих программистов, так и для профессионалов.

АЛГОРИТМЫ И АЛГОРИТМИЗАЦИЯ

Возможности компьютера как технической основы обработки данных связаны с используемым программным обеспечением (программами).

Программа – упорядоченная последовательность команд (инструкций) компьютера для решения задачи.

Процесс создания программ можно представить как последовательность действий, представленных на рисунке.



Постановка задачи – это точная формулировка решения задачи на компьютере с описанием входной и выходной информации.

Алгоритм – система точно сформулированных правил, определяющая процесс преобразования допустимых исходных данных (входной информации) в желаемый результат (выходную информацию) за конечное число шагов.

Таким образом, алгоритмом называют последовательность команд (инструкций) для достижения некоторой цели.

Алгоритм решения задачи имеет ряд обязательных свойств:

дискретность – разбиение процесса обработки информации на более простые этапы (шаги), выполнение которых компьютером или человеком не вызывает затруднений;

определенность алгоритма – однозначность выполнения каждого отдельного шага преобразования информации;

выполнимость – конечность действий алгоритма решения задач, позволяющая получить желаемый результат при допустимых исходных данных за конечное число шагов;

массовость – пригодность алгоритма для решения определенного класса задач.

В алгоритме отражаются логика и способ формирования результатов решения с указанием

необходимых расчетных формул, логических условий, соотношений для контроля достоверности выходных результатов. В алгоритме обязательно должны быть предусмотрены все ситуации, которые могут возникнуть в процессе решения комплекса задач.

Программирование – теоретическая и практическая деятельность с созданием программ, осуществляющая перевод алгоритма решения задачи на язык программирования.

Существует несколько способов представления алгоритмов: словесный, графический и др. Один из них – представление в виде блок-схем.

Блок-схема алгоритма – это графическое представление алгоритма в виде геометрических фигур и стрелок.

ЭВОЛЮЦИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ. КЛАССИФИКАЦИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

В развитии инструментального программного обеспечения рассматривают пять поколений языков программирования (ЯП). Языки программирования как средство общения человека с ЭВМ от поколения к поколению улучшали свои характеристики, становясь все более доступными в освоении непрофессионалам.

Первые три поколения ЯП характеризовались более сложным набором зарезервированных слов и синтаксисом. Языки четвертого поколения все еще требуют соблюдения определенного синтаксиса при написании программ, но он значительно легче для освоения. Естественные ЯП, разрабатываемые в настоящее время, составят пятое поколение и позволят определять необходимые процедуры обработки информации, используя предложения языка, весьма близкого к естественному и не требующего соблюдения особого синтаксиса

Поколения	**Языки программирования**	**Характеристика**
Первое	Машинные	Ориентированы на использование в конкретной ЭВМ, сложны в освоении, требуют хорошего знания архитектуры ЭВМ
Второе	Ассемблеры, макроассемблеры	Более удобны для использования, но по-прежнему машинно-зависимы
Третье	Языки высокого уровня	Мобильные, человеко-ориентированные, проще в освоении
Четвёртое	Непроцедурные, объектно-ориентированные, языки запросов, параллельные	Ориентированы на непрофессионального пользователя и на ЭВМ с параллельной архитектурой
Пятое	Языки искусственного интеллекта, экспертных систем и баз знаний, естественные языки	Ориентированы на повышение интеллектуального уровня ЭВМ и интерфейса с языками

Элементы языков программирования могут рассматриваться на следующих уровнях: *алфавит* - совокупность символов, отображаемых на устройствах печати и экранах и/или вводимых с клавиатуры терминала. Обычно это набор символов Latin-1, с исключением управляющих символов. Иногда в это множество включаются неотображаемые символы, с указанием правил их записи (комбинирование в лексемы); *лексика* - совокупность правил образования цепочек символов (лексем), образующих идентификаторы (переменные и метки), операторы, операции и другие лексические компоненты языка. Сюда же включаются зарезервированные (запрещенные, ключевые) слова ЯП, предназначенные для обозначения операторов, встроенных функций и пр.

Классификация ЯП			
Фактор	Характеристика	Группы	Примеры ЯП
Уровень ЯП	Степень близости ЯП к архитектуре ПК	Низкий	Автокод, ассемблер
		Высокий	Fortran, Pascal, ADA, Basic, C и др.
		Сверхвысокий	Сетл
Специализация ЯП	Потенциальная или реальная область применения	Общего назначения (универсальные)	Fortran (инженерные расчёты), Cobol (Коммерческие задачи), Refal, Lisp(символьная обработка), Modula,

			ADA(программирование в реальном времени)
		Специализированные	
Алгоритмичность (процедурность)	Возможность абстрагироваться от деталей алгоритма решения задачи. Алгоритмичность тем выше, чем точнее приходится планировать порядок выполняемых действий	Процедурные	Ассемблер, Fortran, Basic, Pascal, ADA
		Непроцедурные	Prolog, Langin

БАЗОВЫЕ КОНСТРУКЦИИ ЯЗЫКА С++

Состав языка

Любой естественный язык содержит четыре основных элемента: символы, слова, словосочетания, предложения. Алгоритмический язык также включает символы, на основании которых строятся элементарные конструкции (*лексемы*). Из лексем и символов строятся *выражения*, которые в свою очередь образуют *операторы*.

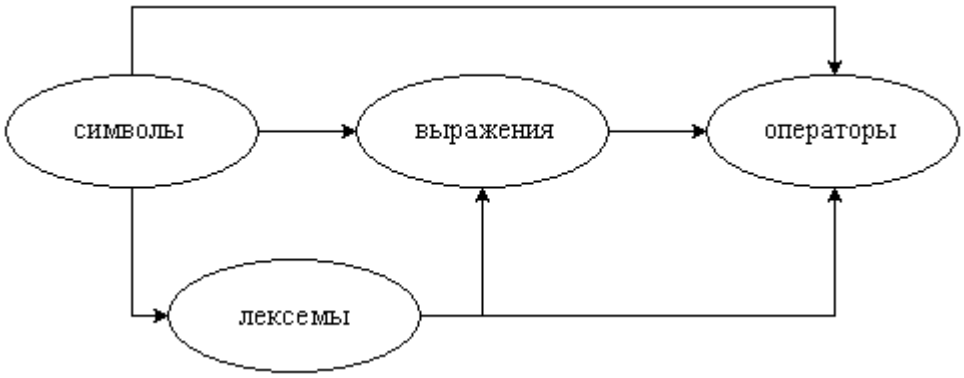


Рисунок 1. Состав алгоритмического языка

Алфавит языка включает в себя основные неделимые знаки, с помощью которых пишутся все тексты программ. Лексема является минимальной единицей языка, имеющей самостоятельный смысл. *Выражение* задает правило вычисления некоторого значения. *Оператор* представляет собой законченное описание некоторого действия. Любое выражение, заканчивающееся точкой с запятой, является оператором, выполнение которого заключается в вычислении выражения.

Операторы бывают исполняемые и неисполняемые. Первые задают действия над данными, а вторые служат для описания данных и называются операторами описания или просто описаниями. Для описания сложного действия требуется последовательность операторов. Операторы могут объединяться в сложный оператор или блок.

Объединенная единым алгоритмом совокупность описаний и операторов образуют программу на алгоритмическом языке.

Алфавит языка С++

Алфавит С++ включает в себя:

- прописные и строчные буквы латинские буквы и знак подчеркивания;
- арабские цифры от 0 до 9;
- специальные знаки: “, { } , | [ ] ( ) + - / % * . \ ‘ ^ ? < = > ! & # ~ $;
- пробельные символы: пробел, табуляция, символы перехода на новую строку.

Переменные, идентификаторы

Идентификаторы используются в С++ для именования различных объектов. *Идентификатор* – имя, связанное с данными или функцией программы, которое используется для обращения к этому объекту или функции.

Идентификатор представляет собой последовательность символов произвольной длины, содержащую буквы, цифры и символ подчеркивания, но начинающуюся обязательно с буквы или символа подчеркивания. Прописные и строчные буквы различаются. Пробелы внутри имен не допускаются.

Переменная – именованная область памяти, в которой хранятся данные определенного типа. У переменной есть имя (идентификатор) и значение. Имя служит для обращения к области памяти, в которой хранится переменная. Значение переменной может изменяться во время выполнения программы. Прежде чем использовать переменную, ее необходимо определить.

Идентификаторы используются для обозначения определенной области памяти, а компилятор транслирует имена в соответствующие адреса.

Каждый элемент данных должен принадлежать к определенному типу. Тип данных определяет, в каком виде они представлены в компьютере, а также какие преобразования компьютер может к ним применять.

Типы данных

Тип данных – множество допустимых значений данных и набор операций, применимых к этим значениям. Встроенные типы данных языка C++ подразделят на простые, структурированные и адресные (рисунок 2).

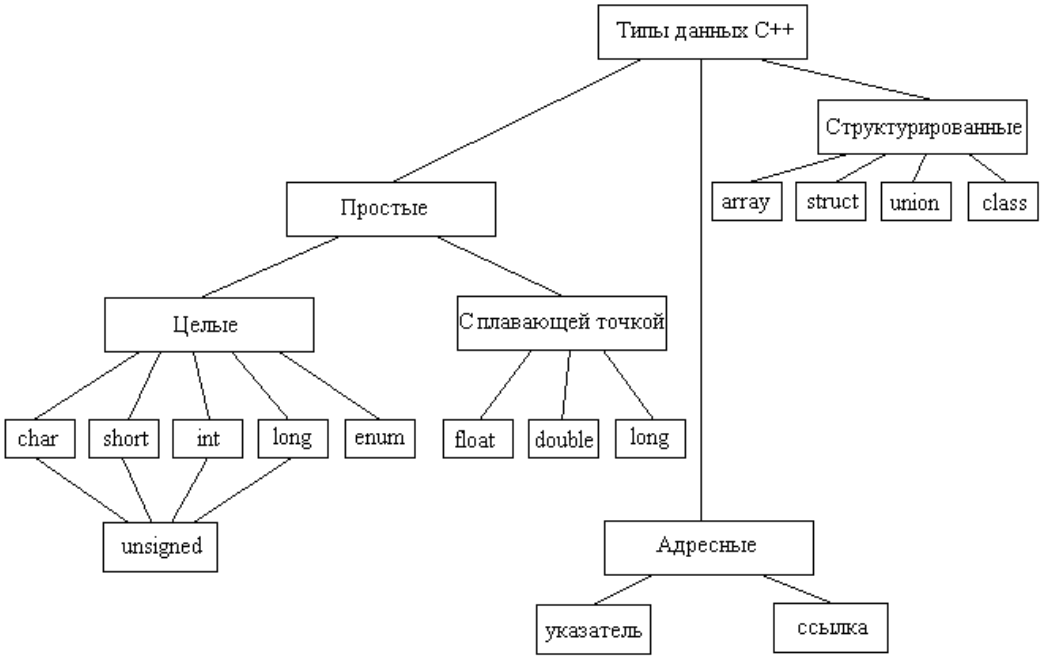


Рисунок 2. Типы данных C++

Объявить переменную означает задать ее имя и тип. Объявление сообщает компилятору, что данный идентификатор связывается с областью памяти, содержимое которого имеет определенный тип.

Синтаксис объявления переменной следующий:

<имя типа> <идентификатор переменной>;

Например,

int k;

Существует возможность объявить сразу несколько переменных одного типа в одном выражении. Для этого имена переменных перечисляются через запятую. Например,

int Number, Count;

float cost1, cost2;

Определяя переменную можно задать также и ее начальное значение, то есть инициализировать переменную. Инициализатор можно записывать в двух формах – со знаком равенства (=)

или в круглых скобках.

Пример:

```
int d, c=5, r=4;
short b(8);
int k, l(145),  m;
unsigned s=0.5;
```

Таким образом можно собрать в один оператор описания переменные одного и того же типа, или наоборот, разбить одно описание на несколько операторов – эффект будет одинаков.

При инициализации символьных переменных их значение требуется заключать в апострофы.

```
char ch='z',  f;
```

Операции и выражения

Знак операции это один или более символов, определяющих действие над операндами. Операции делятся на унарные, бинарные и тернарную по количеству участвующих в них операндов.

Значение переменной можно изменить с помощью операции присваивания =. Например,

```
int summa = 0;
summa = 5;
```

В отличие от алгебраического уравнения в операторе присваивания сначала вычисляется выражение в правой части оператора, а затем оно присваивается отдельной переменной, стоящей слева от знака равенства. Например,

```
int k = 5, m = 1;
m = k;
```

Это означает, что значение переменной m стало равно 5, а не то, что m равно k. Кроме того, выражения m = k и k = m обозначают различные действия. В первом случае обе переменные станут равными пяти, а во втором – единице.

В C++ допускается использование нескольких присваиваний в одном выражении:

```
a = в = c = 0;
```

Основные арифметические операции языка C++ обозначаются стандартными математическими операциями: сложение +, вычитание −, умножение *, деление /. Эти операции, как и операция присваивания, являются бинарными, так как для каждой из них требуется по два аргумента.

Кроме стандартных арифметических операций в C++ введен ряд специальных операций. Из них наиболее часто используемыми являются операция инкремента (увеличение на единицу) ++, и операция декремента (уменьшение на единицу) - - . Эти операции унарные (операции с одной переменной). Они могут использоваться с целым и вещественным аргументом. Использование оператора

```
 j++ ;
```

эквивалентно оператору

```
 j=j+1;
```

А соответственно j- - эквивалентно j = j – 1.

У этих операций существует особенность: они имеют две формы: префиксную, когда ее можно поместить перед переменной и постфиксную – после переменной.

Операции составного присваивания используются для сокращения записи операторов, содержащих в себе присваивание и арифметическую операцию. Оператор вида <операнд1> += <операнд2> эквивалентен записи <операнд1> = <операнд1> + <операнд2>. Существует составное присваивание со сложением, вычитанием, делением, умножением, с остатком от деления: + = ; - = ; * = , / =, % =.

Пример

```
foo + = 3;       // эквивалентно
foo = foo + 3;
```

Операции отношения: больше > , больше или равно > = , меньше < , меньше или равно< = ,

равно = = , не равно! = , не !.

Также определены логические операции: И && и ИЛИ ||. Результатом логической операции является true (истина) или false (ложь). Результатом операции логическое И является значение true, если оба операнда имеют значение true. Результат логического ИЛИ true, если хотя бы один из операндов имеет значение true. Логические операции выполняются слева направо. Например, результат выражения (k>=1)&&(k<10) будет true, если k=5. И это же выражение есть false, если k=0.

Операция определения размера sizeof( ) предназначена для вычисления размера объекта или типа в байтах, и имеет 2 формы: sizeof (выражение) или sizeof(тип).

Операции	***Приоритет операций***
++ – – ! sizeof	наивысший приоритет
* / %	
+ –	
< <= > >=	
= = !=	
&&	
||	
?:	
= += –=	наименьший приоритет

Рисунок 3. Приоритеты операций

Выражения состоят из операндов, знаков операций и скобок и используются для вычисления некоторого значения определенного типа.

Примеры выражений:

(d + 8) / 67

x && y || !z

(t + 5*k) / (f – 56) + 470

Если в одном выражении записано несколько операций одинакового приоритета, унарные операции, условная операция и операция присваивания выполняются *справа налево*, остальные – *слева направо*. Например, a=c=y означает a=(c=y), а a+b+c означает (a+b)+c.

Ввод – вывод на экран. Введение в потоки ввода – вывода

Оператор cout (си-аут) позволяет осуществлять вывод данных на экран монитора. Переменная cout зарезервирована для обозначения выходного потока.

Для того чтобы послать значение на си-аут применяют последовательность cout<< (оператор «направить в» или оператор вставки).

Если необходимо напечатать строку символов, требуется взять ее в кавычки. Можно также напечатать несколько значений одновременно, разделяя их оператором вставки.

cout << “My name is” <<”Tatyana”;

При печати цифр их можно не помещать в кавычки. Возможно объединение текста и цифр.

cout << “ мой адрес: Институтская“ << 26;

В процессе печати можно использовать довольно большое количество специальных символов. Вот некоторые из них:

\n	начало новой строки
\t	табулятор
\b	возврат назад на один пробел
\f	начало новой строки страницы
\\	печать символа обратный слэш
\’	печать символа’
\”	печать символа ’’

Пример:

cout << “ He said:\ ” Hello:\” \ n”;

Оператор вставки использует два аргумента. Аргумент слева от << является потоковым выражением (потоковой переменной). Правый аргумент представляет собой строку или выражение, результат которого имеет простой тип. Оператор вставки преобразует правый операнд в последовательность символов и добавляет их выходной поток. Например,

cout<<”Результат равен ” << 5*n + 90;

Если n равно 10, то на экране появится:

Результат равен 140

Для перехода на новую строку используют манипулятор endl, который также очищает буфер потока. Например,

int x=17, y=21;

cout << “x= ”<<x<<endl<<”y=”<<y;

На экране появится

x=17

y=21

Стандартная библиотека C++ предоставляет пользователю большое количество манипуляторов, позволяющих форматировать ввод-вывод. Манипулятор setw (сокращение от «set width» – «установить ширину») позволяет управлять количеством позиций для вывода следующего за манипулятором элемента данных. Применяется только для форматирования чисел и строк, но не данных типа char. Параметр данного манипулятора – целое выражение, определенное число знаковых позиций для вывода очередного элемента. Данные при выводе выравниваются по правому краю, а свободные позиции слева заполняются пробелами. Например,

int ans=33, num=7132;

cout<<setw(4)<<ans<<setw(5)<<num;

выведет на экран □□33□7132

cout<<setw(1)<<ans<<setw(6)<<num;

33□□7132 - поле автоматически расширяется, чтобы вместить двузначное число.

Установка ширины поля является одноразовым действием и влияет только на ближайший элемент вывода.

Контроль числа десятичных позиций при выводе решается с помощью манипулятора setprecision, указывающего количество знаков после запятой. Например,

int x=4.856;

cout<<setw(6)<<setprecision(2)<<x;

вывод □□□4.85

Для ввода с клавиатуры используют символ cin (си-ин или син) и >> оператор „взять из“

cin >> Number;

Директива #include

Язык C++ содержит очень небольшое число встроенных функций, но он легко расширяется дополнительными библиотеками.

Операторы cin и cout также являются частью библиотеки. Для их использования необходимо включить в программу соответствующие заголовочные файлы. Это делается с помощью команды #include.

Любая команда, начинающая с решетки называется директивой препроцессора. Она не является выражением языка C++ (поэтому не заканчивается точкой с запятой). Директивы препроцессора могут состоять только из одной строки.

Препроцессор – это программа, действующая как фильтр на этапе компиляции. Так препроцессорная директива #include приказывает компилятору загрузить включаемый файл.

Описания операторов cin и cout находятся в файле с именем iostream.h (input output stream – поток ввода-вывода, h - стандартное расширение заголовочного файла (сокращение от header file).

Синтаксис: имя файла помещается в угловые скобки < >, что указывает препроцессору, что этот файл ищется в стандартном каталоге подключаемых файлов:

#include < iostream.h>

Построение программы

При создании алгоритма решения сложной задачи возможно разбиение решения на решение более простых подзадач. Подпрограммы позволяют сначала записать части программы по отдельности, а затем собрать их в единое работоспособное целое. В языке С++ подпрограммы называются *функциями*, а программа представляет собой набор из одной или более функций. Каждая функция выполняет определенное действие, а все вместе они решают задачу в целом.

Итак, любая программа, написанная на языке С++ есть последовательность выполнения функций, причем одна из них обязательно должна называться main( ). Выполнение программы всегда начинается с выполнения ряда операторов этой функции. Когда main () хочет, чтобы другая функция выполнила свое задание, она вызывает (активизирует) необходимую функцию.

Все функции в языке С++ равноправны: каждая из них (даже main( )) может быть любой другой функцией. Функция может вызывать саму себя (явление рекурсии). Компилятор не ограничивает число рекурсивных вызовов, но операционная система может наложить чисто практические ограничения.

Описание функции состоит из заголовка и тела, и в общем случае, выглядит следующим образом:

Директива препроцессора	#include <iostream.h>
Заголовок	<тип результата> имя функции (список аргументов)
Тело функции	{ *пять типов операторов:*
	Описание
	Присваивание
	Функция
	Управление
	Пустой оператор
	}

Предполагается, что любая часть описание функции может не использоваться, кроме имени функции. Пример простейшей программы:

```
#include < iostream.h>
int main ( )
{
 cout << “HeLLo!!! ”;
}
```

Данная программа просто выведет на экран монитора HeLLo!!!

Фигурные скобки отмечают начало и конец тела функции. В круглых скобках в общем случае содержится информация, передаваемая этой функции.

Если тип результата не указывается, то предполагается, что функция возвращает значение типа int. Если функция не возвращает результат, указывается слово void. Некоторые компиляторы требуют обязательного указания типа возвращаемого результата перед именем функции.

Наличие списка аргументов и описаний аргументов также не является обязательным. Но круглые скобки всегда должны присутствовать после имени функции. Аргументы функции – это величины, которые необходимо передать из вызывающей функции в вызываемую функцию.

Часто употребляется описание типов аргументов сразу при их объявлении внутри круглых скобок. Тип аргумента может быть любым. Если аргументов несколько, их описания (тип плюс имя) разделяются запятыми. Если функции не передаются величины, то вместо списка аргументов необходимо задавать ключевое слово void. Локальные переменные отличные от аргументов, описываются внутри тела функции.

Возвращает значение оператор return, с помощью которого можно передать в вызывающую функцию только одно значение. Возвращаемое значение можно брать в круглые скобки после ключевого слова return. Круглые скобки не являются обязательными.

Пример

Напишем программу, содержащую три функции: непосредственно main и функции, вычисляющие значения куба и квадрата некоторого целого числа.

```
#include <iostream.h>
int square (int x)
        { return x*x; }
int cube (int y)
        { return y*y*y; }
  int main ( )
  {     cout<< “Квадрат числа 15 равен”<<square(15)<<endl;
        cout<< “Куб числа 10 равен”<<square(10)<<endl;
  }
```

Любая функции в С++ может возвращать не более одного значения, т.е. либо не возвращать ничего, либо возвращать единственное значение. Аргументов же у функции может быть сколько угодно, в том числе ни одного.

Библиотека математических функций

Библиотека математических функций содержит:

Имя функции	*Параметр*	*Возвращаемое значение*
acos(x)	$-1.0 \le x \le 1.0$	Арккосинус x, в диапазоне от 0.0 до π
asin(x)	$-1.0 \le x \le 1.0$	Арккосинус x, в диапазоне от $-\pi/2$ до $\pi/2$
atan(x)	x	Арктангенс x, в диапазоне от $-\pi/2$ до $\pi/2$
ceil(x)	x	Верхнее значение x (наименьшее целое число ≥ x)
cos(x)	x, в радианах	Тригонометрический косинус x
exp(x)	x	Значение e (2.718…), возведенное в степень x
fabs(x)	x	Модуль x
floor (x)	x	Нижнее значение x (наибольшее целое число ≤ x)
log(x)	$x > 0.0$	Натуральный логарифм от x
log10(x)	$x > 0.0$	Десятичный логарифм от x
pow(x, y)	если x=0, y>0; иначе y - целое	Значение x, возведенное в степень y
sin(x)	x, в радианах	Тригонометрический синус x
sqrt(x)	x>0.0	Корень квадратный из x
tan(x)	x, в радианах	Тангенс

Параметрами и результатами перечисленных математических функций являются переменные типа float.

Использовать библиотечную функцию несложно. Требуется разместить в начале программы директиву #include с указанием требуемого файла заголовков math.h. Затем можно обращаться к функции в любом месте программы.

Пример:

```
#include <iostream.h>
#include <math.h>
int main ( )
{       float alpfa, beta;
        alpfa = sqrt(169.41 + fabs( pow( beta, 5));
        cout<< alpfa; }
```

МЕТОДЫ СТРУКТУРНОГО ПРОГРАММИРОВАНИЯ

Блоки или составные выражения

Блок является последовательностью из нуля или более операторов. Эта последовательность

заключена в фигурные скобки. После окончания блока точка с запятой не ставится.

Блок можно использовать везде, где разрешено использование отдельного оператора.

Блоки часто используются в программах, особенно как составная часть других выражений. Пропуск пары фигурных скобок может кардинально поменять смысл алгоритма программы, повлиять на ее выполнение и полученный результат.

Базовые конструкции структурного программирования

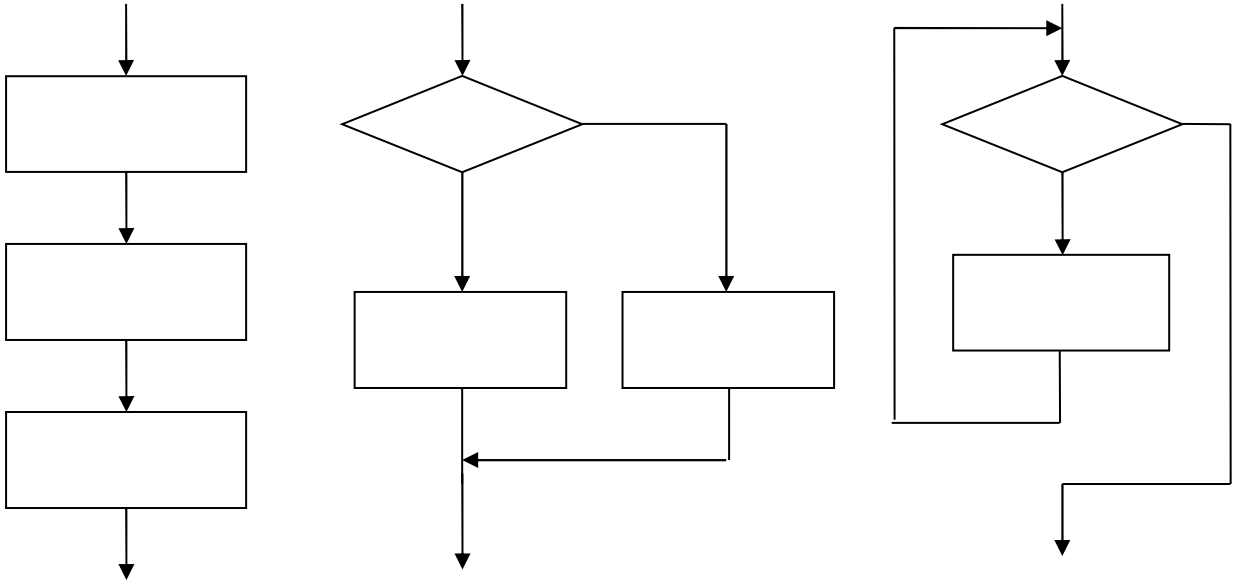
Программу для решения задачи любой сложности можно составить только из трех структур, называемых следованием, ветвлением и циклом.

Следование, ветвление и цикл называют *базовыми конструкциями* структурного программирования.

Следование представляет собой последовательное выполнение нескольких операторов (простых или составных).

Ветвление задает выполнение одного из операторов в зависимости от результата выполнения какого-либо выражения, задающего условие.

Цикл задает многократное выполнение одного или последовательности нескольких операторов.



Следование Ветвление Цикл

Рисунок 4. Базовые конструкции структурного программирования

Любая базовая конструкция содержит только один вход и один выход. Конструкции могут вкладываться друг в друга произвольным образом, тем самым, образуя структуру программы. Например, цикл может содержать следование двух ветвлений, каждое из которых содержит последовательное выполнение циклов.

В большинстве языков высокого уровня существует несколько реализаций базовых конструкций. Язык C++ содержит три вида циклов и два вида ветвлений. Их выбор осуществляется из требований алгоритма и его эффективности.

Операторы ветвления

Условный оператор if

Условный оператор if позволяет разветвлять вычислительный процесс на два направления.

Сначала вычисляется выражение, стоящее в условие. Если оно не равно нулю или имеет значение истина, выполняется первый оператор, иначе второй. После этого управление передается

следующему оператору. Таким образом, с помощью оператора if можно в ходе выполнения программы задать некоторый вопрос и в зависимости от ответа (да или нет) выполнить те или иные действия.

Синтаксис оператора if:

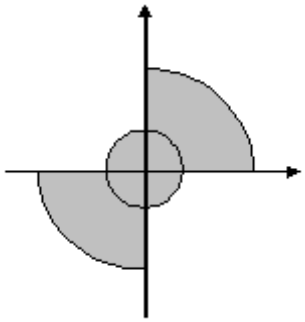
if (<выражение>) <оператор1>; [else <оператор 2>;]

Ветвь с ключевым словом else не является обязательной. Поэтому она взята в квадратные скобки. (В дальнейшем изложении для обозначения необязательных элементов будут всегда использоваться квадратные скобки.)

Примеры:

```
if ( fvalue>=0.0 )  fvalue = fvalue;  else fvalue = -fvalue;   // вычисляется модуль числа
if ( x<10 ) x+ =10; else x *=2;
if ( f != 0 )  c = 100 / f;
```

Пример программы. Производится выстрел по мишени, изображенной на рисунке. Радиус внутреннего круга равен единице, а внешнего – тройке. Попадание в меньший круг дает 10 очков, в сегменты большого – 5 очков. Определить количество очков, набранного выстрелом, координаты которого вводятся с клавиатуры.



```
#include <iostream.h>
int  main ( )
{
float x, y;
 int score;
 cout << “введите координаты выстрела”<<endl;
 cin >>x>>y;
 if  ( x*x + y*y < 1 ) score = 10;
        else     if  ( x*x + y*y < 9 &&  x*y>0 ) score = 5;
                 else kol = 0;
 cout << “Вы набрали” << kol << “  очков!!!”;
}
```

Оператор выбора switch

Оператор switch (переключатель) предназначен для разветвления процесса вычислений на несколько направлений.

Синтаксис оператора switch:

```
switch ( выражение) {
case <константное выражение 1> : <операторы1>; [ break; ]
case <константное выражение 2> : <операторы2>; [ break; ]
…
case <константное выражение n> : <операторы n>; [ break; ]
[default : операторы];
}
```

Выполнение оператора начинается с вычисления выражения, которое должно быть целочисленным. Затем управление передается первому оператору из списка, помеченному константным выражением, значение которого совпало с вычисленным. После этого, если выход из переключателя явно не указан (отсутствует break), последовательно выполняются все нижележащие ветви. Выход из переключателя обычно выполняется с помощью оператора break или return. Оператор break выполняет выход из самого внутреннего из объемлющих его операторов. А оператор return выполняет выход из функции, в которой он описан.

Все константные выражения, расположенные после case, должны быть различны. Если совпадения ни с одним оператором не произошло, выполняются операторы, расположенные после ключевого слова default.

Ветвь default может отсутствовать. В этом случае выполнение программы передается сле-

дующему за switch оператору.

Несколько меток могут следовать подряд, предполагая выполнение одинаковых действий.

```
#include <iostream.h>
int main ( )
{ int x;
 cout << “Введите номер месяца” << endl;
 cin >> x;
 switch (x ) {
        case 1 :        case 2 :        case 12 : cout << “Зима”<< endl;          break;
        case 3 :        case 4 :        case 5   : cout << “Весна”<< endl;        break;
        case 6 :        case 7 :        case 8   : cout << “Лето”<< endl;         break;
        case 9 :        case 10 :       case 11 : cout << “Осень”<< endl;         break;
        default : cout << “Неверный номер” << endl;       }
return 0;
}
```

Инструкции перехода

В языке C++ имеется четыре инструкции перехода: goto, break, continue, return.

Оператор безусловного перехода goto имеет формат:

goto <метка>;

Тогда в теле той же функции должна присутствовать ровно одна конструкция вида:

<метка> : оператор;

Метка является обычным идентификатором. Оператор goto передает управление на помеченный оператор.

Наличие конструкции безусловного перехода рассматривается как плохой стиль программирования, и считается, что в четко структурированной и грамотно написанной программе этой конструкции быть не должно.

Инструкция break используется внутри циклов или переключателей и позволяет переход в точку программы, находящуюся непосредственно за оператором, внутри которого находится. Таким образом, break обеспечивает вывод из цикла еще до того, как условие цикла станет ложным. Управление передается первой строке, следующей за телом оператора.

Инструкция continue заставляет программу пропустить все оставшиеся строки цикла, но сам цикл при этом не завершается. Для решения некоторых задач удобно комбинировать инструкции break и continue.

Иногда необходимо прервать выполнение программы задолго до того, как будут выполнены все ее строки. Для этого используют return, которая завершает выполнение той функции, в которой она была вызвана. Если же вызов произошел в функции main( ), то завершается сама программа.

Операторы цикла

Операторы цикла используются для организации многократно повторяющихся вычислений. Любой цикл состоит из тела, т. е. операторов, которые повторяются несколько раз, начальных установок, модификации параметра цикла и проверки условия продолжения выполнения цикла.

Один повтор выполнения операторов тела цикла называется *итерацией*. Проверка условия выполнения цикла производится на каждой итерации.

Параметрами цикла называются переменные, изменяющиеся в теле цикла и используемые при проверке условия продолжения цикла, называются параметрами цикла.

Начальные установки могут явно не присутствовать в программе, их смысл состоит в том, чтобы до входа в цикл задать значения переменным, которые в нем используются.

Цикл завершается, если условие его выполнения не выполняется. Возможно принудительное завершение, как текущей итерации, так и тела цикла. Для этого используются конструкции перехода.

Для удобства в С++ существуют три разных оператора цикла – while, do while, for.

Цикл с предусловием (while)

В цикле с предусловием проверка условия продолжения цикла выполняется перед телом цикла (рисунок 5).

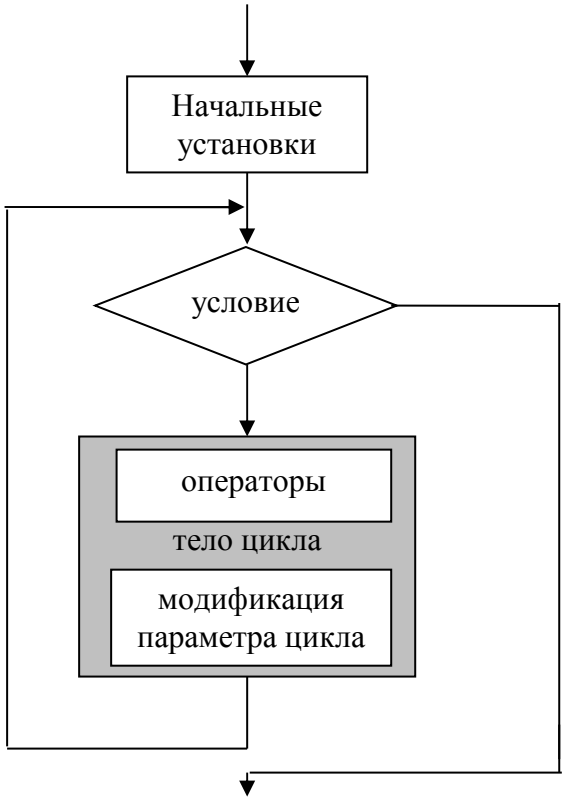


Рисунок 5. Цикл с предусловием

Цикл с предусловием реализован в С++ оператором цикла while.

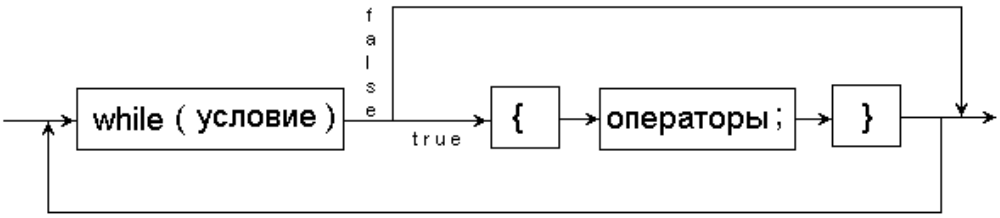


Рисунок 6. Структурная схема оператора цикла while

Выражение, стоящее в круглых скобках, определяет условие повторения тела цикла, представленного простым или составным оператором. Если оператор простой операторные скобки { } не ставятся.

Выполнение оператора цикла начинается с вычисления выражения, стоящего в условии. Если оно истинно, выполняется тело цикла. Если при первой проверке выражение ложно (false), цикл не выполнится ни разу.

Цикл while обычно используется в тех случаях, когда число повторений заранее неизвестно.

Цикл с постусловием (do while)

Тело цикла с постусловием всегда выполняется хотя бы один раз, после чего проверяется, надо ли его выполнять еще раз.

Цикл с постусловием реализован в С++ оператором цикла do … while и имеет вид:

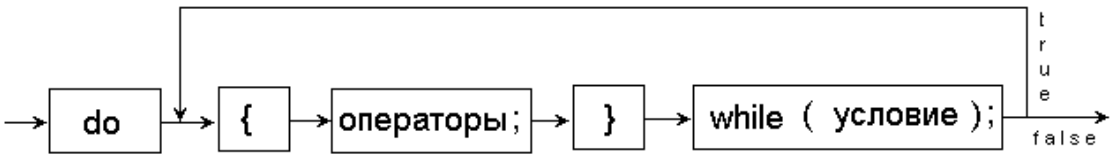


Рисунок 7. Структурная схема оператора цикла do while

Сначала выполняется простой или составной оператор, составляющий тело цикла, а затем вычисляется выражение, составляющее условие выполнение цикла. Даже если условие заведомо ложно, цикл выполнится один раз. Если условие истинно, тело цикло выполнится еще раз. Цикл завершается, когда выражение станет равным false, или в теле цикла будет выполнен оператор передачи управления.

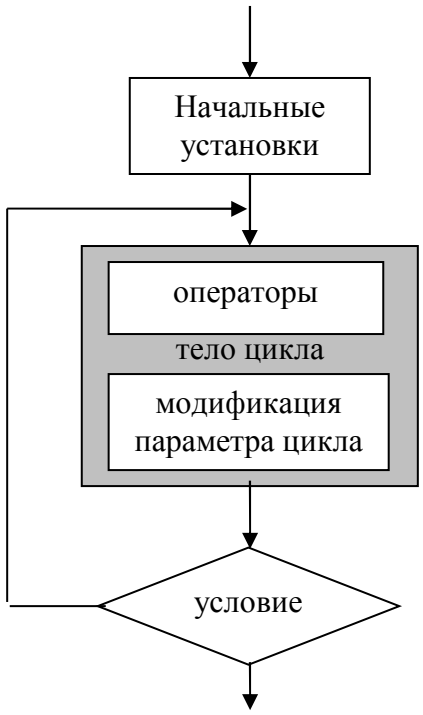


Рисунок 8. Цикл с постусловием

Цикл с параметром (for)

Цикл for называют также циклом с заданным числом повторений. Он имеет следующий формат:

for (инициализация; выражение (условие) ; модификации) оператор;

Инициализация используется для объявления и присвоения начальных значений величинам, используемым в цикле. Инициализация выполняется один раз перед выполнением тела цикла.

Цикл с параметром реализуется как цикл предусловием. Выражение определяет условие выполнения цикла: если его результат равен истине, то цикл выполняется. Модификации выполняются после каждой итерации цикла и служат обычно для изменения параметра цикла.

Тело цикла представляет собой простой или составной оператор.

Любое из трех выражений, указанных в скобках, является необязательным. Точки с запятой должны всегда оставаться на своих местах, даже в случае, если все три выражения отсутствуют.

Примеры

```
for ( ; ; );        // пример бесконечного цикла
for (y=2; y<20; y++)  r+=y;
```

В инициализации и модификации параметра можно писать несколько операторов, разделенных запятой.

for (i = 1, s=1; i<11; i++) s*=i; // вычисления факториала 10

Цикл for обычно используется в тех случаях, когда можно точно определить необходимое число повторов.

Допускается объявление переменной прямо в строке инициализации цикла for. Значение счетчика цикла может не только увеличиваться, но и уменьшаться, причем на произвольный шаг, который может быть не только целым, но и числом с плавающей точкой.

for (float k = 11.5; k>0.5; k - = 0. 5) s+=k;

В качестве параметра цикла можно использовать и символьную переменную.

for (char ch='a'; ch< 'z'; ch++)

Массивы

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой переменных одного типа требуется выполнить различные действия, им дают одно имя и отличают по порядковому номеру. Это позволяет записывать множество действий и операций над этими элементами через циклы.

Конечная именованная последовательность однотипных величин называется массивом.

Для создания массива компилятору необходимо знать тип данных, количество элементов в массиве и требуемый класс памяти. Массивы могут иметь те же типы данных, что и простые переменные.

Синтаксис объявления массива:

<тип данных> <имя массива> [размерность массива];

Примеры:

int array[45]; //массив из 45 элементов целого типа с именем array

double rt[12]; // массив rt, состоящий из 12 элементов типа double

const int ARRAY_SIZE=90;

float alp[ARRAY_SIZE]; // вещественный массив alp из 90 элементов

Количество элементов должно соответствовать размеру массива. Если их окажется меньше, то недостающие элементы будут инициализированы нулями (или мусором, хранящимся в памяти). Если же элементов окажется больше – компилятор выдаст ошибку.

При инициализации допускается использование пустых скобок в объявлении массива, и тогда компилятор сам определяет размерность массива. Например,

float x[ ]= {4, 8, 19}; // размерность x равна 3

Для доступа к элементу массива после его имени указывается номер элемента (индекс), заключенный в квадратные скобки. Нумерация элементов массива начинается с нуля. Следовательно, диапазон значений для объявленного в примере массива x лежит в пределах от 0 до 2. Следовательно,

x[0]=4 x[1]=8 x[2]=19

В памяти компьютера элементы массива располагаются последовательно друг за другом. Место в памяти для хранения элементов массива выделяется компилятором сразу после обработки его объявления.

Так объявление вида

float y[5]= {0.7, 1.9, 8.4, 3.1 };

выделит место для расположения пяти элементов типа float. Так как выполнена инициализация четырех первых элементов, они сразу займут свое место в памяти компьютера, а ячейки памяти для пятого элемента пока останутся свободными:

0. 7	1. 9	8. 4	3. 1	
y[0]	y[1]	y[2]	y[3]	y[4]

В C++ не осуществляется проверка значений индексов на выход за пределы массива, ни в процессе компиляции, ни в процессе выполнения программы. Так если, в тексте программы напишем выражение

y[5]=2;

компьютер сохранит значение 2 в ячейке памяти, следующей за ячейкой, соответствующей последнему элементу массива. При этом старое значение данной ячейки будет затерто. Поэтому проверка значения индекса – обязанность программиста.

К элементам массива можно применять все операции, допустимые для обычной переменной типа, соответствующего типу элементов массива.

Пример: Программа подсчета суммы элементов массива.

```
#include<iostream.h>
int  main ( ) {
  const int n=10;
  int i, m[n], s=0;
  cout << " введите" << n << "чисел" <<endl;
                    // ввод элементов массива с клавиатуры
  for ( i=0; i < n; i++) cin >> m[ i];
  for ( i=0; i < n; i++) s+=m[ i ] ;
  cout << "сумма введенных элементов равна "<<s<<endl;
}
```

Для улучшения эффективности алгоритма ввод элементов массива и подсчет их суммы можно объединить в один цикл:

```
  for ( i=0; i<10; i++) { s+=m[i];    cin >> m[i]; }
```

Многомерные массивы

Для объявления многомерного массива необходимо после его имени задать несколько размеров, заключенных в квадратные скобки. Размерность массивов в С++ не ограничивается. Она может зависеть от возможностей компьютера и особенностей конкретного компилятора. Как правило, на практике используются двумерные массивы.

Двумерный массив в С++ представляет собой массив одномерных массивов.

```
float dam[4] [5];
```

В этом случае массив будет содержать 4 строки и 5 столбцов. Инициализация двумерного массива происходит следующим образом:

```
float art [5] [2]={  { 1. 2, 1. 5 },
                     { -4. 0, 3. 6 },
                     { 2. 3,  -6.1 },
                     { 7. 3,  0. 4 },
                     { 0. 0, -2. 7 }  };
```

При этом внутренние фигурные скобки могут быть опущены. Например,

```
float art [5][2] = {1. 2, 1. 5, -4. 0, 3. 6, 2. 3, -6. 1, 7. 3, 0. 4, 0. 0, -2. 7} ;
```

Все сказанное может быть распространено на трехмерный массив.

```
int rum [3][7][2]; // массив размерности три, каждый элемент которого двумерный массив.
```

Если одномерный массив используется для представления списка, то двумерный массив – для представления таблицы, содержащей строки и столбцы. При этом подразумевается, что все элементы принадлежат одному типу данных. При обращении к отдельному элементу двумерного массива нужно указать его позицию в строке и столбце. Нумерация строк и столбцов начинается с нуля. Массив из примера можно представить как таблицу. Выделенный на рисунке элемент есть art [1][2].

	[0]	[1]
столбцы		
[0]	1. 2	1. 5
[1]	-4. 0	3.6
[2]	2. 3	- 6. 1
[3]	7. 3	0. 4
[4]	0. 0	- 2. 7

Символьные строки

Символьная строка – последовательность одного и более символов, заключенная в двойные кавычки. Для формирования символьных строк, занимающих несколько строк программы, используется комбинация символов \ и \n. Кавычки не являются частью строки, они служат для обозначения ее начала и конца.

Строки представляются в виде массива элементов типа char. Число элементов символьного массива должно быть на единицу больше числа символов, которые предстоит хранить в строке. Дополнительная ячейка памяти требуется для размещения нуль-символа (/0), который автоматически добавляется в качестве последнего байта в памяти для обозначения окончания строки.

Как и любую другую переменную строку можно инициализировать при объявлении. При этом размерность можно не указывать, компилятор определит ее самостоятельно. Инициализировать строку можно всю целиком или посимвольно:

```
char message [10]= {‘с’, ‘о’, ‘о’, ‘б’, ‘щ’, ’е’, ‘н’, ‘и’, ‘е’};
char message [ ]=  “сообщение”;
```

Для ввода строк лучше использовать функцию cin.get ( ), которая имеет два параметра: строковую переменную и целое типа int, отвечающее за количество вводимых символов в строке плюс один (для нуль-символа).

Например:

```
char line[51];
cin.get(line, 51);
```

В этом случае непечатные символы не пропускаются. Функция считывает и сохраняет полностью всю строку ввода (длиной не более чем указано вторым параметром). Чтобы правильно считать две последовательные строки, нужно не забывать о символе передачи строки:

```
char dummy=‘/n’;
cin.get(line1, 51);
cin.get(dummy);        //убрать ‘/n’ из потока ввода перед использованием get
cin.get(line2, 51);
```

Язык С++ предлагает большой ассортимент полезных функций для работы со строками. Прототипы всех функций работы со строками содержатся в файле string.h. Рассмотрим несколько из них.

Функция strlen() вычисляет длину строки без нуль-символа. Функция имеет один аргумент – имя строки.

```
#include <string.h>
#include <iostream.h>
int  main ( )
{
char str[ ]= ”Hello, my friend!”;
cout<<strlen(str);                  // результат 17
}
```

Функция strcmp( ) сравнивает две строки. Она имеет два аргумента и возвращает целое значение, которое равно 0, если строки полностью совпадают.

Значение strcmp(str1, str2) является целым числом меньше нуля, если str1<str2 и целое >0, если str1>str2. Сравнение строк происходит в лексикографическом порядке, т.е. в порядке их расположения в словаре.

Функция strcpy( ) копирует содержимое второй, из указанных в качестве параметра, строки в первую, замещая прежние данные, включая ‘/0’. При этом торая строка не изменяется. Первая из указанных строк должна иметь достаточный размер, иначе копирование не выполняется.

Функция strcat( ) присоединяет строку, указанную в качестве второго параметра, к первой указанной строке.

Работа с символьным массивом аналогична работе с числовым массивом.

Пример. Определение количества вхождений каждого символа в заданную строку.

```
#include <iostream.h>
#include <string.h>
int  main ( )
{ int k;
  char str[100];
  cout<<“Введите любую последовательность символов без пробелов”;
 cin>> str;
  for (char ch=’a’; ch<=’z’; ch++)
        { k=0;
          for ( int i=0; i < strlen(str); i++)
           if (str [ i ] ) == ch) k++;
          if (!k)   cout<<”Количество символа” << ch << “равно”<< k<<endl;
        }
```

Массив символьных строк выглядит как двумерный массив. К его строкам применимы все функции для работы со строками. Можно работать и непосредственно с отдельными его элементами. Так инициализировать символьный массив, содержащий четыре строки можно следующим образом:

static char fruit[4][ ]={“Слива”, “Персик”, “Яблоко”, “Апельсин”};

Теперь компилятор автоматически определит количество элементов в строке – девять, т.к. самое длинное слово содержит восемь символов плюс один для размещения нуль-символа.

Методы сортировки

При разработке программного обеспечения очень распространенной операцией является сортировка значений, т.е. расположение списка элементов в некотором порядке (например, слова по алфавиту или числа – в возрастающем или убывающем порядке).

Существует множество алгоритмов сортировки элементов. Наиболее простой из них – метод «пузырька». Алгоритм реализуется в виде двух вложенных циклов. Во внутреннем цикле просматриваются по порядку все элементы массива, попарно сравниваются рядом стоящие элементы, и если второй больше первого (при сортировке по убыванию) элементы меняются местами. Параметром внутреннего цикла является индекс(номер) элемента массива. Для полной сортировки такая перестановка должна быть выполнена n-1 раз, где n – количество элементов в массиве. Для этого организуется внешний цикл, параметром которого является шаг сортировки.

Пример. Отсортировать элементы одномерного массива целых чисел по возрастанию. На рисунке 9 приведена последовательность перестановки элементов.

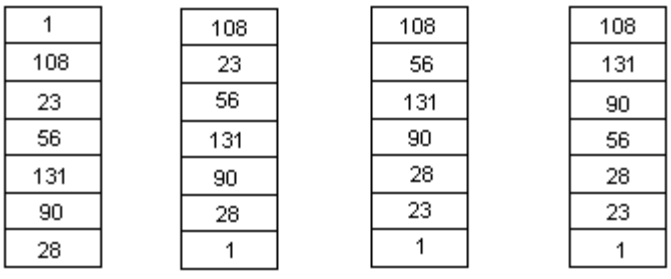


Рисунок 9. Сортировка методом «пузырька»

Хотя в предложенном примере сортировка выполнилась за четыре шага, проверка элементов будет продолжаться еще две итерации, т.к. полное число итераций на единицу меньше размерности массива. На рисунке 10 приведена блок-схема рассмотренного алгоритма.

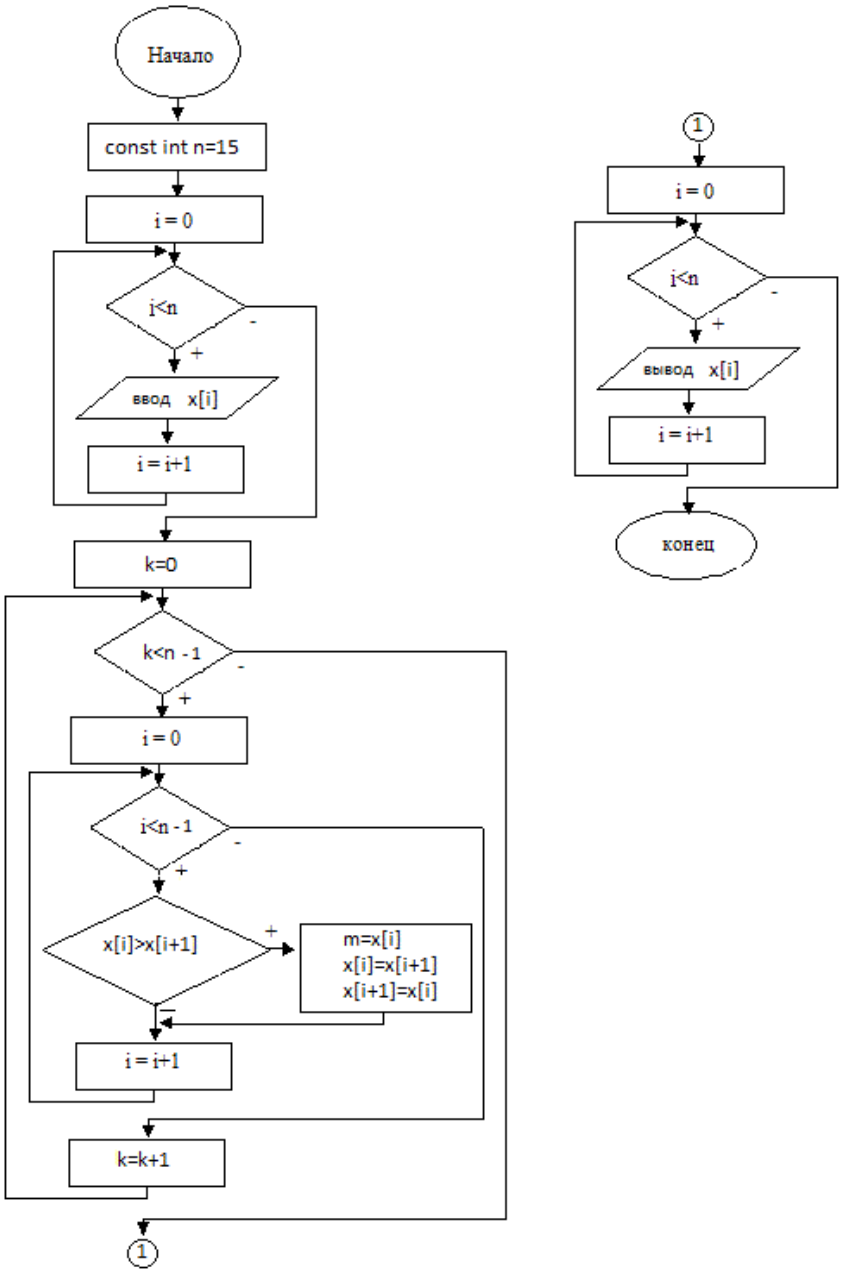


Рисунок 10. Блок-схема алгоритма сортировки методом «пузырька»

Сортировка пузырьковым методом является неэффективным методом, вследствие большого числа сравнений.

Структуры

Структуры позволяют программисту определять новые типы данных путем логического группирования переменных различных типов. В отличие от массива, все элементы которого имеют один тип, структура может содержать элементы различных типов.

Элементы структуры называются полями. Описание полей и их типов содержится в структурном шаблоне. Описание структурного шаблона выглядит следующим образом:

struct имя структуры {

тип1 идентификатор1;
тип2 идентификатор2;
тип3 идентификатор3; };

Например, описание структурного шаблона для описания книги:

struct book{

```
        char title[40];
        char author[30];
        int page, year;
    };
```

Описание структурного шаблона создает новый тип. После его определения можно смело объявлять переменные вновь созданного типа. Правила объявления структурной переменной аналогичны правилам объявления обычной переменной, т.е. указывается имя типа, за которым следуют имена переменных, разделяемые запятыми.

Например,

```
book library;          // описание структурной переменной libry типа book
book lib1, lib2,*ptb;  //описание двух структурных переменных и указателя
book library[100];     //объявлен массив из 10 элементов типа book
```

Теперь каждая из объявленных переменных имеет поля title, author и value.

Для обращения к полю используется операция выбора, обозначаемая точкой. Например,

```
library.page=486;
strcpy (library.author, "Пушкин");
```

Можно совмещать в одном объявлении определение структурного шаблона и структурной переменной.

```
struct complex {
        int Re, Im;
    } z1, z2;
```

Объявлены переменные z1 и z2 типа complex.

Имя структурного шаблона в можно в этом случае не указывать.

Как и любую другую переменную, структурную переменную можно инициализировать при объявлении. Тогда значения ее полей перечисляются в фигурных скобках в порядке их описания в шаблоне, через запятую.

```
struct book libry = {"Просто и ясно о Borland C++", "Бруно Бабэ", 400, 2006};
```

При инициализации массива структур следует заключать в фигурные скобки каждый элемент массива.

Для переменных одного и того же структурного типа определена операция присваивания. При ее использовании происходит поэлементное копирование.

Пример программы.

Создать структурный шаблон, хранящий сведения о некотором человеке: фамилию, пол, возраст. Создать массив, содержащий сведения о 15 людях. Выбрать из списка фамилии мужчин в возрасте от 20 до 35 лет, причем их фамилия должна начинаться с введенной с клавиатуры буквы.

```
#include <iostream.h>
int main( )
{const int n=15;
 struct man {          //описание структурного шаблона
                       char family[20];
                       char sex;
                       int age;
            };
 man people [n];                    //массив структурных переменных
 int i;  char symbol;
 for ( i=0; i < n; i+ + )
        {cout<< "введите фамилию: " ;
        cin>> peole[i].family;
        cout<< "введите возраст: ";
        cin>> peole[i].age;
        cout<< "введите пол: м – мужской или ж – женский: ";
        cin>> peole[i].sex;
```

```
        }
cout<< " введите первую букву фамилии: ";
cin << symbol;
for ( i=0; i < n; i++)
        { if  ( peole[i].family[0]= = symbol)
                if (peole[i].sex = = 'м')
                        if ( peole[i].age > 19)&&( peole[i].age < 36 )
                                cout<< peole[i].family <<endl;
        }
}
```

Текстовые и бинарные файлы

Во всех примерах рассмотренных ранее предполагалось, что ввод в программу осуществляется с клавиатуры, а вывод направляется на экран монитора.

В случаях использования большого объема вводимых и выводимых данных используют файлы.

Файлом называется именованная область внешней памяти, в которой содержится некоторая информация. Хранение информации в файле позволяет не вводить заново данные при повторном запуске программы, а также просматривать данные сколько угодно раз. Содержимое файла может быть просто просмотрено на экране или напечатано на принтере.

Язык С++ позволяет работать с файлами несколькими способами.

Первый из предлагаемых способов применялся в языке С и поддерживается в С++. Язык С выделяет 2 вида файлов: текстовые и двоичные (бинарные). Текстовым считается файл, в котором информация запоминается в виде символов кода ASCII (или аналогичном). Он отличается от бинарного файла, который обычно используется для запоминания кодов машинного языка. Таким образом, содержимое текстового файла можно просмотреть и изменить в любом текстовом редакторе. Для чтения бинарного файла необходима специальная программа.

Для работы с любым видом файла описываем указатель на переменную типа file (иногда используют термин файловая переменная):

```
file *in;
```

Открытие файла происходит с помощью функции fopen( ). Данной функцией управляют три основных параметра.

Имя файла, который следует открыть, является первым аргументом.

Второй аргумент, указывающий режим использования файла, может принимать три значения: "r" – файл используется для чтения, "a" – для дополнения, "w" – для записи (при применении "r" используется существующий файл, для двух других, если файл не существует, то он будет создан; если используется "w" для уже существующего файла, старая версия файла затирается);

Третий параметр является указателем на файл, и это значение возвращается функцией.

```
file *in;
in = fopen( "test", "r" );
```

Теперь in является указателем на файл, хранящийся на диске под именем test. С этого момента программа будет работать с файлом через in, а не по имени test. Если fopen( ) не способна открыть требуемый файл, то она возвращает значение 'NULL' (определенное в файле stdio.h как 0).

Рекомендуется использовать проверять существование файл перед его открытием, например, строкой вида

```
 if ( (in = fopen( "test", "r" ) ) != NULL )
```

Если файл создать невозможно, то указателю in присваивается значение NULL, и условие становится ложным. В этом случае нужно вывести на экран предупреждающее сообщение и завершить программу.

Закрыть файл проще. Для этого используют функцию fclose( ), которая имеет только один аргумент – указатель на файл.

fclose(in);

Ввод текстового файла осуществляется с помощью функций getc( ) или getch( ), работающих с одним символом. Различие этих функций в том, что первая работает с целым типом, а вторая – с типом char. Функции putc( ) и putch( ), соответственно записывают один символ. Прототипы эти функций содержатся в заголовочном файле stdio.h.

```
//Программа печатает содержимое файла на экран и определяет количество точек
#include < stdio.h >
int main( )
{
file *in;          // описание указателя на файл
char ch, point = ‘ . ’;
int count=0;
if ( ( in = fopen( “test”, “rt” )) != NULL )
        {  while ( (ch = getch(in) ) != EOF)  // получает символ из in
              { putch(ch, stdout);        //посылает на stdout (стандартный вывод - экран)
                if (ch = = point) count++;
              }
        fclose(in);
        }
else printf(“файл не открывается /n“);
}
```

При работе с файлом используются понятия начало и конец файла. Когда файл открывается, система помещает файловый указатель на начало файла – его первый элемент. При чтении информации в систему передается первый элемент, и происходит сдвиг указателя на следующий элемент. Таким образом, работа с файлом походит на работу с массивом, размер которого заранее не известен. Указатель перемещается до тех пор, пока не будет достигнут конец файла. Конец файла помечается константой EOF (End of File), которая автоматически формируется при закрытии файла, после записи в него информации. Данная константа возвращается функцией feof( ) в случае достижения конца файла. Ее параметром служит указатель на файл. Следовательно, задание цикла чтения до конца файла может быть записано как:

while (!feof(in)) { … }

Для указания, что файл открыт в двоичном режиме необходимо добавить букву b во второй аргумент функции fopen() (для текстового добавляется t , но это не является обязательным).

Для работы с бинарными файлами существует ряд функций: fread (buffer, size, count,stream) содержит 4 аргумента.Данная функция читает count элементов длины size из входного потока (файла) stream (FILE * stream) и помещает в заданный массив buffer. При этом значение указателя файла увеличивается на число действительно прочитанных байтов.

fwrite ( ) позволяет записывать в файл и имеет такие же аргументы. Функция дописывает count записей по size байтов каждый из области buffer в выходной поток stream.

fseek (stream, offset, origin) перемещает внутренний указатель файла, связанный с потоком stream, на новое место в файле, которое вычисляется по смещению offset и указанию направления отсчета origin. После использования fseek( ) следующая операция ввода/вывода с указанным потоком stream будет выполнена, начиная с той позиции, на которую произведено перемещение. Аргумент offset должен быть типа long, a origin может принимать значение одной из следующих целочисленных констант:

SEEK_SET (значение 0) – начало файла,

SEEK_CUR (значение 1) – текущая позиция указателя файла,

SEEK_END (значение 2) – конец файла.

Функция fseek( ) возвращает целое значение.

Функция позволяет получить текущую позицию указателя файла, связанного с потоком stream. Позиция задается как смещение относительно начала файла.

Запись и чтения файла осуществляется поэлементно. Элементом файла может быть символ,

или целое число, или даже структурная переменная.

Если текстовый файл можно создать с помощью текстового редактора, например WordPad или среды Borland C++, то бинарный файл нужно создавать программно. Так для хранения координат точек на плоскости можно создать файл следующим образом:

```
#include<stdio.h>
#include<stdlib.h>
int  main ( )
{
struct point                //шаблон и переменная для хранения координаты
{       int x, y;
        }z;
FILE *f;
f = fopen( “point .bbb”, ”wb”);
randomize( );
int k=1;
while ( !k )     { printf (“Задать координаты точки? Если да - 1, иначе 0 \n”);
                   z. x= random(100) – 50;
                   z. y = random(100) – 50;
                   fwrite (&z, sizeof(point), 1, f);
        }
fclose(f);
}
```

Таким образом, на диске в текущей директории создан файл с именем point.bbb, и теперь обрабатывать хранящиеся в нем данные. Требуется отметить, что в отличие от массива при работе с файлом нужно помнить, что количество элементов файла неизвестно. Например, в заданном файле определим координат точек, принадлежащих прямой, заданной уравнением $y = kx + b$. Значения констант k, b задаются с клавиатуры.

```
#include <stdio.h>
int  main ( )
{
struct point
{       int x, y;
        }z;
int k, b, count=0;
printf (“Введите параметры уравнения”);
scanf ( “%d   %d  ”, &k, &b);
FILE *f;
f = fopen( “point .bbb” ,”rb” );
while ( !feof(f) ) {
                    fread (&z, sizeof(point), 1, f);
                    if ( z.y = = k*z.x + b )   { count++;
                                                 printf (“точка (%d, %d) лежит на прямой, z.x, z.y);
                                                }
                  }
fclose(f);
printf ( “количество найденных точек  %d, count);
}
```

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

В основе объектно-ориентированного программирования (ООП) лежит идея разбиения задачи на группу объектов. Объекты содержат в себе данные и подпрограммы для их обработки. Данные и стандартные подпрограммы, обрабатывающие данные, объединяются в один блок

(тип), называемый *классом*.

Три ключевых принципа определяют объектно-ориентированное программирование:

- инкапсуляция;
- наследование;
- полиморфизм.

Инкапсуляция (сокрытие информации) – определение пользователем новых типов данных. Каждый такой тип содержит определение набора значений и операций, которые могут быть выполнены над этими значениями, и образует так называемый абстрактный тип данных. При этом никакие другие функции, кроме набора определенных в классе операций, не должны иметь доступа к значениям этого типа. В языке С++ инкапсуляция данных реализуется с помощью механизма классов. Переменные класса являются объектами, из которых строится программа.

Наследование – механизм, позволяющий строить иерархию типов. Это предполагает определение базового типа (или прародителя), а затем использование этого типа для построения производных типов (потомков). Причем каждый производный тип наследует все свойства базового типа, включая как данные, так и набор операций (функции). Новые типы данных могут также иметь свои дополнительные свойства. Таким образом, они не дублируют свойства базового класса, а только имеют возможность их использовать.

Полиморфизм в переводе с греческого означает «много форм», заключается в обозначении общего действия одним именем (функцией), которое используется во всей иерархии типов. Каждый тип в этой иерархии реализует это действие своим собственным, пригодным для него способом. В языке С++ полиморфизм имеет две формы:

- перегрузка операций и функций,
- использование виртуальных функций.

Понятие «класс»

Класс является типом данных, определяемым пользователем. Он представляет собой модель реального объекта в виде данных и функций для работы с ними. Класс есть расширение понятие структуры языка С++. Каждый представитель класса называется объектом.

Объединение данных и функций, которые обрабатывают объект определенного типа, в одном описании называется инкапсуляцией.

Данные класса называются полями или данными-членами класса. Функции класса – методами или функциями-членами класса. Поля и методы называются элементами класса. Описание класса выглядит примерно так:

```
class <имя> {
 private:
< описание скрытых элементов>
 public:
< описание доступных элементов>
 protected:
< описание защищенных элементов>
};
```

Описанию элементов предшествуют ключевые слова, являющиеся спецификаторами доступа: private (закрытый), public(открытый) и protected (защищенный).

Открытые элементы (public) доступны снаружи класса. Они, как правило, отвечают за внешний интерфейс класса и являются методами класса.

Закрытые элементы (private) предназначены только для внутреннего использования в классе. Как правило, они являются полями. Закрытые элементы доступны только методам класса, в состав которого они входят.

Защищенные элементы (protected**)** доступны для методов данного класса и классов, производных от него.

При создании класса программист самостоятельно решает, какие из членов класса будут общедоступными, а какие закрытыми. Однако большинство классов имеет типичную схему: за-

крытая часть содержит данные (поля), а открытая – функции для работы с этими данными (методы).

В классе могут присутствовать многочисленные открытые и закрытые секции сколько угодно раз и в производном порядке.

Спецификатор доступа не является обязательным, и если он не присутствует, по умолчанию элементы класса становятся закрытыми.

Рассмотрим класс, созданный для хранения даты и времени.

```
class TTime {
        private:
                int year,  month, day, hour, minute;
        public:
                  void Dispay( );
                  void SetTime(int d, int m, int y, int hr, int min);
};
```

В рассмотренном примере класс TTime имеет семь элементов класса: пять полей – year, math, day, hour, minute и два метода класса - Dispay( ) и SetTime( ).

Методы класса объявлены прототипами функций. Предположительно, они выполняют какие-то действия над полями. Их тела будут описаны позднее. Тогда их описанию должны обязательно предшествовать имя класса и операция разрешения видимости (::), сообщающая о принадлежности метода классу.

```
// Определение методов
void TTime::SetTime(int d, int m, int y, int hr, int min)  {
          day = d;
          month = m;
          year = y;
          minute = min;
          hour = hr;
}
void TTime::Display( ) {
 char s[40];
cout<< "Дата:"<<  day<<" ."<<month<<"."<year<<" Время: "<< hour<<":
'<<minute<<endl;
}
```

Тело метода может также содержаться внутри определения класса. В этом случае метод является встроенной (inline) функцией.

Рассмотренный пример показывает применение невстроенных методов.

Можно определить встроенную функцию-элемент и вне тела класса, указав в заголовке определения ключевое слово inline.

Объект класса - это переменная типа «класс». Можно объявлять сколько угодно объектов класса, причем синтаксис их объявления аналогичен объявлению переменной любого другого типа.

[class] <идентификатор класса> <идентификатор переменной>;

Можно передавать объекты в качестве параметров функций или возвращать их как значение функции; объявлять массивы, состоящие из объектов класса.

Большинство встроенных операций не могут применяться к классам. Нельзя использовать операцию сложения для двух объектов, или оператор = = для их сравнения. Но все эти операции применимы для полей классов. Для самих объектов справедливы следующие встроенные операции: выбор элемента (.), присваивание (=), получение адреса (&), косвенная адресация (*), и операция ->.

Доступ к элементам объекта аналогичен доступу к полям структуры. Для этого используются операции точка (выбора) при обращении к элементу через имя объекта и операция -> при обращении через указатель. Обращение таким способом возможно только к элементам со специ-

фикатором доступа public. Получить или изменить значения закрытых элементов (private) можно только через обращение к соответствующим методам класса.

Использование методов класса TTime возможно в функции main( ) следующим образом.

```
  int  main ( )
{
    TTime day;                                  //объявление объекта day класса TTime
    day.SetTime( 24, 2, 1996, 4, 20); // вызов метода SetTime для объекта day
    day.Display( );                             // вызов метода  Display для объекта day
};
```

Методы класса могут вызывать другие функции-элементы того же класса, используя имя функции. Обычно функции или функции-элементы других классов могут вызывать элементы класса с помощью операций . и ->, применяемых представителю или указателю на представитель класса.

Конструкторы

Язык C++ имеет две встроенных особенности при работе с классами – конструкторы и деструкторы – помогающие не забывать про инициализацию объектов и про очистку памяти после завершения работы с ними. Использование конструкторов и деструкторов необходимо во избежание массы досадных ошибок.

Конструктор – функция, автоматически вызываемая при создании объекта (инициализации класса).

Внутрь конструкторов могут быть помещены процедуры инициализации для установки необходимых значений полей до использования объекта. В одном классе можно объявлять сразу несколько конструкторов. Это требуется для инициализации объектов различными способами.

Конструктор является методом класса и носит тоже имя. Он вызывается автоматически компилятором при создании каждого представителя класса.

Если конструктор в программе не определен, то компилятор автоматически генерирует конструктор без параметров, называемый конструктор умолчанию.

Рассмотрим пример класса с несколькими конструкторами.

```
enum color {white, black, auburn, spotted};
class kitten {
        int age;
        color skin;
        char *name;
        public:
        kitten ( int a=1);          //объявлен конструктор по умолчанию
        kitten ( color sk);
        kitten (char *nam);
        };
        kitten :: kitten ( int  a)          // описания конструкторов
        { age=a; skin=black; name=0;}

        kitten :: kitten  ( color sk)
{ switch (sk) {
                case black :    age=10; skin=black;  strcpy (name, “черныш” );   break;
                case white :    age=2; skin=white; strcpy (name, “снежок” ); break;
                case auburn :   age=3;  skin=auburn; strcpy (name, “рыжик” );  break;
                case spotted : age=5; skin=spotted; strcpy (name, “ ” );  break;
              }
 }
};
```

При использовании нескольких конструкторов в одном классе необходимо помнить, что

они должны отличаться набором параметров. Каждый из них должен иметь уникальный набор параметров, иными словами, уникальную сигнатуру. Кроме того, если определение конструктора содержит список инициализации элементов, то список может определяться от заголовка определения функции двоеточием. В этом случае поля перечисляются через запятую. Для каждого поля в скобках указывается инициализирующее значение, которое может быть выражением. Без этого способа не обойтись при инициализации полей-констант, полей-ссылок, полей-объектов. Пример использования инициализации полей в конструкторе.

```
kitten :: kitten ( color c) :  age(1), skin(c),  name(0) { };
```

Деструктор

Деструктор является дополнением конструктора. По завершению работы с объектом автоматически вызывается функция, называемая деструктором.

Он также носит имя класса, но ему предшествует префикс-тильда (~). Деструктор вызывается всякий раз, когда уничтожается представитель класса. Для деструктора выполняются те же правила, что и для конструктора.

Пример простейшего деструктора:

```
# include <iostream.h>
  class Pair{
                    int first, second;                  // поля
   public:
          Pair ( int one, int two): first(one), second (two)                  //конструктор
                    {cout<< " объект создан "<< endl;}
                    ~Pair ( ) { cout <<" объект удален "<< endl;}       //деструктор
                    void out ( ) { cout<< first<< " << second; }
   }
int main( )  {
 Pair num(2,3); num. out( );
 Pair num(4,5); num. out( ) }
```

Указатель this

Иногда внутри метода требуется обратиться к указателю на объект. Это можно сделать через ключевое слово this. У каждого объекта указатель this свой. Он содержит в себе адрес объекта, полем которого он является.

Указатель this называют неявным указателем. Он автоматически создается компилятором. Тогда каждый объект класса имеет свою копию полей класса. Методы же класса существуют только в одном экземпляре.

При вызове метода ему передается неявный аргумент, который обозначает конкретный объект класса. Неявный указатель this можно использовать и явно, как и любой другой указатель, для работы с полями объекта.

```
class Simple{
                    int a;
public:
                    Simple( );
                    void Greet( ) { printf("Hello!\n"); }
        };
  Simple::Simple( ) {
                    this->a=15;
                    Greet( );
                    (*this).Greet;            //оба оператора вызывают функцию Greet
  };
```

Наследование

Простое наследование описывает родство между двумя класса. Класс, являющийся праро-

дителем называется базовым классом (родителем). Класс, созданный из базового класса называется производным классом (потомком). Производный класс наследует все элементы базового класса и может обладать новыми элементами, свойственными только ему. На основе одного базового класса можно создавать многие классы. Производный класс сам может быть базовым. Таким способом создается иерархия классов.

Синтаксис механизма наследования:

class <идентификатор>: [ключ доступа] <идентификатор базового класса>

{тело класса};

Разница между использованием различных ключей доступа при объявлении наследования представлена в таблице:

Ключ доступа	Спецификатор в базовом классе	Ключ доступа в производном классе
private	private protected public	нет private private
protected	private protected public	нет protected protected
public	private protected public	нет protected public

Таким образом, закрытые элементы базового класса в производном классе недоступны вне зависимости от ключа. Обращение к ним осуществляется только через методы базового класса.

Элементы protected при наследовании с ключом private становятся в производном классе private, а в остальных случаях доступ к ним не изменяется.

Доступ к открытым элементам при наследовании становится соответственным ключу доступа.

Указание ключа доступа необязательно, по умолчанию для классов используется ключ доступа private.

Производный класс наследует из базового класса поля и методы, а также деструктор, но не конструкторы и операции присваивания.

В общем случае производный класс наследует все данные и функции своих предков, например:

```
class Base {
private:
        int count;                          // поле класса
public:
Base( ) { count = 0; }                  // определение конструктора
void SetCount(int n) { count = n; }   // определение метода
int GetCount( ) { return count; }       // определение метода
};
```

Если требуется класс, имеющий все свойства Base, но, дополнительно, обладающий способностью изменения значения поля объекта на заданную величину, можно объявить производный класс вида

```
class TDerived: public Base {
public:
Derived( ):Base( ){  };
void ChangeCount(int n) {SetCount( GetCount( ) + n ) };
};
```

Порядок вызова конструктора в производном классе определяется следующими правилами. Если в конструкторе производного класса явный вызов конструктора базового класса отсутствует, автоматически вызывается конструктор базового класса по умолчанию (то есть без параметров).

Если конструктор базового класса требует указания параметров, он должен быть вызван явным образом в конструкторе производного класса в списке инициализации.

Для иерархии классов, состоящей из нескольких уровней, конструкторы базовых классов вызываются, начиная с самого верхнего уровня. После выполняются конструкторы элементов классов, которые являются объектами, в порядке их объявления в классе.

В случаях нескольких базовых классов их конструкторы вызываются в порядке объявления.

Вызов функций базового класса предпочтительнее копирования фрагментов кода из базового класса в производный. Кроме сокращения объема программы, этим достигается упрощение ее модификации.

Перегрузка функций и операций

В языке C++ допускается использование перегруженных функций. Под перегрузкой функций (не обязательно методов класса) понимается создание нескольких прототипов функции, имеющих одинаковое имя. Компилятор различает их по набору параметров.

В создаваемом классе можно изменить свойства большинства стандартных операторов, таких как +, -, *, /, заставив их работать не только с данными базовых типов, но и также с объектами.

Перегружать явным образом можно большинство операций, за исключением: . ?: # ## sizeof.

Перегрузка осуществляется с помощью методов специального типа (функций-операций). Синтаксис функции-операции:

<тип> operator <операция> (список параметров) {тело функции}

Например,

```
class angle {
                int degree, minites, seconds;
public:         angle_value (char *);
                int operator > ( angle &a)              // перегруженный оператор
   { if (3600*degree + 60*minites + seconds>3600*a.degree + 60*a.minites + a.seconds)
                return 1;
   else return 0;
   }
};
```

На перегрузку операторов накладываются следующие ограничения:

- сохраняются количество аргументов, приоритет оператора и порядок группировки его операндов, используемые в стандартных типах данных;
- невозможно изменить синтаксис оператора;
- смысл стандартного оператора применительно к базовым классам переопределять нельзя;
- невозможно создавать новые операторы,
- функции-операции не наследуются;
- функции-операции не могут быть объявлены как static.

Виртуальные функции

Виртуальная функция – это функция, которая определена в базовом классе, а в любом производном от него классе, может переопределяться. Виртуальная функция вызывается только через указатель или ссылку на базовый класс. Определение того, какой экземпляр виртуальной функции вызывается, осуществляется во время выполнения программы и зависит от класса объекта, адресуемого указателем или ссылкой.

Метод определяется виртуальным заданием ключевого слова virtual в объявлении функции в базовом классе.

Например,

```
virtual void Set (int x, int y);
```

Правила объявления виртуальных функций:

Если в базовом классе метод определен как виртуальный, то метод, определенный в производном классе с тем же именем и набором параметров, автоматически становится виртуальным, а с отличающимися параметрами - обычным.

Виртуальные методы наследуются, т.е. переопределять их в производном классе требуется только при необходимости задания различающихся действий.

Если виртуальный метод переопределен в производном классе, объекты этого класса могут получить доступ к методу базового класса с помощью операции разрешения видимости.

Виртуальный метод не может быть объявлен как static, но может быть дружественным.

Если в классе вводится описание виртуального метода, он должен быть определен хотя бы как чисто виртуальный.

Виртуальная функция, объявленная в базовом классе иерархии порождения, часто никогда не используется для объектов этого класса, т.е. не имеет определения. Чтобы подчеркнуть это, используется следующая запись:

virtual int init(void) = 0;

Такие функции называются *чисто виртуальными функциями*.

Класс с одной или с большим количеством чисто виртуальных функций называется *абстрактным классом*.

Абстрактный класс может быть использован только как базовый класс для последующих порождений новых классов. Следовательно, нельзя создать объекты абстрактного класса. Невозможно использование абстрактного класса в качестве типа значения, возвращаемого функцией, и типа параметров функции.

Пример

```
class Shapes {
protected:      int x, y;
public:         virtual int S ( ) = 0;
                virtual void remove ( int, int ) = 0;
};
class Circle : public Shapes {
private:        radius;
public:         circle ( int r, int xx, int yy ) { r=radius;  x=xx;  y=yy;}
                int S ( ) { return (3.1415*radius*radius); }
                void remove ( int u, int v ) { x+=u; y+=v; }
};
```

Теперь невозможно создать объекта класса Shapes, поскольку он содержит чисто виртуальные функции и, следовательно, является абстрактным. Объявлять объекты класса Circle возможно. Так как виртуальные функции S( ) и remove ( ) в нем переопределены.

```
int main ( )
{
Circle b(4,10,12);
Shapes *pB;             //указатель на базовый класс
pB=&b;                  // связывание указателя с объектом базового класса
pB-> S( );      // вызов виртуального метода для объекта базового класса
return 0; }
```

Событийно-управляемое программирование

В основу работы операционных систем семейства Windows положен принцип *событийного управления.* Он означает, что все программные приложения, а также сама операционная система, после своего запуска работают в режиме ожидания действий пользователя и реагируют на них заранее заданным образом. *Событием* называется любое действие пользователя – щелчок кнопкой мыши, перемещение курсора мыши, нажатие клавиши на клавиатуре.

Событие воспринимается операционной системой, а затем преобразуется в *сообщение* – запись, содержащую необходимую информацию о произошедшем событии (какая клавиша была

нажата, в каком месте экрана произошел щелчок мышью и т.п.). Сообщения могут поступать от пользователя, от самой операционной системы, от активного приложения или других приложений. Стандартные сообщения образуют иерархию. Пользователь также может определять собственные сообщения.

Все сообщения поступают в общую очередь, затем распределяются по очередям приложений. Каждое приложение содержит *цикл обработки сообщений*, которое выбирает сообщение из очереди и через операционную систему вызывает подпрограмму, предназначенную для его обработки – *обработчик события*.

Таким образом, Windows-приложение состоит из главной программы и набора *обработчиков событий.*

Среда Visual Studio.NET содержит удобные средства разработки Windows-приложений, выполняющие вместо программиста рутинную работу создание шаблонов и заготовок обработчиков событий.

Среда разработки Visual Studio.NET

Платформу .Net корпорации Microsoft включает в себя средства для разработки программ на четырех языках программирования C++, J#, C#, VT.

Платформа является открытой, т.е. компиляторы для нее могут поставляться сторонними разработчиками.

Приложения в процессе разработки называются проекты. Проект объединяет в себе: файлы, папки, ссылки и прочие ресурсы.

Среда Visual Studio.NET позволяет создавать различные проекты:

- консольные приложения, осуществляющие ввод-вывод в окно командного процессора;
- Windows-приложения, использующие элементы интерфейса Windows: кнопки, формы, события и пр.;
- веб-приложения, формирующие веб-страницу, доступ к которой осуществляется через браузер;
- библиотеки классов, объединяющие классы для дальнейшего их использования в других приложениях;
- веб-сервисы – компоненты, методы которых вызываются через Интернет.

В данном пособие рассматриваются возможности для создания Windows-приложений на языке C++.

Компоненты *Win*dows Forms

Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс и упрощающий доступ к элементам интерфейса Microsoft Windows. Создаваемый с их помощью управляемый код – классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как в приложениях C++, так и на C#, VB.Net, J# и др.

Краткие описания некоторых компонент:

Action List (список действий) осуществляет управление взаимодействием между интерфейсными элементами и логикой программы.

Button (кнопка) служит для создания в приложении различных прямоугольных кнопок с текстовой надписью в одной строчке.

Checkbox (ячейка состояния) позволяет создавать на форме приложения ячейку с двумя состояниями (без галочки и с галочкой) и со строкой, содержащей названия. Щелчок левой кнопкой мыши по этому компоненту во время работы программы вызывает каждый раз изменение состояния компонента на противоположное. В программе всегда можно узнать состояние этого компонента и тем самым выполнять то или иное действие.

ComboBox (комбинированный список) позволяет создавать на форме элемент, являющийся комбинацией строки ввода и выпадающего списка для выбора, фактически объединяя в себе компоненты редактируемого поля (ListBox) и списка (Edit).

Edit (редактирование) задает в форме однострочное редактируемое окно для ввода и вывода данных.

GroupBox (окно группы) служит для создания области, визуально объединяющей на форме несколько интерфейсных элементов.

Label (этикетка) создает на форме текстовую метку или надпись.

ListBox (окно списка) создает прямоугольное поле для отображения текстовых строк с возможностью их выбора, добавления или удаления при работе программы.

GroupBox (окно группы) служит для создания области, визуально объединяющей на форме несколько интерфейсных элементов.

Memo отображает на форме многострочное поле для ввода-вывода данных. Обычно служит для создания редакторов и полей, для вывода блоков данных.

Panel (панель) создает пустую область, на которой можно разместить другие компоненты. Как правило, используется для создания панели инструментов в программе. При перемещении панели компоненты перемещаются вместе с ней.

PopupMenu (всплывающее меню) обычно с помощью этого компонента создается контекстное меню.

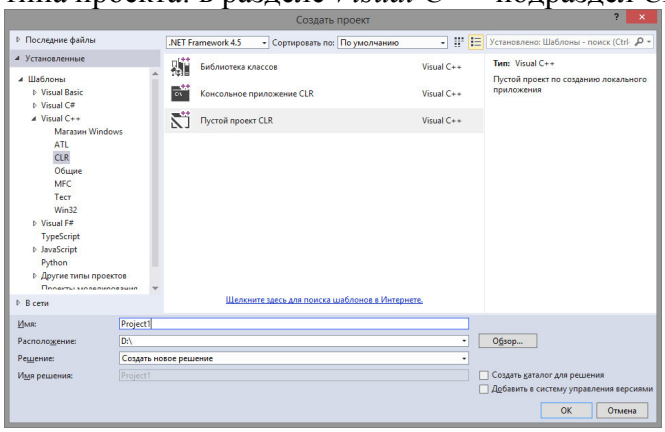
RadioButton (радиокнопка) создает круглое поле с двумя состояниями (с точкой и без точки) и текстовой строкой, поясняющей ее назначение в программе. Обычно несколько таких компонентов, расположенных на форме, позволяют переключить только один элемент из группы. Можно изменить состояние только для одного из этих компонентов, т. к. остальные компоненты переключают при этом свое состояние автоматически.

RadioGroup (группа радиокнопок) позволяет создавать на форме контейнер в виде прямоугольной рамки для объединения группы взаимоисключающих радиокнопок.

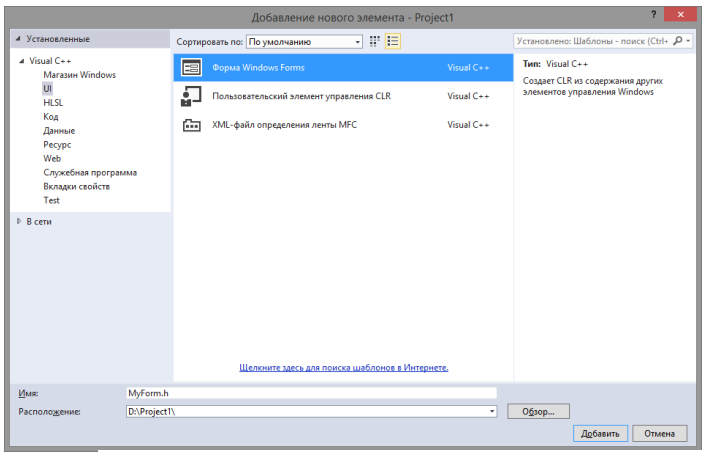
ScrollBar (линейка прокрутки) создает элемент, похожий на линейку с бегунком и кнопками для прокрутки окна, к которому относится этот элемент. Кроме того, с его помощью можно изменять в пределах некоторого заданного интервала значение какого-либо параметра.

Создание проекта Windows Forms в Visual Studio на C++

Для создания Windows-приложения в среде Visual Studio на языке C++ необходимо выполнить последовательность *Файл → Создать → Проект. В* открывшемся окне осуществить выбор типа проекта: в разделе *Visual C++* подраздел *CLR,* а затем – пункт *Пустой проект CLR.*



После создания проекта в обозревателе решений требуется кликнуть по нему правой кнопкой мыши. В открывшемся контекстном меню выбрать: *Добавить → Создать элемент,* а затем в разделе *UI* открывшегося меню: *Форма→Windows→Forms.*



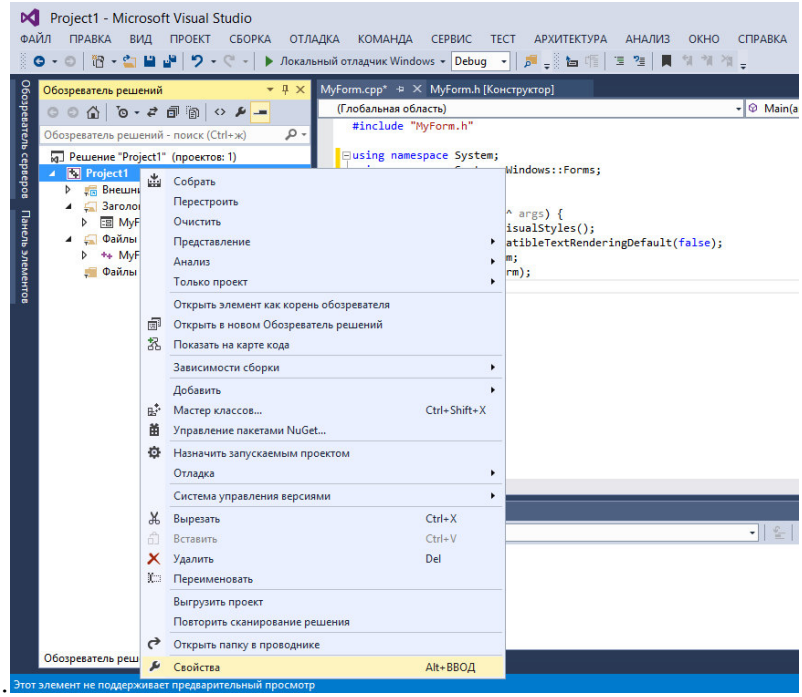
Когда форма будет добавлена, в обозревателе решений выбрать файл MyForm.cpp, откроется новая вкладка с единственной строчкой кода:

```
#include "MyForm.h"
```

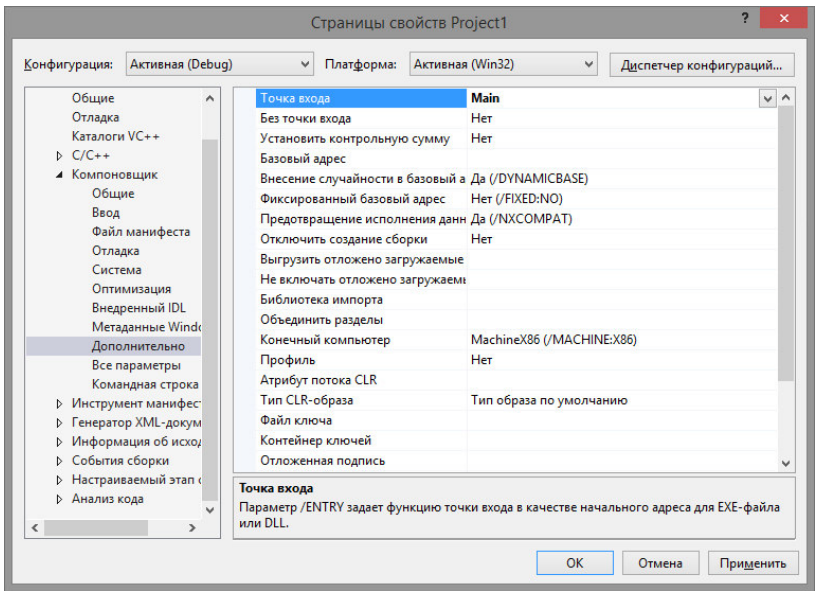
К нему следует добавить программный код вида:

```
using namespace System;
using namespace System::Windows::Forms;

[STAThread]
void Main(array<String^>^ arg) {
	Application::EnableVisualStyles();
	Application::SetCompatibleTextRenderingDefault(false);

	Project1::MyForm form; //Project1 - имя вашего проекта
	Application::Run(%form);
	}
```

После этого в свойствах проекта выбрать подраздел *Система раздела →Компоновщик* и в строке *Подсистема* из выпадающего меню: *Windows (/SUBSYSTEM:WINDOWS),* нажать *Применить*.



Не закрывая окно свойств проекта, нужно перейти в подраздел *Дополнительно* и в строке *Точка входа* ввести **Main** и нажать **ОК.**

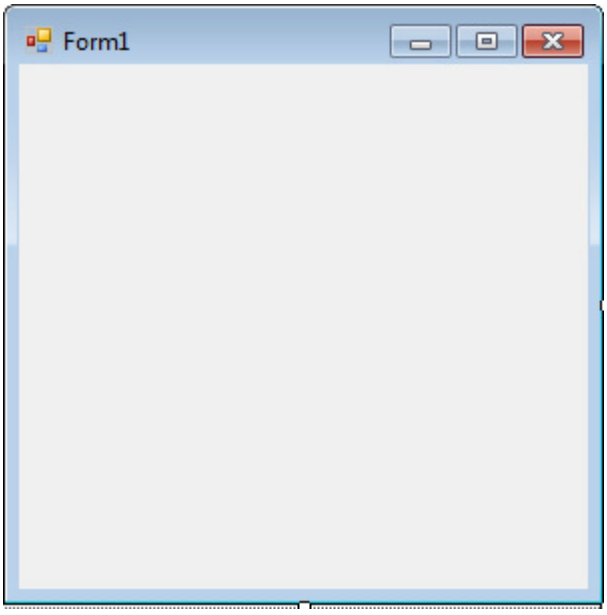


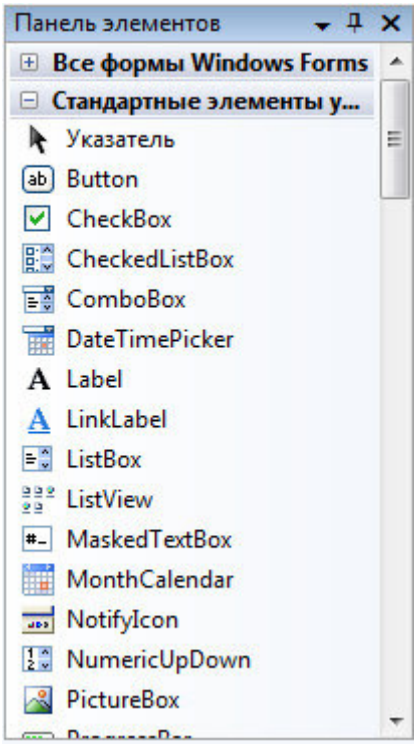
На этом настройки проекта завершены. Для редактирования внешнего вида формы, необходимо перейти во вкладку *MyForm.h [Конструктор]*, кликнув дважды по файлу *MyForm.h* в обозревателе решений.

Пример создания Windows-приложения

После создания проекта на экране должна появиться пустая форма: Справа от нее располагается панель элементов. В случае ее отсутствия включение возможно с помощью меню **Вид → Панель Элементов** или комбинацией клавиш: **Ctrl + Alt + X**.

На панели элементов расположены элементы Windows Forms.





Затем следует задать атрибут **Name** в **Панели Свойств** для текстовых полей (**TextBox**) . Для первого поля пусть будет **x**, а для второго – **y**. Форма готова.

Для добавления заголовочного файла – **fact.h** – в проект, нужно кликнуть правой кнопкой мыши в **Обозревателе решений** на папке **Заголовочные файлы**, далее в меню: **Добавить -> Создать Элемент** и вписать название файла – **fact.h (вкладка код-> заголовочный файл)**, затем выполнить **Добавить**.

В файле будет содержаться прототип функции вычисления факториала:

```
long double fact(int N);
```

Далее требуется подключить этот файл к проекту с помощью директивы **#include** – открыть файл **fact.cpp** и добавить строку кода:

```
#include "fact.h"
```

Затем в файл исходного кода **fact.cpp** добавляется и сама функция вычисления факториала:

```
#pragma once

long double fact(int N)
{
// если пользователь ввел отрицательное число
if (N < 0)
        // возвращаем ноль
        return 0;
// если пользователь ввел ноль
if (N == 0)
        // возвращаем факториал нуля
        return 1;
// Во всех остальных случаях
else
        // делаем рекурсию
        return N * fact(N - 1);
}
```

После компиляции кода, можно приступить к написанию обработчика событий для кнопки **Button**. Открыв файл **Form.h**, нужно щелкнуть двойным щелчком по кнопке **Button** в визуальном представлении формы и перейти к исходному коду.

Необходимо помнить, что любой набор символов, вводимый с клавиатуры в текстовое поле программы, является строкой. Однако сама программа работает с числами. Поэтому нужно вытащить значение из текстового поля и привести его к целочисленному типу. Для этого надо воспользоваться одной из множества функций, выполняющих конвертирование типов данных. Для этого следует описать и инициализировать переменную, в которой будет храниться значение числа, для которого будет вычислен факториал.

```
int  number = System :: Convert :: ToDouble (x–>Text);
```

Эта строка инициализирует переменную, которая хранит в себе введенное значение числа. Далее нужно вызвать функцию **fact** с переменной **number** в качестве фактического параметра. Результат работы функции передается в переменную **factor** следующим кодом:

```
double factor = fact(number);
```

Затем следует выполнить обратное преобразование – полученное значение факториала в строку – и присвоить его второму текстовому полю:

```
y –> Text = System :: Convert :: ToString (factor);
```

В итоге, обработчик события для **Button** будет выглядеть следующим образом:

```
int  number = System :: Convert :: ToDouble (x–>Text
double factor = fact(number);
y –> Text = System :: Convert :: ToString (factor);
```

Скомпилировав и запустив готовую программу, можно проверить ее работоспособность.
Обработка исключительных ситуаций.

Отношения между классами. Диаграммы классов на языке UML

Язык унифицированного моделирования UML (Unified Modeling Language) является визуальным средством представления моделей программ, ставшим уже стандартом. UML широко применяется для визуализации, специфицирования, конструирования и документирования различных программных систем.

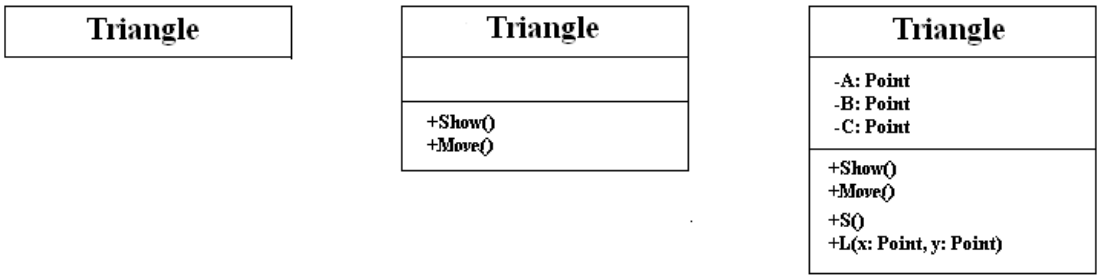
Под моделями программ понимается их графическое представление в виде различных диа-

грамм, отражающих связи между объектами в программном коде. Одной из основных диаграмм UML является диаграмма классов, которая очень удобна для сопоставления различных вариантов проектных решений. Кроме того, она используется для описания шаблонов проектирования, которые в сочетании с объектно-ориентированной парадигмой программирования лежат в основе современного подхода к разработке программного обеспечения.

На диаграмме UML класс изображается в виде прямоугольника, состоящего из трех частей, расположенных вертикально. В верхней части указывается имя класса. В средней части приводится список полей, называемых в диаграмме UML атрибутами. Возможно, указание типов и атрибутов полей после двоеточия. Список методов (операций) приводится в нижней части, возможно, с указанием возвращаемого значения.

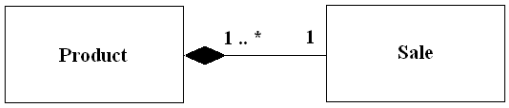
Имена абстрактных классов и операций принято обозначать курсивом. Перед именами полей и методов указывается спецификатор доступа с помощью одного из трех символов: + для public, - для private, # для protected. Для статических элементов класса после спецификатора доступа записывается символ $.

Во второй и третьих частях могут указываться не все элементы класса, а только представляющие интерес на данном уровне абстракции. Эти части могут быть совсем пусты, и в этом случае их можно не указывать.



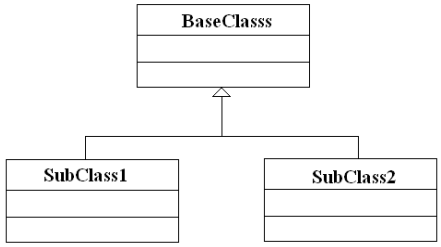
Существуют следующие отношения между классами: ассоциация, наследование, агрегация, зависимость.

Если два класса взаимодействуют друг с другом концептуально, то их взаимодействие называется ассоциацией. На диаграмме ассоциация показывается соединительной линией, над которой может быть указана кратность – то есть сколько объектов данного класса может быть связано с одним объектом другого класса. В случае произвольного количества указывается символ звездочка.



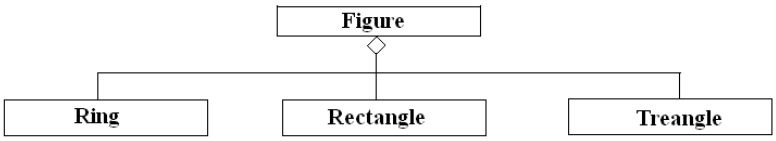
Отношение «ассоциация» выявляется на ранней стадии проектирования, и в дальнейшем, как правило, подлежит конкретизации.

Наследование на диаграмме классов показывается линией со стрелкой в виде не закрашенного треугольника, которая указывает на базовый класс. Допускается объединение несколько стрелок в одну для разгрузки диаграммы.



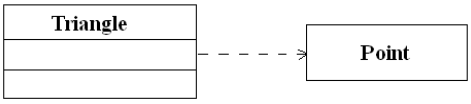
Из диаграммы видно, что классы SubClass1 и SubClass2 являются потомками класса BaseClass.

Отношение «агрегация» подразумевает включение в качестве составной части объектов другого класса (отношение целое/часть). На диаграмме такая связь обозначается стрелкой в виде не закрашенного ромба. При этом стрелка указывает на «целое». В программе для реализации нестрогой агрегации «часть» включается в «целое» по ссылке. Такой компонент может динамически появляться и исчезать.



Строгая агрегация имеет название – композиция. Она означает, что компонент не может исчезнуть, пока объект «целое» существует. Композиция обозначается линией со стрелкой в виде закрашенного ромба.

Отношение зависимости (использования) между классами показывает, что один класс пользуется некоторыми услугами другого класса. На диаграмме она обозначается пунктирной линией со стрелкой.



Современное программирование базируется не только на самих идеях объектно-ориентированного подхода, но и на применении шаблонов проектирования, включающих в себя наиболее удачные решения типичных проблем, возникающих при разработке программ.

Документация по сопровождению программных средств

Программные средства являются одним из наиболее гибких видов промышленных изделий и эпизодически подвергаются изменениям в течение всего времени их использования.

Для сохранения и повышения качества программного обеспечения необходимо регламентировать процесс модификации и поддерживать его соответствующим тестированием и контролем качества. В результате программное изделие со временем обычно улучшается как по функциональным возможностям, так и по качеству решения отдельных задач.

Документирование программных изделий:

При разработке программного обеспечения (ПО) создается и используется большой объем разнообразной документации.

Она необходима:

1) как средство передачи информации между разработчиками ПО,
2) как средство управления разработкой ПО,
3) как средство передачи пользователям информации, необходимой для применения и сопровождения ПО.

На создание этой документации приходится большая доля стоимости ПО.

Эту документацию можно разбить на две группы:

1. Документы управления разработкой ПО (*software process documentation*) управляют и протоколируют процессы разработки и сопровождения ПО, обеспечивая связи внутри коллектива разработчиков ПО и между коллективом разработчиков и *менеджерами ПО* (*software managers*) – лицами, управляющими разработкой ПО. Эти документы могут быть следующих типов:

• *планы, оценки, расписания.* Эти документы создаются менеджерами для прогнозирования и управления процессами разработки и сопровождения ПО.

• *отчеты об использовании ресурсов в процессе разработки.* Создаются менеджерами.

• *стандарты.* Эти документы предписывают разработчикам, каким принципам, правилам,

соглашениям они должны следовать в процессе разработки ПО. Эти стандарты могут быть как международными или национальными, так и специально созданными для организации, в которой ведется разработка ПО.

• *рабочие документы.* Это основные технические документы, обеспечивающие связь между разработчиками. Они содержат фиксацию идей и проблем, возникающих в процессе разработки, описание используемых стратегий и подходов, а также рабочие (временные) версии документов, которые должны войти в ПО.

• *заметки и переписка.* Эти документы фиксируют различные детали взаимодействия между менеджерами и разработчиками.

- документы, входящие в состав ПО.

2. *Документы, входящие в состав* ПО *(software product documentation),* описывают программы ПО как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением ПО). Следует отметить, что эти документы будут использоваться не только на стадии эксплуатации ПО (в ее фазах применения и сопровождения), но и на стадии разработки для управления процессом разработки (вместе с рабочими документами). Во всяком случае, они должны быть проверены (протестированы) на соответствие программам ПО.

Существует четыре основных типа документации на ПО:

1. архитектурная/проектная – обзор программного обеспечения, включающий описание рабочей среды и принципов, которые должны быть использованы при создании ПО;

2. техническая – документация на код, алгоритмы, интерфейсы, API;

3. пользовательская – руководства для конечных пользователей, администраторов системы и другого персонала;

4. маркетинговая.

Архитектурная/проектная документация*:*

Проектная документация обычно описывает продукт в общих чертах. Не описывая того, как что-либо будет использоваться, она скорее отвечает на вопрос «почему именно так?».

Техническая документация*:*

При создании программы, одного лишь кода, как правило, недостаточно. Должен быть предоставлен некоторый текст, описывающий различные аспекты того, что именно делает код. Такая документация часто включается непосредственно в исходный код или предоставляется вместе с ним.

Подобная документация имеет сильно выраженный технический характер и в основном используется для определения и описания API, структур данных и алгоритмов.

Пользовательская документация*:*

В отличие от технической документации, сфокусированной на коде и том, как он работает, пользовательская документация описывает лишь то, как использовать программу.

Обычно, пользовательская документация представляет собой руководство пользователя, которое описывает каждую функцию программы, а также шаги, которые нужно выполнить для использования этой функции.

Пользовательская документация ПО (user documentation) объясняет пользователям, как они должны действовать, чтобы применить данное ПО.

Различают две категории пользователей ПО: ординарных пользователей ПО и администраторов ПО.

Типичным можно считать следующий состав пользовательской документации для достаточно больших ПО:

- общее функциональное описание ПО. Дает краткую характеристику функциональных возможностей ПО. Предназначено для пользователей, которые должны решить, насколько необходимо им данное ПО.

- руководство по инсталляции ПО. Предназначено для системных администраторов. Должно детально предписывать, как устанавливать системы в конкретной среде, содержать описание машинно-считываемого носителя, на котором поставляется ПО, файлы, представляющие ПО, и тре-

бования к минимальной конфигурации аппаратуры.

- инструкция по применению ПО. Предназначена для ординарных пользователей. Содержит необходимую информацию по применению ПО, организованную в форме удобной для ее изучения.

- справочник по применению ПО. Предназначен для ординарных пользователей. Содержит необходимую информацию по применению ПО, организованную в форме удобной для избирательного поиска отдельных деталей.

- руководство по управлению ПО. Предназначено для системных администраторов. Оно должно описывать сообщения, генерируемые, когда ПО взаимодействует с другими системами, и как реагировать на эти сообщения. Кроме того, если ПО использует системную аппаратуру, этот документ может объяснять, как сопровождать эту аппаратуру.

Разработка пользовательской документации начинается сразу после создания внешнего описания. Качество этой документации может существенно определять успех ПО. Она должна быть достаточно проста и удобна для пользователя (в противном случае это ПС, вообще, не стоило создавать). Поэтому, хотя черновые варианты (наброски) пользовательских документов создаются основными разработчиками ПО, к созданию их окончательных вариантов часто привлекаются профессиональные технические писатели. Кроме того, для обеспечения качества пользовательской документации разработан ряд стандартов, в которых предписывается порядок разработки этой документации, формулируются требования к каждому виду пользовательских документов и определяются их структура и содержание.

Маркетинговая документация*:*

Для многих приложений необходимо располагать рядом с ними рекламные материалы, с тем, чтобы заинтересовать людей, обратив их внимание на продукт. Такая форма документации имеет целью:

- подогреть интерес к продукту у потенциальных пользователей
- информировать их о том, что именно делает продукт, с тем, чтобы их ожидания совпадали с тем, что они получат
- объяснить положение продукта по сравнению с конкурирующими решениями

Формальные требования к документации программного обеспечения описаны в ЕСПД (Единая система программной документации), неформально: состав документации к программному обеспечению состоит из описания внешнего эффекта ПО и описания его внутреннего устройства.

Первая часть документации (пользовательская), так называемая «Инструкция пользователю» (или «Руководство пользователю») предназначена для того, кто собирается использовать программное обеспечение, не вникая в подробности его внутреннего устройства; вторая («Руководство программисту») необходима при модификации ПО или при необходимости исправить в нем ошибку.

Разработка технического задания на создание программ

Стандарт ГОСТ 19.201-78. устанавливает порядок построения и оформления технического задания на разработку программы или программного изделия для вычислительных машин, комплексов и систем независимо от их назначения и области применения.

Общие положения

1 Техническое задание оформляют в соответствии с ГОСТ 19.106-78 на листах формата А4 и А3 по ГОСТ 2.301-68, как правило, без заполнения полей листа. Номера листов (страниц) проставляют в верхней части листа над текстом.

2 Лист утверждения и титульный лист оформляют в соответствии с ГОСТ 19.104-78. Информационную часть (аннотацию и содержание), лист регистрации изменений допускается в документ не включать.

3 Для внесения изменений и дополнений в техническое задание на последующих стадиях разработки программы или программного изделия выпускают дополнение к нему. Согласование и утверждение дополнения к техническому заданию проводят в том же порядке, который установлен для технического задания.

4 Техническое задание должно содержать следующие разделы:

•название программы и область применения;

•основание для разработки;

•назначение разработки;

•технические требования к программе или программному изделию;

•технико-экономические показатели;

•стадии и этапы разработки:

•порядок контроля и приемки;

•приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

5 Содержание разделов

5.1. В разделе «Наименование и область применения» указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

5.2. В разделе «Основание для разработки» должны быть указаны:

•документ (документы), на основании которых ведется разработка,

•организация, утвердившая этот документ, и дата его утверждения;

•наименование и (или) условное обозначение темы разработки.

5.3. В разделе «Назначение разработки» должно быть указано функциональное и эксплуатационное назначение программы или программною изделия.

5.4. Раздел «Технические требования к программе или программному изделию» должен содержать следующие подразделы:

•требования к функциональным характеристикам;

•требования к надежности;

•условия эксплуатации;

•требования к составу и параметрам технических средств;

•требования к информационной и программной совместимости;

•требования к маркировке и упаковке;

•требования к транспортированию и хранению;

•специальные требования.

5.5. В подразделе «Требования к функциональным характеристикам» должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам и т.п.

5.6. В подразделе «Требования к надежности» должны быть указаны требования к обеспечению надежного функционирования (обеспечение устойчивого функционирования, контроль входной и выходной информации, время восстановления после отказа и т. п.).

5.7. В подразделе «Условия эксплуатации» должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность и т. п. для выбранных типов носителей данных), при которых должны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.

5.8. В подразделе «Требования к составу и параметрам технических средств» указывают необходимый состав технических средств с указанием их технических характеристик.

5.9. В подразделе «Требования к информационной и программной совместимости» должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования. При необходимости должна обеспечиваться защита информации и программ.

5.10. В подразделе «Требования к маркировке и упаковке» в общем случае указывают требования к маркировке программного изделия, варианты и способы упаковки.

5.11. В подразделе «Требования к транспортированию и хранению» должны быть указаны для программного изделия условия транспортирования, места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

5.12. В разделе «Технико-экономические показатели» должны быть указаны: ориентировочная экономическая эффективность предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами.

5.13. В разделе «Стадии и этапы разработки» устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных документов, которые должны быть разработаны, согласованы и утверждены), а также, как правило, сроки разработки и определяют исполнителей.

5.14. В разделе «Порядок контроля и приемки» должны быть указаны виды испытаний и общие требования к приемке работы.

5.15. В приложениях к техническому заданию при необходимости приводят:

• перечень научно- исследовательских и других работ, обосновывающих разработку;

• схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые могут быть использованы при разработке;

• другие источники разработки.

Пример разработки технического задания.

1. Введение.

Работа выполняется в рамках проекта «Автоматизированная система оперативно-диспетчерского управления электро-, теплоснабжением корпусов предприятия».

2 Основание для разработки

1. Основанием для данной работы служит договор № 12-к от 10 января 2017 г.

2. Наименование работы:

«Модуль автоматизированной системы оперативно-диспетчерското управления теплоснабжением корпусов предприятия».

3. Исполнители: ОАО « ProgramLabs».

4. Соисполнители: нет.

3 Назначение разработки

Создание модуля для контроля и оперативной корректировки состояния основных параметров теплообеспечения корпусов предприятия «Темп».

4 Технические требования

4.1. Требования к функциональным характеристикам.

4.1.1. Состав выполняемых функций. Разрабатываемое программное обеспечение должно обеспечивать:

• сбор и анализ информации о расходовании тепла, горячей и холодной воды по данным теплосчетчиков SA-94 на всех тепловых выходах;

• сбор и анализ информации с устройств управления системами воздушного отопления и кондиционирования типа РТ1;

• предварительный анализ информации на предмет нахождения параметров в допустимых пределах и сигнализирование при выходе параметров за пределы допуска;

• выдачу рекомендаций по дальнейшей работе;

• отображение текущего состояния набора параметров – циклически постоянно (режим работы круглосуточный), при сохранении периодичности контроля прочих параметров;

• визуализацию информации по расходу теплоносителя:

- текущей (аналогично показаниям счетчиков);

- с накоплением за прошедшие сутки, неделю, месяц - в виде почасового графика для информации за сутки и неделю;

- суточный расход - для информации за месяц.

Для устройств управления приточной вентиляцией текущая информация должна содержать номер приточной системы и все параметры, выдаваемые на собственный индикатор.

По отдельному запросу осуществляются внутренние настройки. В конце отчетного периода система должна архивировать данные.

4.1.2. Организация входных и выходных данных.

Исходные данные в систему поступают в виде значений с датчиков, установленных в помещениях предприятия. Эти значения отображаются на компьютере диспетчера. После анализа поступившей информации оператор диспетчерского пункта устанавливает необходимые параметры для устройств, регулирующих отопление и вентиляцию в помещениях.

Возможна также автоматическая установка некоторых параметров для устройств регулирования.

Основной режим использования системы – ежедневная работа.

1.2. Требования к надежности.

Для обеспечения надежности необходимо проверять корректность получаемых данных с датчиков.

4.3. Условия эксплуатации и требования к составу и параметрам технических средств.

Для работы системы должен быть выделен ответственный оператор. Требования к составу и параметрам технических средств уточняются на этапе эскизного проектирования системы.

4.4. Требования к информационной и программной совместимости.

Программа должна работать на платформах Windows 8.

4.5. Требования к транспортировке и хранению. Программа поставляется на лазерном носителе информации.

Программная документация поставляется в электронном и печатном виде.

4.6. Специальные требования.

Программное обеспечение должно иметь дружественный интерфейс, рассчитанный на пользователя (в плане компьютерной грамотности) средней квалификации. Ввиду объемности проекта задачи предполагается решать поэтапно, при этом модули ПО, созданные в разнос время, должны предполагать возможность наращивания системы и быть совместимы друг с другом, поэтому документация на принятое эксплуатационное ПО должна содержать полную информацию, необходимую для работы программистов с ним.

Язык программирования – по выбору исполнителя, должен обеспечивать возможность интеграции программного обеспечения с некоторыми видами периферийного оборудования.

5 Требования к программной документации

Основными документами, регламентирующими разработку будущих программ, должны быть документы Единой Системы Программной Документации (ЕСПД);

руководство пользователя,

руководство администратора, описание применения.

6 Технико-экономические показатели

Эффективность системы определяется удобством использования системы для контроля и управления основными параметрами теплообеспечения помещений предприятия, а также экономической выгодой, полученной от внедрения аппаратно-программного комплекса.

7 Порядок контроля и приемки

После передачи Исполнителем отдельного функционального модуля программы Заказчику, последний имеет право тестировать модуль в течение 7 дней. После тестирования Заказчик должен принять работу по данному этапу или в письменном виде изложить причину отказа от принятия. В случае обоснованного отказа Исполнитель обязуется доработать модуль.

8 Календарный план работ.

Наименование этапа	Сроки этапа	Результат выполнения этапа
1 Изучение предметной области	01.02.2017 – 28.02.2017	Предложения по разработке программного обеспечения Проектирование системы. Выбор средства реализации. Разработка системы. Акт сдачи-приемки предложений по реализации системы.

2 Разработка программного модуля по сбору и анализу информации со счётчиков и устройств управления.	01.03.2017 – 31.08.2017	Завершенный программный комплекс. Внедрение системы на одном из корпусов предприятия.
3 Тестирование и отладка модуля.	01.09.2017 – 30.11.2017	Готовая система контроля теплообеспечения.
4. Внедрение автоматизированной системы.	01.12.2017 – 30.12.2017	Готовая система контроля системы во всех корпусах, установленная в диспетчерском пункте. Программная документация. Акт сдачи- приёма работ.

Руководство пользователя

Руководство пользователя – документ, назначение которого – предоставить людям помощь в использовании некоторой системы. Документ входит в состав технической документации на систему и, как правило, подготавливается техническим писателем.

Большинство руководств пользователя помимо текстовых описаний содержат изображения. В случае программного обеспечения, в руководство обычно включаются снимки экрана, при описании аппаратуры — простые и понятные рисунки. Используется стиль и язык, доступный предполагаемой аудитории, использование жаргона сокращается до минимума либо подробно объясняется.

Структура и содержание документа Руководство пользователя автоматизированной системы регламентированы подразделом 3.4 документа РД 50-34.698-90. Структура и содержание документов Руководство оператора, Руководство программиста, Руководство системного программиста регламентированы ГОСТ 19.505-79, ГОСТ 19.504-79 и ГОСТ 19.503-79 соответственно.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ ЗАНЯТИЯМ

Общие рекомендации по созданию программы

Главная цель при создании программы с использованием структурного и процедурного методов программирования – получение легко читаемого кода программы на простой структуры.

Первый шаг состоит в продумывании и записи алгоритма будущей программы на естественном языке или с использованием блок-схемы. Это позволяет продумать алгоритм в деталях, разбить программу на логические блоки, определить их последовательность, продумать комментарии к программе.

Необходимо стараться (если это возможно) разбить алгоритм на последовательность законченных действий. Каждое из них можно оформить в виде функции.

Функция не должна быть слишком большой, как правило, ее текст должен не превышать два экрана.

Если некоторые действия в программе повторяются несколько раз, их также требуется оформить в виде функции, что сделает программу нагляднее и сократит ее размер.

При расположении последовательности функций нужно помнить, что функция может быть вызвана только после ее объявления. Если программа содержит две-три функции, такой порядок достаточно легко определить. Иначе лучше использовать прототипы функций, расположив их в начале программы, сразу после директив препроцессора.

Всю информацию, требуемую для работы функции, необходимо передавать в качестве параметров. В вызове функции строго соблюдать соответствие количества, типов и порядок следования фактических параметров – формальным.

Правильный выбор имен улучшают читаемость программы. Для этого существуют несколько рекомендаций. Считается, что несколько первых символов имени должны объяснять содержимое переменной, тем самым, создавая документированность программы.

Обычно, чем больше область видимости переменной, тем ее имя длиннее. Для параметров коротких циклов лучше использовать однобуквенные имена. Имена макросов и констант предпочтительнее записывать заглавными буквами.

Нельзя использовать в качестве имен переменных имена типов и ключевые слова языка.

Желательно инициализировать переменные при их объявлении, а объявлять их как можно ближе к месту непосредственного использования. В небольших функциях удобно все объявления локальных переменных располагать в начале блока.

Если ввод переменных осуществляется через клавиатуру, требуется предварять его выводом сообщения на экран, которое обязательно должно быть информативным. Лучше предусмотреть и правильность ввода переменных. Например, при введении даты нужно проверить, чтобы число не превышало 31, а номер месяца 12.

Использование локальных переменных предпочтительнее глобальных. Если глобальная переменная необходима, лучше объявить ее статической, что ограничит область ее действия одним файлом. Изменение глобальных переменных сложно отслеживать в большой программе.

Не рекомендуется использования в программе чисел в явном виде. Лучше использовать константу. Особенно это актуально в случаях, неоднократного ее использования. Тогда при изменении кода программы, легко изменить ее значение. Константы также должны иметь осмысленные имена.

При использовании ветвления следует избегать проверки лишних условий. Неэффективным кодом считается проверка на равенство нулю. Например, вместо if (k = = 0) лучше писать if (k).

При организации циклов лучше размещать инициализацию и приращения счетчика, проверку условия выхода из цикла в одном месте. Требуется предусматривать аварийный выход из цикла по достижению заданного максимального количества итераций.

Вложенные циклы и блоки должны иметь отступ друг от друга в три-четыре символа, причем блоки одного уровня вложенности должны быть выровнены по вертикали. Желательно, чтобы закрывающаяся фигурная скобка была расположена строго под соответствующей ей открываю-

щейся.

Необходимо проверять коды возврата ошибок и предусматривать печать сообщений в тех точках программы, куда управление программой при ее нормальной работе передаваться не должно. Например, оператор switch должен иметь ветвь default, в случае, когда в нем не перечислены все возможные значения переключателя.

Сообщение об ошибке должно быть информативным и подсказывать пользователю методы ее исправления. Например, при вводе неверного значения в сообщении об ошибке должен быть указан допустимый диапазон.

Написанный программный код нужно тщательно отредактировать. Убрать ненужные фрагменты, сгруппировать описания, оптимизировать проверки условий, проверить условия выхода из циклов, проверить оптимальность разбиения на функции.

Следует сопроводить программу комментариями, которые должны представлять собой правильные предложения без сокращений и со знаками препинания. Но они и не должны подтверждать очевидное. Если комментарий занимает несколько строк, лучше его разместить до комментируемого фрагмента. Абзацный отступ комментария должен соответствовать отступу комментируемого блока. Для улучшения читаемости можно помечать комментарием окончание длинного составного оператора.

Не следует размещать в одной строке программы множество операторов. Важно, чтобы строка программы не выходила за пределы экрана. Между крупными блоками и функция лучше располагать пустую строку.

Для грамотно написанной программы недостаточно подтверждения ее работы и даже получения верного результата. Программа должна иметь четкую структурированность, наглядность, читаемость, сопровождение комментариями, возможность легкой модификации, и желательно, грамотно разработанный, эффективный алгоритм.

ТИПОВЫЕ ВАРИАНТЫ ЗАДАНИЙ И РЕКОМЕНДАЦИИ ДЛЯ ВЫПОЛНЕНИЯ

Тема **Линейные программы**

Задание. Написать программу для расчета двух формул.

Методические рекомендации

Перед созданием программы четко определить входные и выходные данные, их типы. Ввод данных с клавиатуры предварять сообщением. Для вывода данных использовать форматирование (библиотеки iostream.h, iomanip.h). При записи выражений не забывать про приоритет операций. Использовать библиотеку математических функций math.h.

Подготовить в тетради тестовые задания и проверить результат с помощью калькулятора. Запустить программу несколько раз. Сверить с результатами, полученными вручную.

Результаты вычислений по первой и второй формуле должны совпадать.

Варианты

1	$z_1 = 2\sin^2(3\pi - 2\alpha)\,\cos^2(5\pi + 2\alpha)$ $z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right)$	6	$z_1 = \cos^4 x + \sin^2 y + \frac{1}{4}\sin^2 2x - 1$ $z_2 = \sin(x+y)\sin(y-x)$
2	$z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha$ $z_2 = 2\sqrt{2}\cos\alpha\sin\left(\frac{\pi}{4} + 2\alpha\right)$	7	$z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2$ $z_2 = -4\sin^2\frac{\alpha-\beta}{2}\cos(\alpha+\beta)$
3	$z_1 = \frac{1 - 2\sin^2\alpha}{1 + \sin 2\alpha}$ $z_2 = \frac{1 - tg\alpha}{1 + tg\alpha}$	8	$z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)}$ $z_2 = ctg\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right)$
4	$z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha - \cos 3\alpha + \cos 5\alpha}$ $z_2 = tg3\alpha$	9	$z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2\sin^2 2\alpha}$ $z_2 = 2\sin\alpha$
5	$z_1 = \frac{\sin\alpha + \cos(2\beta - \alpha)}{\cos\alpha - \sin(2\beta - \alpha)}$ $z_2 = \frac{1 + \sin 2\beta}{\cos 2\beta}$	10	$z_1 = \frac{\sin 4\alpha}{1 + \cos 4\alpha} \cdot \frac{\cos 2\alpha}{1 + \cos 2\alpha}$ $z_2 = ctg\left(\frac{3}{2}\pi - \alpha\right)$

Тема **Разветвляющиеся программы**

Задание. Написать программу для вычисления значения функции F. Значения величин *a*, *b*, *c*, *x* вводить с клавиатуры. Учитывать область определения функции F. Предусмотреть вывод сообщения в случае, если введенные пользователем значения не принадлежат ОДЗ. Протестировать программу, контрольные значения записать в тетрадь.

Методические рекомендации

Перед созданием программы четко определить входные и выходные данные, их типы. Ввод данных с клавиатуры в программе предварять сообщением. Использовать условный оператор.

Варианты

1. $F=\begin{cases} ax^2+b & при\ \ x<0\ и\ b\neq 0 \\ \dfrac{x-a}{x-c} & при\ \ x>5\ и\ b=0 \\ \ln\dfrac{x}{c} & в\ остальных\ случаях \end{cases}$

2. $F=\begin{cases} \dfrac{1}{ax}-b & при\ \ x<0\ и\ c=0 \\ \dfrac{x-ac^2}{x} & при\ \ x>15 \\ \dfrac{10bx}{c-4a} & в\ остальных\ случаях \end{cases}$

3. $F=\begin{cases} ax^2+b+c & при\ \ a<0\ и\ c\neq 0 \\ \dfrac{-a}{x-b} & при\ \ a>0\ и\ c=0 \\ \dfrac{(x+c)}{bc} & в\ остальных\ случаях \end{cases}$

4. $F=\begin{cases} -ax-cb & при\ \ c<0 \\ \dfrac{x^3-a}{-c} & при\ \ c>0\ и\ x>10 \\ \dfrac{bx}{\lvert b-a\rvert} & в\ остальных\ случаях \end{cases}$

5. $F=\begin{cases} a-\dfrac{cx}{\sqrt{10+b}} & при\ \ x<0\ и\ b>0 \\ x+c+b & при\ \ x>10\ ,b<0, \\ 3x^a+\dfrac{2b}{c} & в\ остальных\ случаях \end{cases}$

6. $F=\begin{cases} ax^2+b^2x+c & при\ \ c<0\ и\ b\neq 0 \\ \dfrac{x+a}{\lvert x+c\rvert} & при\ \ c>0\ и\ b=0 \\ \dfrac{x}{c} & в\ остальных\ случаях \end{cases}$

7. $F=\begin{cases} ax^2+bc & при\ \ x<0\ и\ b\neq 0 \\ \dfrac{x-a}{x-c} & при\ \ x>0\ и\ b=0 \\ \dfrac{\lvert bx-2a\rvert}{c} & в\ остальных\ случаях \end{cases}$

8. $F=\begin{cases} -ax^2+cb & при\ \ c<0\ и\ b\neq 0 \\ \dfrac{-x-a}{\sqrt{xc}} & при\ \ c>0\ и\ b=0 \\ \dfrac{xb}{a^3-8} & в\ остальных\ случаях \end{cases}$

9. $F=\begin{cases} ax^2-b^2x^3 & при\ \ a<0\ и\ x\neq 0 \\ bc-\dfrac{a}{c} & при\ \ a>3\ и\ x=0 \\ 1+\dfrac{xb}{c+a} & в\ остальных\ случаях \end{cases}$

10. $F=\begin{cases} ax^2-bx+c & при\ \ x<3\ и\ b\neq 0 \\ \dfrac{\sqrt{x-a}}{x-c} & при\ \ x>3\ и\ b=0 \\ \dfrac{x^b}{c} & в\ остальных\ случаях \end{cases}$

Тема **Разветвляющиеся программы. Оператор switch.**

Задание. Написать программу с использованием оператора множественного выбора switch.

Методические рекомендации

Определите параметр, относительно которого меняется значение вычисляемого выражения. Не забудьте, что он должен быть целочисленным. Предусмотреть вывод сообщения при вводе отсутствующего в перечне значения.

1. Создать программу, вычисляющую площадь фигуры на плоскости.

$$S=\begin{cases} \pi R^2, & при \quad k=1 \\ a\sqrt{3}/4, & при \quad k=3 \\ a^2, & при \quad k=4 \\ 3\sqrt{3}a/2, & при \quad k=6 \end{cases}$$

Значение k вводить с клавиатуры.

2. С клавиатуры ввести число y, обозначающее день недели. В зависимости от введенного

значения на экран вывести название дня недели. Предусмотреть напоминание о выходных днях.

3. С клавиатуры ввести число x, обозначающее номер месяца. В зависимости от введенного значения на экран вывести название сезона (лето, весна, осень, зима).

4. Создать программу, содержащую справочную информацию о значениях тригонометрических функций. По введенной с клавиатуры величине угла x, выраженной в градусах, программа должна выдавать соответствующие значения cos x и sin x.

градусы	0	30	45	60	90
sin x	0	1/2	$\sqrt{2}/2$	$\sqrt{3}/2$	1
cos x	1	$\sqrt{3}/2$	$\sqrt{2}/2$	1/2	0

5. С клавиатуры ввести число m, обозначающее номер месяца. В зависимости от введенного значения программа должна выводить на экран количество дней в месяце (без учета високосных года).

6. С клавиатуры ввести число m, обозначающее номер месяца. В зависимости от значения введенного числа программа должна выводить на знак по гороскопу: (1 - козерог, 2 - водолей, 3 – рыбы, 4 – овен, 5 – телец, 6 – близнецы, 7 – рак, 8 – лев, 9 – дева, 10 – весы, 11- скорпион, 12 – стрелец).

7. С клавиатуры ввести число y, обозначающее количество углов в правильном многоугольнике. В зависимости от введенного значения на экран вывести величину угла, выраженного в градусах.

8. Вычислить значение переменной A, в зависимости от величины B.

$$A=\begin{cases} 34x & B=10 \\ 25.x-x^2 & B=20 \\ 10x-45/37 & B=30 \\ 98(x-1) & B=40 \end{cases}$$

Значение x вводить с клавиатуры.

9. С клавиатуры ввести целое число (3, 4, 5, 6, 7, 8) в зависимости от значения которого на экран вывести название многоугольника, содержащего указанное число сторон, например 3 – треугольник.

10. С клавиатуры ввести целое число, означающее номер класса в школе, в зависимости от его значения на экран вывести следующие сообщения 1-4 – младший класс, 5-8 – средняя школа, 9-11 – старший класс.

Тема **Оператор цикла с предусловием**

Задание. Написать программы на языке C++ для вычисления значения функции $f(x)$. Значение аргумента x вводить с клавиатуры.

Методические рекомендации

В программе использовать циклический алгоритм. Не забудьте задать начальные значения для счетчика цикла и значению, подсчитываемому внутри него.

№ варианта	Задание	№ варианта	
1	$f(x)=\sum_{n=10}^{28}\frac{(3+x)^4}{n}$	6	$f(x)=3\prod_{n=20}^{30}\frac{x}{m^3-5}$
2	$f(x)=2\sum_{n=3}^{15}\frac{x^5}{n+1}$	7	$f(x)=\sum_{n=20}^{30}\frac{nx^2}{(2+n)}$

3	$f(x)=\prod_{k=5}^{22}(\cos^2 kx)$	8	$f(x)=\prod_{k=3}^{18}(k-x)^2/(k+x)^3$
4	$f(x)=\sum_{n=1}^{7}\frac{(-x)^{n-1}}{(2n+1)(x+1)}$	9	$f(x)=-\sum_{n=0}^{16}\frac{x+7}{(2n-1)}$
5	$f(x)=\prod_{k=17}^{35}k^3/(10-x^5)$	10	$f(x)=\sum_{n=12}^{26}\frac{x^3-xn}{(2n+x)}$

Тема **Оператор цикла с постусловием**

Задание. Вычислить и вывести на экран в виде таблицы все значения функции y на заданном интервале с шагом Δt. Составить блок-схему алгоритма.

Методические рекомендации

Таблица должна содержать заголовок и «шапку». В каждой строке таблицы должно содержаться значение аргумента и значение функции. Для выравнивания границ таблицы при выводе на экран использовать библиотеку iomanip.h. Значение констант задать в программе.

Варианты

1	$y=\begin{cases}at^2\ln t, & 1\le t\le 2\\ 1, & t<1\\ e^{at}\cos bt, & t>2\end{cases}$ a = -0.5, b = 2, $t\in[0;3]$, $\Delta t=0.15$	6	$y=\begin{cases}\pi t^2-7/t^2, & t<1.3\\ at^3+7\sqrt{t}, & t=1\\ \lg(t+7\sqrt{t}), & t>1.3\end{cases}$ a=1.5, $t\in[0.8;2]$, $\Delta t=0.1$
2	$y=\begin{cases}at^2+bt+c, & t<1.4\\ a/t+\sqrt{t^2+1}, & t=1.4\\ (a+bt)/\sqrt{t^2+1}, & t>1.4\end{cases}$ a=2.8, b=-0.2, c=4 $t\in[1;2]$, $\Delta t=0.05$	7	$y=\begin{cases}\pi t^2-7t^2, & t<1.4\\ at^3+7\sqrt{t}, & t=1.4\\ \ln(t+7\sqrt{\lvert t+a\rvert}, & t>1.4\end{cases}$ a=1.65, $t\in[0.7;2]$, $\Delta t=0.1$
3	$y=\begin{cases}1.5\cos^2 t, & t<2\\ 1.8at, & t=2\\ (2-t)+3tgt, & t>2\end{cases}$ a=2.3, $t\in[0.2;2.8]$, $\Delta t=0.2$	8	$y=\begin{cases}t\sqrt{t-a}, & t>a\\ t\sin at, & t=a\\ e^{-at}\cos at & t<a\end{cases}$ a=2.5, $t\in[1;5]$, $\Delta t=0.5$
4	$y=\begin{cases}bt-\lg bt, & bt<1\\ 1, & bt=1\\ bt+\lg bt, & bt>1\end{cases}$ b=1.5 $t\in[0.1;1]$, $\Delta t=0.1$	9	$y=\begin{cases}\sin at^2, & t>3.5\\ \lg t+\sqrt{t^2+1}, & t=3.5\\ \cos^2 t, & t<3.5\end{cases}$ a=2.8, $t\in[2;5]$, $\Delta t=0.25$
5	$y=\begin{cases}\lg(t+c), & t>1\\ a^3, & t=1\\ \sin^2\sqrt{\lvert at\rvert}, & t<1\end{cases}$ a=-20.8, c=4, $t\in[0.5;2]$, $\Delta t=0.1$	10	$y=\begin{cases}(\ln^3 t+t^2)/\sqrt{t+c}, & t<0.5\\ 1/t+\sqrt{t+c}, & t=0.5\\ \cos t+t\sin^2 t, & t>0.5\end{cases}$ c=2.2, $t\in[0.2;2]$, $\Delta t=0.2$

Тема **Одномерные массивы**

Задание. Написать программу на языке С++. Элементы массива ввести с клавиатуры.

Методические рекомендации

В программе использовать цикл с заданным числом повторений for. Предусмотреть подсказку для пользователя при вводе данных. Подготовить тестовые задания. Записать результаты тестирования в отчет

Варианты

1. В массиве A, содержащем 15 элементов, определить сумму положительных элементов, расположенных после минимального элемента данного массива.
2. В массиве B, содержащем 12 элементов, определить среднее арифметическое отрицательных элементов, расположенных после минимального элемента данного массива.
3. В массиве D, содержащем 14 элементов, определить среднее арифметическое элементов, расположенных после минимального элемента данного массива.
4. В массиве D, содержащем 14 элементов, определить разность между минимальным и максимальным элементами данного массива.
5. В массиве Y, содержащем 14 элементов, заменить минимальный элемент нулем, а максимальный увеличить в десять раз.
6. В массиве K из 15 элементов, найти сумму элементов, расположенных после минимального и стоящих на нечетных местах
7. В массиве C, содержащем 14 элементов, определить произведение элементов, удовлетворяющих условию a<c[i]<b и расположенных до максимального элемента этого массива. Значения переменных a и b вводить с клавиатуры.
8. В массиве A, содержащем 12 элементов, определить номер элемента, являющегося минимальным и заменить нулем все элементы, расположенные после него.
9. В массиве X из 15 элементов посчитать сумму неположительных элементов, расположенных до максимального элемента.
10. Посчитать произведение положительных элементов, расположенных после минимального элемента.

Тема Работа со строками

Методические рекомендации

Символьная строка является массивом типа char, последний элемент которого автоматически заполняется нуль-символом. Для работы с символьными строками воспользуйтесь функциями библиотеки string.h

Варианты

1. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Сравнить данные строки по длине. Если они не равны, присоединить к меньшей строке - большую. Определить количество слов в полученной строке и наименьшее из них вывести на экран.
2. Ввести две строки с клавиатуры. Посчитать в каждой из них количество гласных букв. В строке, содержащей большее число гласных, удалить все согласные буквы.
3. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Соединить их содержимое. Определить длину полученной строки. Вывести на экран первую половину полученной строки, удаляя из нее гласные буквы.
4. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Если их содержимое одинаково, оставить строки без изменения, иначе соединить содержимое строк. При этом первыми должны быть символы той строки, в которой первый символ по алфавиту раньше. Определить длину полученной строки.
5. Одну строку инициализировать в программе, другую – ввести с клавиатуры. В каждой из строк переместить первый символ в конец строки, после чего соединить строки, расположив вначале наибольшую из них по алфавиту. Определить в новой строке количество гласных букв.
6. В строке символов найти самое длинное и самое короткое слово. Сформировать новую

строку, расположив в ее начале самое короткое слово, затем самое длинное, а потом все остальные слова.

7. Определить в строке количество предложений. Если предложений более одного, копировать второе предложение в отдельную строку и вывести ее содержимое на экран. В новой строке первое слова поставить последним.

8. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Сформировать из них новую строку, чередуя в ней слова из заданных строк. Определить длину строки, количество в ней гласных.

9. Две строки инициализировать в программе, третью – ввести с клавиатуры. Сформировать из них новую строку по следующему алгоритму: сначала соединить строки в порядке возрастания их длины, затем исключить из нее первое и среднее слова. Посчитать количество слов в итоговой строке, начинающихся с гласной буквы.

10. Одну строку инициализировать в программе, другую – ввести с клавиатуры. Сравнить строки, и если они не равны, соединить их. Отредактировать вновь полученную строку, исключив множественные пробелы.

Тема **Двумерные массивы**

Задание. Написать программу на языке C++. Значения элементов массива вводить по желанию пользователя либо с клавиатуры, либо случайным образом.

Методические рекомендации

Размерности матрицы определить константами, что значительно упростит отладку программы. Для проверки правильности работы вывести на экран все промежуточные значения переменных. Осуществить вывод значений элементов исходного и измененного массива в виде таблицы. При форматировании вывода элементов матрицы использовать манипуляторы. Для реализации возможности ввода элементов матрицы различными способами использовать оператор switch.

Варианты

1. В матрице размерности 8 на 6 определить номер первого из столбцов, содержащих хотя бы один нулевой элемент. Найти сумму положительных элементов матрицы.

2. В матрице размерности 5 на 10 в каждой строке поменять местами максимальный и минимальный элементы. Заменить положительные элементы последней строки нулями.

3. В матрице размерности 8 на 8 определить сумму элементов в столбцах, не содержащих отрицательные элементы. Найти минимум среди элементов главной диагонали матрицы.

4. В матрице размером 6 на 9 определить сумму модулей его отрицательных элементов каждой строки. Заменить положительные элементы второй и третьей строк единицами.

5. В матрице размером 10 на 10 заменить отрицательные элементы матрицы их модулями. В измененной матрице найти произведение элементов, расположенных ниже главной диагонали.

6. В матрице размером 8 на 8 определить номера минимальных элементов в каждой второй строке. Посчитать произведение отрицательных элементов, расположенных выше главной диагонали.

7. В матрице размерности 12 на 8 в каждой третьей строке матрицы заменить минимальный элемент нулем, а максимальный сотней. Посчитать произведение элементов матрицы, по модулю не превосходящих 5.

8. В матрице размерности 8 на 9 определить произведение элементов в каждом втором столбце. Найти минимум среди всех элементов матрицы.

9. В матрице размером 9 на 9 определить количество строк, содержащих хотя бы один положительный элемент. Определить номер столбца, в котором содержится минимальный элемент матрицы.

10. В матрице размерности 8 на 8 определить сумму минимального и максимального из элементов. В каждой второй строке заменить положительные элементы нулями.

Тема **Функции**

Задание. В программе создать функцию, грамотно определив ее параметры и тип, продемонстрировать ее вызов. Обязательно использовать прототип функции.

Методические рекомендации

Помните, что в вызове функции количество фактических параметров и их типы должны совпадать с формальными параметрами.

Варианты

1. Определить периметр треугольника, заданного координатами вершин. Длину стороны треугольника вычислять в функции. Посчитать среднее арифметическое периметров трех треугольников, координаты которых ввести с клавиатуры.
2. Определить функцию нахождения объема конуса по известному радиусу основания и высоте. С помощью данной функции найти объемы трех усеченных конусов. Среди них определить наибольший объем.
3. Создать функцию нахождения скалярного произведения двух векторов в трехмерном пространстве. Использовать ее для нахождения среднего арифметического трех скалярных произведений произвольных векторов.
4. Вычислить значение $f = \dfrac{\max(a,b,c) * \min(a,c,d) - \max(b,c,d)}{\min(a,b,c)}$,

где *a, b, c, d* – некоторые значения, введенные с клавиатуры. При вычислении значения учитывать область допустимых значений функции. Для нахождения максимального и минимального значений использовать функции.

5. Создать функцию вычисления площади треугольника по трем заданным сторонам. Функция должна содержать проверку возможности построения треугольника. Продемонстрировать работу функции.
6. Создать функцию, определяющую принадлежит ли точка с заданными координатами (x_1 , y_1) уравнению прямой $y = kx + b$. Использовать функцию для определения из некоторого набора точек, лежащих на одной прямой.
7. Создать функцию нахождения факториала произвольного числа. С помощью ее посчитать 5!+(x+8)!/y! Значения переменных x и y ввести с клавиатуры.
8. По известным длинам сторон треугольника вывести на экран длины его высот в порядке убывания. В программе определить функцию нахождения длины одной высоты.
9. Создать функцию, определяющую по известным координатам трех вершин площадь ромба. Вывести на экран в порядке убывания площади трех, заданных случайным образом ромбов.
10. Создать функцию, находящую площадь треугольника по известным длинам двух сторон и углу между ними. Случайным образом задать четыре треугольника. Вывести на экран их площади в порядке возрастания.

Тема **Типы данных, определяемые пользователем. Структурный шаблон**

Задание. Структурный шаблон определить в соответствии с вариантом. Создать массив типа структурного шаблона. Продемонстрировать ввод данных для элементов массива и вывод их на экран в виде таблицы. При вводе данных обеспечить проверку правильности ввода. Реализовать поиск и вывод на экран информации по неполным данным

Методические рекомендации

При вводе данных количество записей в массиве не задавать явным образом. После ввода каждой записи программа должна выводить запрос, например такого вида: Вводить еще? Если да – введите 1, иначе – 0.

Проверка правильности ввода подразумевает проверку на достоверность количественных

значений, например, номер месяца не более 12, стоимость товара не может быть отрицательной, возраст человека не превышает 100 лет и пр. В случае неверного ввода, программа должна запросить повтор на ввод.

Варианты

1. Структурный шаблон включает: номер автобуса, фамилия и инициалы водителя, номер маршрута, дата действия техосмотра, номер автобуса в автопарке.
2. Структурный шаблон содержит сведения квитанции ЖКХ: ФИО владельца, адрес, площадь квартиры, текущие и предыдущие показатели счетчиков холодной и горячей воды, сумма и наличие платежа по месяцам. Сумма платежа рассчитывается программно. Тарифы в программе указаны как константы.
3. Структурный шаблон содержит сведения квитанции об оплате электроэнергии: ФИО владельца, адрес, текущие и предыдущие показатели счетчика сумма и наличие платежа по месяцам. Сумма платежа рассчитывается программно. Тариф в программе указан как константа.
4. Структурный шаблон содержит о заказах Пиццерии: название и стоимость, количество для каждого наименования, наличие скидки, ее величина, сумма заказа, рассчитанная программно, адрес доставки, дата. Величина скидки зависит от суммы заказа – 2% при сумме выше 2 тыс. руб., 5% - при сумме заказа выше 5 тыс. руб.
5. Структурный шаблон содержит сведения о комплектации ноутбука: модель, объем оперативной памяти, объем жесткого диска, название Операционной системы, дата истечения срока лицензии, стоимость.
6. Структурный шаблон должен содержать фамилию, имя студента; номер группы и курса; пять оценок, полученных на экзаменах с указанием названия предметов.
7. Структурный шаблон должен содержать фамилию, имя студента; номер группы; пять оценок, полученных на экзаменах с указанием названия предметов, размер стипендия в руб. Стипендия назначается программно. Студентам, получившим хотя бы одну тройку или двойку, стипендия не назначается. Студентам, получившим все пятерки назначить стипендию в 4000 руб., пятерок больше, чем четверок – 3700 руб., все четверки – 3500 руб.
8. В структурном шаблоне должны содержаться следующие сведения: Фамилия, Имя, дата рождения, номер телефона.
9. Структурный шаблон должен содержать пункт назначения, номер рейса, дату и время вылета, время в полете, стоимость билета, наличие билетов кассе. Время прибытия в пункт назначения высчитывается программно.
10. Структурный шаблон должен содержать: наименование ; название модели; характеристики памяти; скорость работы процессора; количество.

Тема **Текстовые файлы**

Задание. Согласно варианту напишите программу для обработки текста.

Методические рекомендации

С помощью текстового редактора Блокнот создать файл, содержащий текст, длина которого не превышает 1000 символов (длина строки не превышает 70 символов). Имя файла должно иметь расширение txt. Написать программу, которая выводит содержимое файла на экран и выполняет действия в соответствии с вариантом. Выделение фрагмента в тексте выделить произвольными символами (звездочками, слэшем и т.п)

Варианты

1. Определить количество предложений в тексте; по нажатию произвольной клавиши выделить третье предложение.
2. Определить количество слов в тексте; по нажатию произвольной клавиши выделить десятое слово.

3. Определить количество слов в тексте, начинающихся с гласной буквы; выделить шестое слово в тексте по нажатию произвольной клавиши.

4. Определить количество слов в тексте, у которых первый и последний символы совпадают; выделить по нажатию произвольной клавиши первое из найденных слов.

5. Определят количество предложений, начинающихся с гласной буквы; по нажатию произвольной клавиши выделить первое из найденных предложений.

6. Отредактировать текст, удаляя лишние символы пробелов между словами; по нажатию произвольной клавиши выделить первое предложение текста.

7. Отредактировать текст, заменяя буквы «о» на «а»; по нажатию произвольной клавиши выделить первое исправленное слово.

8. Отредактировать текст, заменяя двойную букву «н» на одинарную. По нажатию произвольной клавиши выделить первое исправленное слово.

9. Определить количество восклицательных предложений; выделить первое найденное предложение по нажатию произвольной клавиши.

10. Подсчитать количество слов в самом длинном предложении; выделить по нажатию произвольной клавиши найденное предложение.

Тема **Бинарные файлы**

Задание. В каждом из заданий исходный файл формировать программно. Количество элементов исходного файла неизвестно.

Методические рекомендации

Целесообразно для создания бинарного файла создать отдельную программу. Числа могут вводиться либо пользователем, либо счетчиком случайных чисел. Количество чисел в файле задавать случайным образом, и оно должно быть неизвестно в основной программе, т.е. цикл выполняется при условии «пока не конец файла». Для проверки правильности работы программы на экран выводить и первоначальное содержимое файла, и результат.

Варианты

1. В файле содержится некоторое количество чисел. Сформировать из них матрицу, содержащую 4 столбца. Недостающие элементы последней строки задать равными нулю. Вывести на экран матрицу в общепринятом виде, ее размерность и суммы элементов главной диагонали и побочной диагонали.

2. В файле содержатся числа. Сформировать из них матрицу, содержащую пять элементов в строке. Числа неполной строки отбросить. Вывести на печать матрицу в общепринятом виде и посчитать сумму элементов, содержащихся в предпоследней строке.

3. Из цифр, содержащихся в файле сформировать новый файл следующим образом, сначала расположить трехзначные цифры, затем двузначные. Найти максимальное и минимальное число в полученном наборе.

4. В файле содержатся числа, сформировать из них матрицу 4Х4, взяв первые 8 чисел сначала файла, а остальные – с конца файла. Посчитать произведение элементов под главной диагональю матрицы.

5. В файле содержатся числа. Сформировать из них матрицу, содержащую пять элементов в строке. Первые десять элементов взять из начала файла, последние пять с его конца. Вывести на печать матрицу в общепринятом виде и посчитать сумму элементов, содержащихся в последней строке.

6. Описать структурный шаблон для представления комплексного числа, содержащий поля: вещественная и мнимая часть. Написать программу, записывающую данные о комплексных числах в бинарный файл и отображающую на экран информацию о числах, модуль которых меньше десяти.

7. Описать структурный шаблон для представления комплексного числа, содержащий по-

ля: вещественная и мнимая часть. Написать программу, записывающую данные о комплексных числах в бинарный файл и отображающую на экран информацию о числах, аргумент которых меньше $\pi / 2$.

8. Описать структурный шаблон для представления комплексного числа, содержащий поля: вещественная и мнимая часть. Написать программу, записывающую данные о комплексных числах в бинарный файл и отображающую на экран информацию о числах, мнимая и вещественная части которых имеют одинаковый знак.

9. В файле заданы множество точек A и точка d вне его. Найти все пары точек, лежащих с точкой d на одной прямой.

10. В файле задано множество точек на плоскости. Подсчитать количество точек, лежащих внутри указанной окружности. Окружность задавать с клавиатуры координатами центра и радиусом. Вывести на экран координаты найденных точек.

Тема **Объектно-ориентированное программирование. Конструкторы**

Задание. В соответствии с вариантом составить описание класса. В классе определить не менее трех конструкторов различного вида. Объявить объекты класса и продемонстрировать работу всех его методов класса.

Методические рекомендации

Выбирая набор полей класса необходимо помнить, что он должен однозначно задавать все характеристики реального прототипа объекта. Для выполнения принципа инкапсуляции поля должны являться закрытыми элементами. Интерфейс представления элементов класса описывается его методами.

Варианты

1. Организовать класс *точка,* содержащий координаты точки на плоскости и методы: вывода координаты точки, нахождения расстояния от точки до начала координат, определения принадлежности точки некоторой прямой (коэффициенты уравнения прямой задать как параметры метода).

2. Организовать класс *точка,* содержащий координаты точки в пространстве и методы: вывода координаты точки, нахождения расстояния от точки до начала координат, метод определения принадлежности точки некоторой прямой (коэффициенты уравнения прямой задать как параметры метода).

3. Организовать класс *дробь,* содержащий методы вывода дроби в общепринятом виде и функцию выделения целой части.

4. Организовать класс *дробь,* содержащий методы вывода дроби в общепринятом виде и функцию приведения дроби к несократимому виду. Перегрузить операцию деления двух дробей.

5. Описать класс *вектор* на плоскости, содержащий координаты его начала и конца и методы вывода координат на экран и нахождения длины.

6. Описать класс *вектор* в пространстве, содержащий координаты его начала и конца и методы вывода на экран значений координат и нахождения длины.

7. Создать класс *комплексное число* с методами нахождения аргумента комплексного числа и вывода числа на экран в общепринятом виде.

8. Создать класс *комплексное число* с методами нахождения модуля комплексного числа и вывода числа на экран в общепринятом виде.

9. Описать класс *одномерный массив*, содержащий его элементы и их количество, а также метод вывода всех значений на экран и нахождения среднего арифметического всех элементов и количества отрицательных и положительных элементов.

10. Описать класс *одномерный массив*, содержащий его элементы и их количество, а также метод вывода всех значений на экран и нахождения минимального и максимального его элементов.

Тема **Наследование. Перегрузка операций**

Задание. Измените программу предыдущего задания, организовав на основе существующего класса производный класс, дополнительно включив в него операции сравнения (больше, меньше, равно) и математические операции, применив их к объектам. Продемонстрируйте работу всех методов базового и производного классов.

Методические рекомендации

Необходимо помнить, что хорошим стилем программирования является вызов конструкторов базовых классов из конструкторов производных классов. Перегруженные операции должны иметь тот же смысл и то же количество аргументов, что и исходные. В бинарные операции второй объект передавать в качестве параметра.

Тема 18 **Полиморфизм: виртуальные функции и абстрактные классы**

Задание. В соответствии с вариантом организовать классы. В производных классах не забыть объявление конструкторов и функций вывода данных на печать. Продемонстрировать работу всех методов производных классов.

Варианты

1. Описать абстрактный класс Сотрудник. Класс должен содержать характеристики: имя, фамилия, должность, отдел; функцию вывода всех данных на экран. На его основе реализовать классы Администратор, Начальник отдела, Руководитель организации. Функцию вывода данных на экран реализовать в классах различным способом, чтобы руководителям подразделений могли быть доступны данные только об их подчиненных, а рядовой сотрудник мог видеть только себя, администратор всех, кроме руководителя организации.
2. Создать абстрактный класс *средство передвижения*. На его основе реализовать классы *самолет*, *машина*, *корабль*. Все классы должны хранить параметры средств передвижения: скорость, расход топлива, наименование производителя, год выпуска, метод вывода на экран всех данных, определения срока службы. Индивидуально для самолета указать высоту и максимальную дальность полета, для машины – объем двигателя, для самолета и корабля – количество посадочных мест, для корабля – водоизмещение.
3. Создать абстрактный класс *линия второго порядка* с полями – коэффициенты уравнения второго порядка. На его основе создать классы окружность, парабола (с методом нахождения директрисы), гипербола, эллипс (с методом нахождения эксцентриситета). Предусмотреть виртуальные методы нахождения центра (вершин или фокусов) линий и функции вывода данных на экран.
4. Описать абстрактный класс *фигура на плоскости*. На его базе создать классы круг, треугольник, прямоугольник. Предусмотреть виртуальные методы создания объектов, вычисление площади фигур, периметра для треугольника и прямоугольника, длины окружности – для круга.
5. Создать абстрактный класс *правильный многоугольник*. На его основе создать классы треугольник, квадрат, восьмиугольник. Предусмотреть виртуальные методы создания объектов, вычисления их периметра, площади, величины угла.
6. Создать абстрактный класс *правильный многогранник* с полями длина ребра и число ребер. На его основе создать классы тетраэдр, куб, октаэдр (восьмигранник). Предусмотреть виртуальные методы создания объектов, вычисления их площади поверхности и объема.
7. Создать абстрактный класс *вектор*. На его основе создать классы вектор на плоскости, в трехмерном пространстве, в пятимерном пространстве. Предусмотреть виртуальные методы создания объектов, вычисления их длины, вывода на экран их координат.
8. Создать абстрактный класс *человек с* полями год рождения, пол, фамилия, имя. На его основе создать классы школьник (с указанием номера школы и класса), студент (специальность,

курс), преподаватель (стаж работы, должность). Предусмотреть виртуальный метод вывода данных на экран и вычисления возраста с указанием у молодежи совершеннолетний или нет, у взрослых пенсионер или нет.

9. Создать абстрактный класс *ЭВМ с* полями модель, тактовая частота процессора, объем оперативной памяти. На его основе создать классы Компьютер (с указанием характеристик монитора), суперкомпьютер (указать количество процессоров и их характеристики), Компьютерная сеть (с указанием топологии сети, используемого стандарта, IP-адресов рабочих станций). Предусмотреть виртуальный метод вывода данных на экран.

10. Описать абстрактный класс *трехмерная фигура*. На его базе создать классы цилиндр, конус, пирамида. Предусмотреть методы создания объектов, вычисление площади поверхности фигур, объема, площади основания.

Тема **Событийное программирование в C#**

Событийно-управляемое программирование

В основу ОС Windows положен принцип *событийного управления.* Это значит, что и сама система, и приложения после запуска ожидают действий пользователя и реагируют на них заранее заданным образом. Любое действие пользователя (нажатие клавиши на клавиатуре, щелчок кнопкой мыши, перемещение мыши) называется *событием.*

Событие воспринимается Windows и преобразуется в *сообщение* запись, содержащую необходимую информацию о событии (например, какая клавиша была нажата, в каком месте экрана произошел щелчок мышью). Сообщения могут поступать не только от пользователя, но и от самой системы, а также от активного или других приложений. Определен достаточно широкий круг стандартных сообщений, образующий иерархию, кроме того, можно определят собственные сообщения.

Сообщения поступают в общую очередь, откуда распределяются по очередям приложений. Каждое приложение содержит *цикл обработки сообщений*, которое выбирает сообщение из очереди и через операционную систему вызывает подпрограмму, предназначенную для его обработки. Таким образом, Windows-приложение состоит из главной программы, содержащей цикл обработки сообщений, инициализацию и завершение приложения, и набора *обработчиков событий.*

Среда Visual Studio.NET содержит удобные средства разработки Windows-приложений, выполняющие вместо программиста рутинную работу создание шаблонов и заготовок обработчиков событий.

Задание

Написать Windows-приложение, заголовок главного окна которого содержит Ф. И. О., группу и номер варианта.

В приложение включить реализацию класса из второй лабораторной работы.

Компоненты формы выбрать самостоятельно.

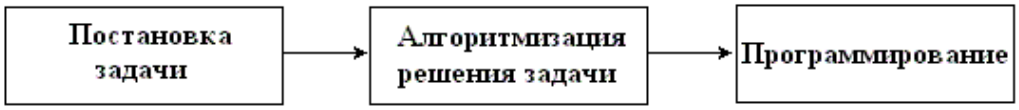
МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Алгоритмы и алгоритмизация

Возможности компьютера как технической основы обработки данных связаны с используемым программным обеспечением (программами).

Программа – упорядоченная последовательность команд (инструкций) компьютера для решения задачи.

Процесс создания программ можно представить как последовательность действий, представленных на рисунке.



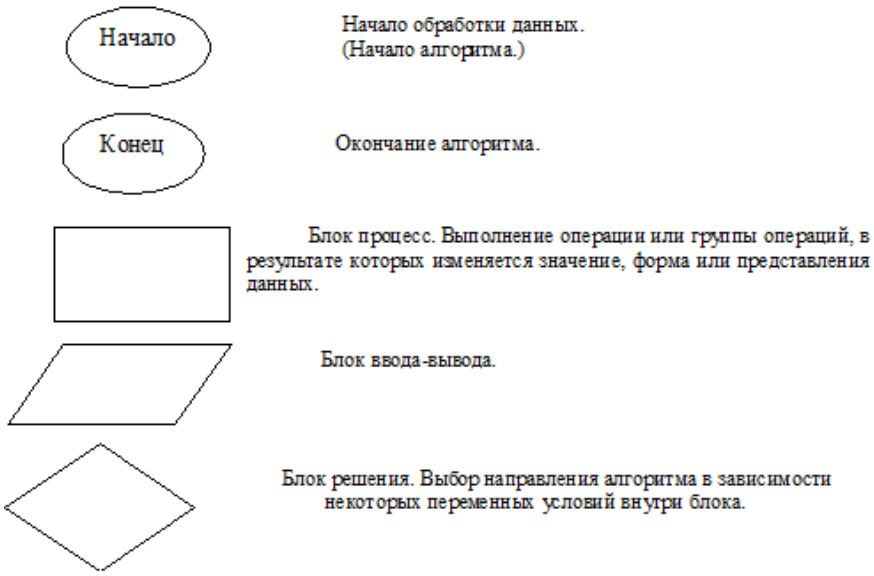
Постановка задачи – это точная формулировка решения задачи на компьютере с описанием входной и выходной информации.

Алгоритм – система точно сформулированных правил, определяющая процесс преобразования допустимых исходных данных (входной информации) в желаемый результат (выходную информацию) за конечное число шагов.

Программирование – теоретическая и практическая деятельность с созданием программ, осуществляющая перевод алгоритма решения задачи на язык программирования.

Существует несколько способов представления алгоритмов: словесный, графический и др. Один из них – представление в виде блок-схем.

Блок-схема алгоритма – это графическое представление алгоритма в виде геометрических фигур и стрелок. Основные блоки, используемые в блок-схемах:



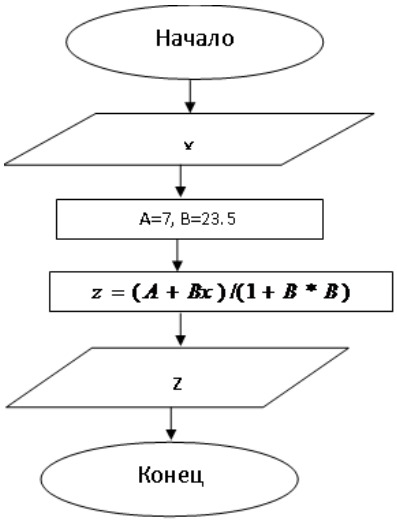
Особенностью базовых конструкций является то, что любая из них имеет только один вход и один выход, поэтому конструкции могут вкладываться друг в друга произвольным образом, например, цикл может содержать следование двух ветвлений, каждое из которых содержит вложенные циклы.

ЛИНЕЙНЫЙ АЛГОРИТМ

Линейным называется алгоритм, в котором все действия последовательно выполняются друг за другом.

Составить блок-схему алгоритма вычисления значения

$z = \dfrac{A + Bx}{1 + \hat{A}^2}$, где A=7, B=23.5 – константы, x – переменная вводимая с клавиатуры.



Задание

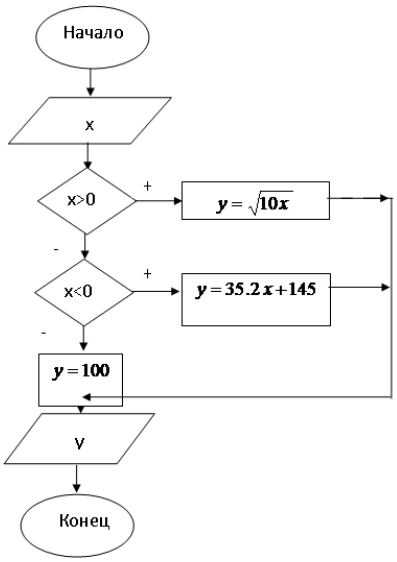
Составить блок-схему алгоритма вычисления значений величины:

1. $y(z) = z^3 - 8, \quad z(x) = \dfrac{125x + 321}{x^2 + 1},$
2. $h(a) = 38a\ - 123a / 7,\ q(a) = \sin(a) - h(a)$
3. $r(b) = 294b - \sin b + \cos 2b, \quad b = 1 - 7c$

РАЗВЕТВЛЯЮЩИЙСЯ АЛГОРИТМ

Алгоритм является разветвляющимся, если в нем предусмотрено несколько направлений. Пример. Составить блок-схему алгоритма для вычисления значения функции:

$$y = \begin{cases} 35.2x + 145, & x < 0 \\ 100, & x = 0 \\ \sqrt{10x}, & x > 0 \end{cases}$$



.

Задание

I. Составить блок-схему алгоритма вычисления заданной величины по введенному с клавиатуры аргументу.

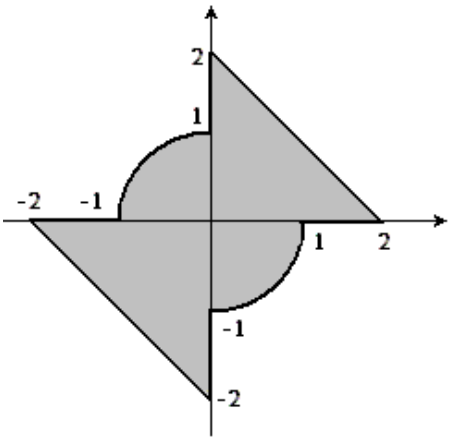
1. $$\upsilon(t)=\begin{cases} t^{-2}+t+3, & 25 \le t < 50 \\ t+25, & 10 \le t < 25 \\ 0, & 0 \le t < 10 \\ t^3-100, & -50 \le t < 0 \end{cases}$$

2. $$z(x)=\begin{cases} \sqrt{x^3-15}, & x \ge 3 \\ x+25, & x < 3 \end{cases}$$

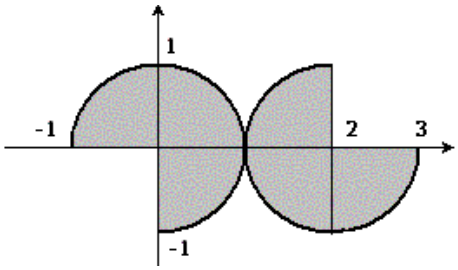
3. $$r(k)=\begin{cases} (k+25)/(k-25), & k > 25 \\ k+25, & 0 \le k < 25 \\ (k-25)/(k+25) & -25 < k < 0 \end{cases}$$

II. Составить алгоритм программы, которая по введенному значению координат точки определяет, попадает ли точка в мишень, представленную рисунком:

1.



2.



Циклический алгоритм

Циклический алгоритм используют для организации многократно повторяющихся вычислений. Циклический алгоритм существенно сокращает объем программы.

Любой цикл состоит из *тела цикла* (действий, которые повторяются несколько раз), начальных установок, модификации параметра цикла и проверки условия продолжения выполнения цикла.

На схеме изображены типовые составляющие циклического алгоритма.

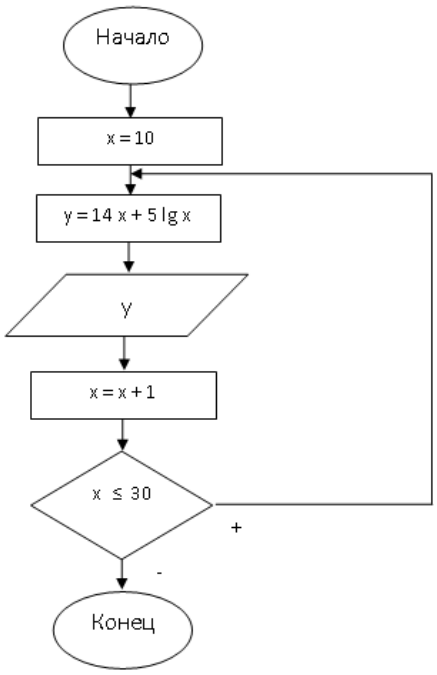
Один повтор выполнения операторов тела цикла называется *итерацией*. Проверка условия выполняется на каждой итерации.

Переменные, изменяющиеся в теле цикла и используемые при проверке условия продолжения цикла, называются *параметрами цикла*.

Начальные установки используются для того, чтобы до входа в цикл задать значения переменным, которые в нем используются, в том числе и начальные значения параметров цикла.

Цикл завершается, если условие его выполнения становится ложным.

Составить блок-схему алгоритма вычисления $y = \sum_{x=10}^{30} (14x + 5\lg x)$.



Задание

I. Посчитать значение функции

1. $y = \sum_{1}^{14} (x^2 - x)/2$
2. $y = 25\sum_{5} \sin^2 x$
3. $y = \sum_{3}^{20} (9x - 5)^2$

II. Посчитать все значения функции на указанном интервале

1. $y = t^2 - 5t + 1 \qquad t \in [2, 12], \qquad \Delta t = 0.5$
2. $y = \begin{cases} \sin 2t, & t < 0 \\ \cos 3t, & t \geq 0 \end{cases} \quad t \in [-4, 6], \quad \Delta t = 1$
3. $y = \begin{cases} 2t - 5, & t > 0 \\ 0, & t = 0 \\ |t|, & t < 0 \end{cases} \qquad t \in [-10, 45], \ \Delta t = 5$

Программная реализация алгоритмов разветвляющейся и циклической структуры

Пример 1. Программа печатает таблицу значений функции $y = x^3 - x$ во введенном диапазоне.

```
#include <iostream.h>
#include <iomanip.h>
int main ( )
{ float x1, xn, d;
  cout << " введите диапазон и шаг изменения x"<< endl;
  cin >> x1 >> xn >> d;
  cout << "|        x        |        y      |" << endl;     // шапка таблицы
  float x = x1;
  while ( x <=  xn )
        { cout << "|" << setw(6) << setprecision(3) << x << "|";
          cout << "|" << setw(6) << setprecision(3)<< x*x*x – x << "|" << endl;
          x+=d;
        }
}
```

Пример 2. Программа угадывания загаданного числа.

```
#include <iostream.h>
#include <stdlib.h>
int main ( )
{ float x, y;
  y = rand( )%10;
do
  { cout << " введите произвольное число меньшее 10"<< endl;
    cin >> x;
    if ( x = = y) {cout <<" вы угадали!"; break; }
   else          if  ( x< y ) cout<< "введите меньшее"<<endl;
         else     cout<< "введите большее"<<endl;

  } while (x!=y);
}
```

Массивы

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой переменных одного типа требуется выполнить различные действия, им дают одно имя и отличают по порядковому номеру. Это позволяет записывать множество операций через циклы. Конечная именованная последовательность однотипных величин называется массивом. Для создания массива компилятору необходимо знать тип данных, количество элементов в массиве и требуемый класс памяти. Массивы могут иметь те же типы данных, что и простые переменные.

Примеры:

```
int array[45];          //массив из 45 элементов целого типа с именем array
double rt[12];          // массив rt, состоящий из 12 элементов типа double
const int ARRAY_SIZE=90;
float alp[ARRAY_SIZE]; // вещественный массив alp из 90 элементов
```

Размерность массива предпочтительнее задавать с помощью именованных констант, как это сделано в примере, так как при таком подходе для ее изменения во всей программе достаточно изменить значение константы в одном месте.

Возможна также инициализация массива при его объявлении.

```
int a[5]={4, 90, 71, 45, 3};
```

При инициализации допускается использование пустых скобок в объявлении массива, и тогда компилятор сам определяет размерность массива. Например,

float x[ ]= {4, 8, 19};

В памяти компьютера элементы массива располагаются последовательно друг за другом. Место в памяти для хранения элементов массива выделяется компилятором сразу после обработки его объявления.

Так объявление вида

float y[5]= {0.7, 1.9, 8.4, 3.1 };

выделит место для расположения пяти элементов типа float. Так как выполнена инициализация четырех первых элементов, они сразу займут свое место в памяти компьютера, а ячейки памяти для пятого элемента пока останутся свободными:

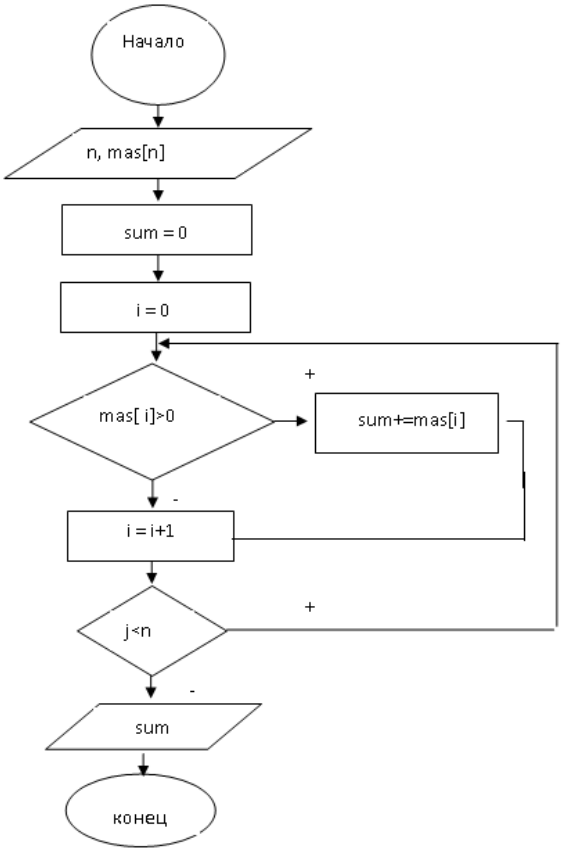
0. 7	1. 9	8. 4	3. 1	
y[0]	y[1]	y[2]	y[3]	y[4]

Для доступа к элементу массива после его имени указывается номер элемента (индекс), заключенный в квадратные скобки. Нумерация элементов массива начинается с нуля. Следовательно, диапазон значений для объявленного в примере массива x лежит в пределах от 0 до 2. Следовательно,

x[0]=4 x[1]=8 x[2]=19

Работа с элементами массивов осуществляется на основе циклических алгоритмов.

Пример: Составим алгоритм подсчета суммы положительных элементов массива, содержащего 10 значений.




```
#include<iostream.h>
int main ( )
{const int n=10;
  int i, mas[n], sum=0;
  cout << " введите" << n << "чисел" <<endl;
                    // ввод элементов массива с клавиатуры
```

```
  for ( i=0; i < n; i++) cin >> mas[i];
  for ( i=0; i < n; i++)
        if ( mas[i] > 0)     sum+= mas[i];
  cout << ”сумма положительных элементов равна “<<s<<endl;
}
```

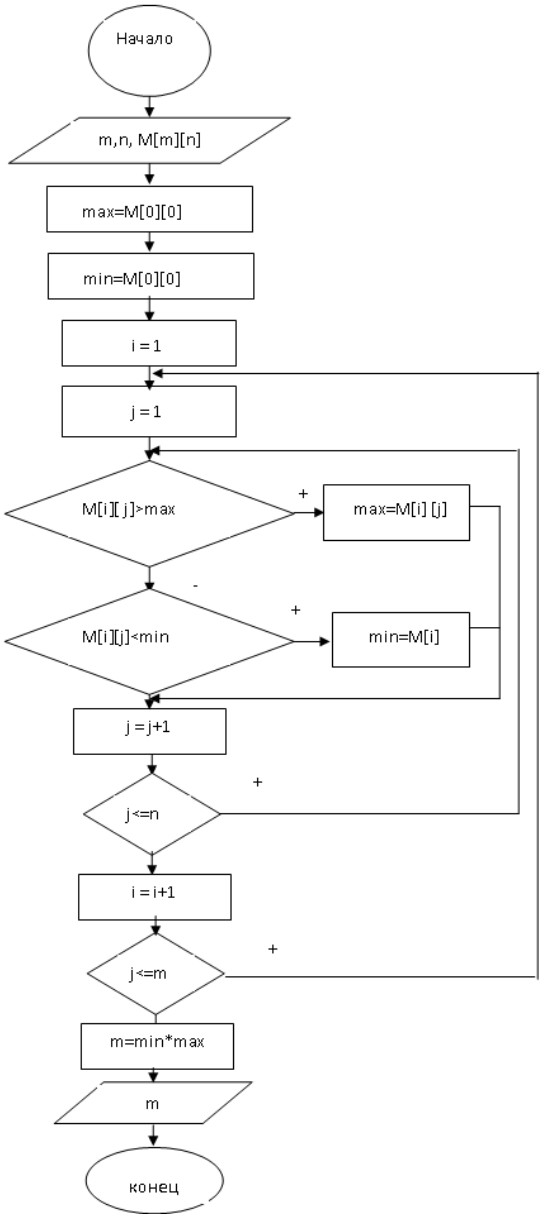
Для улучшения эффективности алгоритма ввод элементов массива и подсчет их суммы можно объединить в один цикл.

Для объявления многомерного массива необходимо после имени массива задать несколько размеров, заключенных в скобки. Двумерный массив в С++ представляет собой массив одномерных массивов.

float dam[4] [5];

В этом случае массив будет содержать 4 строки и 5 столбцов.

Пример. В массиве определить произведение максимального и минимального его элементов. Составим блок-схему алгоритма и реализуем алгоритм программно.




```
#include <iostream. h>
#include <stdlib.h>
int main ( )
```

```
{
const int n = 6; const int m = 8;
int M[m][n], i, j, min, max;
for (  i = 0; i <m; i++ )
for (   j = 0; j < n; j++)
          M [ i ][ j ]=rand( )%10 - 10;  //задание значений
for (  i = 0; i < n; i++ )                              // вывод на экран полученной матрицы
          {         for ( int  j = 0; j < n; j++)       cout << M [ i ][ j ]<< “  “;
          cout << endl;
          }
min = M[0] [0], max = M[0] [0],;
for (  i = 1; i < m; i++ )
          for (   j =1; j < n; j++)
          if ( M[ i ][ j ]<min)  min = M[ i ][ j];
          else if ( M[ i ][ j ]>max)  max = M[ i ][ j];
 cout<< “значение произведения минимума на максимум ”<<min*max;
}
```

Задание

Составить алгоритм и программу на языке С++ :

1. Определите номер максимального элемента одномерного массива.
2. Найти произведение отрицательных элементов одномерного массива.
3. В одномерном массиве поменять местами максимальный и минимальный элементы
4. Посчитать среднее арифметическое отрицательных элементов, расположенных под главной диагональю матрицы размерности 5 на 5.
5. В каждом третьем столбце матрицы размерности 5 на 7 определить произведение элементов
6. В каждой строке матрицы размерности 6 на 8 поменять местами максимальный и минимальный элементы.

Функции

Компьютерная программа состоит из команд, которые сообщают компьютеру, что он должен делать. Задача большинства программ заключается в распознавании данных, их обработке и вывода их на экран.

Любая программа, написанная на языке С++, предполагает последовательность выполнения нескольких функций, причем одна из них обязательно должна носить имя main( ). Выполнение программы всегда начинается с выполнения ряда операторов этой функции. Операторы языка отделяются друг от друга точкой с запятой.

Описание функции состоит из заголовка и тела.

Заголовок <тип результата> main (список аргументов) имя функции

{ *Тело функции* }

После заголовка функции точка с запятой не ставится. Тело функции может содержать любое количество операторов языка.

Если тип результата не указывается, предполагается, что функция возвращает (передает в вызываемую функцию) значение типа int. Функция не обязательно возвращает значение – она просто может выполнять какие-либо действия. Если функция не возвращает результат – в заголовке перед ее именем указывается ключевое слово void (пустой).

Аргументы – значения, которые можно передавать в функцию, их количество аргументов может быть любым. Наличие списка аргументов необязательно, но круглые скобки всегда должны присутствовать после имени функции.

Часто употребляется описание типов аргументов сразу при их объявлении внутри круглых

скобок. Если аргументов несколько, их описания (тип и имя) разделяются запятыми. Если функции не передаются величины, то вместо списка аргументов можно задавать ключевое слово void. Локальные переменные, отличные от аргументов, описываются внутри тела функции. Процесс использования функции называется вызовом (или запуском) функции. Вызывать функцию можно сколько угодно раз. Функция должна быть определена до того, как начнется ее использование. Для вызова функции, ничего не возвращающей, достаточно написать ее имя, в круглых скобках указать значения фактических параметров – значений аргументов, которые непосредственно участвуют в работе функции. Если функция не имеет аргументов, при ее вызове также ставятся пустые круглые скобки.

Все функции в языке C++ равноправны: каждая из них (даже main) может быть вызвана любой другой функцией. Функция может также вызывать самое себя (явление рекурсии). Количество рекурсивных вызовов не ограничивается, а определяется лишь возможностями компьютера.

Программа, демонстрирующая применение рекурсии при вычислении факториала.

```
#include <stdio. h>
double factorial (int number);          // объявление прототипа функции factorial
int main ( )
{
int n;
double result;
printf (“Введите число\n”);
scanf (“%d”, &n);
result = factorial(n);
printf(“факториал %d равен %15.0f\”, n, result);
return (0);
}
double factorial (int number)   // описание функции factorial
{
if (number<=1)                  return (1.0);
else return (number*factorial (number – 1));
}
```

Задание

1. Определить периметры двух треугольников, заданных координатами их вершин. Длину стороны треугольника вычислить в функции.
2. Вычислить среднее арифметическое объемов шаров с радиусами r_1, r_2, r_3. В программе выделить функцию нахождения объема шара.
3. Заданы три конуса (радиусы основания и высота). Определить конус с наибольшим объемом. В программе выделить функцию нахождения объема конуса.
4. Создать функцию вывода на экран таблицы умножения для числа по запросу пользователя.
5. Создать функцию, определяющую, принадлежит ли точка с заданными координатами уравнению прямой $y = kx + b$. Использовать функцию для определения из набора точек.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Самостоятельная работа по дисциплине включает изучение учебной литературы, подготовка к выполнению лабораторных заданий и отчетов по их выполнению, а также подготовку к тестированию по теоретическим разделам и непосредственно к зачету и экзамену.

Задания к лабораторным работам выдаются заранее, как правило, на первом занятие текущего семестра, и для их успешного их выполнения необходимо предварительное освоение теоретического материала и разбор, приведенных на лекции примеров программ, проработка алгоритма решения разобранных задач и составление собственных алгоритмов. Для этого наряду с конспектами можно воспользоваться учебно-методическим обеспечением для самостоятельной работы, указанным в рабочей программе, и самопроверкой с помощью тестовых заданий, размещенных там же.

Для подготовки к выполнению лабораторных работ и повторения, усвоения (изучения пропущенного) теоретического материала студентам рекомендуется самостоятельно организовать по месту проживания дополнительное рабочее место, оборудованное персональным компьютером, подключённым к сети Интернет, и установленным программным обеспечением, необходимым для разработки программ и указанном в рабочей программе. Общие методические рекомендации по составлению программ на языке программирования дополнительные рекомендации по каждой теме представлены в предыдущих разделах.

В отчете по выполнению индивидуального варианта заданий должны содержатся следующие сведения: формулировка задания, входные и выходные данные, текст программы, тестовые (контрольные) значения входных данных и рассчитанные выходные данные.

Итоговый контроль – зачет и экзамен проводятся на основании перечней вопросов, представленных в рабочей программе.

Экзамен проводится по билетам. Билет включает два теоретических вопроса и задачу. Ответы на поставленные вопросы студент дает после предварительной подготовки. Преподаватель имеет право задать дополнительные вопросы, если ответ дан неполный или затруднительно однозначно оценить ответ.

Подготовка к зачету(экзамену) заключается в изучении и тщательной проработке студентом конспектов по всем видов занятий в соответствии с перечнем вопросов к зачету(экзамену), представленном в рабочей программе дисциплины. При подготовке к зачету рекомендуется использовать конспекты лекций, рекомендованную в рабочей программе литературу, ЭВМ и все теоретические знания, и практические навыки, полученные во время проведения лабораторных работ.

СОДЕРЖАНИЕ