# ПОНЯТИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ.

**ЯЗЫК ПРОГРАММИРОВАНИЯ** – формальный язык для описания данных и алгоритмов их обработки, определяющий набор правил, задающих внешний вид программы и действия, которые выполняет компьютер.

**ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ**

1) алфавит, синтаксис, семантика;

2) типы, структуры, классы данных, значения, литералы, переменные, константы;

3) операторы, операции и выражения, ввод-вывод данных;

4) подпрограммы;

5) библиотеки;

6) стандартизация языков и мобильность программ.

# 2. КЛАССИФИКАЦИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ.



По степени зависимости от аппаратных средств:

1. Языки низкого уровня (Ассемблер – язык, команды которого максимально соответствуют командам микропроцессора);
2. Языки высокого уровня (поддерживают сложные типы данных, объектно-ориентированы)

По принципам программирования

1. Процедурные
2. Непроцедурные
3. Объектно-ориентированные
4. Структурные

По ориентации на класс задач

1. Универсальные
2. Специализированные

По виду обработки кода

1. Компилируемые
2. Интерпретируемые

По назначению

1. Алгоритмические (Pascal, C, C++, C#, Java)
2. Языки описания данных (HTML)

# 3. ТРАНСЛЯЦИЯ ПРОГРАММ. ПОНЯТИЯ «ТРАНСЛЯТОР», «КОМПИЛЯТОР», «ИНТЕРПРЕТАТОР». ОСНОВНЫЕ ЭТАПЫ КОМПИЛЯЦИИ/ИНТЕРПРЕТАЦИИ.

**ТРАНСЛЯТОР** - программа или тех. средство, выполняющее трансляцию – обработку программы и её перевод на язык, близкий к машинному (бинарному) коду. Существуют следующие виды трансляторов:

**КОМПИЛЯТОР** - (англ. compiler — составитель, собиратель) читает всю программу целиком, делает ее перевод и создает законченный вариант программы на машинном языке (исполняемый .exe файл), который затем выполняется.

**ИНТЕРПРЕТАТОР**  - (англ. interpreter — истолкователь, устный переводчик) переводит и выполняет программу строка за строкой. В отличие от компилятора, интерпретатор не создает на выходе программу на машинном языке. Распознав команду исходного языка, он тут же выполняет ее.

И там, и там используются одинаковые методы анализа исходного текста программы, однако интерпретатор позволяет сделать процесс разработки/отладки программ более гибким и быстрым. С другой стороны, интерпретируемый код сложнее распространять, в отличие от исполняемых файлов, создаваемых компилятором, и скорость работы интерпретируемых программ в итоге получается ниже, чем у компилируемых. В настоящее время идёт объединение понятий компиляции и интерпретации.

**ОСНОВНЫЕ ЭТАПЫ КОМПИЛЯЦИИ**

1) лексический анализ с помощью программы-сканера: последовательность символов исходного кода преобразуется в последовательность лексем – наборов символов, распознаваемых компилятором языка. Результат – поток лексем, эквивалентный исходному тексту программы;

2) синтаксический анализ – проверка предложений, из которых составлена исходная программа, на соответствие правилам грамматики языка. Параллельно происходит генерация внутренней формы представления программы;

3) семантический анализ - установления семантики (смысла) программы. Например, проверка совместимости типов выражений, привязка декларации и др. Результат –промежуточный код;

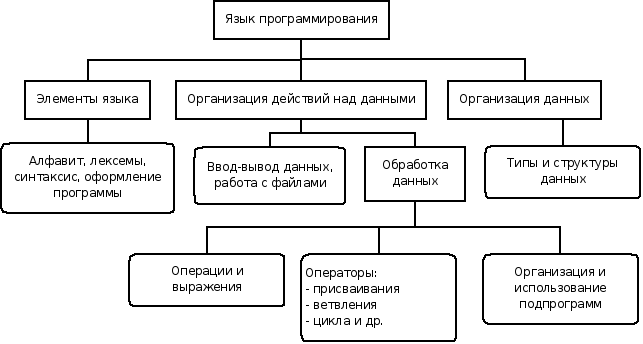
4) оптимизация – удаление излишних конструкций, упрощение кода с целью сохранения его смысла;

5) генерация машинного кода – из промежуточного представления порождается результирующий машинный код.

На всех этапах происходит работа с различного рода таблицами (например, таблица зарезервированных ключевых слов, таблица операторов, таблица идентификаторов и типов). Возможна раздельная компиляция (трансляция частей по отдельности с последующим объединением их компоновщиком в единый загрузочный модуль).

# СТРУКТУРА ЯЗЫКА ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ. ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯПВУ

Во всяком языке программирования определены способы организации данных и способы организации действий над данными. Кроме того, существует понятие «элементы языка», включающее в себя множество символов ([алфавит](http://wiki.mvtom.ru/index.php?title=%D0%90%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82&action=edit&redlink=1)), [лексемы](http://wiki.mvtom.ru/index.php?title=%D0%9B%D0%B5%D0%BA%D1%81%D0%B5%D0%BC%D0%B0&action=edit&redlink=1) и другие изобразительные средства языка программирования. Всякий язык программирования имеет три основные составляющие: **алфавит, синтаксис и семантику**. Синтаксис – набор правил построения фраз алгоритмического языка, позволяющий определить, осмысленные предложения в этом языке. Смысловое содержание языковой конструкции называется семантикой.



# СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ ПАСКАЛЬ.

Паскаль — язык структурного программирования, что означает, что его синтаксисом предусмотрена определенная структура программы:

**[ ЗАГОЛОВОК ПРОГРАММЫ ]**

program Primer;

**[ РАЗДЕЛ ОПИСАНИЙ ]**

раздел меток ( label )  
раздел констант ( const )  
раздел типов ( type )  
раздел переменных ( var )

**[ РАЗДЕЛ ПРОЦЕДУР И ФУНКЦИЙ ]**

(раздел может быть пропущен, если в программе не предусмотрено использование процедур или функций)

**[ РАЗДЕЛ ОПЕРАТОРОВ ]**

begin

операторы;

end.

# ОПЕРАЦИИ ЯЗЫКА ПАСКАЛЬ. ВЫРАЖЕНИЯ.

Любое выражение имеет определенный тип и после вычисления возвращает некоторое значение. Простейшими выражениями являются переменные и константы. Более сложные выражения строятся из более простых с использованием операций, скобок, вызовов функций, индексов и приведений типов. Данные, к которым применяются операции, называются операндами.

**АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ**

К арифметическим относятся бинарные операции + - \* / для вещественных и целых чисел, бинарные операции div и mod для целых чисел и унарные операции + и - для вещественных и целых чисел. Выражение, имеющее числовой тип, называется арифметическим. Тип арифметического выражения определяется по следующему правилу: если все операнды целые и в выражении отсутствует операция деления /, то выражение имеет тип integer, в противном случае выражение имеет тип real. Например, если b имеет тип byte, c имеет тип 1..9, то b+c и -b имеют тип integer, а 2.0+b и 1/2 - тип real.

**ЛОГИЧЕСКИЕ ОПЕРАЦИИ**

К логическим относятся бинарные операции and, or и xor, а также унарная операция not, имеющие операнды типа boolean и возвращающие значение типа boolean. Выражение, имеющее тип boolean, называется логическим.

**ОПЕРАЦИИ ОТНОШЕНИЯ**

Операции отношения <, >, <=, >=, =, <> возвращают значение типа boolean и применяются к операндам целого, вещественного, символьного, логического и строкового типов, а также к типизированным указателям. Операции = и <> также применяются к операндам типа pointer, к операндам, являющихся объектами классов и к константе nil. При сравнении символа и строки символ преобразуется в строку длины 1.

**ПОБИТОВЫЕ ОПЕРАЦИИ**

Побитовые операции and, or, not, xor, shl, shr производят побитовые манипуляции с операндами целого типа. Для унарной операции not результат имеет тот же тип, что и операнд. Для остальных операций результат имеет тип integer. Например, если b имеет тип byte и b=1, то b shl 10=1024, а not b=254.  
  
**СТРОКОВЫЕ ОПЕРАЦИИ**

Помимо операций отношения <, >, <=, >=, =, <>, к строковым и символьным операндам применима операция конкатенации (слияния) +. Ее результат имеет строковый тип. Например, 'a'+'b'='ab'. Поскольку строки могут содержать максимум 255 символов, то если сливаются строки суммарной длины больше 255, то программа завершается сообщением об ошибке.

**ПРИОРИТЕТ ОПЕРАЦИЙ**

Приоритет определяет порядок выполнения операций в выражении. Первыми выполняются операции, имеющие высший приоритет. Операции, имеющие одинаковый приоритет, выполняются слева направо.

**ТАБЛИЦА ПРИОРИТЕТОВ ОПЕРАЦИЙ**

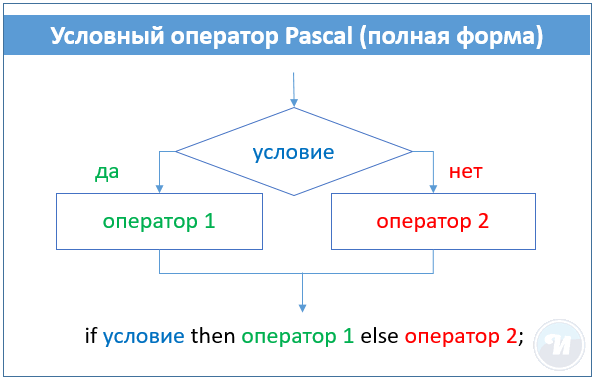
1. Наивысший @, not, ^
2. \*, /, div, mod, and
3. +, -, or, xor -
4. Низший =, <>, <, >, <=, >=, in, :=

# ТИПЫ ДАННЫХ ЯЗЫКА ПАСКАЛЬ

# 8. ОПЕРАТОР IF

**УСЛОВИЕ** – это логическое выражение, в зависимости от которого выбирается одна из двух альтернативных ветвей алгоритма. Если значение условия истинно (TRUE), то будет выполняться оператор 1, записанный после ключевого слова then. В противном случае будет выполнен оператор 2, следующий за словом else, при этом оператор 1 пропускается. После выполнения указанных операторов программа переходит к выполнению команды, стоящей непосредственно после оператора if.

В качестве условия может стоять: операция отношения, логическая переменная, логическое выражение. Else-часть в операторе if может отсутствовать. Следует помнить, что синтаксис языка допускает запись только одного оператора после ключевых слов then и else, поэтому группу инструкций обязательно надо объединять в составной оператор (окаймлять операторными скобками begin ... end).



# 9. ОПЕРАТОРЫ ЦИКЛА ЯЗЫКА ПАСКАЛЬ



ЦИКЛ FOR**называют циклом со счетчиком.** В Pascal этот цикл используется, когда количество повторений известно заранее. Цикл for существует в двух формах:

**for** счетчик **:=** начало **to** конец **do** тело;

**for** счетчик **:=** начало **downto** конец **do** тело;

Если между начальным и конечным значением счетчика указано ключевое слово **to**, то на каждом шаге цикла значение счетчика будет увеличиваться на единицу. Если же указано **downto**, то значение счетчика будет уменьшаться на единицу.

ЦИКЛ WHILE является циклом с предусловием. В заголовке цикла находится логическое выражение. Если оно возвращает true, то тело цикла выполняется, если false – то нет.

**while** условие **do** тело;

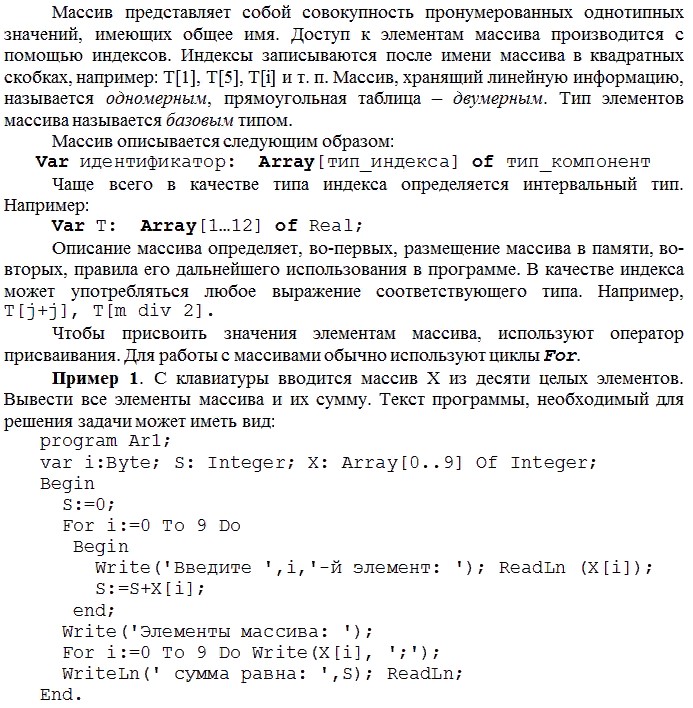
Когда тело цикла было выполнено, то ход программы снова возвращается в заголовок цикла. Условие выполнения тела снова проверяется (находится значение логического выражения). Тело цикла выполнится столько раз, сколько раз логическое выражение вернет истину. Поэтому очень важно в теле цикла предусмотреть изменение переменной, которая используется в заголовке цикла, таким образом, чтобы когда-нибудь обязательно наступала ситуация логической лжи. Иначе произойдет так называемое зацикливание.

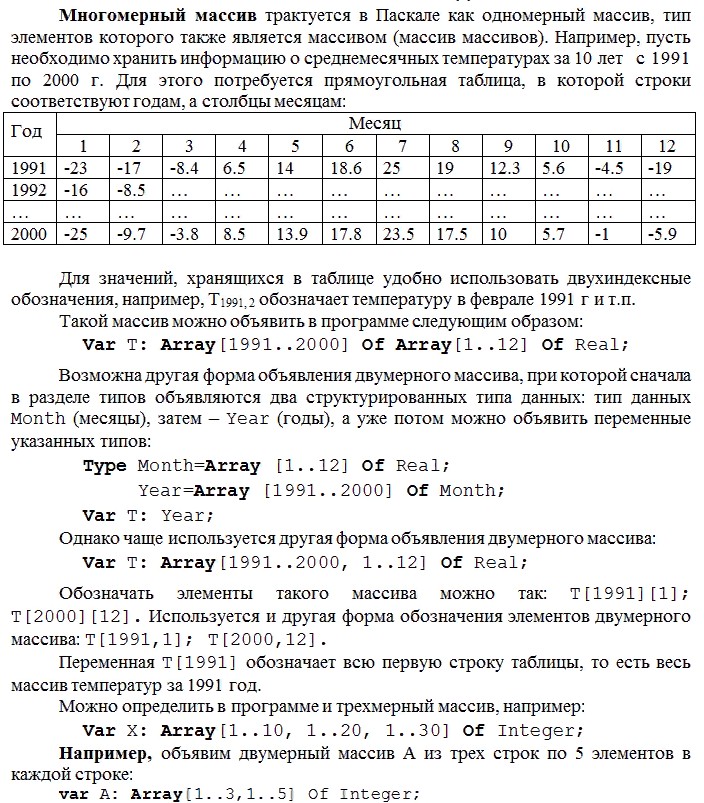
Цикл repeat выполняется хотя бы один раз, не зависимо оттого, что вернет логическое выражение – это цикл с постусловием.

В цикле repeat логическое выражение стоит после тела цикла. Причем, в отличие от цикла while, здесь всё наоборот: в случае true происходит выход из цикла, в случае false – его повторение.

**repeat** тело **until** постусловие;

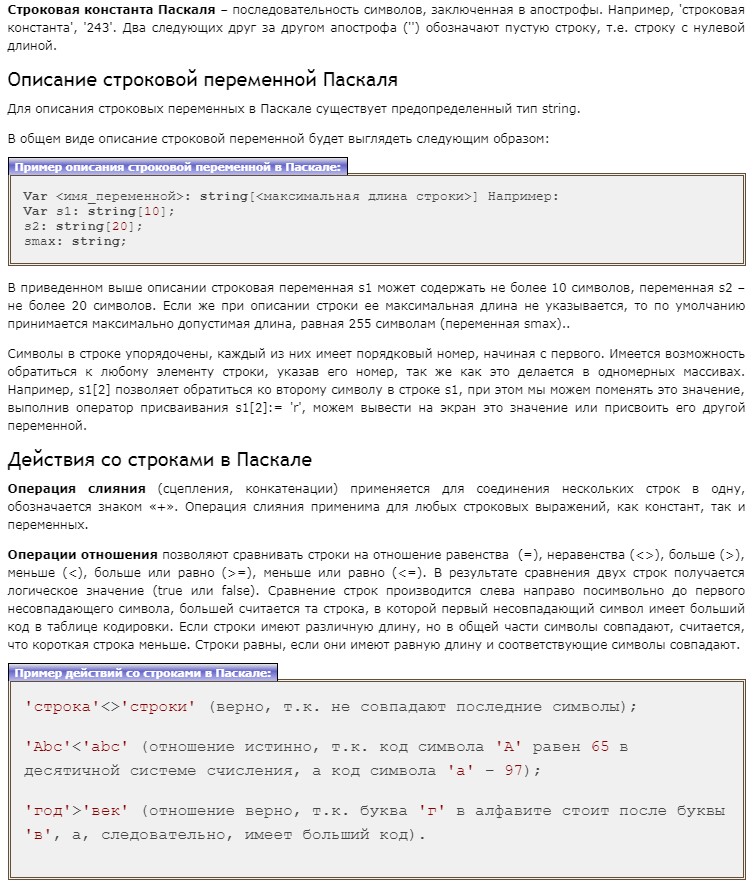
# 10. МАССИВЫ В ПАСКАЛЕ

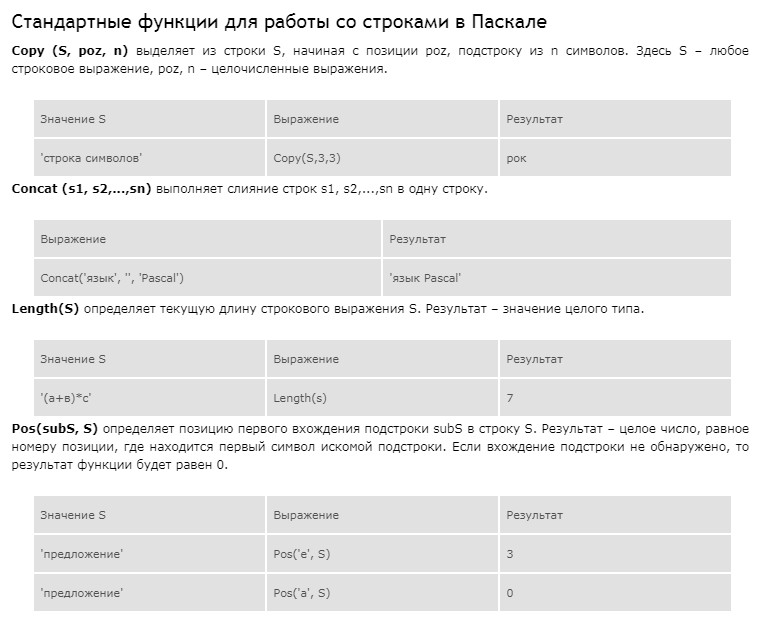




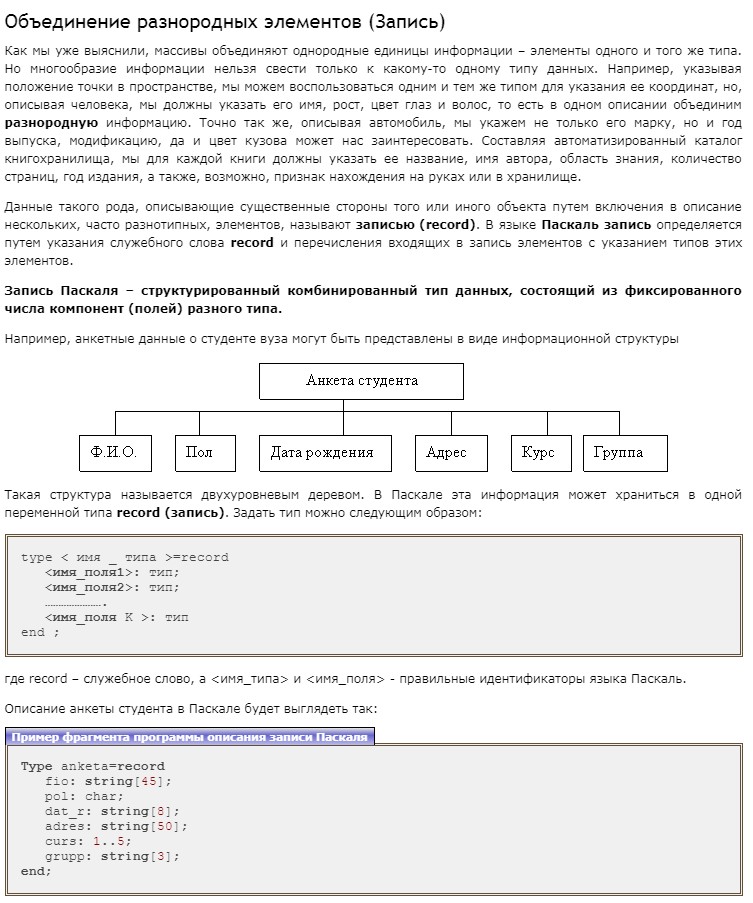
# 11. СТРОКОВЫЙ ТИП ДАННЫХ ЯЗЫКА ПАСКАЛЬ.

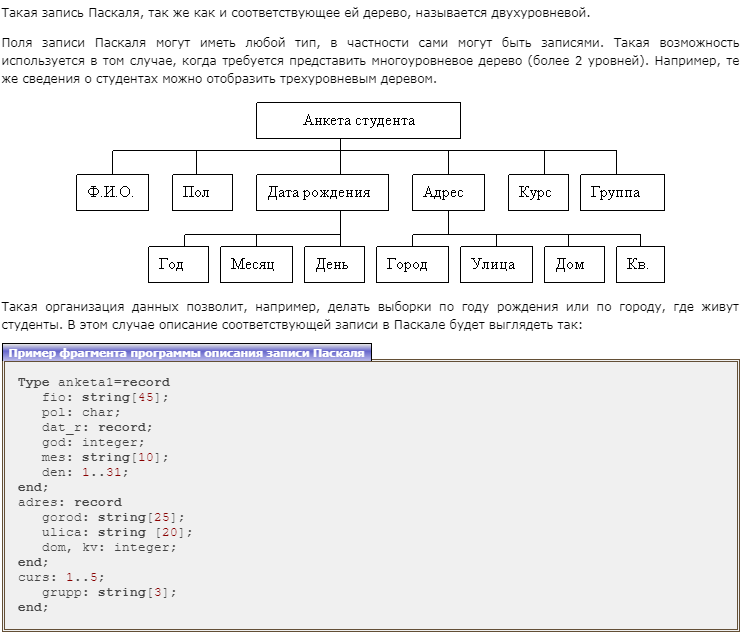
Строка представляет собой особую форму одномерного массива символов, имеющего фиксированную длину, которая определяется при описании. Количество символов в строке называется ее длиной. Длина строки в Паскале может лежать в диапазоне от 0 до 255. Каждый символ строковой величины занимает 1 байт памяти и имеет числовой код в соответствии с таблицей кодов ASCII.



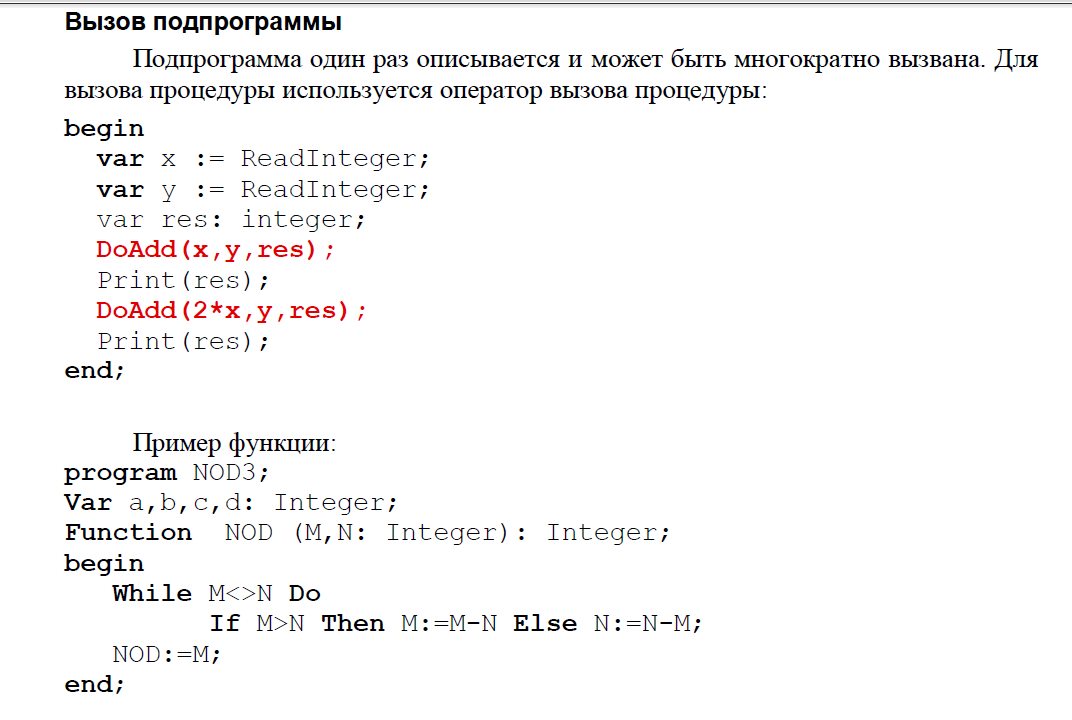
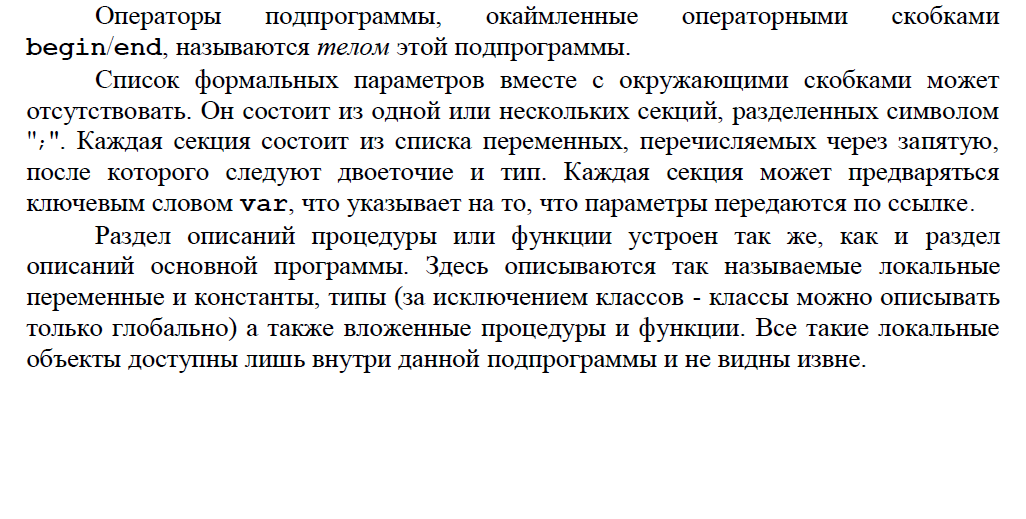


# 12. ТИП ДАННЫХ «ЗАПИСЬ» В ПАСКАЛЕ

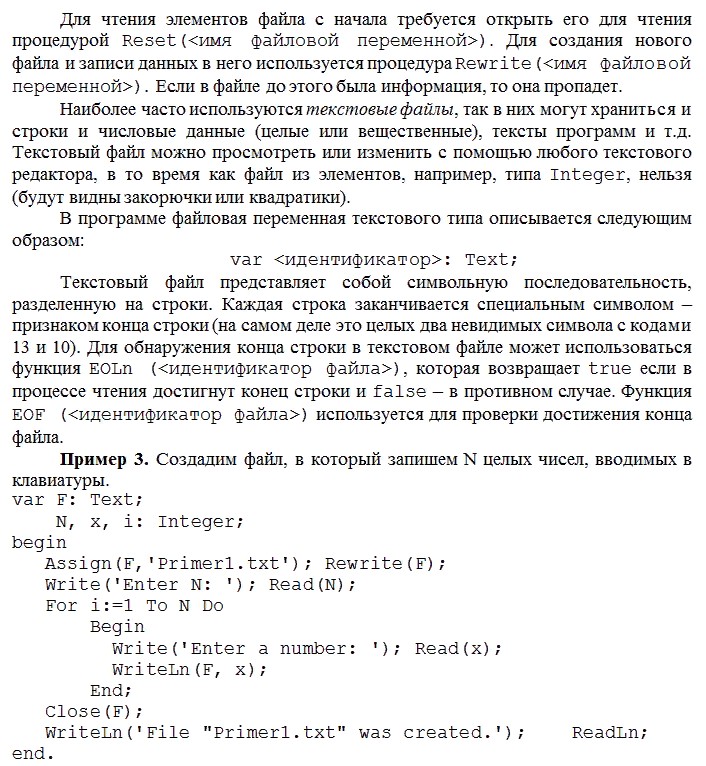
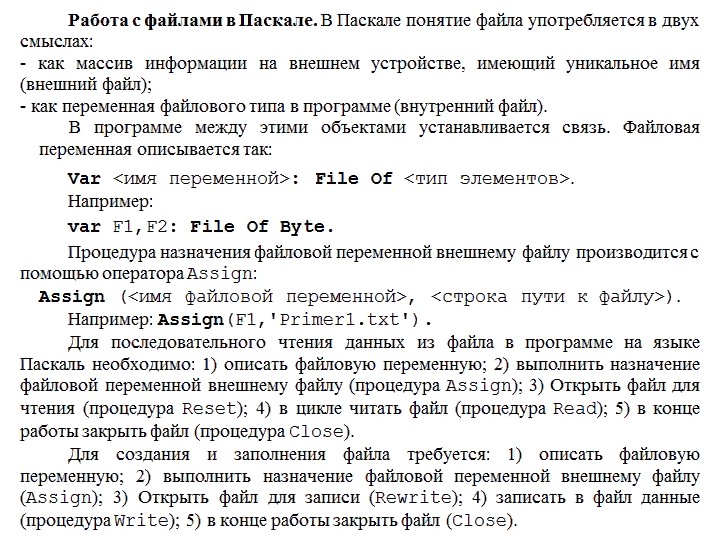


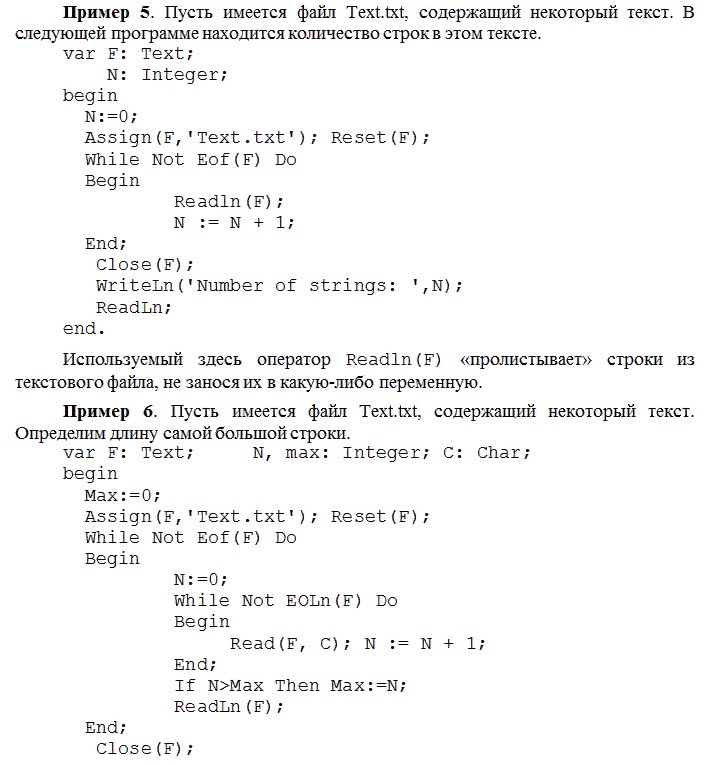
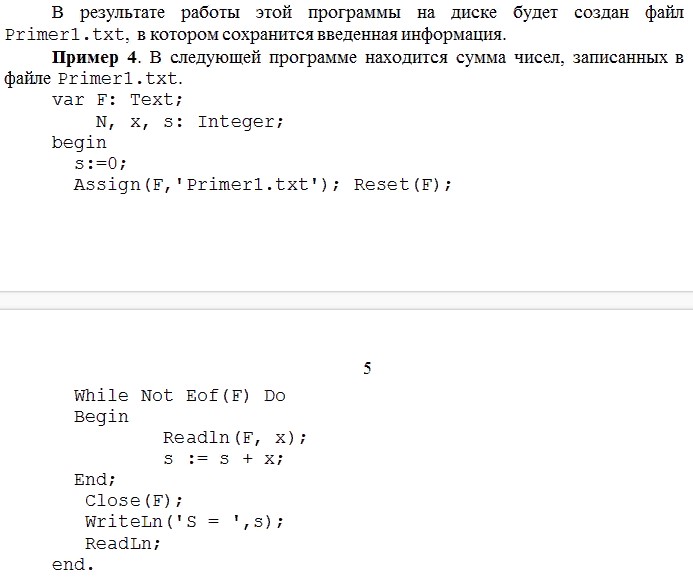


# 13. ФУНКЦИИ И ПРОЦЕДУРЫ В ПАСКАЛЕ



# 14. ЧТЕНИЕ ТЕКСТОВЫХ ФАЙЛОВ В ПРОГРАММЕ НА ПАСКАЛЕ. ЗАПИСЬ ТЕКСТОВЫХ ФАЙЛОВ В ПРОГРАММЕ НА ПАСКАЛЕ.





# 15. СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ С++.

Программа на языке С/С++ состоит из:

1. Директив процессора
2. Описаний
3. Функций

Главная функция носит имя main и должна быть обязательно.  
Общую структуру программы на языке С/С++ можно представить следующим образом:

|  |  |
| --- | --- |
|  | **директивы препроцессора //1** |
|  |  |
|  | **описание глобальных переменных //2** |
|  |  |
|  | **тип\_результата main (параметры) //3** |
|  |  |
|  | **{** |
|  | **тело главной функции** |
|  | **}** |
|  |  |
|  | **тип\_результата F1 (параметры 1) {** |
|  | **тело функции F1;** |
|  | **}** |
|  |  |
|  | **тип\_результата F2 (параметры 2) {** |
|  | **тело функции F2;** |
|  | **}** |
|  |  |
|  | **тип\_результата FN (параметры N) {** |
|  | **тело функции FN;** |
|  | **}** |

*Директивы препроцессора* – определяют действия по преобразованию программы перед компиляцией, а также включают инструкции, которым компилятор следует во время компиляции;

**ДИРЕКТИВЫ ПРОЦЕССОРА**

Понятие препроцессора является одним из ключевых в языке С++.  
*Препроцессор* – это программа, действующая как фильтр на этапе компиляции.  
*Исходная программа => Препроцессор => Расширенная исходная программа => Компилятор*  
Перед тем, как попасть на вход компилятора, исходная программа проходит через препроцессор.  
  
Основному тексту программы предшествуют *директивы препроцессора* (preprocessor directive), которые начинаются с символа решётки #, не являются выражением языка С++ (и потому не заканчивается точкой с запятой). Пример #include.  
  
*Команды (директивы) препроцессора* представляют собой инструкции, записанные в исходном тексте программы, используемые для того, чтобы облегчить модификацию программ и сделать их независимыми от особенностей различных реализаций компилятора.

*Объявления* – описания переменных, функций, структур, классов и типов данных.  
*Описание функций*, выполняемых в проекте. В языке С++ существует только один вид подпрограмм – подпрограмма-функция.  
Функция состоит из *заголовка* и *тела функции*.  
Заголовок состоит из:

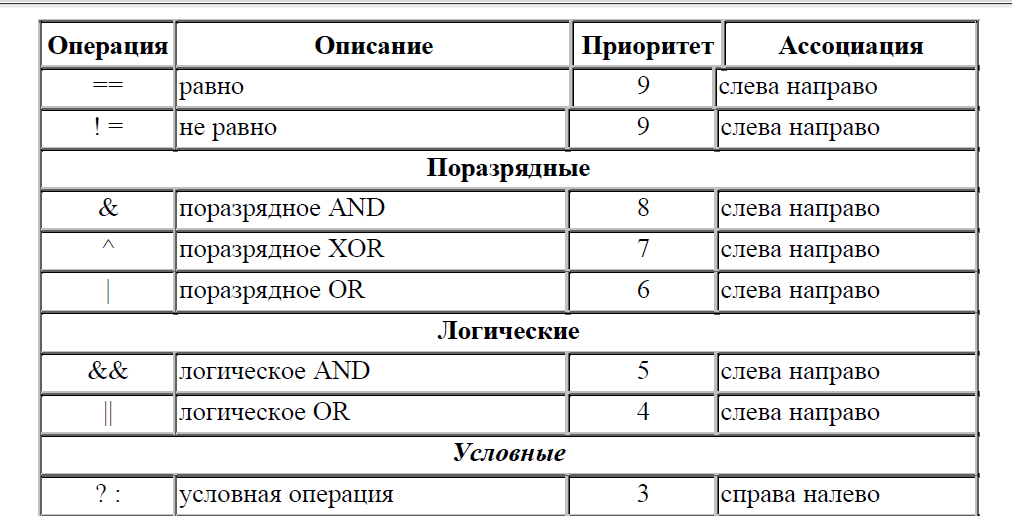
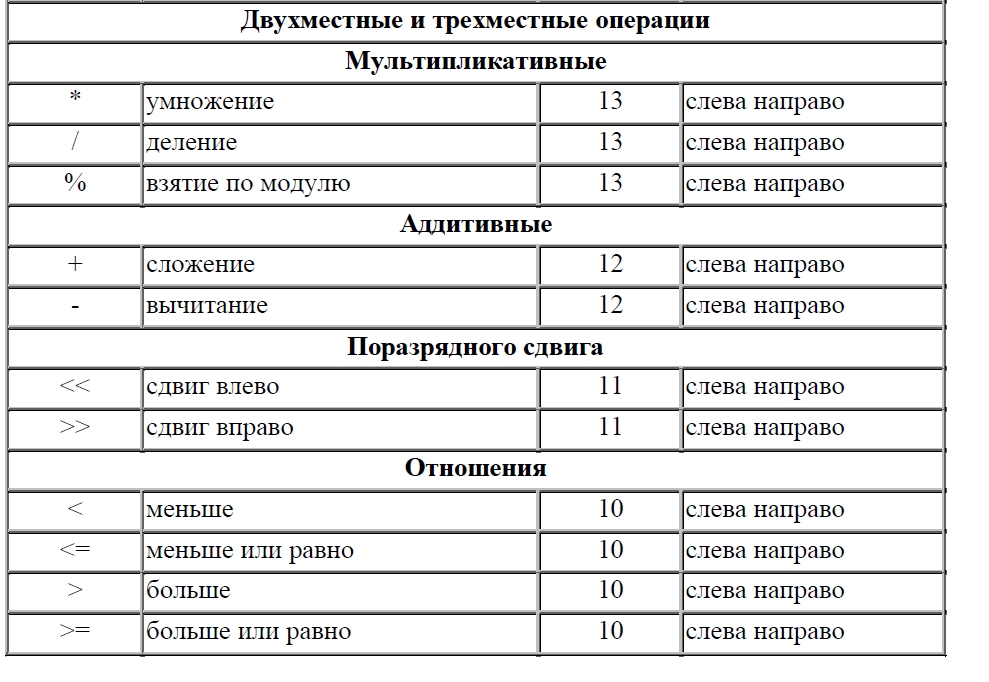
1. Типа результата, возвращаемого функцией;
2. Имени функции;
3. Списка параметров, необходимых для выполнения функции.

**ТЕЛО ФУНКЦИИ** заключается в операторные скобки {}, и содержит *описание локальных данных*и *операторы функции*.

# 16. ОПЕРАЦИИ ЯЗЫКА С++. ВЫРАЖЕНИЯ.

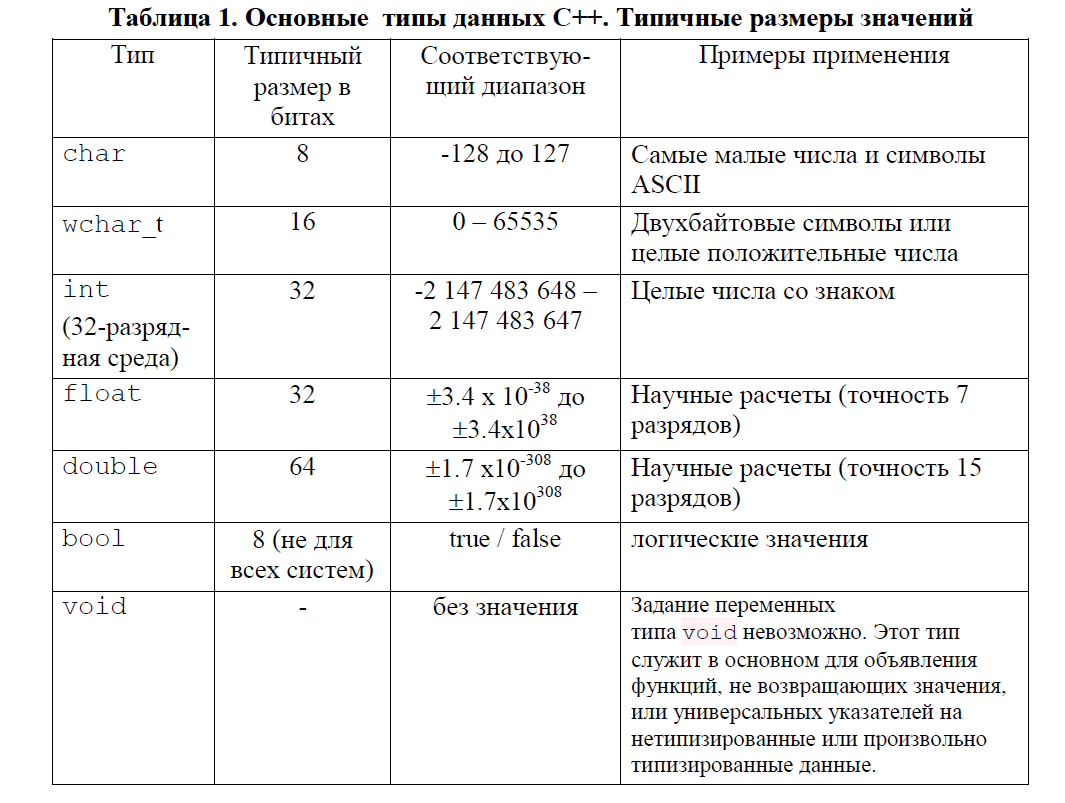
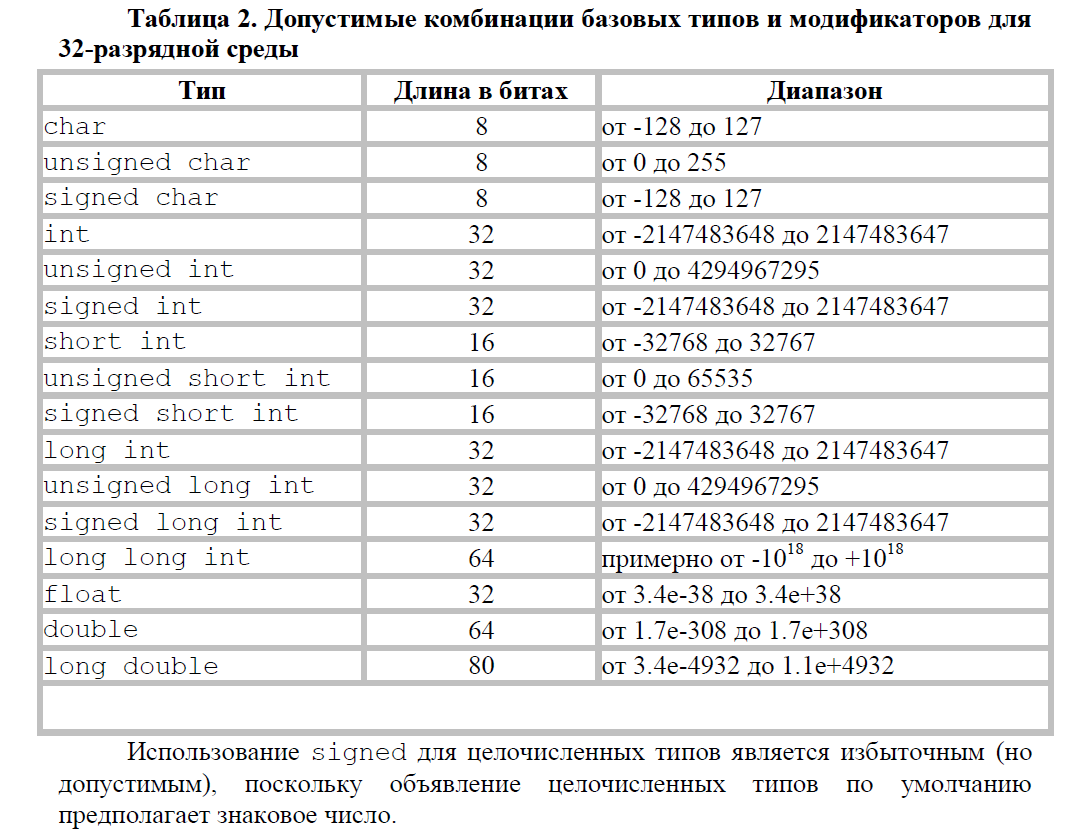
В языке C++ следующим уровнем представления данных после одиночных переменных и констант являются своего рода формулы, называемые *выражениями.*

Операции характеризуются своим *приоритетом,* определяющим порядок, в котором производится оценка выражения, и правилом *ассоциации,* задающим направление последовательных оценок идущих друг за другом операций одного приоритета.

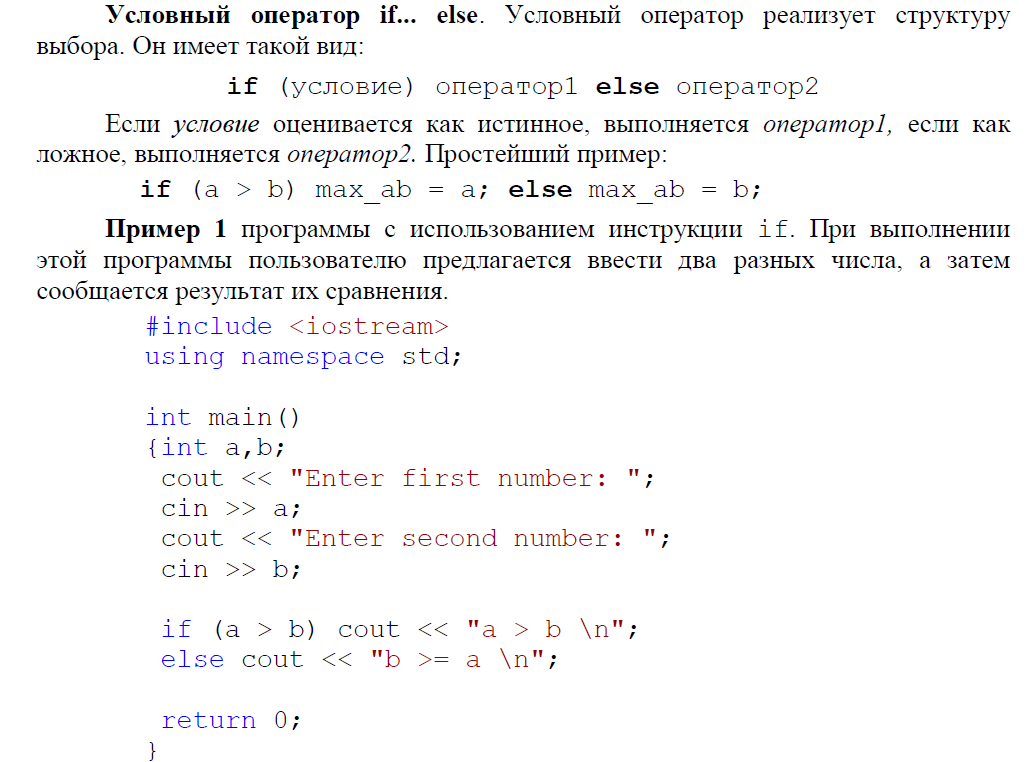
Как и в обычных формулах, для изменения порядка оценки выражения могут применяться круглые скобки. Знак равенства здесь также является *операцией присваивания,* которая сама (и, соответственно, все выражение в целом) возвращает значение. В этом отличие С от других языков, в частности Pascal, где присваивание является оператором, а не операцией. Оператором выражение станет, если поставить после него точку с запятой. 

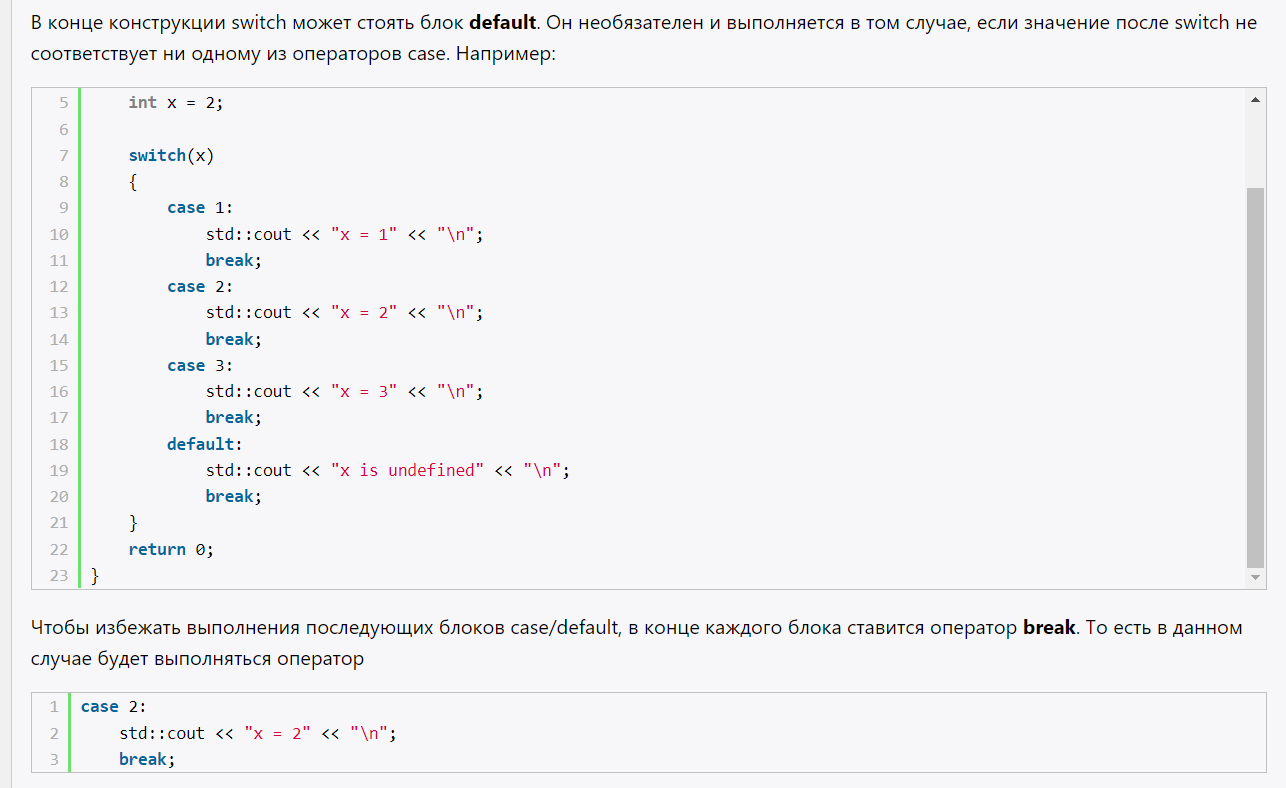
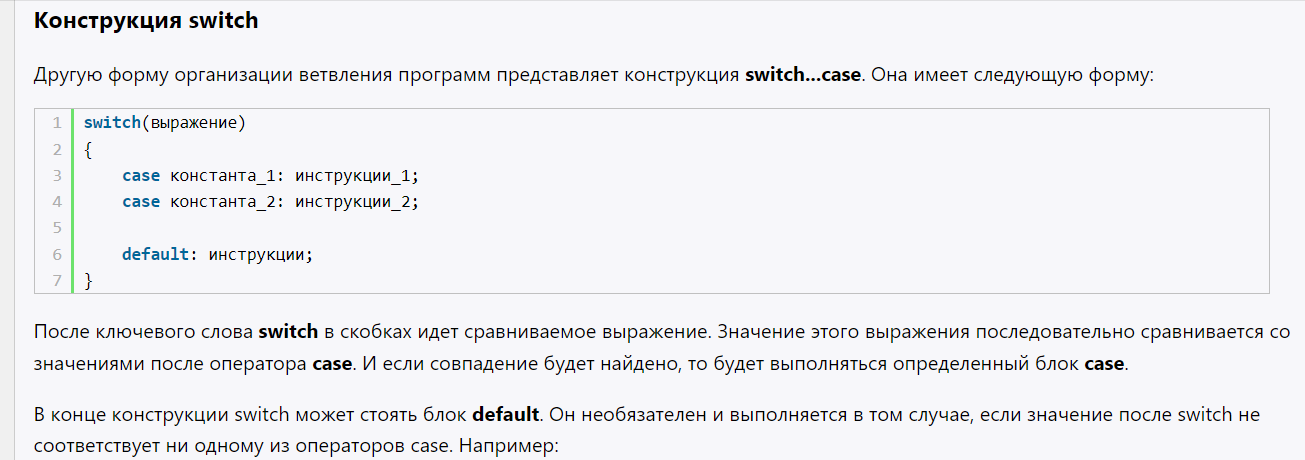


# 17. ТИПЫ ДАННЫХ ЯЗЫКА С++

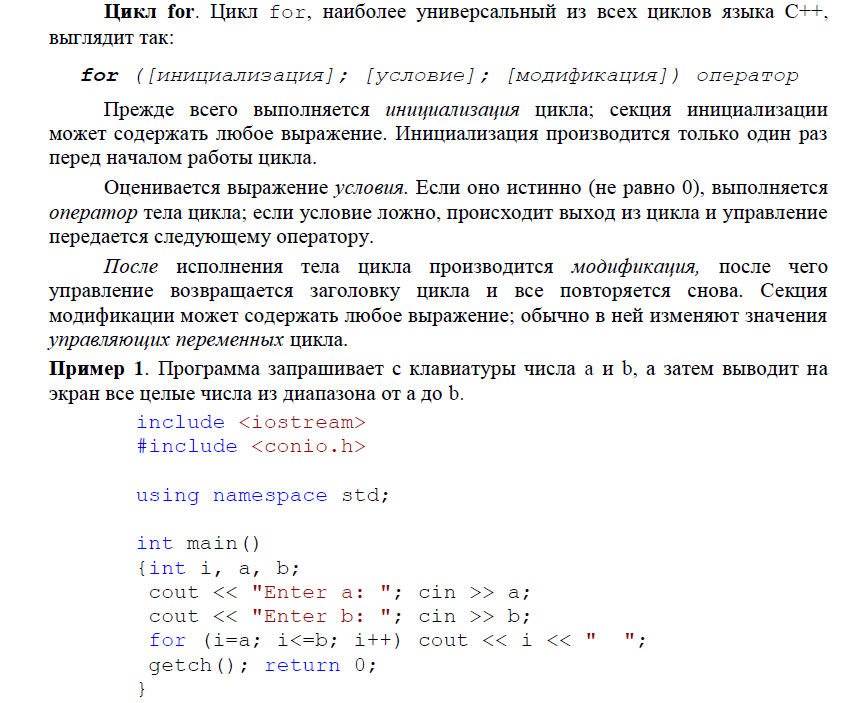
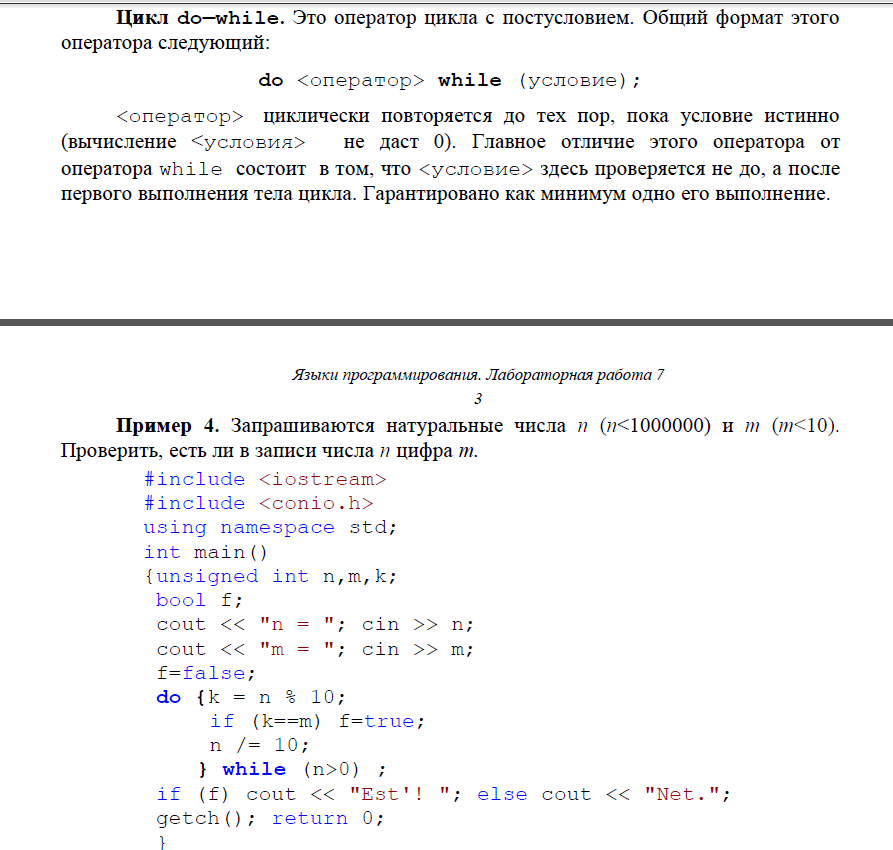
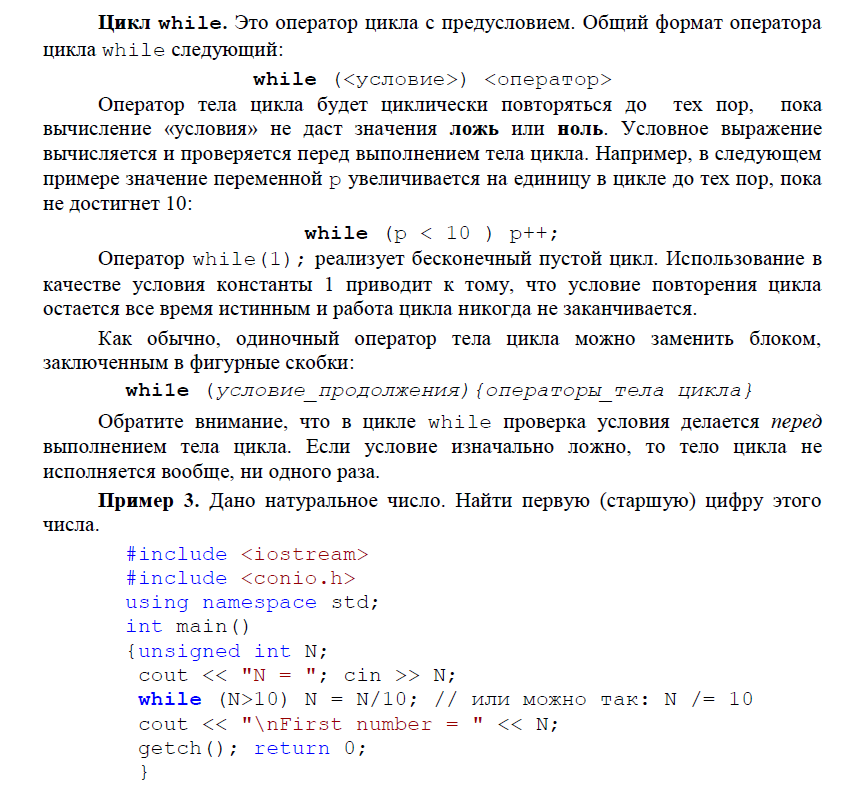
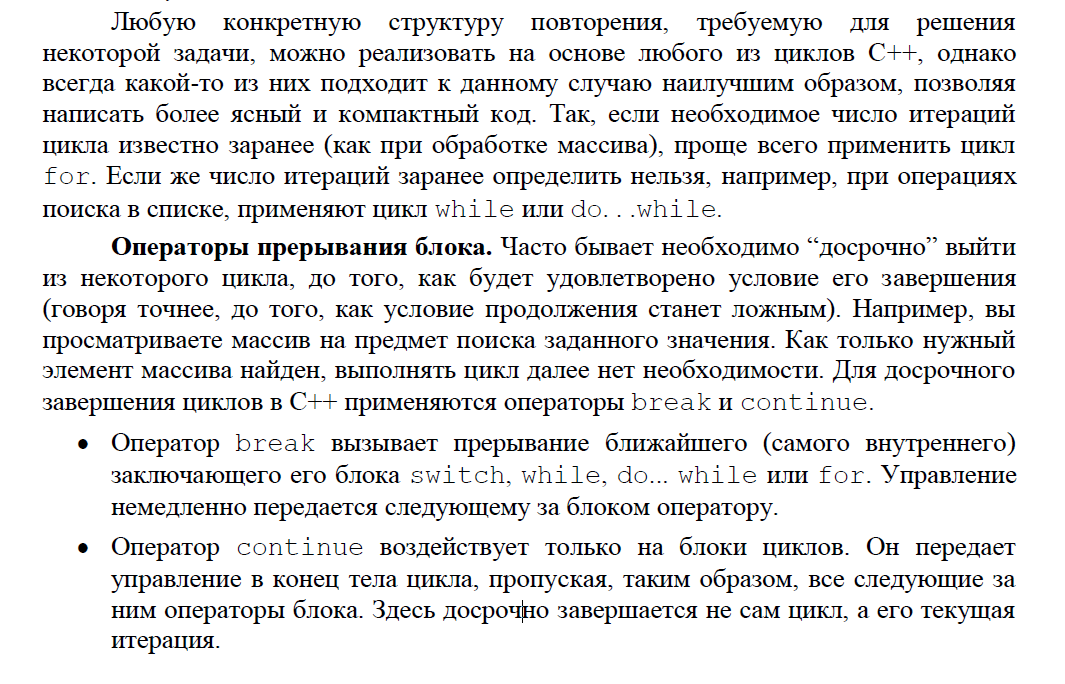


# 18. ОПЕРАТОРЫ УСЛОВНОГО ВЫПОЛНЕНИЯ (IF, SWITCH)

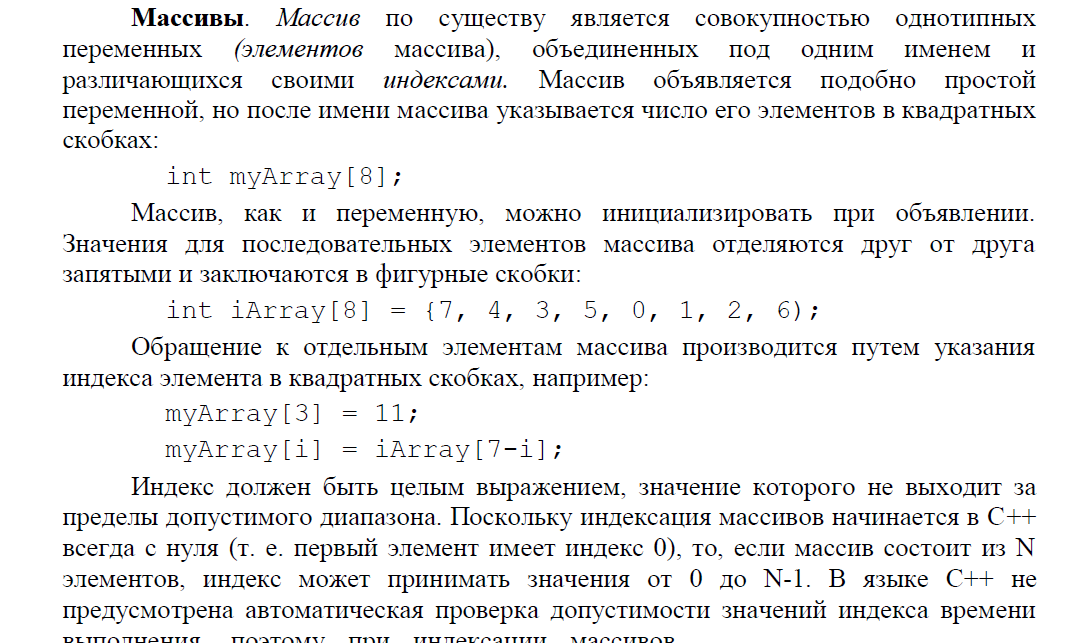
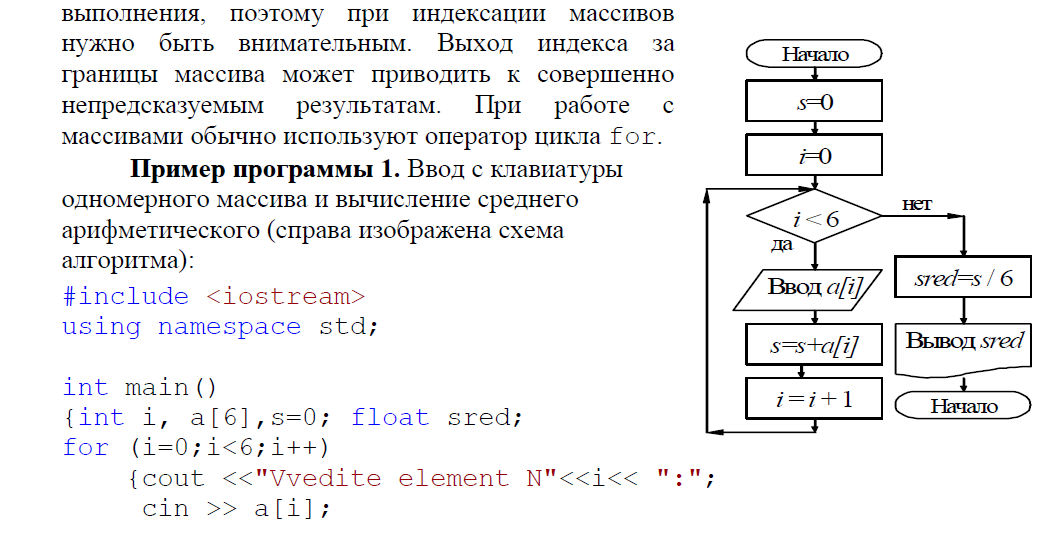


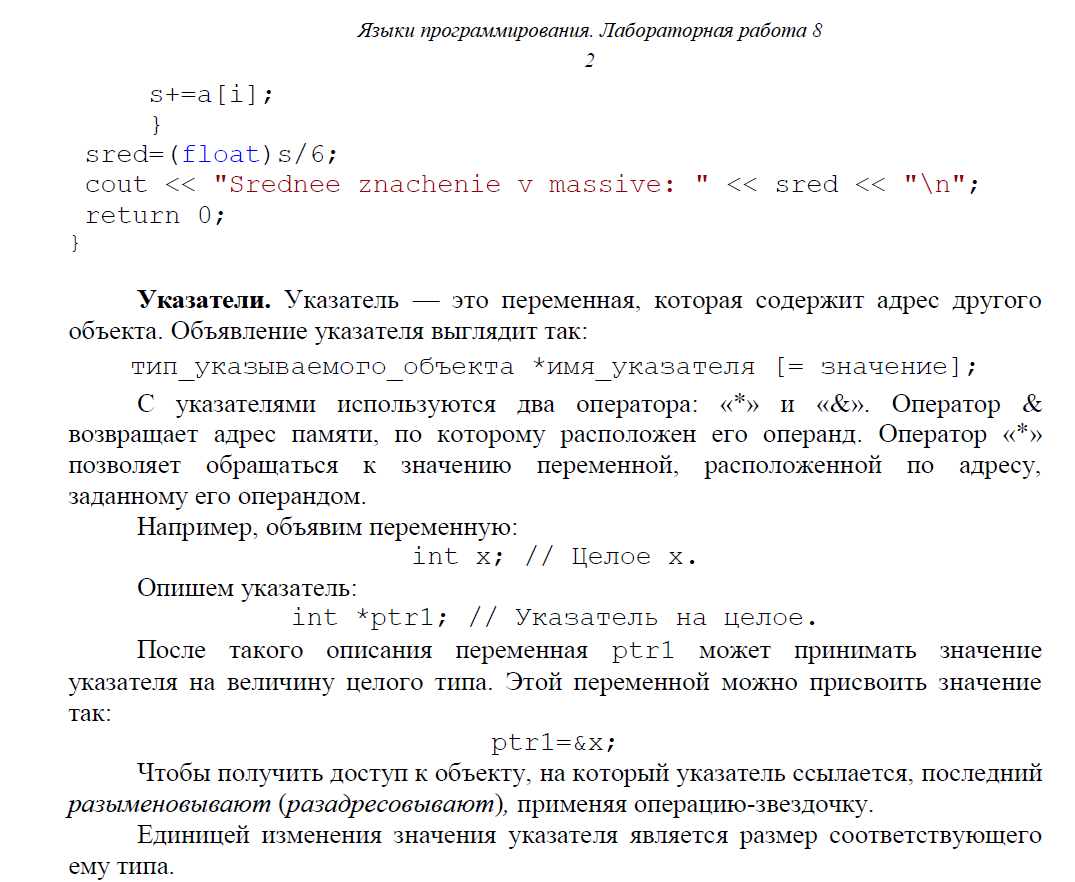


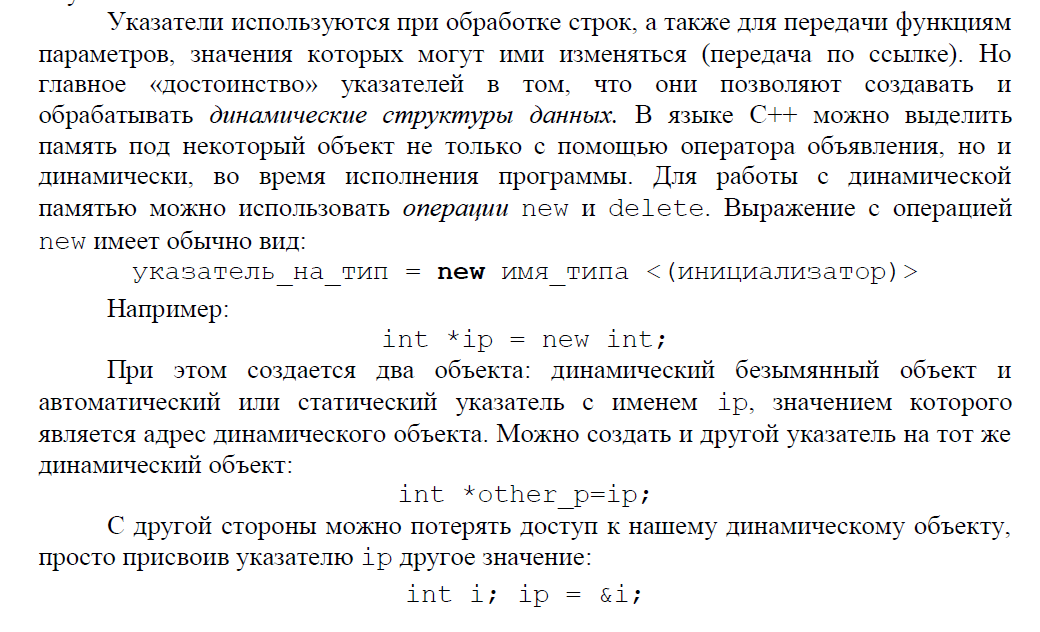
# 19. ОПЕРАТОРЫ ЦИКЛА ЯЗЫКА C++

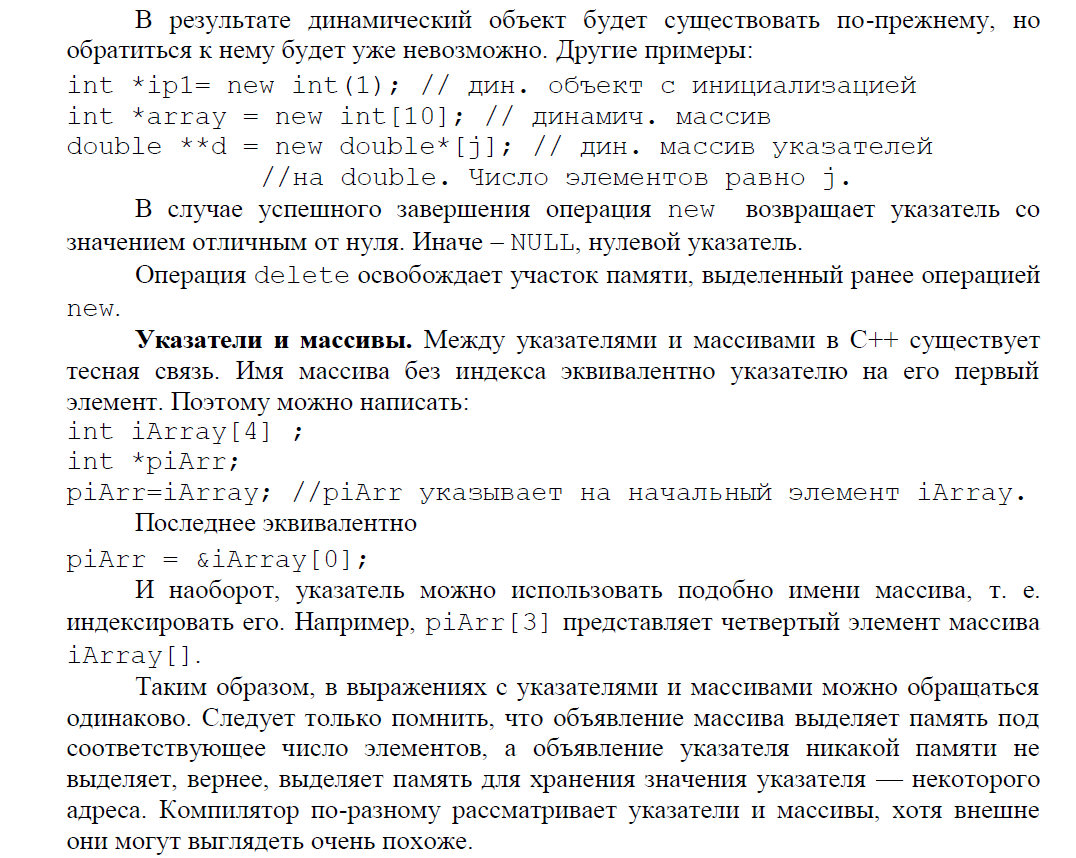
  

# 20. МАССИВЫ В C++: СТАТИЧЕСКИЕ, ДИНАМИЧЕСКИЕ. УКАЗАТЕЛИ







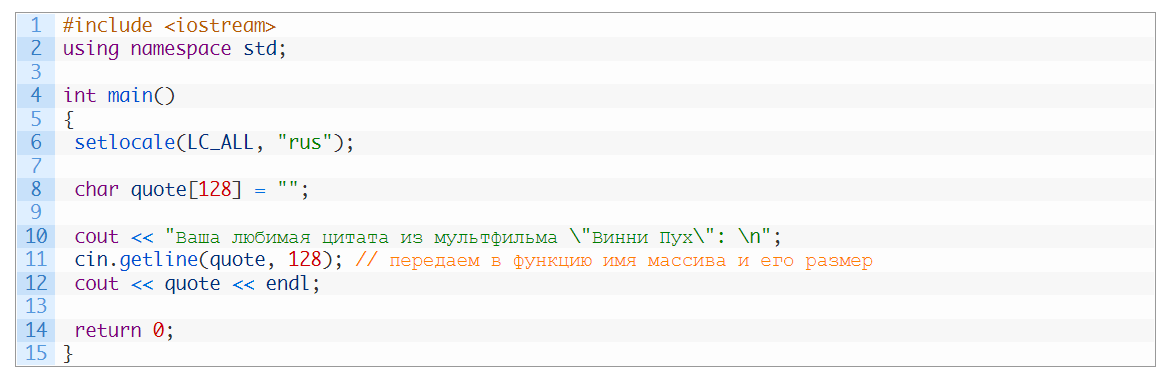


# 21. ОБРАБОТКА СТРОК В С++.

В C++ для хранения строк  используют символьные массивы. Последним символом каждой такой строки является символ **\0** (нулевой символ, нуль-терминатор). Его необходимо учитывать при выделении памяти в массиве, поскольку он тоже занимает одно место.

строки в С++, символьные массивы в С++

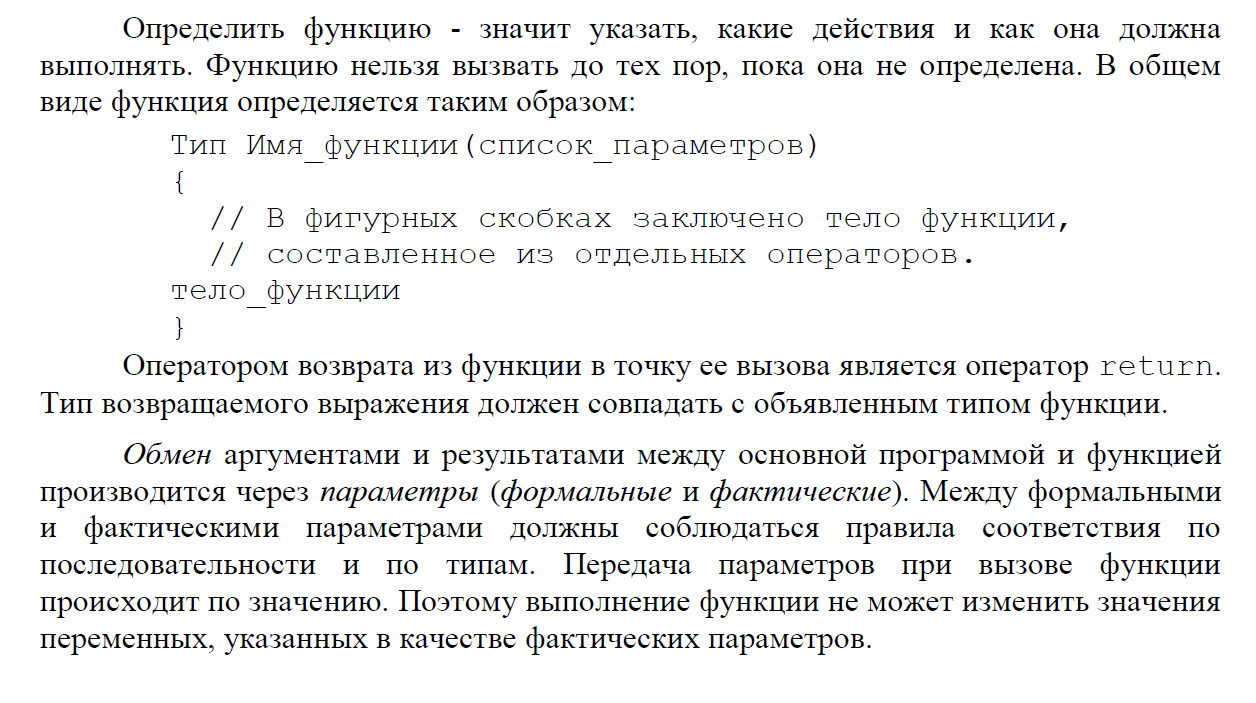
Для взятия символов из потока в C++ есть функции fgets()  **[** fgets(st, n, stdin); **]** и getline(). Например:



Поскольку строковый массив ничем не отличается от обычных массивов, можно делать его и динамическим тоже. В сущности, работа со строковым массивом аналогична обработке обычного массива. В качестве альтернативного метода существует также отдельный строковый класс string, в котором используются свои функции. Он подключается с помощью отдельной библиотеки.

# 22. ФУНКЦИИ В С++. ПЕРЕГРУЗКА ФУНКЦИЙ. РЕКУРСИВНЫЕ ФУНКЦИИ

Функция – часть программы, имеющая собственное название. Любая программа на С++ содержит функцию main() или WinMain(), которая является ее входной точкой. Функция main() — это частный случай *функции* вообще. Функции являются основными ―строительными блоками‖ программы. В программах на С++ можно использовать уже готовые, написанные ранее, «стандартные» библиотечные функции. Например, включив программу строку #include <cmath> можно использовать большое количество математических функций.



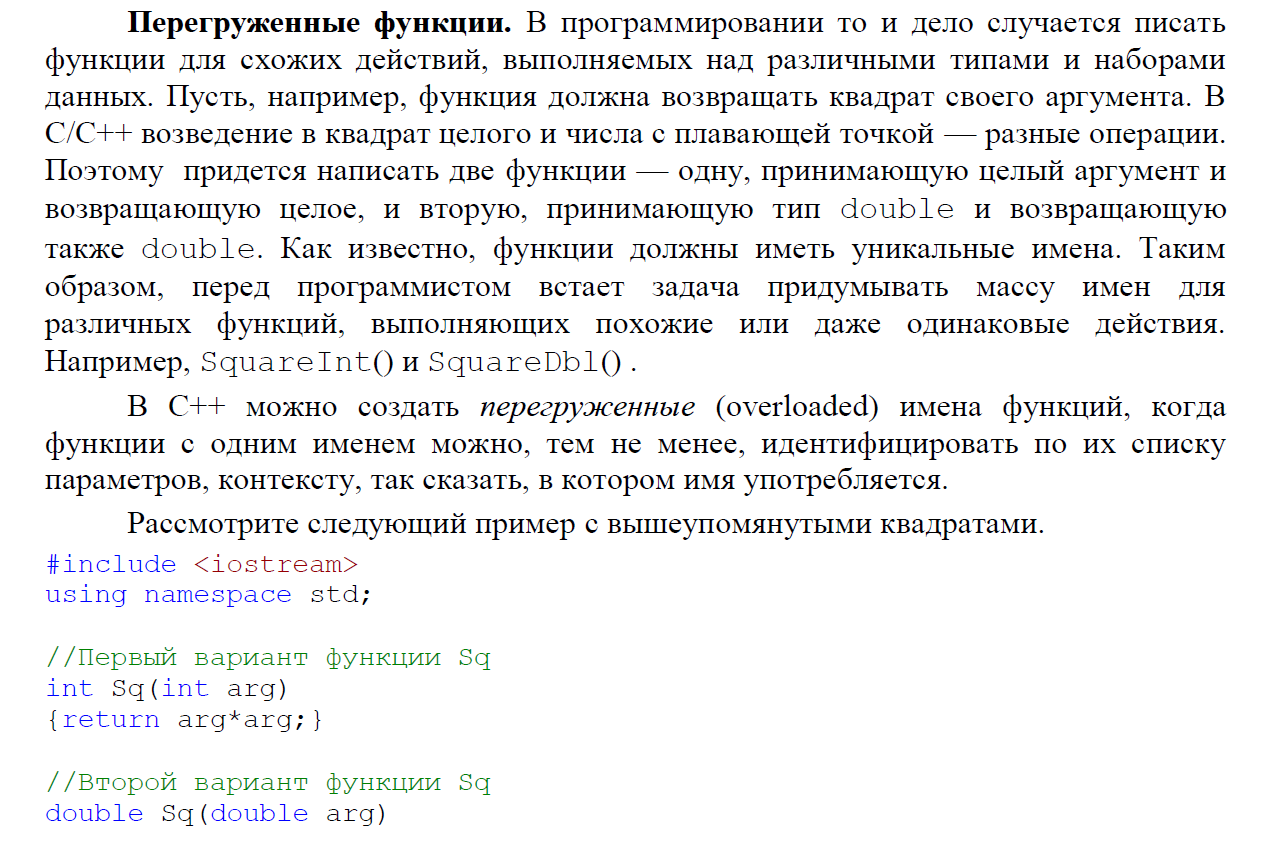


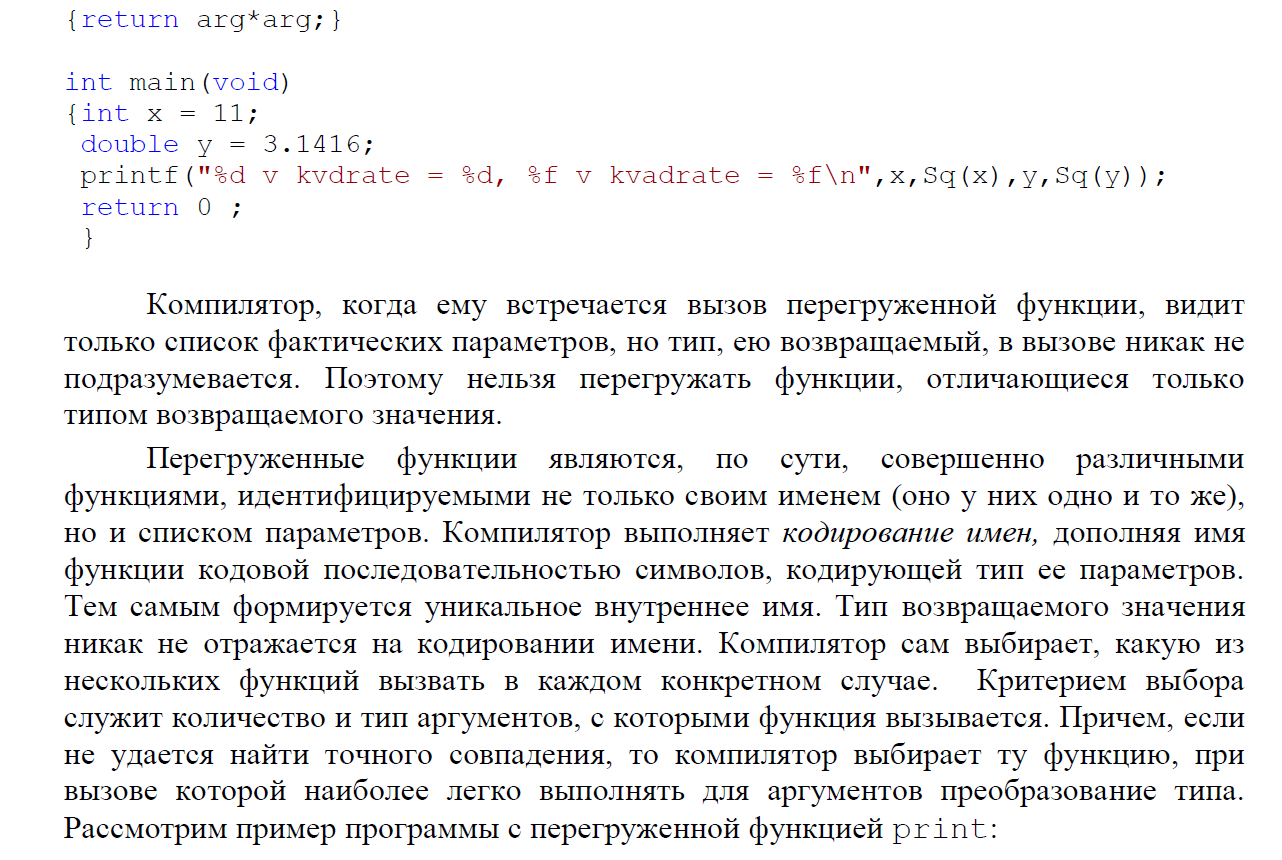


Функция, которая не возвращает результата, объявляется как void. Например, в следующем примере используется функция hello(), которая выводит на экран приветствие. Так как она не возвращает в основную программу никаких результатов, она объявлена как void:

#include <iostream>   
using namespace std;   
void hello(void)   
{cout<<"Hello All\n"; return;}   
int main()   
{cout<<"Primer: ";   
hello();   
return 0; }

Значения аргументов функции по умолчанию. В прототипах функций можно задавать значения аргументов, передаваемых по умолчанию. Задавать значения по умолчанию надо ―начиная справа, например функция DrawCircle (int x, int y=100, int R)





#include <stdio.h>

#include <iostream>

using namespace std;

void print(int i) {printf("%d ", i);}

void print(double x) {printf("%f ", x);}

void print(char\* s) {printf("%s ", s);}

int main()

{int j=5;

float e=2.7183;

double pi=3.1415926;

print(j); print(e); print(pi);

print("Hello All!");

return 0;

}

Рекурсивные функции - это функции, которые вызывают сами себя. Например, так можно организовать вычисление факториала:

int factorial(int n)

{

    if(n>1)

        return n \* factorial(n-1);

    return 1;

}

В функции factorial задано условие, что если число n больше 1, то это число умножается на результат этой же функции, в которую в качестве параметра передается число n-1. То есть происходит рекурсивный спуск. И так далее, пока не дойдем того момента, когда значение параметра не будет равно 1. В этом случае функция возвратит 1.

