1. Структура языка программирования: алфавит языка программирования, идентификаторы, зарезервированные идентификаторы, литералы, ключевые слова, определения. Правила составления идентификаторов в C++ и других языков программирования. Примеры. ЛК\_02

|  |
| --- |
| **алфавит языка**: набор разрешенных символов, кодировка символов исходного кода программ; символы времени трансляции, символы времени выполнения; |
|  **идентификаторы**: правила образования идентификаторов; зарезервированные идентификаторы; |
|  |

**Алфавит языка** **программирования:** **набор** символов, разрешенных к использованию языком программирования. Основывается на одной из кодировок.

Совокупность символов, допускаемых в языке – алфавит языка

**Идентификатор: *имя компонента*** программы (переменной, функции, метки, типа и пр.), составленное программистом по определенным **правилам составления** для С++:

**Идентификатор C++:**

* + - идентификаторы должны начинаться с буквы или подчеркивания;
    - идентификатор не может совпадать с ключевыми словами С++ или с именами библиотечных функций;
    - идентификаторы могут состоять из любого количества символов, но компилятор **гарантирует**, что будет считать значащими: o 31 первых символов идентификаторов, не имеющих внешней связи;o не более 6 значащих символов идентификаторов с внешней связью; идентификаторы чувствительны к регистру.

Длина идентификатора по стандарту не ограничена. Идентификатор создается при объявлении переменной, функции, типа и т. п.

Для Python:

Python Используются символы Unicode. Идентификаторы начинаются с латинской буквы в любом регистре или символа подчёркивания, могут содержать цифры. Не могут совпадать с ключевыми словами (их список можно узнать по import keyword; print(keyword.kwlist), нежелательно переопределять встроенные имена. Имена, начинающиеся с символа подчёркивания, имеют специальное значение. В Python 3 – в идентификаторе допустимы символы любого алфавита в Юникоде, например, кириллицы

**Зарезервированные идентификаторы:** идентификаторы, которые предварительно определены в системе программирования.

1. **Зарезервированные идентификаторы:** идентификаторы, которые предварительно определены в системе программирования.

**Python**

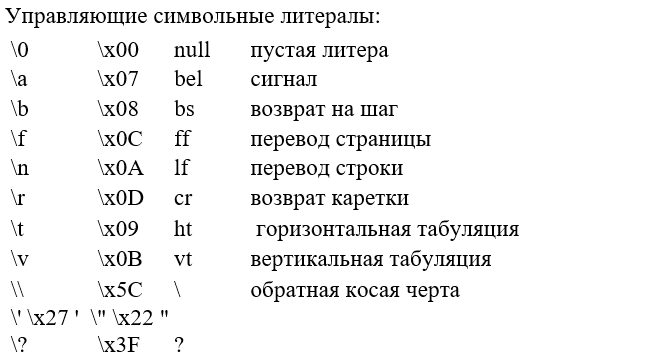
имеет особое значение идентификатор «\_» – используется для хранения результата последнего вычисления.

1. **Зарезервированные идентификаторы С++**: 
   * + - все имена с двумя подчеркиваниями считаются зарезервированным;
       - Кроме того: **is**xxxx, **mem**xxxx, **str**xxxx, **t**oxxxx, **wcs**xxxx, **Eцифра**xxxx, **LC\_X**xxx, **SIGX**xxx, **SIG\_X**xxxx.

1. **Литерал:** элемент программы, который непосредственно представляет значение.

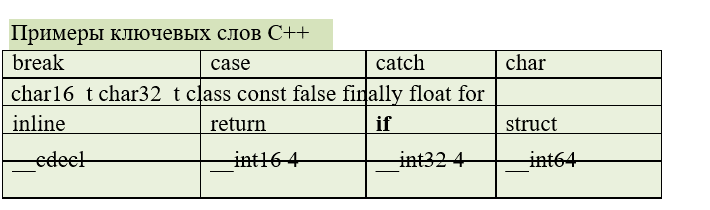
В C++ существует четыре типа литералов:

* + - целочисленный литерал,
    - вещественный литерал, - символьный литерал, - строковый литерал.



**Ключевые слова:**  последовательности символов алфавита языка, имеющие специальное назначение.

Ключевые слова зарезервированы компилятором для обозначения типов переменных, класса хранения, элементов операторов и т.д.



1. Структура языка программирования: фундаментальные типы данных. Что определяет тип данных? Определения, примеры (C++ и другие языки). ЛК\_02

**Фундаментальные типы** (встроенные типы) являются базовыми типами на которых строятся производные типы и типы определённые пользователем.

**Примеры**: Для описания основных типов определены следующие ключевые слова:

* int (целый); (4 байта)
* char (символьный); (1 байт)
* wchar\_t (расширенный символьный); (2 байта)
* bool (логический); (1 байт)
* float (вещественный); (4)
* double (вещественный с двойной точностью). (8)

Существует четыре модификатора основных типов, уточняющих внутреннее представление и диапазон значений стандартных типов:

* short (короткий); (2 байта)
* long (длинный); (4 байта)
* signed (знаковый);
* unsigned (беззнаковый).

— Указатель (pointer) — это переменная, в которой хранится адрес другого объекта (как правило, другой переменной).

— \*void: В C++ существует специальный тип указателя, который называется указателем на неопределённый тип.

Указатель на void применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен (например, если в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов)

Указателю на void можно присвоить значение указателя любого типа, а также сравнивать его с любыми указателями, но перед выполнением каких-либо действий с областью памяти, на которую он ссылается, требуется преобразовать его к конкретному типу явным образом.

— Ссылка (ссылочный тип) - это задание альтернативного имени объекта. Всякое изменение ссылки преобразует не ее, а тот объект, на который она ссылается.

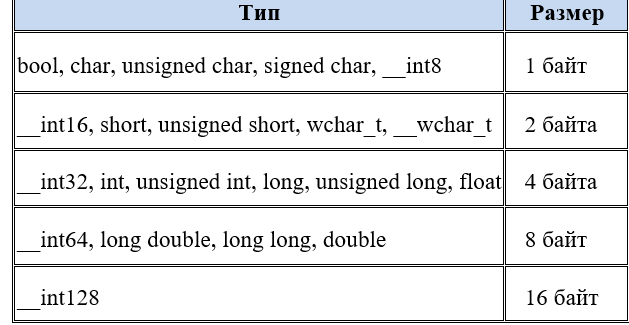
— Указатель на функцию - переменная, которая содержит адрес некоторой функции.

***Тип данных определяет:***

* + - * внутреннее представление данных в памяти компьютера;
      * диапазон значений, которые могут принимать величины этого типа; - операции и функции, которые можно применять к величинам этого типа.

Фундаментальные типы (или встроенные типы) задаются стандартом языка и встроены в компилятор.

Длины и диапазоны значений примитивных типов определяются **стандартом**, а не реализацией:

****

**Фундаментальные типы C++** определены следующие ***ключевые*** слова:

* + - * **int** (целый);
      * **char** (символьный);
      * **wchar\_t** (расширенный символьный);
      * **bool** (логический);
      * **float** (вещественный);
      * **double** (вещественный с двойной точностью); - тип [**void**.](https://ru.wikipedia.org/wiki/Void)

***Модификаторы основных типов***, уточняющие внутреннее представление и диапазон значений стандартных типов:

* + - * **short** (короткий);
      * **long** (длинный);
      * **signed** (знаковый);

**unsigned** (беззнаковый).

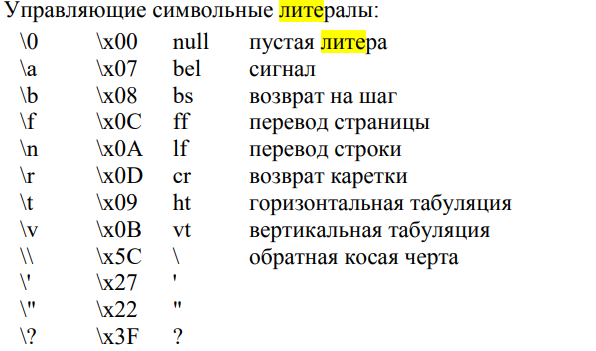
**Java** **8** примитивные типы: [boolean,](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B8%D0%BF) [byte,](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82) [char,](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF) [short,](https://ru.wikipedia.org/w/index.php?title=Short&action=edit&redlink=1) [int,](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF) [long,](https://ru.wikipedia.org/w/index.php?title=Long_%28%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%29&action=edit&redlink=1) [float,](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D0%BE%D0%B4%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D0%BE%D0%B9_%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8) [double.](https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D0%B4%D0%B2%D0%BE%D0%B9%D0%BD%D0%BE%D0%B9_%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8)

Длины и диапазоны значений примитивных типов определяются **стандартом**, а не реализацией:

|  |  |  |
| --- | --- | --- |
| **Тип** | **Длина (в байтах)** | **Диапазон или набор значений** |
| boolean | 1 в массивах, 4 в переменных | true, false |
| byte | 1 | −128..127 |
| char | 2 | 0..216−1, или 0..65535 |
| short | 2 | −215..215−1, или −32768..32767 |
| int | 4 | −231..231−1, или −2147483648..2147483647 |
| long | 8 | −263..263−1, или примерно −9.2·1018..9.2·1018 |
| float | 4 | -(2-2−23)·2127..(2-2−23)·2127, или примерно  −3.4·1038..3.4·1038, а также NaN |
| double | 8 | -(2-2−52)·21023..(2-2−52)·21023, или примерно  −1.8·10308..1.8·10308, а также NaN |

1. Структура языка программирования: литералы, типы литералов, способы задания. Строки. Массивы данных фундаментального типа. Примеры (С++).

**Литерал**-элемент программы, который непосредственно представляет значение. В C++ существует четыре типа литералов: - целочисленный литерал, - вещественный литерал, - символьный литерал, - строковый литерал.



**Строки:** В С++ поддерживаются два типа строк – встроенный тип и класс string из стандартной библиотеки С++.

Встроенный строковый тип перешел к С++ по наследству от С. Строка символов хранится в памяти как массив, и доступ к ней осуществляется при помощи указателя типа char\*.

Класс string (<string>):

минимальный набор операций, которыми должен обладать класс string:

* инициализация массивом символов (строкой встроенного типа) или другим объектом типа string. Встроенный тип не обладает второй возможностью;
* копирование одной строки в другую. Для встроенного типа приходится использовать функцию strcpy();
* доступ к отдельным символам строки для чтения и записи. Во встроенном массиве для этого применяется операция взятия индекса или косвенная адресация;
* сравнение двух строк на равенство. Для встроенного типа используется функция strcmp();
* конкатенация двух строк, получая результат либо как третью строку, либо вместо одной из исходных. Для встроенного типа применяется функция strcat(), однако чтобы получить результат в новой строке, необходимо последовательно задействовать функции strcpy() и strcat();
* вычисление длины строки. Узнать длину строки встроенного типа можно с помощью функции strlen();
* возможность узнать, пуста ли строка. У встроенных строк для этой цели приходится проверять два условия

Класс string стандартной библиотеки С++ реализует все перечисленные операции.

**Массивы данных фундаментальных типов:** коллекция однородных данных, размещенных последовательно в памяти и допускающие доступ по индексу (смещение = индекс\*sizeof(базовый\_тип)).

Массив(array) — это совокупность переменных, имеющих одинаковый тип и объединенных под одним именем. Доступ к отдельному элементу массива осуществляется с помощью индекса.

Многомерные массивы: Доступ к элементу, стоящему на пересечении первой строки и третьего столбца, можно получить двумя способами: либо индексируя массив — mm[0][2], либо используя указатель — \*((int \*)mm+2), где int – это базовый тип массива.

Для двухмерного массива справедлива следующая формула:

mm [j] [к] =\* ((базовый\_тип \*) mm + (i \* длина строки) + к)

Правила адресной арифметики требуют приведения типа указателя на массив к его базовому типу.

1. Структура языка программирования: фундаментальные типы данных. Внутреннее представление величины целого типа, спецификаторы, диапазоны значений целого типа. ???

**Фундаментальные типы** (встроенные типы) являются базовыми типами на которых строятся производные типы и типы определённые пользователем.

**Примеры**: Для описания основных типов определены следующие ключевые слова:

* int (целый); (4 байта)
* char (символьный); (1 байт)
* wchar\_t (расширенный символьный); (2 байта)
* bool (логический); (1 байт)
* float (вещественный); (4)
* double (вещественный с двойной точностью). (8)

Существует четыре модификатора основных типов, уточняющих внутреннее представление и диапазон значений стандартных типов:

* short (короткий); (2 байта)
* long (длинный); (4 байта)
* signed (знаковый);
* unsigned (беззнаковый).

— Указатель (pointer) — это переменная, в которой хранится адрес другого объекта (как правило, другой переменной).

— \*void: В C++ существует специальный тип указателя, который называется указателем на неопределённый тип.

Указатель на void применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен (например, если в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов)

Указателю на void можно присвоить значение указателя любого типа, а также сравнивать его с любыми указателями, но перед выполнением каких-либо действий с областью памяти, на которую он ссылается, требуется преобразовать его к конкретному типу явным образом.

— Ссылка (ссылочный тип) - это задание альтернативного имени объекта. Всякое изменение ссылки преобразует не ее, а тот объект, на который она ссылается.

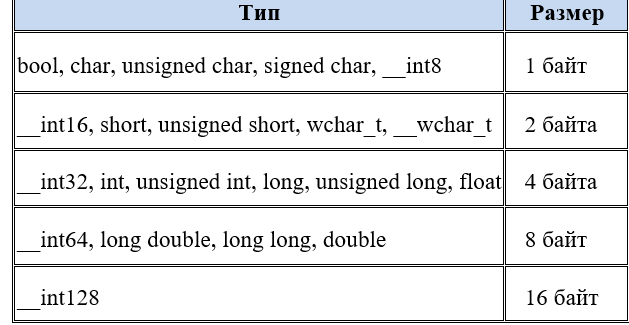
— Указатель на функцию - переменная, которая содержит адрес некоторой функции.

***Тип данных определяет:***

* + - * внутреннее представление данных в памяти компьютера;
      * диапазон значений, которые могут принимать величины этого типа; - операции и функции, которые можно применять к величинам этого типа.

Фундаментальные типы (или встроенные типы) задаются стандартом языка и встроены в компилятор.

Длины и диапазоны значений примитивных типов определяются **стандартом**, а не реализацией:

****

**Фундаментальные типы C++** определены следующие ***ключевые*** слова:

* + - * **int** (целый);
      * **char** (символьный);
      * **wchar\_t** (расширенный символьный);
      * **bool** (логический);
      * **float** (вещественный);
      * **double** (вещественный с двойной точностью); - тип [**void**.](https://ru.wikipedia.org/wiki/Void)

Внутреннее представление величины целого типа:

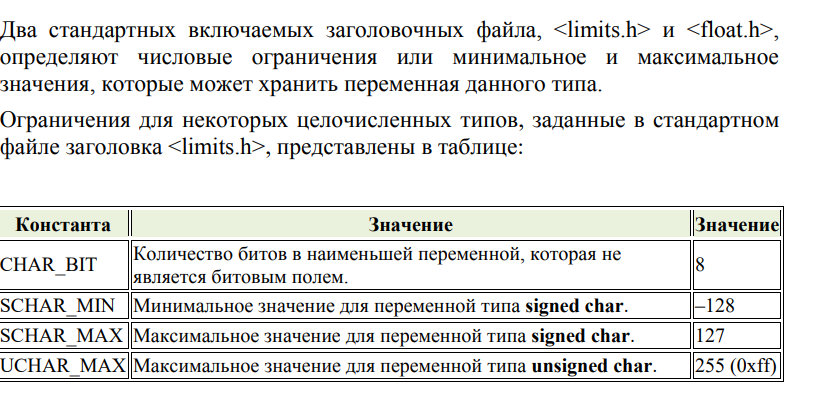
- целое число в двоичном коде.

- спецификатор signed

– старший разряд (бит) числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное)

- спецификатор unsigned: старший разряд (бит) рассматривается как значащий, позволяет представлять только положительные числа.

По умолчанию все целочисленные типы считаются знаковыми, то есть спецификатор signed можно опускать. Диапазон значений зависит от реализации



1. Структура языка программирования: система типов языка программирования, Вывод типов. Преобразование типов: автоматическое преобразование, явное преобразование. Определения и примеры (С++).

**Преобразование типов**  — преобразование значения одного типа в значение другого типа.

Выделяют приведения типов:

* явные
* неявные(автоматическое).

Явное преобразование задаётся программистом в тексте программы.

Явное преобразованиечасто применяется там, где это можно сделать неявно для того, чтобы указать, что преобразование делается осознано.

Механизм неявных преобразований может быть отключён посредством явного указания в тексте программы требуемого преобразования типов.

Неявное преобразование выполняется транслятором (компилятором или интерпретатором) по правилами, описанным в стандарте языка.

1. Структура языка программирования: система типов языка программирования, Вывод типов. Преобразование типов: расширяющее преобразование, сужающее преобразование. Назначение оператора sizeof языка программирования С++. Примеры.
2. Структура языка программирования: пользовательские типы данных (структуры, перечисления, объединения). Определение, примеры (C++). Массивы данных пользовательского типа. Перегрузка операторов для пользовательских типов.

**Пользовательские типы данных:** типы, создаваемые пользователем, на основе того, что доступно в языке; всегда есть объявление типа.

**С++: struct** addres{ //**структура** - это набор переменных, объединенных общим именем. Она обеспечивает удобный способ организации взаимосвязанных данных.

string street;

int number\_of\_house;

};

**struct** rgb\_color{ //**битовое поле** – особый вид полей структуры, используемый для плотной упаковки данных.

unsigned red\_value: 3;

unsigned green\_value: 3;

unsigned blue\_value: 3;

};

**union** bit{ //**объединение** - это пользовательская переменная, которая может хранить объекты различного типа и размера, для их размещения выделяется одна общая память, размерность определяется размерностью максимального элемента объединения.

char ch;

int i;

};

**enum** coin{ //**перечисление** - набор именованных целых констант

penny, nickel, dime

};

**typedef** unsigned int nomer; // определяет новое имя типа данных, новый тип при этом не создается, уже существующий тип получает новое имя

***Массивы данных пользовательского типа:***  это совокупность данных (переменных) пользовательского типа и объединенных под одним именем. Доступ к отдельному элементу массива осуществляется с помощью индекса.

* **Массив struct:**

struct{

int i;

char ch;

} S[5];

Обращение: S[0].i;

* **Массив union:**

union{

float f;

int i;

} U[5];

Обращение: U[1].f;

**Перегрузка операторов для пользовательских типов:**

(возможность перегрузки есть только для пользовательских типов/классов) реализация в однойобласти видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются

1. Структура языка программирования: пользовательские типы данных. Определение структуры, примеры (C++). Перегрузка операторов для пользовательских типов.
2. Структура языка программирования: понятие конфликта имен, область видимости переменных, пространства имен, псевдонимы пространства имен.

**Конфликт имен** возникает, когда два одинаковых идентификатора находятся в одной области видимости, и компилятор не может понять, какой из этих двух следует использовать в конкретной ситуации. Компилятор или линкер выдаст ошибку, так как у них недостаточно информации, чтобы решить эту неоднозначность.

**Область видимости переменных в C++**: доступность переменных по их идентификатору в разных частях (блоках программы).

Переменная должна быть объявлена до ее использования; переменная объявленная во внутреннем блоке (локальная переменная {…}) не доступна во внешнем; переменная объявленная во внешнем блоке доступна во внутреннем; во внутреннем блоке переменная может быть переобъявлена.

**Пространство имен:** именованная область видимости. Применяется для разрешения конфликтов имен. Имена, объявленные в одном пространстве имен, не будут вступать в конфликт с аналогичными именами, объявленными в другой области.

Примеры**:** namespace, using, псевдонимы пространства имен.

**Псевдоним пространства имен**: Имена пространств имен должны быть уникальными, из-за чего зачастую они получаются не слишком короткими. Если длинное имя оказывается трудночитаемым или его сложно вводить в файле заголовка, где нельзя использовать директиву using, можно создать псевдоним пространства имен, который будет служить в качестве сокращения фактического имени.

Пространства имен помогают предотвратить конфликты имен, но не устранить их полностью. Такой конфликт может произойти, когда одно и то же имя объявляется в двух разных пространствах имен и затем предпринимается попытка сделать видимым оба пространства. В таком случае для указания предполагаемого пространства имен явным образом можно воспользоваться описателем псевдонима пространства имен «::».

1. Структура языка программирования: определение выражения, его состав, порядок вычисления выражения. Символ окончания последовательности. Примеры (С++).
2. Структура языка программирования: выражения (lvalue, rvalue, побочные эффекты, точка последовательности, унарные, бинарные и тернарные выражения), константные выражения, укороченное вычисление. Примеры (С++).

**lvalue:** именующее выражение -это ссылка на значение – могут использоваться в левой и правой части оператора присваивания (имя переменной, ссылка на элемент массива по индексу, вызов функции возвращающей указатель, всегда связаны с областью памяти, адрес которой известен)

**Пример:** int n; n = 3; выражение n – lvalue, ссылается на объект int

**rvalue**: значащее выражение – могут использоваться только в правой части оператора присваивания (не связано с адресом, а только со значением, литералы, вызов функции, возвращающей значение)

**Пример:** литералы

**Побочные эффекты:** любые действия**,** изменяющие состояние среды выполнения; если при вычислении выражения значение переменной, входящей в это выражение, изменился, то говорят, что произошел побочный эффект.

**Точка последовательности:** момент времени, когда побочные эффекты от вычисленных выражений осуществились, а побочные эффекты следующих в последовательности выражений не начались.

**С++**: точка последовательности обозначается символом точка с запятой (;)

Величины, над которыми выполняются операции, называются **операндами**. В зависимости от количества операндов операции могут быть унарными (один операнд) и бинарными (два операнда).

* **Унарные:** выражения с одним операндом
* **Бинарные:** выражения с двумя операндами
* **Тернарные:** выражения, возвращающие свой второй или третий операндов зависимости от значения логического выражения, заданного первым операндом

**Константные выражения:** выражение, которое должно быть вычислено на этапе компиляции.

**Пример:** const int k = 5;

1. Структура языка программирования: инструкции языка программирования, объявление, определение, инициализация, простые и составные инструкции, инструкции выбора, инструкции циклов, инструкции переходов, примеры (С++).

**Инструкция (оператор)** — наименьшая автономная часть языка программирования; команда или набор команд. Программа обычно представляет собой последовательность инструкций.

**Объявление** включает в себя указание идентификатора, типа, а также других аспектов элементов языка, например, переменных и функций. Объявление используется, чтобы уведомить компилятор о существовании элемента. (Определение – реализация, при этом компилятор выделяет место. Переменная может быть объявлена несколько раз, а определена только один раз. Объявление предоставляет основные свойства символа: его тип и его название. Определение предоставляет все детали этого символа — если это функция, что она делает; если это класс, какие у него поля и методы; если это переменная, где эта переменная находится. Инициализация – придание начального значения переменной при её объявлении)

**Простые инструкции** завершаются точкой с запятой.

**Составные инструкции**, или блоки ({ }), состоят из одной или более инструкций (каждая из которых сама может быть составной), заключенных в фигурные скобки.

**Инструкции выбора** *if*и *switch*, позволяют выполнять те или иные участки кода в зависимости от выполнения условий.

**Инструкции циклов** используются для многократного повторения фрагментов кода. Цикл *for* мы можем использовать, если знаем точное количество действий (итераций). Когда мы не знаем, сколько итераций должен произвести цикл, нам понадобится цикл *while* или *do…while*. Цикл *do…while* почти ничем не отличается от цикла *while*, за исключением того, что тело цикла гарантированно выполняется хотя бы один раз.

**Инструкции переходов** выполняют немедленную локальную передачу контроля. Оператор *break* завершает выполнение ближайшего внешнего цикла или условного оператора, в котором он находится. Управление передается оператору, который расположен после оператора, при его наличии. Инструкцию *continue* можно располагать только внутри цикла. Она вызывает переход к следующей итерации самого внутреннего содержащего ее цикла. С помощью *return* функция возвращает управление в программу, откуда была вызвана. Если за *return* следует выражение, то его значение возвращается вызвавшей эту функцию программе. Оператор *goto* осуществляет безусловную передачу управления оператору, метка которого задана идентификатором. Метка оператора имеет значение только для оператора *goto*; в остальных случаях метки операторов игнорируются. Повторное объявление меток невозможно.

1. Структура языка программирования: инструкции языка программирования, инструкции обработки исключений, примеры (С++).

**Инструкция (оператор)** — наименьшая автономная часть языка программирования; команда или набор команд. Программа обычно представляет собой последовательность инструкций.

**Инструкции обработки исключений** используют три ключевых слова: *try*, *catch* и *throw*. Те инструкции программы, где ожидается возможность появления исключительных ситуаций, содержатся в бло­ке *try*. Если в блоке *try* возникает исключение, т. е. ошибка, то генерируется исключение throw. Исклю­чение перехватывается, используя *catch*, и обрабатывается.

1. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Функции: передача параметров по значению и по ссылке, передача переменного числа параметров в функции C++. Примеры.

**Программные конструкции:**

**Блоки**: логически сгруппированный набор идущих подряд инструкций в [исходном коде](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) программы, является основой парадигмы [структурного программирования](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)

**Пример С++:**

{

int a = 1;

int b = 2;

int c = a + b;

return c;

}

**Функции:** фрагмент программного кода, к которому можно обратиться из другого места программы

**Пример С++:**

void name(char\* text)

{

std::cout<<text<< std::endl;

}

**Процедуры:** функции не возвращающие значения, return не обязателен, но желателен для явного указания завершения функции

**Пример С++:**

 void ValidMessage()

 {

         cout << "Пароль введен верно" << endl;

 }

**Передача параметров:** передача параметров в функцию происходит через стек. Код, вызывающий функцию, знает, сколько параметров ей передать и каковы значения этих параметров

**Передача параметров**

**по значению:** они при выходе из функции не изменятся

**Пример С++:**

int func(int k){

k\*=2;

return k;

}

void main(){

int z=1, y=3, k;

k=func(z)+func(y);

cout<<z<<" "<<y;

}

Z = 1, y = 3

**по ссылке:** при выходе из функции из значения могут измениться

int func(int &k){

k\*=2;

return k;

}

void main(){

int z=1, y=3, k;

k=func(z)+func(y);

cout<<z<<" "<<y;

}

Z = 2, y = 6

**Передача переменного числа параметров в функции С++:**

По умолчанию параметры передаются в функцию через стек. Поэтому, технически, нет ограничения на количество передаваемых параметров. Функции с переменным числом параметров объявляются как обычные функции, но вместо недостающих аргументов ставится многоточие. Количество параметров и их типы известны только при вызове функции.

Передать функции параметры можно двумя способами:

* явно передать обязательный аргумент, задающий число параметров;
* добавить в конец списка параметр с уникальным значением, по которому будет определяться конец списка параметров.

Общий принцип работы следующий: в функции для доступа к списку параметров устанавливается указатель, значением которого будет адрес явного параметра в списке, далее изменяется значение этого указателя, чтобы переместиться на следующий параметр.