**ЛЕКЦИЯ 1. ВВЕДЕНИЕ. ТИПЫ. ЗНАЧЕНИЯ. УПРАВЛЯЮЩИЕ СТРУКТУРЫ**

Содержание

[1 Основы программирования 1](#_Toc145522236)

[1.1 Модель обработки информации на компьютере 1](#_Toc145522237)

[1.2 Ключевые понятия и определения 2](#_Toc145522238)

[1.3 Значения и типы 8](#_Toc145522239)

[1.4 Константы и переменные 9](#_Toc145522240)

[1.5 Выражения и операторы 9](#_Toc145522241)

[1.6 Советы профессионала 10](#_Toc145522242)

[1.7 Примеры простых программ 13](#_Toc145522243)

# Основы программирования

* 1. Модель обработки информации на компьютере

Обработка информации – это практическая реализация некоторой функции **F**, которая отображает множество данных **D** во множество возможных результатов R.

**F** – произвольная функция, которую надо «вычислить»: перевод текста с русского на английский, нахождение максимума, расчет траектории ракеты, построение оптимального плана и т. д.

Чтобы выполнить обработку информации на компьютере, необходимо располагать тремя физическими представлениями (рисунок 1): **D'** – физическое представление данных **D; R'** – физическое представление результатов **R**; **F'** – физическое представление функции обработки **F**.



Рисунок 1 – Модель обработки информации на компьютере

Таким образом, обработка информации на компьютере включает в себя следующие процессы.

**Ввод данных** – чтение данных с устройства ввода (например, клавиатура) в оперативную память компьютера. При этом данные из «внешнего» представления кодируются в представление на уровне компьютера.

**Вычисление** – выполнение последовательности действий, которые необходимы для преобразования данных в результаты. Данная последовательность должна быть описана таким образом, чтобы ее мог выполнить процессор компьютера.

**Вывод данных** – отображение (запись) полученных результатов на устройстве вывода (например, дисплей). При этом результаты из представления на уровне компьютера преобразуются к виду, понятному человеку.

* 1. Ключевые понятия и определения

**Программа** – это совокупность данных, результатов и автоматизированного преобразования такого, что каждому **x** из множества данных **D** ставится в соответствие результат **y**, принадлежащий множеству результатов **R**.

В зависимости от уровня представления данных и функции автоматизированного преобразования информации говорят об исходной программе (данные **D, R** и функция **F**) и машинной программе (данные **D', R'** и функция **F'**). Описание *исходной программы* выполняется на языке программирования высокого уровня (например, C++), описание *машинной программы* – на языке низкого уровня (например, Assembler).

**Язык программирования** – это система описания программ, достаточно близкая к человеку, чтобы программу можно было легко написать, понять и изменить, но в то же время достаточно строго определенная, чтобы ее мог выполнить компьютер.

Что делает язык программирования высокого уровня? Он освобождает программу от значительной доли необязательной сложности. Исходная (абстрактная) программа состоит из концептуальных конструкций: операций, типов данных, последовательностей и других абстрактных компонентов. Машинная программа связана с битами, регистрами, условиями, переходами, каналами, дисками и пр.

**Компилятор** – это специальная программа, которая переводит текст исходной программы, написанной на языке программирования, в последовательность команд процессора. Компилятор создает машинную программу на языке процессора, эквивалентную исходной программе.

**Синтаксис языка программирования** – это совокупность правил, которые определяют правильные последовательности синтаксических элементов программы.

Синтаксическое правило может быть записано как в текстовом представлении (например, нормальная форма Бэкуса – **НФБ**), так и в графическом (например, синтаксический граф). И в том и в другом случае для задания правила используются *терминальные символы,* определенные в языке программирования,и *нетерминальные,* вспомогательные символы (заключаются в угловые скобки – **< >**).

**Пример НФБ: <цифра>::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

Это правило читается: «Цифра по определению есть 0 или 1 или 2 или … или 9».

Символы этого правила:

**<цифра>** – нетерминальный символ;

**::=** –операция «по определению есть»;

**0, 1, 2, 3, 4, 5, 6, 7, 8, 9** –терминальные символы;

**|** – операция «или».

Данное правило определяет, что в языке допустимыми – правильными – являются только десять арабских цифр (от 0 до 9).

Синтаксические элементы языка программирования:

* **набор символов** – буквы и цифры (например, символы ASCII, американского стандарта обмена информацией), а также ряд специальных символов;
* **идентификаторы** – имена объектов языка (типов, констант, переменных и других). Идентификатор состоит из последовательности букв, цифр и знака подчеркивания **\_.** Идентификатор не может начинаться с цифры. Отметим, что **abc** и **Abc** – два разных идентификатора, т.е. заглавные и строчные буквы различаются;
* **символы (знаки) операций, ограничители и скобки** – символы арифметических и логических операций, например: **+, -, /, \*, !, &, |**; символы операций отношения: **!=, ==, >=** и другие; символ **;** –ограничитель операторов; парные ограничители: **(…), […]** и операторные скобки – **{…};**

Таблица 1. Основные операции языка **C++**

| **Операция** | **Краткое описание** |
| --- | --- |
| **Унарные операции** | |
| **++** | увеличение на 1 |
| **--** | уменьшение на 1 |
| **sizeof** | размер |
| **~** | поразрядное отрицание |
| **!** | логическое отрицание |
| **-** | арифметическое отрицание (унарный минус) |
| **+** | унарный плюс |
| **&** | взятие адреса |
| **\*** | разыменование |
| **new** | выделение памяти |
| **delete** | освобождение памяти |
| **(type)** | преобразование типа |
| **Бинарные операции** | |
| **\***  **/**  **%** | умножение  деление  остаток от деления |
| **+**  **-** | сложение  вычитание |
| **<<**  **>>** | сдвиг влево  сдвиг вправо |
| **<**  **<=**  **>**  **>=** | меньше  меньше или равно  больше  больше или равно |
| **==**  **!=** | равно  не равно |
| **&** | поразрядная конъюнкция (И) |
| **^** | поразрядное исключающее ИЛИ |
| **|** | поразрядная дизъюнкция (ИЛИ) |
| **&&** | логическое И |
| **||** | логическое ИЛИ |
| **=**  **\*=**  **/=**  **%=**  **+=**  **-=**  **<<=**  **>>=**  **&=**  **|=**  **^=**  **,** | присваивание  умножение с присваиванием  деление с присваиванием  остаток от деления с присваиванием  сложение с присваиванием  вычитание с присваиванием  сдвиг влево с присваиванием  сдвиг вправо с присваиванием  поразрядное И с присваиванием  поразрядное ИЛИ с присваиванием  поразрядное исключающее ИЛИ с присваиванием  последовательное вычисление |
| **Тернарная операция** | |
| **? :** | условная операция |

* **ключевые слова –** идентификаторы, используемые в качестве фиксированной части синтаксиса какой-либо конструкции языка. Например: **int, char, if, while, new;**
* **выражения –** конструкции языка для вычисления и изменения значений. Выражения включают символы операций и операнды (константы, переменные, функции и другие объекты языка);
* **управляющие структуры** – алгоритмические конструкции (схемы), состоящие из функциональных, предикатных узлов и узлов слияния, имеющие один вход и один выход. Базовые управляющие структуры: **Последовательность, ЕслиТоИначе, ЦиклДо**(с предусловием);
* **операторы –** конструкции языка, необходимые для обработки данных и управления порядком действий. Оператор задает законченное описание некоторого действия. Различают операторы присваивания, ввода-вывода, операторы, реализующие управляющие структуры;
* **комментарии –** пояснения, включаемые в текст программы, но не обрабатываемые компилятором. Комментарии в **С++:**

**//** – однострочный

**/\*** – многострочный, включающий несколько строк

**...**

**\*/**

**Определения структурного программирования.**

**Простая программа –** программа, которая имеет один вход и один выход, и через каждый узел программы можно провести путь от входа к выходу.

**Элементарная программа** – программа, которая не содержит простых программ, более чем из одного узла. Все управляющие структуры – элементарные программы.

**Составная программа** – элементарная программа, в которой функциональный узел заменен элементарной программой

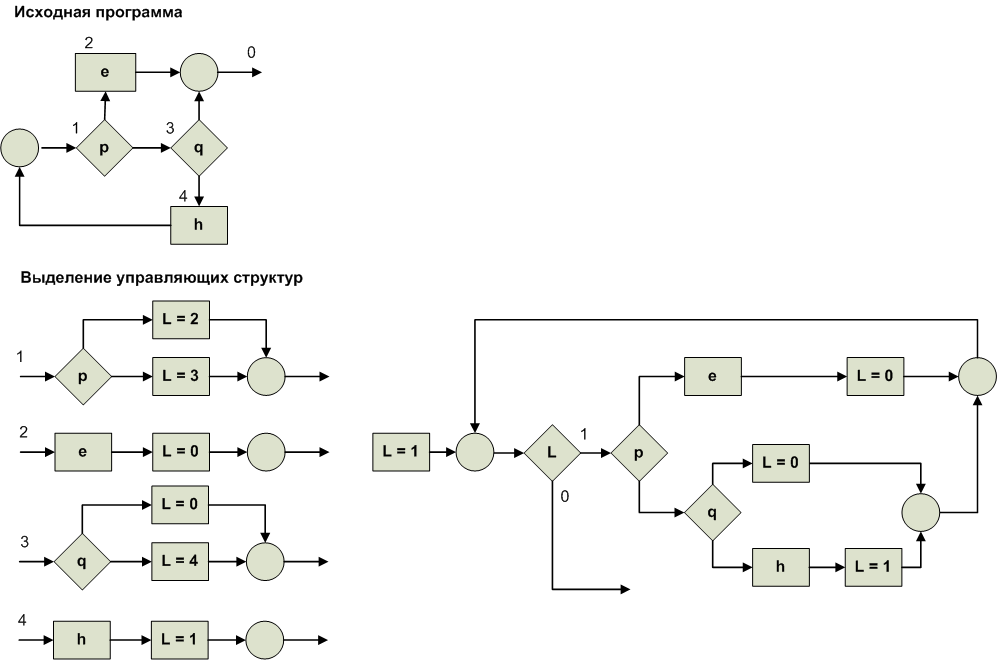
**Структурированная программа** – составная программа, построенная на основе базисного множества элементарных программ (управляющих структур)

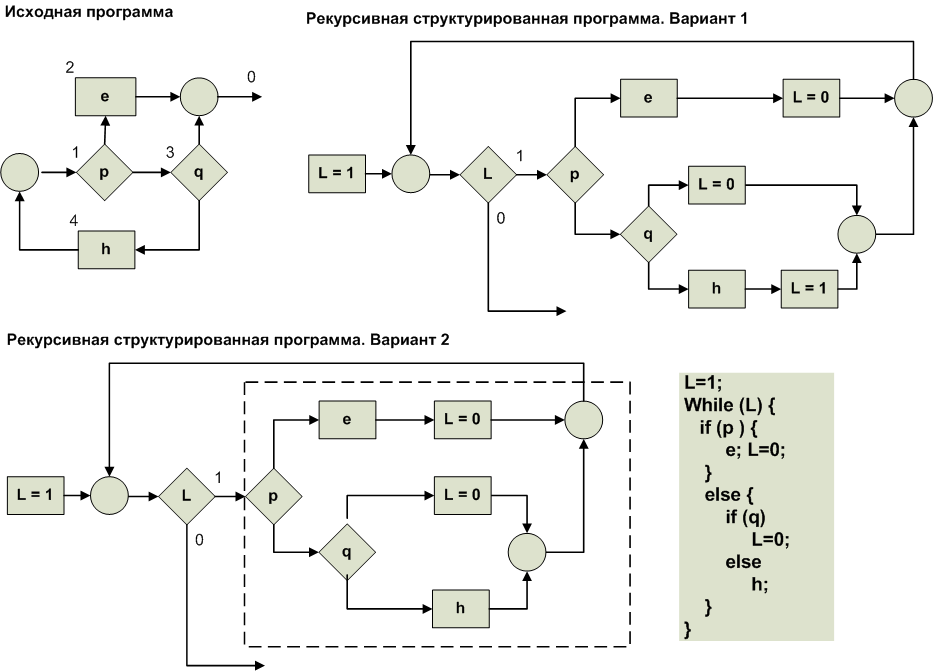
**Теорема о структурировании**. ***Любая простая программа функционально эквивалентна структурированной программе, составленной из базисного множества управляющих структур, с использованием функций и предикатов исходной программы, а также присваиваний и тестов над дополнительным счетчиком*.**

Таблица 1.3 Управляющие структуры

| **Управляющая структура** | **Оператор языка C++** | **Пример** |
| --- | --- | --- |
| **Последовательность**  Последовательность | **A; B;**  Сначала выполняется **A***,* затем**B***.*  **A, B** – операторы | **sum = int1 + int2;**  **cout << "Сумма равна "**  **<< sum << endl;** |
| **Если\_То\_Иначе**  Если_То_Иначе | **if (P)**  **A;**  **else**  **B;**  Если значение логического выражения **P** истинно (**не ноль**), то выполняется **A**,  иначе – **B**. | **If( x>0)**  **y++;**  **else**  **y--;**  **int num, div;**  **if (!(num%div))**  **cout<<"делится"<<endl;**  **else**  **cout<<"не делится"<<endl;** |
| **Если\_То**  Если_То | **if (P ) A;**  Если значение логического выражения **P** истинно(**не ноль**), то выполняется **A**. | **if (x>0) y++;** |
| **Выбор**  Выбор | **switch (K) {**  **case 1: A; break;**  **case 2: B; break;**  **case 3: C; break;**  **default:**  **D; break;**  **};**  **K** – переключатель- выражение, значение которого определяет выполнение операторов:  если **K = 1**, то **A***;*  если **K = 2,** то **B**;  если **K = 3**, то **C**  иначе **– D**; | **char op;**  **bool f=true;**  **int a, b, res;**  **cin >>a>>b>>op;**  **switch (op) {**  **case '+': res=a+b; break;**  **case '-' : res=a-b; break;**  **default: cout<<**  **"\nНеизвестная операция"**  **<< endl; f=false; break;**  **}**  **If (f) cout<<res<< endl;**  Если выход из переключателя явно не указан – **отсутствует** **break** –  тогда **последовательно выполняются все остальные ветви!** |
| **Цикл\_Пока**  **(цикл с предусловием)**  Цикл_Пока | **while (P)**  **A;**  Пока значение логического выражения **P**истинно(**не ноль**), оператор **A** выполняется. | **while (N>0)**  **N--;** |
| **Цикл\_со\_счетчиком**  **(цикл с предусловием)** | **for (инициализация;**  **выражение;**  **модификация)**  **A;**  Инициализация применяется для объявления и задания начальных значений величинам, используемым в цикле.  Выражение определяет условие продолжения цикла.  Модификация выполняется после каждой итерации цикла. | **for (int i = 1, s = 0;**  **i <=100; ++i)**  **s += i;**    **//Последоательность**  **//вычислений:**  **//i=1, s=0+1=1**  **//i=2, s=1+2**  **//i=3, s=1+2+3**  **//…**  **//i=100, s=1+2+…+100**  **//i=101** – выход из цикла |
| **Цикл\_До**  **(цикл с постусловием)** | **do**  **A;**  **while P;**  Оператор **A**выполняется до тех пор, пока логическое выражение **P**истиннo(**не ноль**). | **do**  **N--;**  **while N>0;** |

**Пример**





**ВНИМАНИЕ!**

**Самостоятельно: разработать программу без использования переменной L.**

* 1. Значения и типы

Цель программы состоит в вычислении значений.

**Значение** **данных** может представлять собой число, символ или указатель на другой объект данных. При этом компьютер оперирует с физическими представлениями значений, которыми являются совокупности битов, байтов или слов оперативной памяти.

Одной из главных характеристик данных является тип.

**Тип данных** определяет:

* внутренне представление данных в памяти компьютера;
* множество значений, которые могут принимать данные этого типа;
* операции и функции, которые можно применять к значениям этого типа.

**Преобразование типов** в **С++** выполняется с помощью специальных *операторов приведения типов*. Для приведения целых и вещественных типов при вычислении выражений используется оператор **static\_cast**.

Например:

**float f = 5.25;**

**int i = static\_cast<int>(f); //i=5**

**int first = 5, second = 10;**

**double result=static\_cast<double>(first)/second;//result=0.5**

**//Внимание! double result=first/second; //result=0!**

* 1. Константы и переменные

**Константы**  – это данные, значения которых не изменяется при выполнении программы. Константы в языке **С++** характеризуется типом, наименованием и значением:

**const int X = 240;** **//** константа **X,** тип– целый, значение **240**

**const int Z = 0xF0; //** константа **Z,** тип– целый, значение в шестнадцатеричной

**//** системе счисления **F0**

При объявлении констант с помощью **const** обязательно укажите их значения!

При разработке программ в ряде случаев удобно объявить наименования констант с помощью перечисляемого типа:

**enum {Red, Yellow, Green}; //** константам **Red, Yellow, Green** автоматически будут

**//** присвоены целые значения **0, 1, 2** соответственно

**enum {EMPTY=-1, FULL=100}; //** константа **EMPTY** имеет значение **-1**, **FULL** – **100**

**Переменные** – это данные, значения которых изменяется при выполнении программы. Во время выполнения программы c каждой переменной связана определенная область памяти, где хранится текущее значение переменной.

**Класс памяти** определяет время жизни и область видимости программного объекта, в частности и переменной. Область видимости чаще всего совпадает с областью действия. Если класс памяти не указан явным образом, он определяется компилятором исходя из контекста объявления

**Пространства имен** – это поименнованные области, которые служат для логического группирования объявлений и ограничения доступа к ним. Формат описания пространства имен следующий:

**namespace имя\_области{объявления}**

или

**namespace имя\_области;**

* 1. Выражения и операторы

**Логические выражение** выполняет логические операции над логическими операндами или операции отношения над заданными операндами.

**Пример.** Присвоить переменной f истина, если a>b, и ложь – в противном случае:

**bool f = a>b; //оператор if не нужен!!!**

**Условная операция (?:)** имеет три операнда. Ее формат:

**логическое выражение ? операнд\_1 : операнд\_2**

Если результат вычисления условного выражения равен **true** (**не 0**), то результатом условной операции будет значение операнда\_1, иначе – операнда\_2. Условная операция **?:** является сокращенной формой оператора **if.**

**Пример:**

**i = (i < n) ? i + 1 : 0;**

**grade >= 60 ? cout<<"Зачет\n" : cout<<"Незачет\n";**

* 1. Советы профессионала

**IconAttention**

**Как правильно использовать типы переменных?**

* **Избегайте сравнения переменных разных типов.**

Например, дляпеременных **float x; int i;** сравнение **x == i** почти гарантированно не работает, так как преобразование одного типа к другому всегда связано с округлением и получить точное равенство (с точностью до двоичного разряда) нельзя!

* **Избегайте сравнения на равенство переменных вещественных типов**

Если десять раз сложить **0**.**1**,то значение 1.**0** получится очень редко: почти всегда – **0**.**99999**...Поэтому для вещественных переменных **x1**, **x2** вместо сравнения **x1 == x2** следует сравнить абсолютное значение их разности с заданной точностью, например:

**abs(x1 – x2) <= 0**.**00001**

**Как правильно использовать константы?**

* **Избегайте «магических» чисел.**

Используйте в программе в качестве числовых констант только значения **0** или **1**.Все другие значения (**«магические» числа**) должны быть описаны, как именованные константы. Например, **Const N = 10;** и везде в коде программы вместо числа **10** используйте наименование **N**.

* **Используйте символьные константы**, **а не их порядковый номер.**

Вместо **Chr(65)** напишите в коде программы просто **'A'**.

**Как правильно задать имя переменной?**

* **Самый важный принцип – имя должно полно и точно описывать сущность**, **представляемую переменной.**
* **Имена переменных должны облегчать чтение кода программы**
* **Выбирайте длину наименования исходя из времени жизни переменной**

Считается, что отладка программы требует меньше усилий при использовании имен переменных, состоящих от **8** до **20** символов.

Короткие имена – значит переменные второстепенные с коротким временем жизни.

* **Добавляйте Спецификаторы вычисляемых значений.**

Спецификаторы вычисляемых значений – **Count** (количество), **Max** (максимум), **Average** (среднее), **Total** (общее число) и другие – добавляются в конце имени переменной. Например, **customerCount**, **customerTotal** – общее число заказчиков.

* **Используйте переменные i**, **j**, **k для показателей циклов.**

Как правило, наименование показателей циклов – **i**, **j**, **k**.Если вы используете эти наименования для показателей циклов,то **не используйте их для других целей в программе!**

* **Используйте общепринятые наименования для логических переменных.**
* **Done** – признак завершения цикла. Присвойте **false** до цикла и **true** – после завершения
* **Error** – признак ошибки: **false** – нет ошибки, **true** – ошибка
* **Found** – признак обнаружения некоторого значения. Например, поиск значения в массиве. Присвойте **false** до начала поиска и **true** – как только значение найдено
* **Success** или **ОК** – признак успешного завершения операции. Присвойте **false**, если операция завершилась неудачно, и **true**, если операция завершилась успешно.
* **НЕ используйте похожие символы в наименовании.**

Например: **G** или **6**, **1**, или **l**, **Q**, или **0**, **2**, или **Z**, **S**, или **5** и другие

* **Начинайте наименование переменной со строчной буквы**, **а наименования типов – с прописной**.

**Упрощайте логические выражения в операторах if**

* **Избегайте вызова «лишних» функций и сравнений.**

Вместо **(!(x>x\_left)) || (!(x<x\_right))** следует написать выражение

**(x<=x\_left) || (x>=x\_right)**

Если вы хотите выяснить, равно ли значение логической переменной **b истина** –НЕ пишите в операторе **if: b==True,** напишите просто: **if (b)**  …

* **Используйте дополнительные логические переменные для понимания и упрощения кода.**

Например, оператор **if** имеет вид

**if ((elementIndex<0) || (elementIndex>MAX\_ELEMENT)**

**|| (elementIndex == LAST\_ELEVENT))**

Добавим в программу две логических переменных **violation** (нарушение) и **completion** (завершение) и определим их следующим образом:

**violation = (elementIndex<0) || (elementIndex>MAX\_ELEMENT);**

**completion = (elementIndex == LAST\_ELEVENT);**

Тогда оператор **if** можно записать так:

**if (violation || completion)**

**Упрощайте структуру цикла:**

* **Используйте оператор цикла for**, **если известно число повторений**, **и оператор while в других случаях**.
* **Не изменяйте в «теле» цикла for значение показателя цикла!**
* **Вычисления**, **не изменяемые в цикле**, **выносите за его пределы**.

Например, переменная **value** вычисляется следующим образом:

**for (int i=0; i<Count; i++) {**

**value = (i+1) 🞸 BaseValue 🞸 DELTA;**

...

**}**

Видно, что значение произведения **BaseValue🞸DELTA** не зависит от показателя цикла (переменная **i)** и не изменяется в цикле.

Добавим в код переменную **increment (**приращение**)** и перепишем цикл следующим образом:

**increment:= BaseValue 🞸 DELTA;**

**for (int i=0; i<Count; i++) {**

**value = (i+1) 🞸 increment;**

**. . .**

**}**

* **Заменяйте умножение сложением.**

В рассмотренном выше примере переменная **value** на каждом шаге цикла увеличивается на значение переменной **increment**,поэтомуцикл можно организовать так:

**Increment = BaseValue 🞸 DELTA; value =0;**

**for (int i=0; i<Count; i++) {**

**value += increment; //Сравните с первоначальным вариантом!**

**. . .**

**}**

* **Используйте «размыкание» цикла.**

Вам необходимо выполнять в цикле похожие действия, отличающиеся небольшой частью, которая выполняется только один раз в начале цикла:

**for (int i=0; i<Count; i++) {**

**if (i == 0){**

**операторы 1;**

**}**

**else {**

**операторы 2;**

**}**

**}**

«Размыкание» цикла заключается в том, что действия, выполняемые в цикле один раз (в примере это **операторы 1**), следует выполнить за его пределами перед началом цикла:

**i = 0; //если это необходимо!**

**операторы 1**

**for (int i =1; i<Count; i++){**

**операторы 2;**

**}**

* **Тщательно анализируйте задание перед конструированием цикла.**

Рассмотрим вычисление **ex**с заданной точностью **tolerance** по формуле

****

При решении «в лоб» в цикле придется выполнять не только вычисление суммы слагаемых, но и возведение **x** в целую степень, а также вычисление факториала (**2!**, **3!**, **…**). Однако при выполнении простого анализа исходной формулы (разделите последующее слагаемое на предыдущее!) видно, что каждое слагаемое, начиная с **x**, может быть получено умножением предыдущего слагаемого на коэффициент **x / I** , где **i = 1**, **2**,**…** – номер слагаемого в исходном выражении (слагаемое **x** имеет номер **1**).

Тогда вычисление **ex** можно сконструировать так:

**i =1; // номер слагаемого**

**summa =1; // значение суммы**

**increment =1; // значение слагаемого**

**while (increment > tolerance){**

**increment \*= x/i;**

**summa += increment;**

**i++;**

**}**

* 1. Примеры простых программ

Примеры простых программ, построенных на основе управляющих структур смотрите в методических указаниях «Введение в программирование на языке С++» разделы **1.8** и **1.9**.

**Примеры операторов ввода/вывода**

**#include <iostream>**

**setlocale(LC\_ALL, "");**

**int i;**

**std::cout << "Введите целое число\n";**

**std::cin >> i;**

**std::cout << "Вы ввели число " << i << std::endl;**

**или**

**#include <iostream>**

**using std::cout;**

**using std::cin;**

**using std::endl;**

**setlocale(LC\_ALL, "");**

**int i;**

**cout << "Введите целое число\n";**

**cin >> i;**

**cout << "Вы ввели число " << i << endl;**

**или**

**#include <iostream>**

**using namespace std; // подключение всего пространства имен std**

**setlocale(LC\_ALL, "");**

**int i;**

**cout << "Введите целое число\n";**

**cin >> i;**

**cout << "Вы ввели число " << i << endl;**

**Пример форматированного вывода**

