Оглавление

[1. Современные подходы к разработке программного обеспечения (ПО): понятие конструирования ПО, основные этапы конструирования ПО. 1](#_Toc105856641)

[2. Современные подходы к разработке программного обеспечения: понятие конструирования ПО; результат конструирования; решения, которые принимаются при конструировании. 1](#_Toc105856642)

[3. Система программирования: определение, состав, назначение. Классический жизненный цикл разработки ПО. Текстовый редактор. Интегрированная среда разработки. Программный продукт. Общая схема преобразования исходного кода в процесс операционной системы. 1](#_Toc105856643)

[4. Интегрированная среда разработки: определение, состав, назначение. Транслятор: виды трансляторов. Транслятор: определение, назначение, примеры. Интерпретатор: определение, назначение, примеры. 2](#_Toc105856644)

[5. Язык программирования: определение, назначение, примеры. Исходный код. Язык ассемблера. Транслятор: определение, назначение, примеры. Алфавит языка. Ассемблер. Объектный код, объектный модуль. Общая схема преобразования исходного кода в процесс операционной системы. 3](#_Toc105856645)

[6. Компоновщик: определение, назначение. Загрузочный модуль. Загрузчик: определение, назначение. Общая схема преобразования исходного кода в процесс операционной системы. 3](#_Toc105856646)

[7. Иерархическая структура компонентов в Visual C++: глобальный контейнер, компоненты, входящие в его состав. Структура проекта Visual C++. 4](#_Toc105856647)

[8. Иерархическая структура компонентов в Visual C++: многофайловый проект, предварительно откомпилированные заголовки: назначение, создание, применения. Пример (C++). 4](#_Toc105856648)

[9. Кодировка: определение, назначение, примеры. Кодировка ASCII. Структура кодировки Windows-1251. Отличие ASCII и Windows-1251. Символы времени выполнения: установка кодовой страницы языкового стандарта. 5](#_Toc105856649)

[10. Кодировка: определение, назначение, примеры. Кодировка UNICODE: назначение, структура, UCS, UTF. Прямой (LE) и обратный (BE) порядок байт. BOM: определение, назначение, примеры. 5](#_Toc105856650)

[11. Структура языка программирования: алфавит языка программирования, идентификаторы, зарезервированные идентификаторы, литералы, ключевые слова, определения. Правила составления идентификаторов в C++ и других языков программирования. Примеры. 6](#_Toc105856651)

[11.Структура языка программирования: алфавит языка программирования, идентификаторы, зарезервированные идентификаторы, литералы, ключевые слова, определения. Правила составления идентификаторов в C++ и других языков программирования. Примеры. 8](#_Toc105856652)

[12.Структура языка программирования: фундаментальные типы данных. Что определяет тип данных? Определения, примеры (C++ и другие языки) 11](#_Toc105856653)

[13.Структура языка программирования: литералы, типы литералов, способы задания. Строки. Массивы данных фундаментального типа. Примеры (С++). 14](#_Toc105856654)

[14.Структура языка программирования: фундаментальные типы данных. Типы данных для хранения вещественных значений. Стандарт IEEE 754. Что определяет стандарт IEEE 754 16](#_Toc105856655)

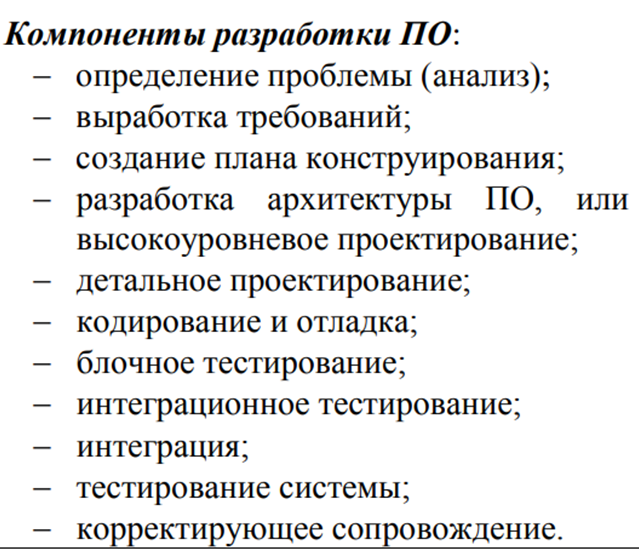
[15.Структура языка программирования: фундаментальные типы данных. Внутреннее представление величины целого типа, спецификаторы, диапазоны значений целого типа. 19](#_Toc105856656)

[16. Структура языка программирования: система типов языка программирования, Вывод типов. Преобразование типов: автоматическое преобразование, явное преобразование. Определения и примеры (С++) 22](#_Toc105856657)

[17. Структура языка программирования: система типов языка программирования, Вывод типов. Преобразование типов: расширяющее преобразование, сужающее преобразование. Назначение оператора sizeof языка программирования С++. Примеры. 25](#_Toc105856658)

## 1. Современные подходы к разработке программного обеспечения (ПО): понятие конструирования ПО, основные этапы конструирования ПО.

**Конструирование** – единственный процесс, который выполняется всегда – это процесс создания какого-нибудь объекта, может включать в себя некоторые аспекты планирования, проектирования и тестирования.



## 2. Современные подходы к разработке программного обеспечения: понятие конструирования ПО; результат конструирования; решения, которые принимаются при конструировании.

**Конструирование** – единственный процесс, который выполняется всегда – это процесс создания какого-нибудь объекта, может включать в себя некоторые аспекты планирования, проектирования и тестирования.

**Результат конструирования** – исходный код – часто является единственным верным описанием программы

**Основные решения, которые принимаются при конструировании**:

− выбор языка программирования;

− конвенции программирования (правила и стандарты кода)

− выбор технологий;

− выбор основных методик конструирования.

## 3. Система программирования: определение, состав, назначение. Классический жизненный цикл разработки ПО. Текстовый редактор. Интегрированная среда разработки. Программный продукт. Общая схема преобразования исходного кода в процесс операционной системы.

**Определение**: комплекс программных средств, предназначенный для реализации программного обеспечения (написания и отладки программного кода).

**Состав:**

* **трансляторы**
* **компоновщики**
* **профилировщики**
* **программные библиотеки**,
* **редакторы кода**,
* **системы поддержки версий**.

**Назначение:** система программирования представляет собой совокупность средств разработки программ, обеспечивающих автоматизацию составления и отладки программ пользователя.

**Жизненный цикл ПО** – непрерывный процесс с момента принятия решения о создании ПО до снятия его с эксплуатации.

**Интегрированная система разработки:** набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

**Программный продукт:** программа, работающая без авторского присутствия; исполняется, тестируется, конфигурируется без присутствия автора; сопровождается документацией.

## 4. Интегрированная среда разработки: определение, состав, назначение. Транслятор: виды трансляторов. Транслятор: определение, назначение, примеры. Интерпретатор: определение, назначение, примеры.

**Интегрированная среда разработки (integrated development environment – IDE):** – набор инструментов для разработки и отладки программ, имеющий общую интерактивную графическую оболочку, поддерживающую выполнение всех основных функций жизненного цикла разработки программы.

**Состав**

* [текстовый редактор](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80),
* [Транслятор](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) ([компилятор](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) и/или [интерпретатор](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80)),
* средства автоматизации сборки,
* [отладчик](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA).

**Интегрированная среда разработки**, используется программистами для разработки создания и тестирования программ или программного обеспечения.

**Трансля́тор** — программа или техническое средство, выполняющее трансляцию программы.

**Виды**

* *Диалоговый*
* *Синтаксически-ориентированный*
* *Однопроходной*
* *Многопроходной*
* *Оптимизирующий*
* *Тестовый*
* *Обратный*

**Интерпретатор** – разновидность транслятора. Переводит и выполняет программу с языка высокого уровня в машинный код строка за строкой.

## 5. Язык программирования: определение, назначение, примеры. Исходный код. Язык ассемблера. Транслятор: определение, назначение, примеры. Алфавит языка. Ассемблер. Объектный код, объектный модуль. Общая схема преобразования исходного кода в процесс операционной системы.

**Язык программирования**: формальная знаковая система, предназначенная для записи компьютерных программ. Знаковая система определяет набор лексических, синтаксических и семантических правил написания программы (программного кода). Язык программирования представляется в виде набора спецификаций, определяющих его синтаксис и семантику.

Примеры: процедурно-ориентированные, объектно-ориентированные, декларативные, операторные, функциональные, скриптовые, проблемно-ориентированные, машинно-зависимые (ассемблеры) и т.п.

**Исходный код (исходная программа)**: текст программы, написанный на языке программирования.

Программа на исходном языке (исходный код) готовится с помощью текстовых редакторов (программ, позволяющих подготовить исходный код программы) и в виде текстового файла или раздела библиотеки поступает на вход транслятора.

**Язык ассемблера:** машинно-ориентированный язык программирования (для конкретной архитектуры компьютера, команды которого соответствуют машинным командам)

**Транслятор:** программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке.

**Трансляторы** (компиляторы) полностью обрабатывают весь исходный текст программы (исходный код). Они просматривают его в поисках синтаксических ошибок (иногда несколько раз), проверяют семантические правила языка.

**Интерпретатор** - разновидность транслятора. Переводит и выполняет программу с языка высокого уровня в машинный код строка за строкой

**Примеры:** диалоговый, синтаксически-ориентированный, однопроходной, многопроходной, оптимизирующий, тестовый, обратный

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

**Ассемблер:** транслятор с исходного кода на языке ассемблера в программу на машинном языке (язык, который может интерпретироваться процессором).

**Объектный код:** результат работы транслятора. Один файл объектного кода – **объектный модуль**.

## 6. Компоновщик: определение, назначение. Загрузочный модуль. Загрузчик: определение, назначение. Общая схема преобразования исходного кода в процесс операционной системы.

**Компоновщик** (linker, редактор связей): программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль. Если программа собирается из нескольких объектных файлов, компоновщик может собирать эти файлы в единый исполнимый модуль, вычисляя и подставляя адреса вместо символов, в течение времени компоновки (статическая компоновка) или во время исполнения (динамическая компоновка).

**Загрузочный код:** результат работы компоновщика. Один файл загрузочного кода – **загрузочный модуль**.

**Загрузчик** (loader): программа, обычно входящая в состав операционной системы, предназначенная для запуска процесса операционной системы на основе загрузочного модуля.

## 7. Иерархическая структура компонентов в Visual C++: глобальный контейнер, компоненты, входящие в его состав. Структура проекта Visual C++.

Глобальный контейнером (компонент, включающий в себя другие компоненты) является ***Решение***. Решение может содержать один или несколько проектов.

Проекты являются независимыми компонентами. Они имеют собственную структуру, состоящую из четырех основных каталогов:

***Внешние зависимости*** – содержит ссылки на все модули, которые использует программа.

***Файлы заголовков*** – содержит файлы кода С++ с расширением h.

***Исходные файлы*** – содержит файлы кода С++ с расширением срр.

***Файлы ресурсов*** – содержит файлы, непосредственно не относящиеся к языку С++, но необходимые для работы приложения. Например, мультимедийные файлы.

Код программного проекта может иметь сложную структуру и состоять из нескольких файлов исходного кода и конфигурационных файлов.

## 8. Иерархическая структура компонентов в Visual C++: многофайловый проект, предварительно откомпилированные заголовки: назначение, создание, применения. Пример (C++).

Общие – ключ Уровень предупреждений позволяет отключить все предупреждения (/W0), либо ужесточить уровень проверок и считать все предупреждения ошибками (/Wall).

**Предварительно откомпилированные заголовки** позволяют их включить/отключить, определить имя создаваемого предварительно откомпилированного заголовочного файла и местоположение для полученного выходного файла (с расширением pch).

/C Сохраняет комментарии на этапе предварительной обработки.

/FI Выполняет предварительную обработку указанного включаемого файла

/U Удаляет предварительно определенный макрос

## 9. Кодировка: определение, назначение, примеры. Кодировка ASCII. Структура кодировки Windows-1251. Отличие ASCII и Windows-1251. Символы времени выполнения: установка кодовой страницы языкового стандарта.

**Кодировка** - таблица, задающая кодировку конечного множества символов алфавита (обычно элементов текста: букв, цифр, знаков препинания). Такая таблица сопоставляет каждому символу последовательность длиной в один или несколько символов другого алфавита (точек и тире в коде Морзе,  нулей и единиц (битов) в компьютере).

Примеры: ASCII, Windows-1251, Кодировки, UTF-8, UTF-16 и UTF-32 набора символов Юникод

**ASCII —** американский стандартный код для обмена информацией. ASCII представляет собой 8-битную кодировку для представления десятичных цифр, латинского и национального алфавитов, знаков препинания и управляющих символов. Нижнюю половину кодовой таблицы (0 — 127) занимают символы US-ASCII, а верхнюю (128 — 255) — разные другие нужные символы (CP866, CP1251).

**Windows-1251** — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий MW.

Кодировка **ASCII**  - кодировка для представления латинского алфавита, десятичных цифр, некоторых знаков препинания, арифметических операций и управляющих символов. **Windows-1251** – для символов русского алфавита.

**Набор символов времени выполнения:** символы, отображаемыми в среде выполнения. Любые дополнительные символы зависят от локализации

## 10. Кодировка: определение, назначение, примеры. Кодировка UNICODE: назначение, структура, UCS, UTF. Прямой (LE) и обратный (BE) порядок байт. BOM: определение, назначение, примеры.

**Кодировка** - таблица, задающая кодировку конечного множества символов алфавита (обычно элементов текста: букв, цифр, знаков препинания). Такая таблица сопоставляет каждому символу последовательность длиной в один или несколько символов другого алфавита (точек и тире в коде Морзе,  нулей и единиц (битов) в компьютере).

Примеры: ASCII, Windows-1251, Кодировки, UTF-8, UTF-16 и UTF-32 набора символов Юникод

**UNICODE:** это стандарт кодирования символов, позволяющий представить знаки почти всех письменных [языков](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA), состоит из 2х разделов:

**UCS** расположены в 17 плоскостях (0-16), 216 (65 536) символов в каждой плоскости, плоскость 0 – основная (основные символы), 1-14 – дополнительные, 15-16 – для частного использования.

**UTF-8** — представление Юникода, обеспечивающее совместимость со старыми системами, использовавшими 8-битные символы.

В **UTF-16** символы кодируются двухбайтовыми словами с использованием всех возможных диапазонов значений (от 0 до FFFF16).

**LE** (Little endian order, прямой порядок, от младшего к старшему), BE (Big endian order, обратный порядок, от старшего к младшему).

**BOM**: Для определения формата представления Юникода в начало текстового файла записывается сигнатура (обозначение) — символ U+FEFF — маркер последовательности байтов.

## 11. Структура языка программирования: алфавит языка программирования, идентификаторы, зарезервированные идентификаторы, литералы, ключевые слова, определения. Правила составления идентификаторов в C++ и других языков программирования. Примеры.

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

Совокупность символов, допускаемых в языке – алфавит языка.

***Базовый набор символов исходного кода:***

1. строчные и прописные буквы латинского и национального алфавитов
2. цифры
3. знаки операций
4. символы подчеркивания \_ и пробела
5. ограничители и разделители
6. специальные символы

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

***Примеры*** правил составления идентификаторов в с++:

* + 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:
    - 31 первых символов идентификаторов, не имеющих внешней связи;
    - 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.

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

**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.

**Литерал: элемент** программы, который непосредственно представляет значение. В C++ существует четыре типа литералов:

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

|  |  |  |  |
| --- | --- | --- | --- |
| Управляющие символьные литералы:  \0 \x00 null пустая литера | | | |
| \a | \x07 | bel | сигнал |
| \b | \x08 | bs | возврат на шаг |
| \f | \x0C | ff | перевод страницы |
| \n | \x0A | lf | перевод строки |
| \r | \x0D | cr | возврат каретки |
| \t | \x09 | ht | горизонтальная табуляция |
| \v | \x0B | vt | вертикальная табуляция |
| \\ | \x5C | \ | обратная косая черта |
| \' | \x27 | ' |  |
| \" | \x22 | " |  |
| \? | \x3F | ? |  |

**Ключевые слова**: последовательности символов алфавита языка, имеющие специальное назначение. Ключевые слова зарезервированы компилятором для обозначения типов переменных, класса хранения, элементов операторов и т.д.

**С++**

|  |  |  |  |
| --- | --- | --- | --- |
| [break](https://docs.microsoft.com/en-us/cpp/cpp/break-statement-cpp?view=vs-2017) | [case](https://docs.microsoft.com/en-us/cpp/cpp/switch-statement-cpp?view=vs-2017) | [catch](https://docs.microsoft.com/en-us/cpp/cpp/try-throw-and-catch-statements-cpp?view=vs-2017) | [char](https://docs.microsoft.com/en-us/cpp/cpp/fundamental-types-cpp?view=vs-2017) |
| [char16\_t](https://docs.microsoft.com/en-us/cpp/cpp/char-wchar-t-char16-t-char32-t?view=vs-2017) | [char32\_t](https://docs.microsoft.com/en-us/cpp/cpp/char-wchar-t-char16-t-char32-t?view=vs-2017) | [class](https://docs.microsoft.com/en-us/cpp/cpp/class-cpp?view=vs-2017) | [const](https://docs.microsoft.com/en-us/cpp/cpp/const-cpp?view=vs-2017) |
| [false](https://docs.microsoft.com/en-us/cpp/cpp/false-cpp?view=vs-2017) | [finally](https://docs.microsoft.com/en-us/cpp/dotnet/finally?view=vs-2017) | [float](https://docs.microsoft.com/en-us/cpp/cpp/fundamental-types-cpp?view=vs-2017) | [for](https://docs.microsoft.com/en-us/cpp/cpp/for-statement-cpp?view=vs-2017) |
| [inline](https://docs.microsoft.com/en-us/cpp/cpp/inline-functions-cpp?view=vs-2017) | return | **if** | struct |
| [cdecl](https://docs.microsoft.com/en-us/cpp/cpp/cdecl?view=vs-2017) | [int16](https://docs.microsoft.com/en-us/cpp/cpp/int8-int16-int32-int64?view=vs-2017) 4 | [int32](https://docs.microsoft.com/en-us/cpp/cpp/int8-int16-int32-int64?view=vs-2017) 4 | [int64](https://docs.microsoft.com/en-us/cpp/cpp/int8-int16-int32-int64?view=vs-2017) |

## 11.Структура языка программирования: алфавит языка программирования, идентификаторы, зарезервированные идентификаторы, литералы, ключевые слова, определения. Правила составления идентификаторов в C++ и других языков программирования. Примеры.

алфавит языка: набор разрешенных символов, кодировка символов исходного кода программ; символы времени трансляции, символы времени выполнения;

Алфавит языка программирования:

набор символов, разрешенных к использованию языком программирования.

Основывается на одной из кодировок.

Совокупность символов, допускаемых в языке – алфавит языка.

Базовый набор символов исходного кода:

1) строчные и прописные буквы латинского и национального алфавитов

2) цифры

3) знаки операций

4) символы подчеркивания \_ и пробела

5) ограничители и разделители

6) специальные символы

3

С помощью символов алфавита записываются служебные слова, которые

составляют словарь языка.

Алфавит языка программирования служит для построения слов в языке

программирования, которые называют лексемами. Примеры лексем:

Лексемы

идентификаторы;

ключевые (зарезервированные) слова;

знаки операций;

константы;

разделители (скобки, знаки операций, точка,

запятая, пробельные символы и т.д.).

Границы лексем определяются с помощью других лексем, таких, как

разделители или знаки операций.

. 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;

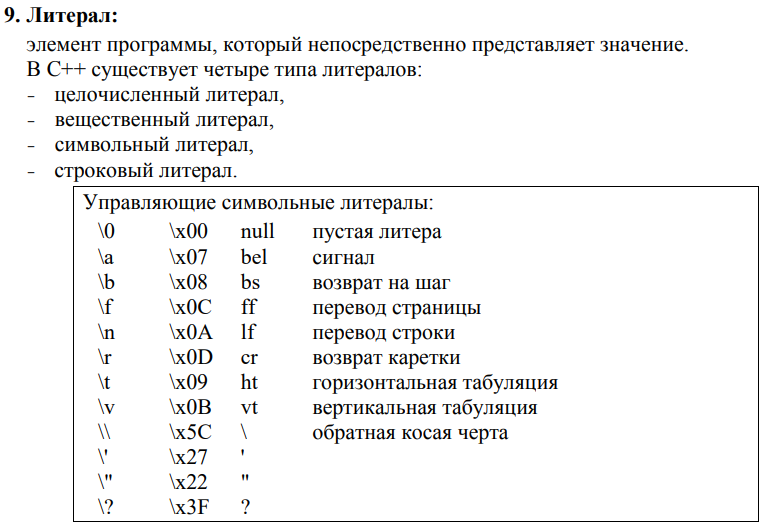
- идентификаторы чувствительны к регистру.

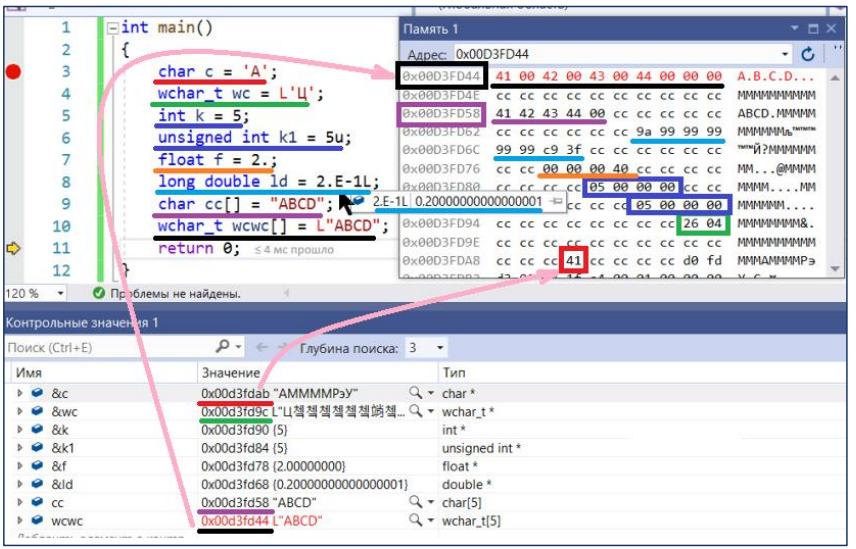
Длина идентификатора по стандарту не ограничена.

Идентификатор создается при объявлении переменной, функции, типа и т. п.

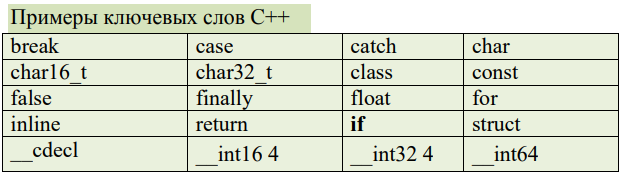
Зарезервированные идентификаторы: идентификаторы, которые предварительно определены в системе программирования.

Зарезервированные идентификаторы С++: − все имена с двумя подчеркиваниями считаются зарезервированным; − Кроме того: isxxxx, memxxxx, strxxxx, toxxxx, wcsxxxx, Eцифраxxxx, LC\_Xxxx, SIGXxxx, SIG\_Xxxxx.



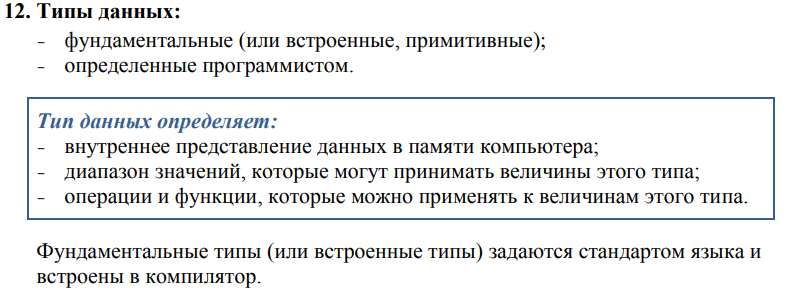


Ключевые слова: последовательности символов алфавита языка, имеющие специальное назначение. Ключевые слова зарезервированы компилятором для обозначения типов переменных, класса хранения, элементов операторов и т.д.





## 12.Структура языка программирования: фундаментальные типы данных. Что определяет тип данных? Определения, примеры (C++ и другие языки)



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.

Модификаторы основных типов, уточняющие внутреннее представление и

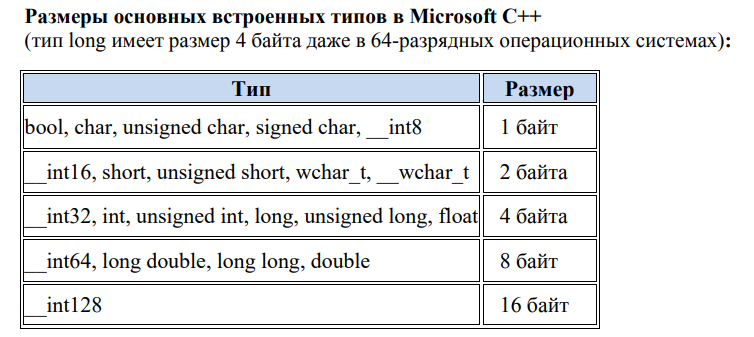
диапазон значений стандартных типов:

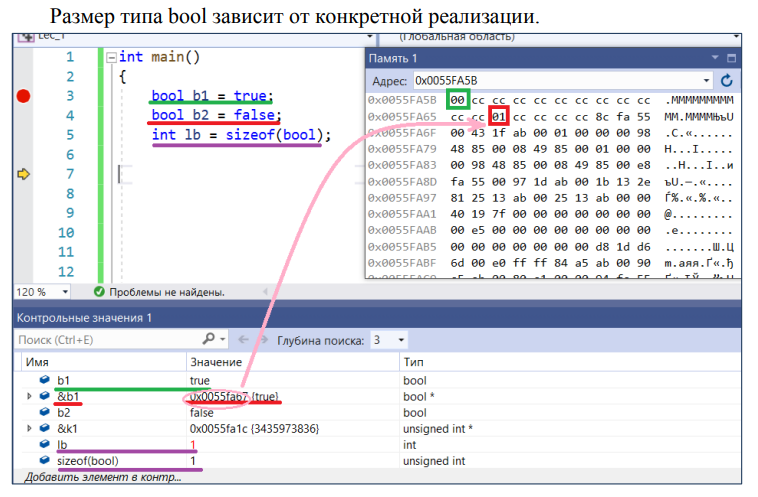
- short (короткий);

- long (длинный);

- signed (знаковый);

-unsigned(беззнаковый).





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.

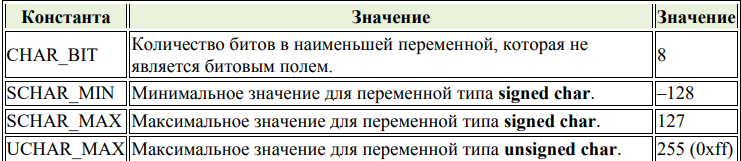
По умолчанию все целочисленные типы считаются знаковыми, то есть

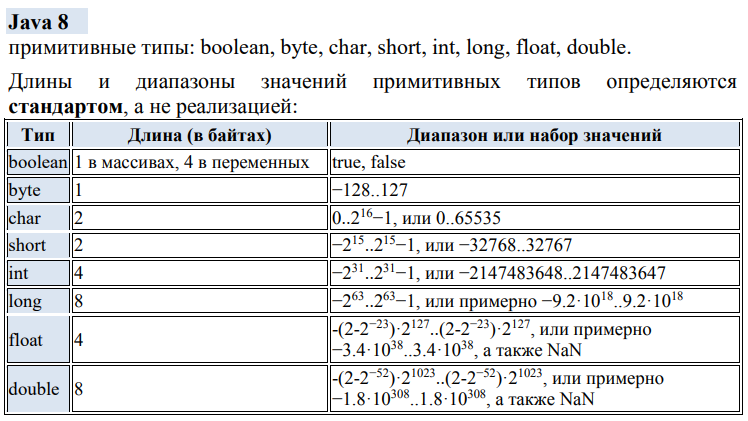
спецификатор signed можно опускать. Диапазон значений зависит от

реализации.

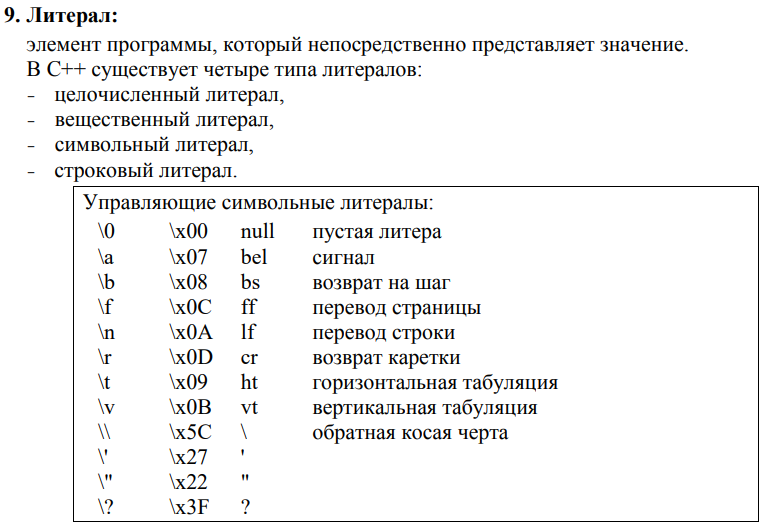
|  |  |
| --- | --- |
|  |  |
|  |  |

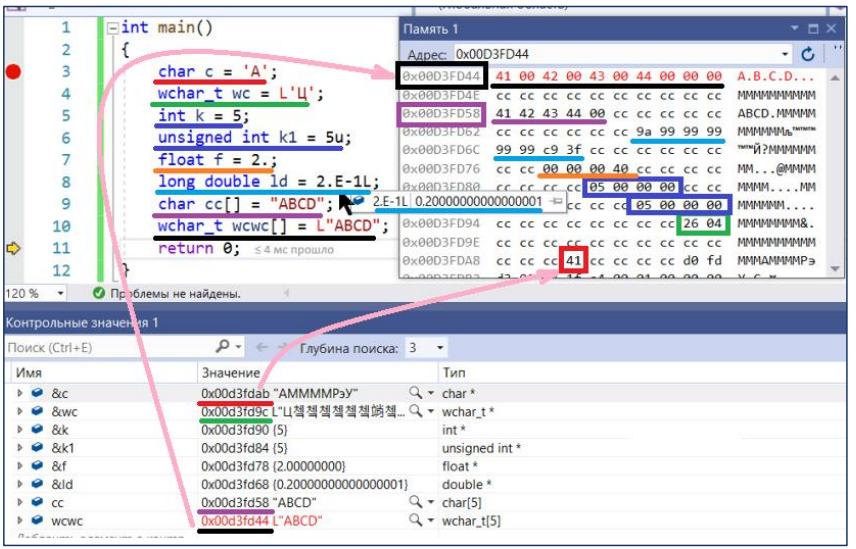
Два стандартных включаемых заголовочных файла, и , определяют числовые ограничения или минимальное и максимальное значения, которые может хранить переменная данного типа. Ограничения для некоторых целочисленных типов, заданные в стандартном файле заголовка , представлены в таблице:





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





Массивы данных фундаментальных типов: коллекция однородных

данных, размещенных последовательно в памяти и допускающие доступ по

индексу (вычисляется: смещение = индекс\*sizeof(базовый\_тип)).

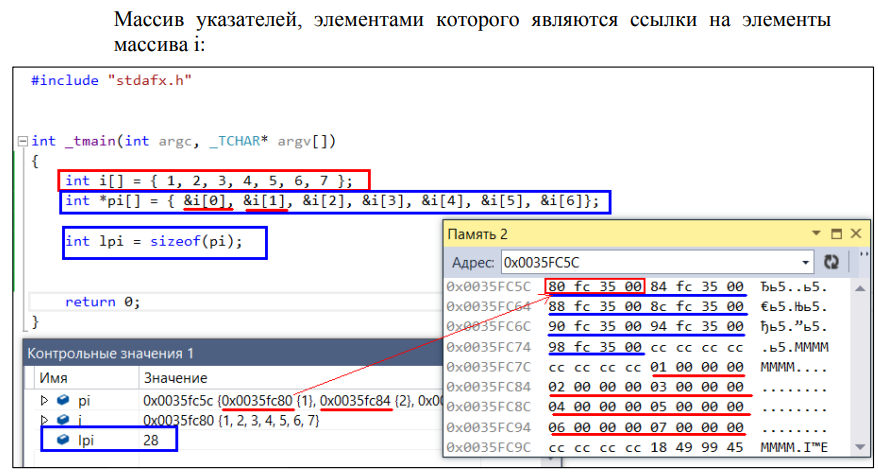
Массив (array) − это совокупность переменных, имеющих одинаковый тип и объединенных под одним именем. Доступ к отдельному элементу массива осуществляется с помощью индекса.

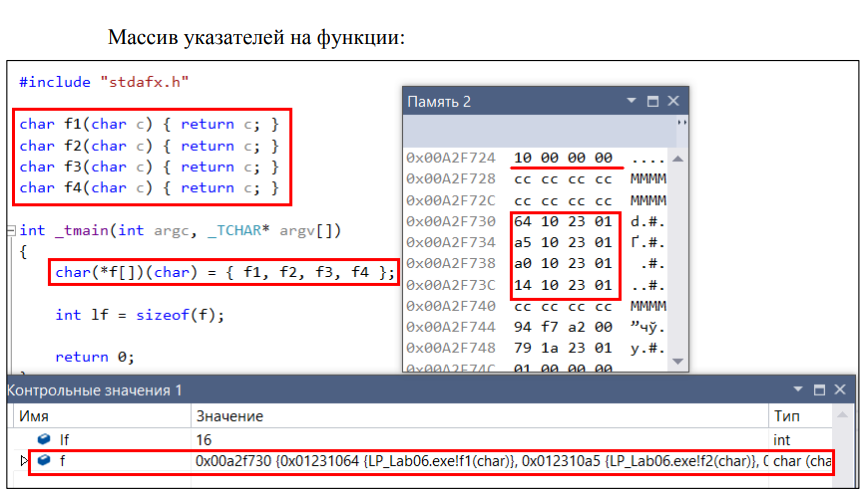
Объем памяти, необходимый для хранения массива, зависит от его типа и

количества элементов в нем.

Размер одномерного массива в байтах вычисляется по формуле:

количество\_байтов = sizeof(базовый\_тип) \* количество\_элементов





Многомерные массивы С/С++: многомерность моделируется в виде «массива массивов» Доступ к элементу, стоящему на пересечении первой строки и третьего столбца, можно получить двумя способами: − либо индексируя массив – mm[0][2]; − либо используя указатель – \*(( \*)mm+2). Правила адресной арифметики требуют приведения типа указателя на массив к его базовому типу. Двухмерный массив можно представить с помощью указателей на одномерные массивы.

14.Структура языка программирования: фундаментальные типы данных. Типы данных для хранения вещественных значений. Стандарт IEEE 754. Что определяет стандарт IEEE 754**.**

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.

Модификаторы основных типов, уточняющие внутреннее представление и

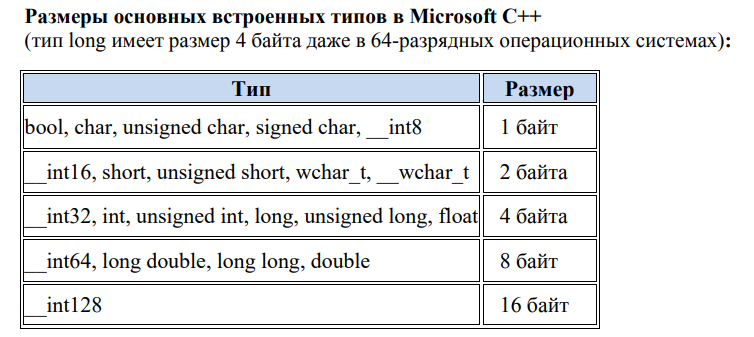
диапазон значений стандартных типов:

- short (короткий);

- long (длинный);

- signed (знаковый);

- unsigned (беззнаковый)



Типы float, double

Стандарт языка C++ определяет три типа данных для хранения

вещественных значений: float, double и long double.

Стандарт IEEE 754 описывает формат представления чисел с плавающей

точкой. Используется в программных (компиляторы разных языков

программирования) и аппаратных (CPU и FPU) реализациях арифметических

действий (математических операций).

Стандарт описывает:

-формат чисел с плавающей точкой: мантисса, показатель (экспонента),

знак числа;

-представление положительного и отрицательного нуля, положительной

и отрицательной бесконечностей, а также нечисла (англ. Not-a-Number, NaN);

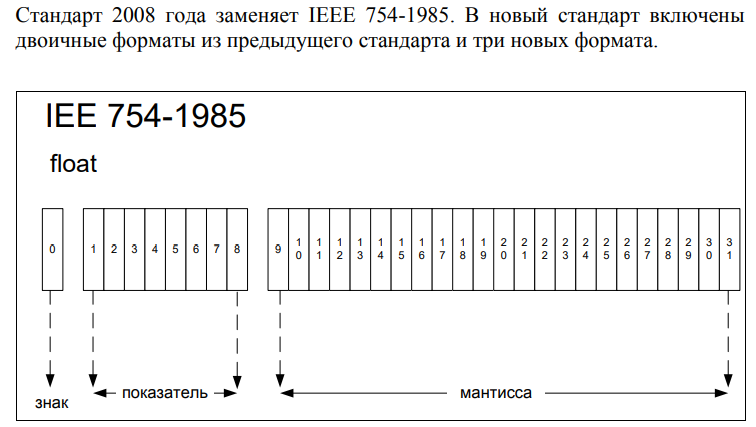
-методы, используемые для преобразования числа при выполнении

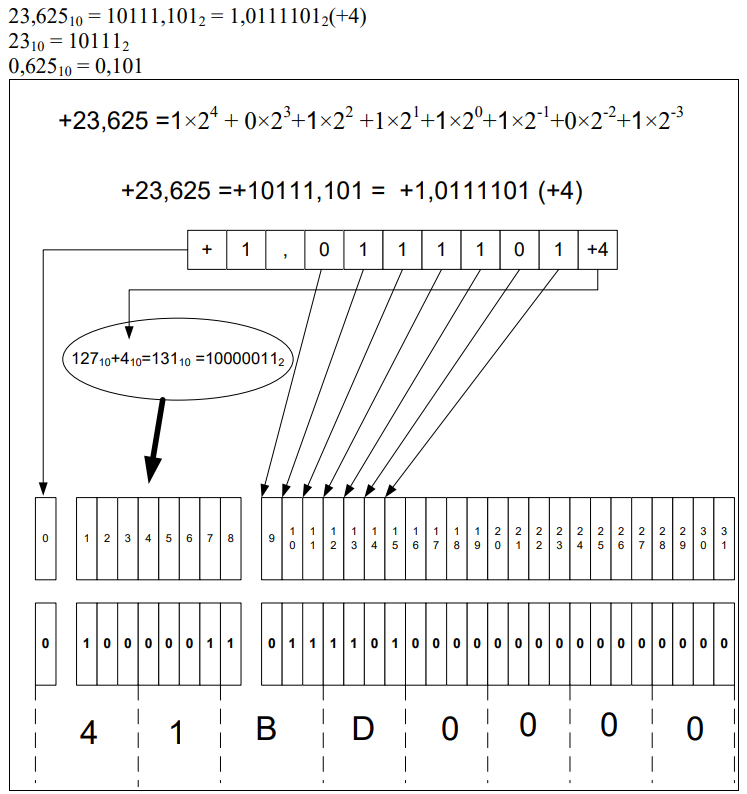
математических операций;

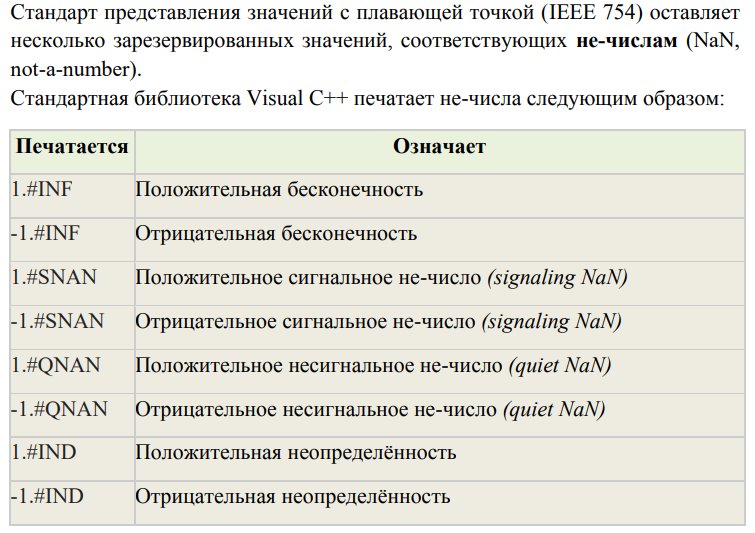
-исключительные ситуации: деление на ноль, переполнение, потеря

значимости, работа с денормализованными числами и другие;

-операции: арифметические и другие.







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

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.

Модификаторы основных типов, уточняющие внутреннее представление и

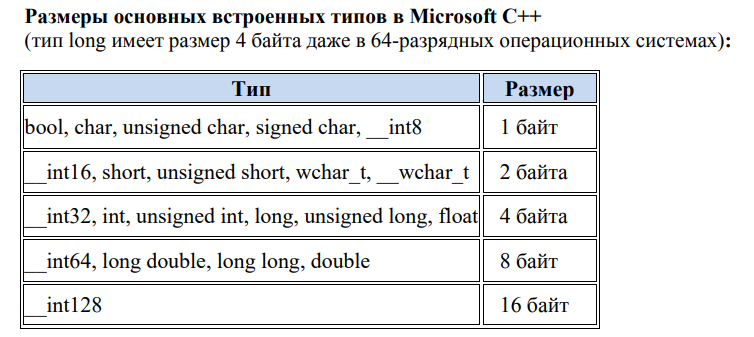
диапазон значений стандартных типов:

- short (короткий);

- long (длинный);

- signed (знаковый);

- unsigned (беззнаковый



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.

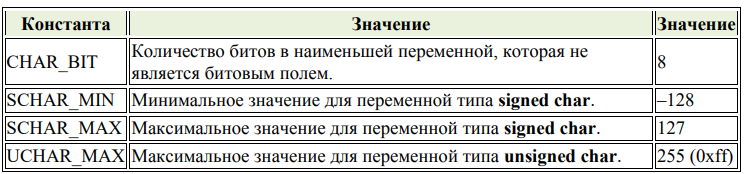
По умолчанию все целочисленные типы считаются знаковыми, то есть

спецификатор signed можно опускать. Диапазон значений зависит от

реализации.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Два стандартных включаемых заголовочных файла, и , определяют числовые ограничения или минимальное и максимальное значения, которые может хранить переменная данного типа. Ограничения для некоторых целочисленных типов, заданные в стандартном файле заголовка , представлены в таблице:



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

Тип данных – множество значений и операций над этими значениями. (IEEE Std 1320.2-1998).

Тип данных определяет: • внутреннее представление данных в памяти компьютера; • множество значений, которые могут принимать величины этого типа; • операции и функции, которые можно применять к величинам этого тина.

Система типов – совокупность правил в языках программирования, назначающих свойства, называемыми типами, различным конструкциям, составляющим программу (переменным, выражениям, функциям, модулям) для возможности выполнения проверки типов во время компиляции или во время выполнения,

Основные функции системы типов данных: − обеспечение безопасности: проверяется каждая операция на получение аргументов именно тех типов, для которых она имеет предназначена; − оптимизация: на основе типа выбирается способ эффективного хранения значения и алгоритмов его обработки; − документация: подчеркивается намерения программиста; − абстракция: использование типов данных высокого уровня позволяет программисту думать о значениях как о высокоуровневых сущностях, а не как о наборе битов.

Два основных вида типизации в языках программирования:

типизированные (C, C++, Java, Python, Scala, Rust, PHP, C#, F#, другие)

нетипизированные (бестиповые) (языки ассемблера, Forth и Brainfuck (этозерический язык), другие)

В бестиповых языках программирования все сущности – это последовательности битов, различной длины.

Примеры:

• в ассемблере единственный тип — последовательность битов;

• в лямбда-исчислении единственный тип — функция.

Выравнивание данных в оперативной памяти компьютеров: требование для объектов определенного типа располагаться на границах ячеек памяти с адресами, кратными своему же размеру.

По умолчанию компилятор выравнивает элементы структуры, класса по значению размера:

bool и char в однобайтовых границах;

short на 2-байтовых границах;

int, long и float в пределах 4-байтового диапазона;

long long, double и long double в пределах 8-байтовых границ

Автоматическое (неявное) преобразование типов:

Для базовых типов

bool, [unsigned/signed] char, short, int, long, float, double,

long double

преобразование типов выполняется без потери точности.

Пример безопасного преобразования:

символ ‘a’ → целое 0х41 →символ ‘a’

Если выбранное преобразование является расширяющим, компилятор выполняет

его, не информируя о выполнении такого преобразования. Расширяющие

преобразования всегда являются надежными.

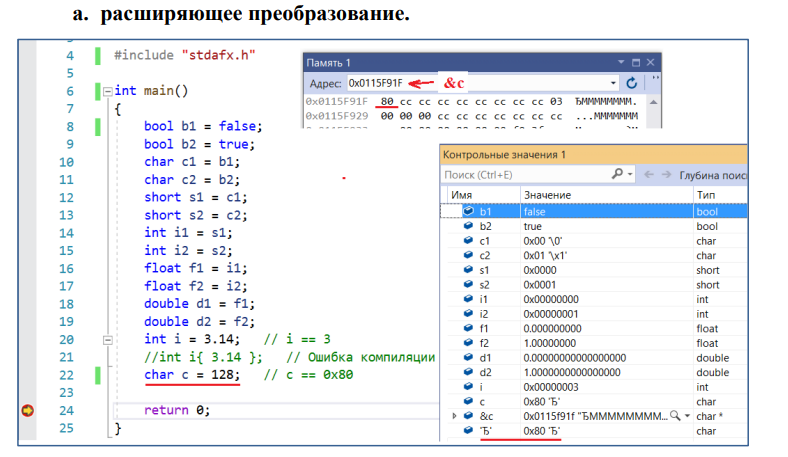
Если преобразование является сужающим, компилятор выдает предупреждение о

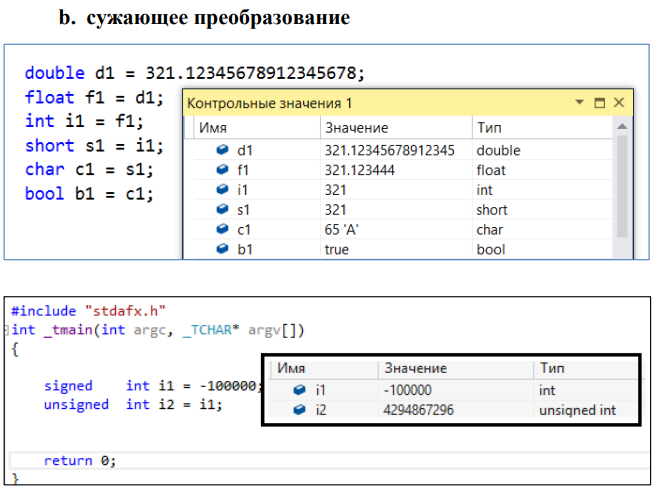
возможной потере данных.

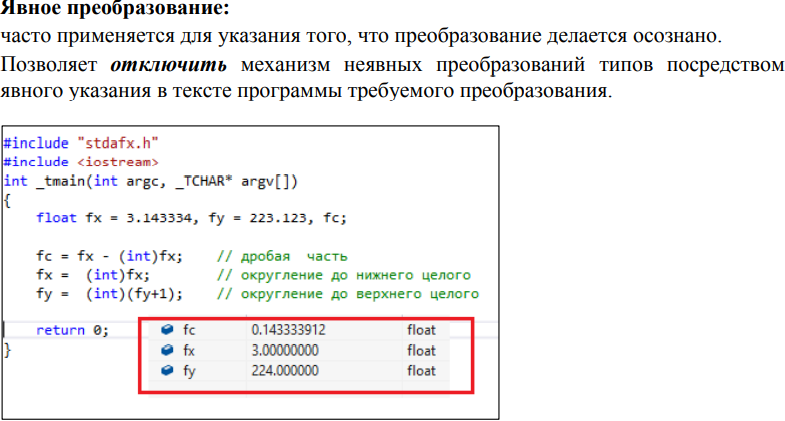
Происходит ли фактическая потеря данных, зависит от фактических значений.

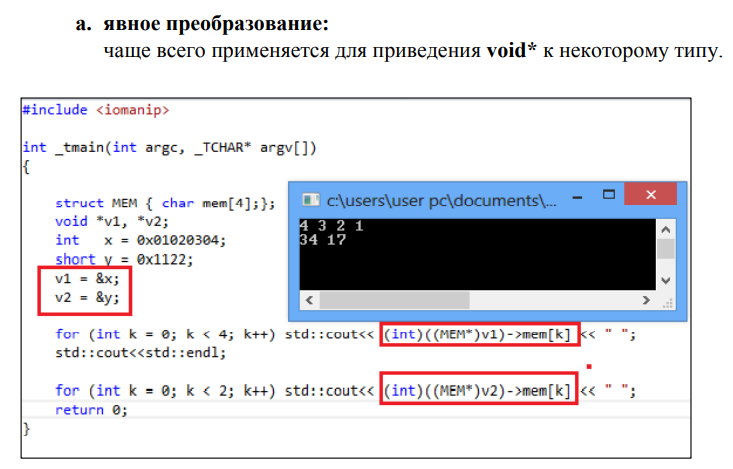
Рекомендуется рассматривать это предупреждение кок ошибку.

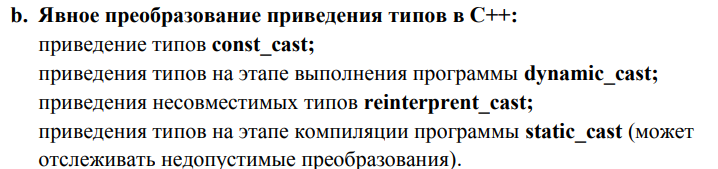
Если компилятору не удается найти допустимое преобразование, то выдается ошибка и объектный код не создается.











## 17. Структура языка программирования: система типов языка программирования, Вывод типов. Преобразование типов: расширяющее преобразование, сужающее преобразование. Назначение оператора sizeof языка программирования С++. Примеры.

Тип данных – множество значений и операций над этими значениями. (IEEE Std 1320.2-1998).

Тип данных определяет: • внутреннее представление данных в памяти компьютера; • множество значений, которые могут принимать величины этого типа; • операции и функции, которые можно применять к величинам этого тина.

Система типов – совокупность правил в языках программирования, назначающих свойства, называемыми типами, различным конструкциям, составляющим программу (переменным, выражениям, функциям, модулям) для возможности выполнения проверки типов во время компиляции или во время выполнения,

Основные функции системы типов данных: − обеспечение безопасности: проверяется каждая операция на получение аргументов именно тех типов, для которых она имеет предназначена; − оптимизация: на основе типа выбирается способ эффективного хранения значения и алгоритмов его обработки; − документация: подчеркивается намерения программиста; − абстракция: использование типов данных высокого уровня позволяет программисту думать о значениях как о высокоуровневых сущностях, а не как о наборе битов.

Два основных вида типизации в языках программирования:

типизированные (C, C++, Java, Python, Scala, Rust, PHP, C#, F#, другие)

нетипизированные (бестиповые) (языки ассемблера, Forth и Brainfuck (этозерический язык), другие)

В бестиповых языках программирования все сущности – это последовательности битов, различной длины.

Примеры:

• в ассемблере единственный тип — последовательность битов;

• в лямбда-исчислении единственный тип — функция.

Выравнивание данных в оперативной памяти компьютеров: требование для объектов определенного типа располагаться на границах ячеек памяти с адресами, кратными своему же размеру.

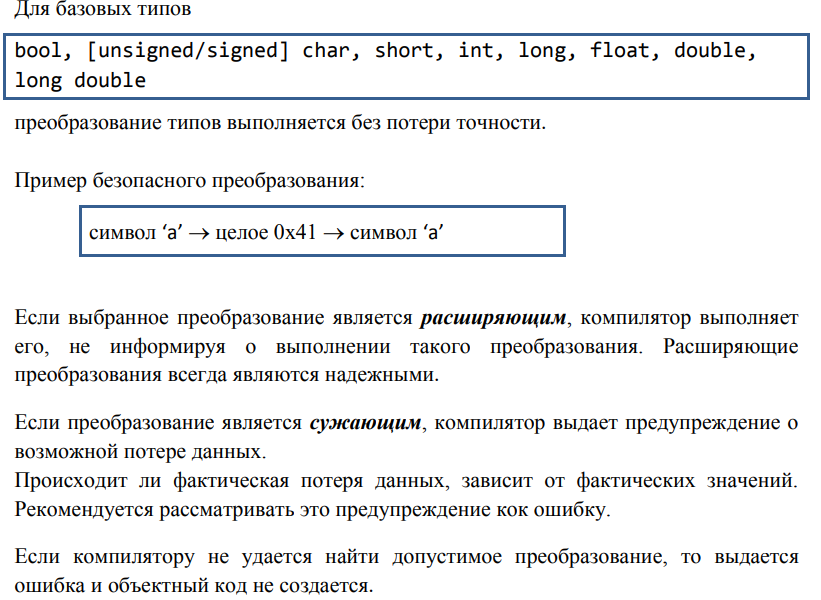
По умолчанию компилятор выравнивает элементы структуры, класса по значению размера:

bool и char в однобайтовых границах;

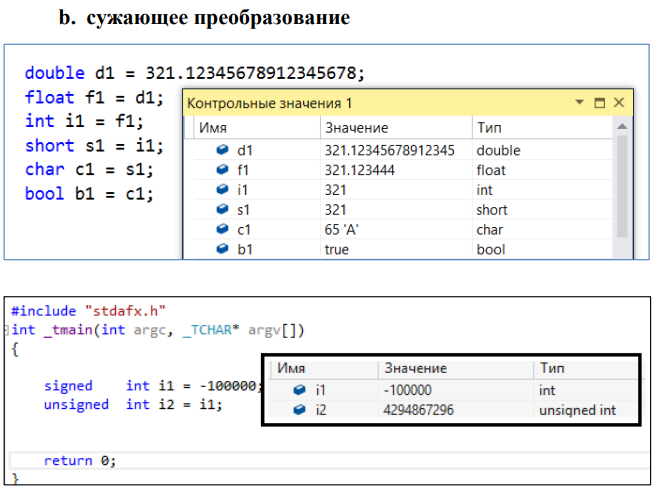
short на 2-байтовых границах;

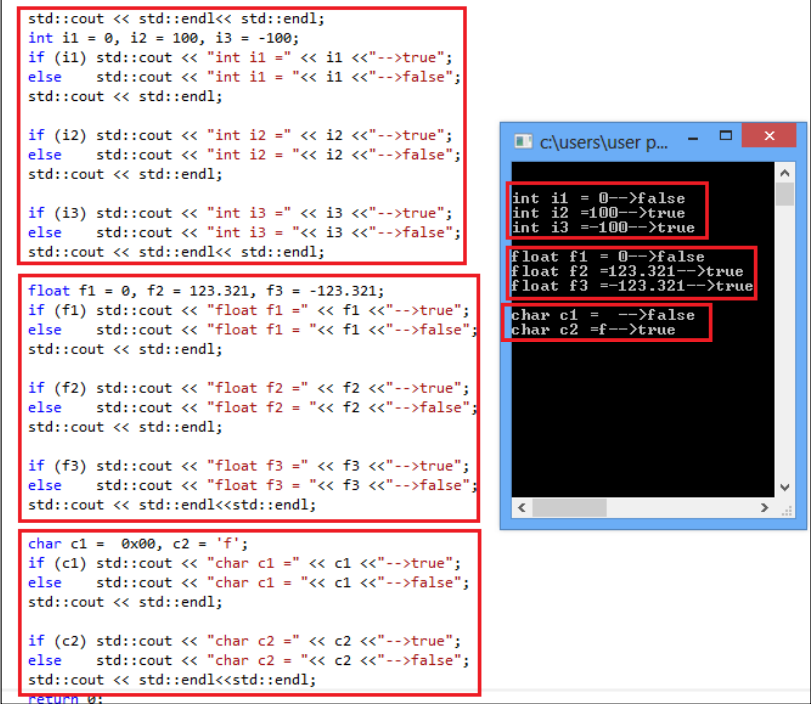
int, long и float в пределах 4-байтового диапазона;

long long, double и long double в пределах 8-байтовых границ.









В С/С++ размер переменной любого типа данных зависит от компилятора и/или архитектуры компьютера. Фактический размер переменных может отличаться на разных компьютерах. Для его определения используют оператор sizeof.

Стандарт задает отношение размера между целыми типами:

1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)

и для переменных с плавающей запятой:

sizeof(float) <= slzeof(double) <= sizeof(long double)

Также поддерживаются целочисленные типы с указанием их размера:

\_\_int8, \_\_int16, \_\_int32, \_\_int64 и \_\_int8, \_\_int16, \_\_int32,

## 18. 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 одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются

## 19. Структура языка программирования: пользовательские типы данных. Определение структуры, примеры (C++). 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 одной области видимости нескольких различных вариантов применения оператора, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются

## 20. Структура языка программирования: понятие конфликта имен, область видимости переменных, пространства имен, псевдонимы пространства имен.

**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.

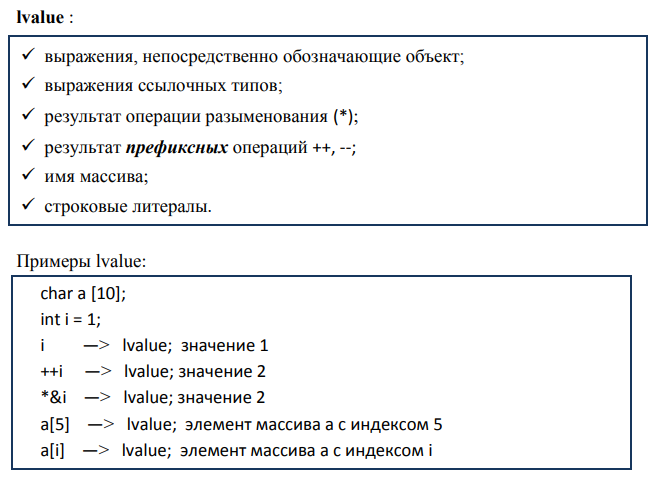
Пространства имен помогают предотвратить конфликты имен, но не устранить их полностью. Такой конфликт может произойти, когда одно и то же имя объявляется в двух разных пространствах имен и затем предпринимается попытка сделать видимым оба пространства. В таком случае для указания предполагаемого пространства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

## 21. Структура языка программирования: определение выражения, его состав, порядок вычисления выражения. Символ окончания последовательности. Примеры (С++).

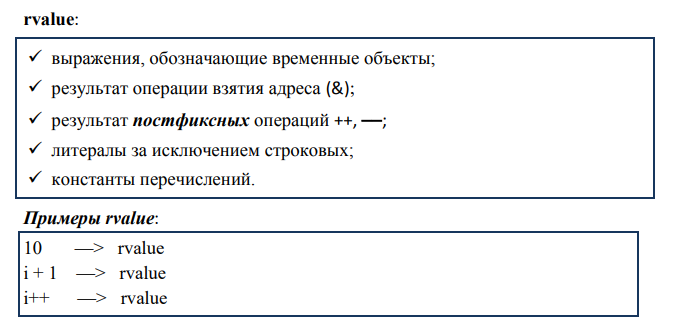
Выражение – объединение литералов, имен (переменных, функций и пр.), операторов и специальных символов, служащих для вычисления выражения или достижения побочных эффектов (например: при применении в выражении функций). Выражения состоят из операндов, знаков операций и скобок. Порядок вычисления выражения с операторами одинакового приоритета не определен. Символ окончания последовательности (в языке программирования C++ – это точка с запятой) определяет точку последовательности (sequence point), в которой завершились все вычисления и побочные эффекты. Точка последовательности – момент времени, когда побочные эффекты вычисленных выражений уже случились, а побочные эффекты следующих в последовательности выражений еще не начались.

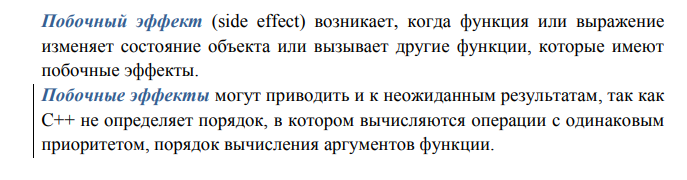
## 21 Структура языка программирования: выражения (lvalue, rvalue, побочные эффекты, точка последовательности, унарные, бинарные и тернарные выражения), константные выражения, укороченное вычисление. Примеры (С++).

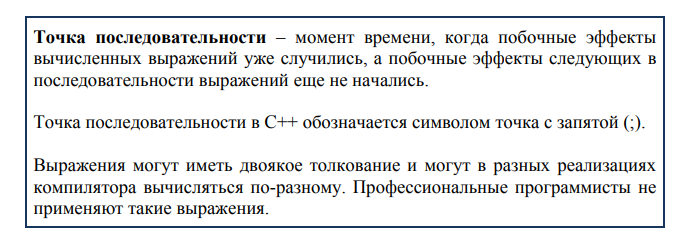
lvalue (именующее выражение) – это ссылка на значение; могут использоваться в левой и правой части оператора присваивания. Имя переменной, ссылка на элемент массива по индексу, вызов функции возвращающей указатель, всегда связаны с областью памяти, адрес которой известен

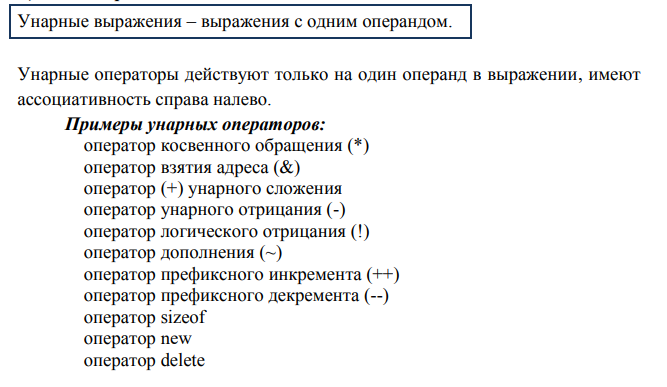


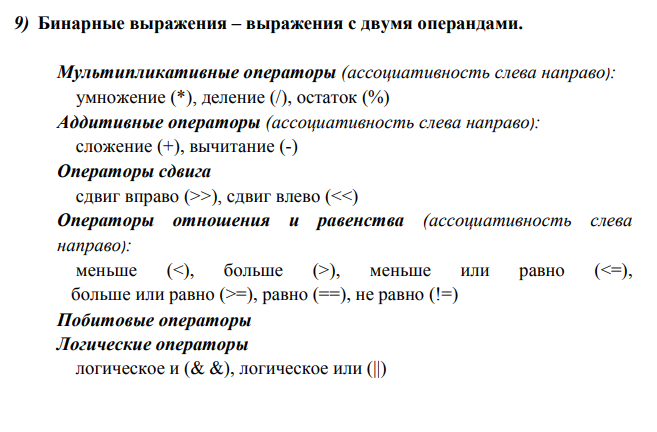
rvalue (значащее выражение) – может использоваться только в правой части оператора присваивания (не связано с адресом, связано только со значением; это могут быть литералы, вызов функции, возвращающей значение).

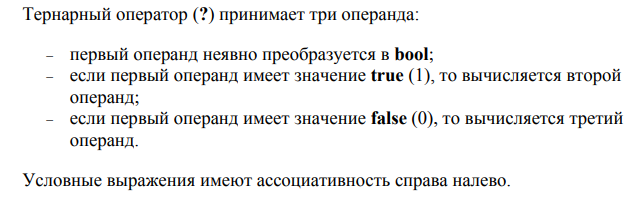












# Структура языка программирования: инструкции языка программирования, объявление, определение, инициализация, простые и составные инструкции, инструкции выбора, инструкции циклов, инструкции переходов, примеры (С++).

**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.

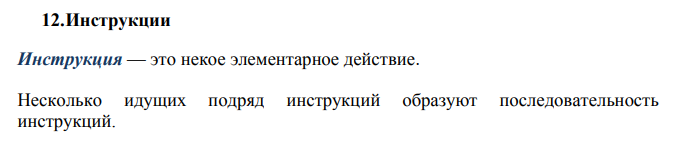
**Инструкции выбора** *if*и *switch*, позволяют выполнять те или иные участки кода в зависимости от выполнения условий.

**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.

**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.

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



**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.

**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.

## 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.

## Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Передача параметров по значению и по ссылке, передача значений параметров по умолчанию 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(){

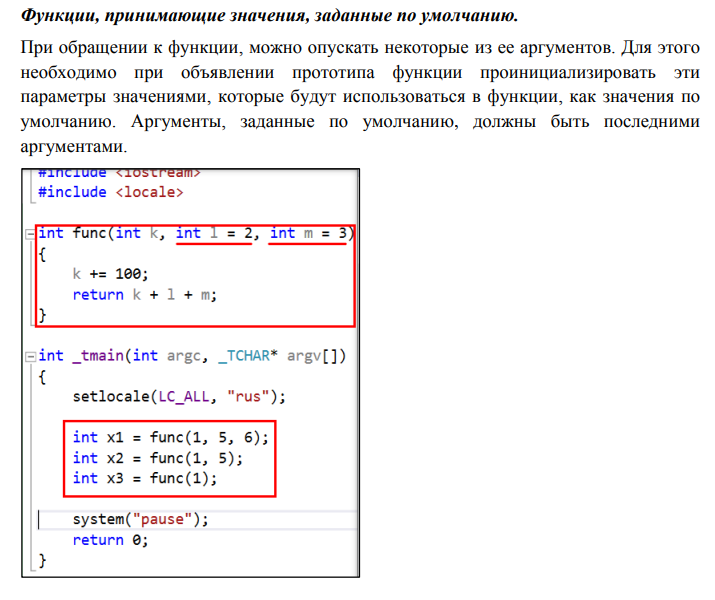
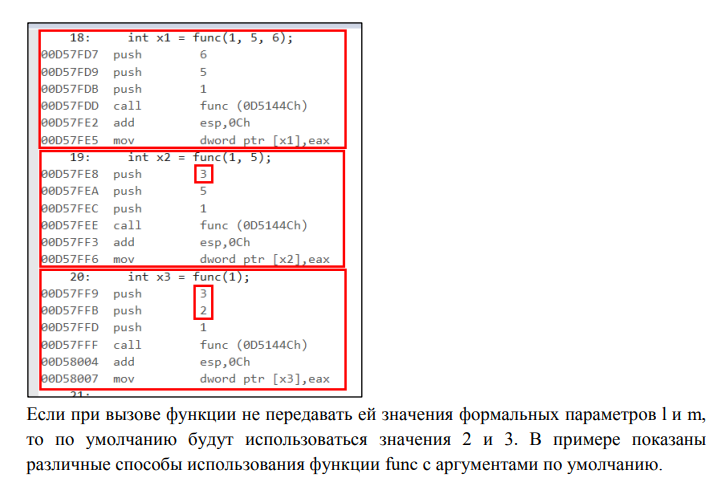
int z=1, y=3, k;

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

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

}

Z = 2, y = 6

## Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций: лямбда-функции в С++. Примеры.

**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++:**

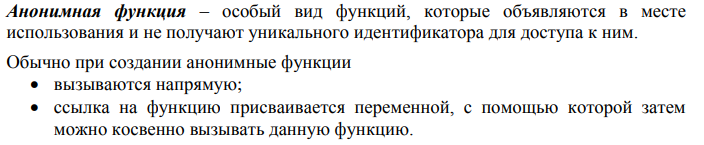
 void ValidMessage()

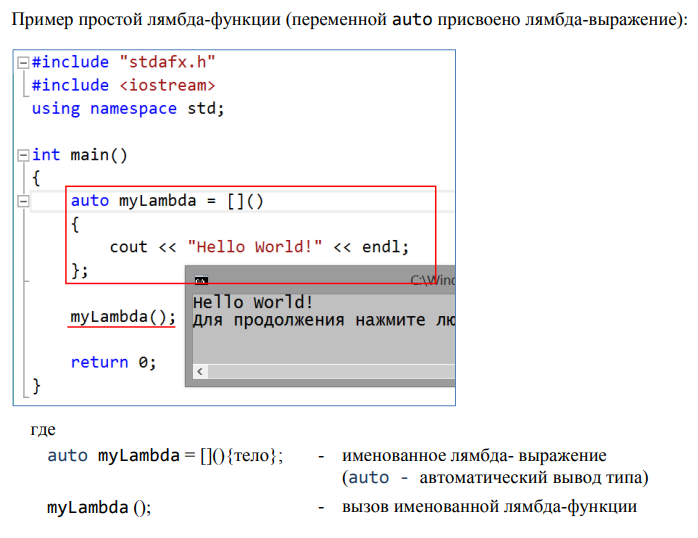
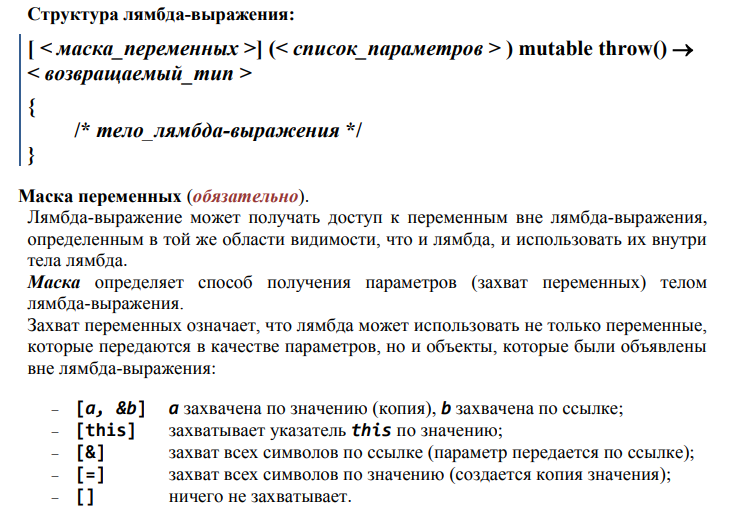
 {

         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



## Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций в C++. Перегружаемые функции в C++. Inline-функции в C++. Шаблоны функций С++.

**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()

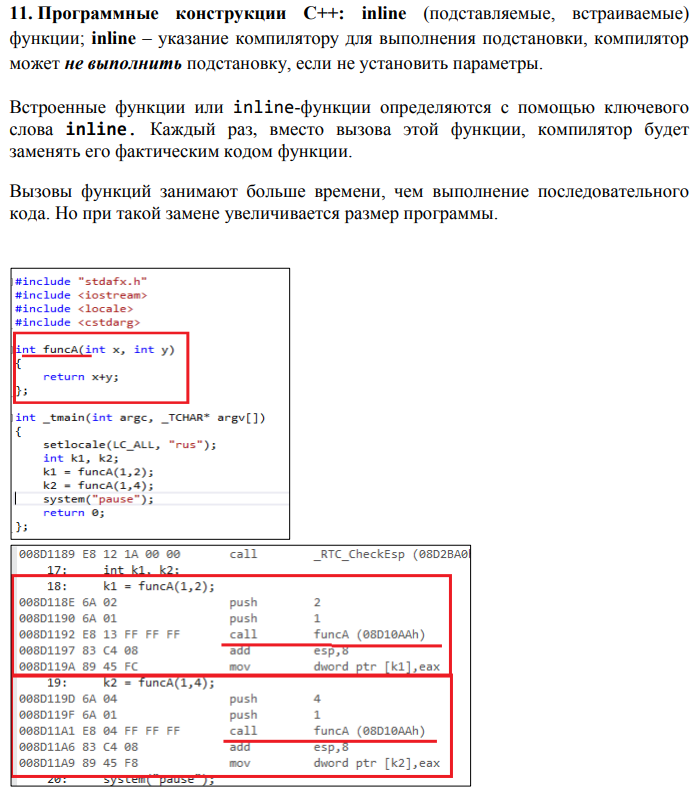
 {

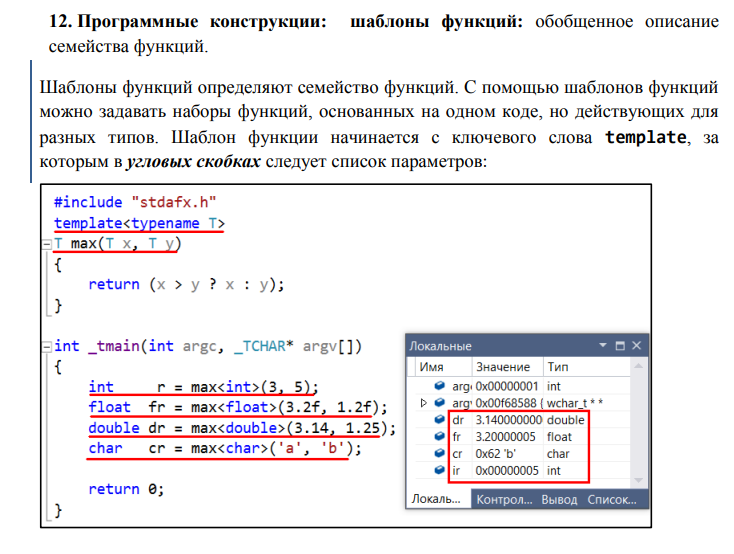
         cout << "Пароль введен верно" << endl;

 }

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







1. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций в C++: соглашения о вызовах (\_cdecl).

**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



**\_cdecl:** Параметры функций помещаются в стек, порядок параметров "справа налево". Параметры, размер которых меньше 4-х байт, расширяются до 4-х байт. Адрес возврата кладется в стек поверх параметров.

1. Структура языка программирования: программные конструкции (блоки, функции, процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций в C++: соглашения о вызовах (\_stdcall).

**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



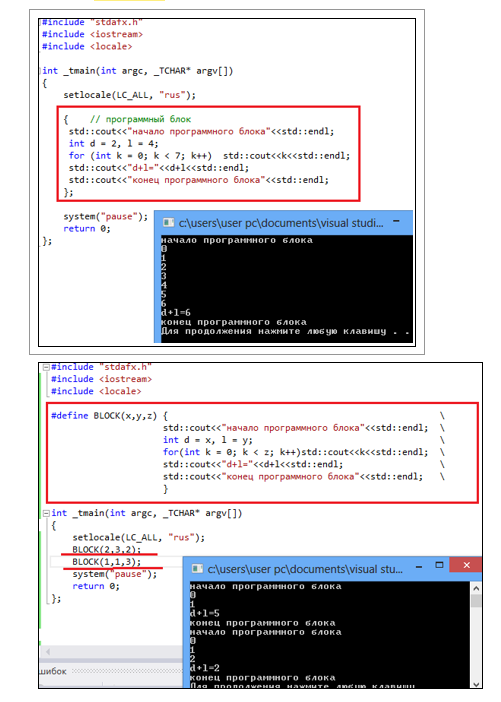
**\_stdcall:** Параметры помещаются в стек, порядок параметров "справа налево". Адрес возврата кладется в стек поверх параметров.

Стек освобождает вызываемый код. В последней инструкции вызываемого кода RET указывается суммарный размер в байтах всех параметров процедуры. Команда RET после извлечения адреса возврата прибавляет к регистру ESP указанное значение. Целостность стека восстанавливается вызываемым кодом.

Возвращаемый параметр передается через регистр EAX.

**31. Структура языка программирования: программные конструкции (блоки, функции,  
процедуры и пр.). Передача параметров в функцию. Реализация программных конструкций  
в C++: соглашения о вызовах (\_fastcall).**

Программные блоки представлены:

****

Функции:

***Функция*** – фрагмент программного кода, к которому можно обратиться из другого места программы. Функция должна быть ***объявлена*** и ***определена***.

***Функция*** – подпрограмма, выполняющая какие-либо операции и возвращающая значение.

***Процедура*** – подпрограмма, которая выполняет операции, и не возвращает значения.

**Передача параметров в функцию**

Передача параметров в функцию происходит через стек.

***Код, вызывающий функцию***, знает, сколько параметров ей передать и каковы значения этих параметров. В нашем примере, если код вызывает функцию sum, которая принимает два параметра типа int, то вызывающий код:

* + кладет два параметра в стек с помощью двух инструкций push. В результате этого указатель стека (ESP) **уменьшается на 2\*4** байта (вершина стека сдвигается на **8** байт);
  + выполняет инструкцию call, которая передает управление функции sum. При этом значение ESP **уменьшается еще на 4 байта**, потому что в стек помещается ***адрес точки возврата***.

Соглашение вызовах \_fastcall (не стандартизированный, для внутренних вызовов), параметры передаются через регистры (первые 2 – через регистры, остальные справа налево в стек), стек освобождает вызываемый код, возврат через регистр EAX. В Borland (Delphi) (параметры слева на право.

## 32. Структура языка программирования: стандартная библиотека. Реализация стандартной библиотеки STL в C++. Понятие контейнера, итератора и алгоритма. Стандартные функции STL для работы со строками: копирование, сравнение, вычисление длины, поиск символа и подстроки.

В составе языка программирования, как правило, есть обязательный (стандартный) набор функций. Такие функции называют встроенными.

Встраиваться функции могут тремя способами:

* + прямо в код транслятора;
  + находиться в отдельной библиотеке;
  + сочетание первого и второго случаев.

Основные требования к набору средств стандартной библиотеки (Бьёрн Страуструп):

* + ***эффективность***;
  + ***независимость от алгоритмов*** — должна предоставлять возможность задавать алгоритмы в качестве параметров;
  + ***удобство и безопасность***;
  + ***завершённость*;**
  + ***органично сочетаться с языком*;**
  + ***типобезопасность*;**
  + ***поддержка общепринятых стилей программирования***;
  + ***расширяемость*** — способность единообразно работать со встроенными типами данных и с типами, определяемыми пользователем

Подходы к разработке стандартных библиотек языков программирования:

* + должна содержать в себе только те процедуры и функции, которые используются практически всеми и обладают максимальной универсальностью;
  + должна содержать в себе максимально возможное количество типичных алгоритмов, обеспечивать простую работу с большинством объектов (в идеале, со всеми), с которыми может взаимодействовать программа. Пример реализации этого подхода является язык Python с девизом «Batteries included» (батарейки в комплекте).

Стандартная библиотека C++ STL:

библиотека стандартных шаблонов (STL) – набор согласованных обобщённых алгоритмов, контейнеров, средств доступа к их содержимому и различных вспомогательных функций в C++.

STL (Standard Template Library) – стандартная библиотека шаблонов. Библиотека содержит универсальные шаблонные классы и функции, реализующие большое количество распространенных универсальных алгоритмов и структур данных. Т.к. библиотека STL состоит из шаблонных классов, ее алгоритмы и структуры можно применять практически к любым типам данных.

Частью *стандартной библиотеки С++* является библиотека *STL*. Библиотека STL содержит пять основных видов компонентов:

* *контейнер* (*container*): управляет набором объектов в памяти.
* *итератор* (*iterator*): обеспечивает для алгоритма средство доступа к содержимому контейнера.
* *алгоритм* (*algorithm*): определяет вычислительную процедуру.
* *функциональный объект* (*function object*): инкапсулирует функцию в объекте для использования другими компонентами.
* *адаптер* (*adaptor*): адаптирует компонент для обеспечения различного интерфейса.

Функции для работы со строками:

strlen() (от слова length – длина)

strcpy()

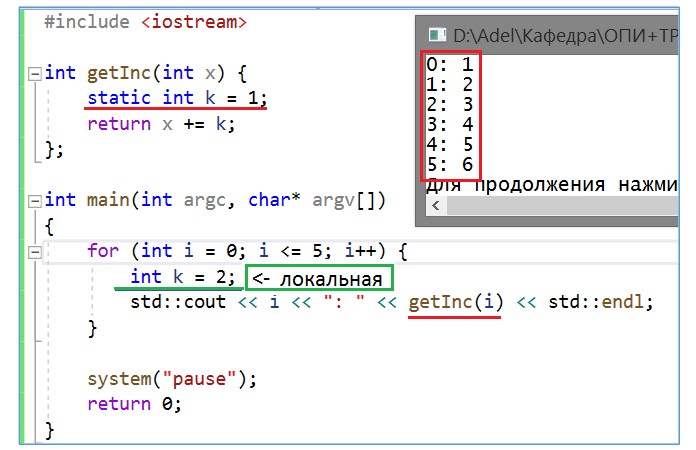
strcmp()

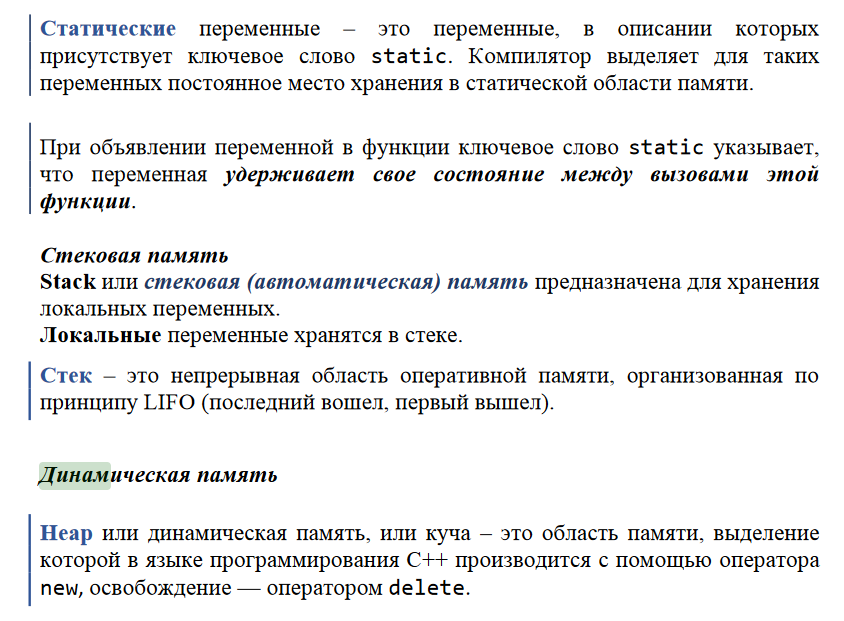
.find()

## 33. **Классы памяти: код, стек, статические данные, динамическая область памяти. Структура** **динамической памяти (Heap) C++. Фрагментация динамической памяти C++.**

Область кода – память, в которой размещается код программы.

Локальная статическая память.

  
  
Переменная с ключевым словом static — это статическая переменная.  
Время ее жизни — постоянное.  
Область видимости статической переменной ограничена одним файлом,  
внутри которого она определена, ее можно использовать только после ее  
объявления.  
Ключевое слово static в языке С/С++ используется для двух различных  
целей:  
− как указание типа памяти: переменная располагается в статической  
памяти;  
− как способ ограничить область видимости переменной рамками одного  
файла (в случае описания переменной вне функции).

****

## 34. Механизм обработки исключений: определение, назначение, применение. Реализация обработки исключений в C++. Пример.

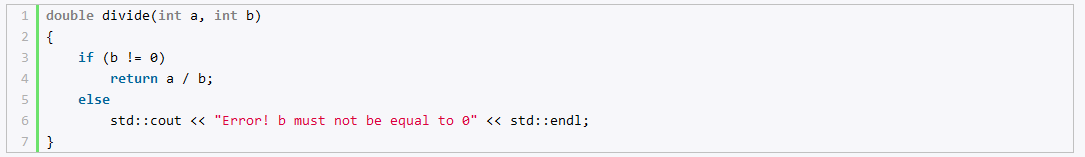
В процессе работы программы могут возникать различные ошибки. Например, при передаче файла по сети оборвется сетевое подключение или будут введены некорректные и недопустимые данные, которые вызовут падение программы. Такие ошибки еще называются исключениями. Если исключение не обработано, то при его возникновении программа прекращает свою работу.

Например, в следующей программе происходит деление чисел:



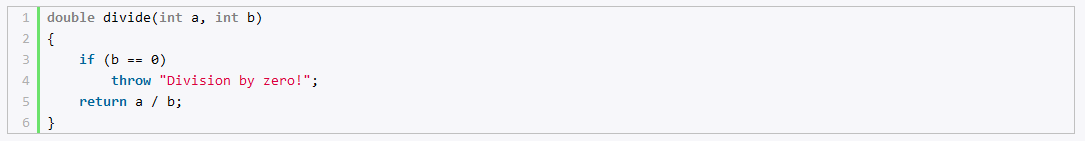
Эта программа успешно скомпилируется, но при ее выполнении возникнет ошибка, поскольку в коде производится деление на ноль, после чего программа аварийно завершится.

С одной стороны, мы можем в функции divide определить проверку и выполнять деление, если параметр b не равен 0. Однако нам в любом случае надо возвращать из функции divide некоторый результат - некоторое число. То есть мы не можем просто написать:



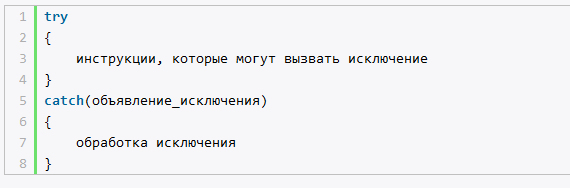
И в этом случае нам надо известить систему о возникшей ошибке. Для этого используется оператор throw.

Оператор throw генерирует исключение. Через оператор throw можно передать информацию об ошибке. Например, функция divide могла бы выглядеть следующим образом:

****

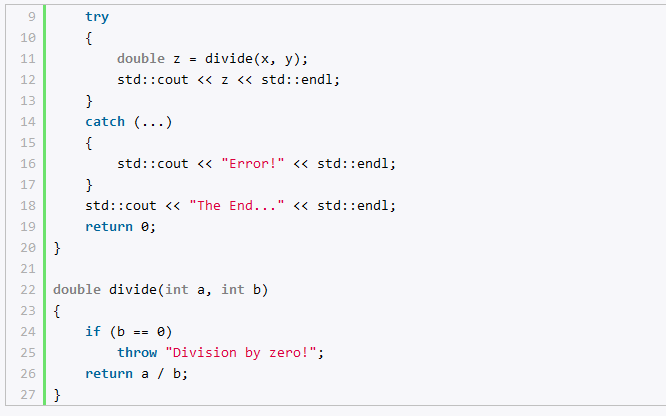
То есть если параметр b равен 0, то генерируем исключение.

Но это исключение еще надо обработать в коде, где будет вызываться функция divide. Для обработки исключений применяется конструкция try...catch. Она имеет следующую форму:

**** В блок после ключевого слова try помещается код, который потенциально может сгенерировать исключение.

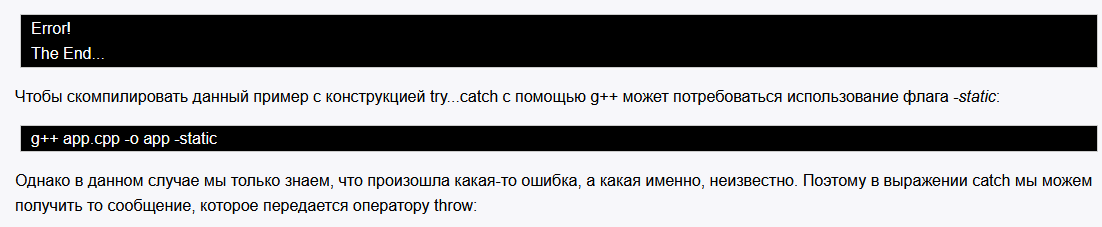
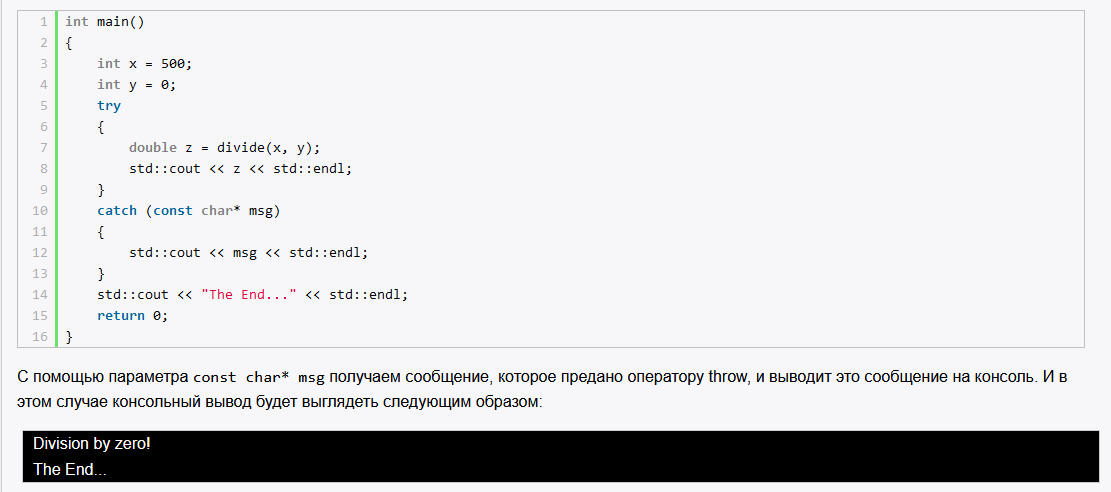
После ключевого слова catch в скобках идет параметр, который передает информацию об исключении. Затем в блоке производится собственно обработка исключения.

Так изменим весь код следующим образом:

**** од, который потенциально может сгенерировать исключение - вызов функции divide помещается в блок try.

В блоке catch идет обработка исключения. Причем многоточие в скобках после оператора catch (catch(...)) позволяет обработать любое исключение.

В итоге когда выполнение программы дойдет до строки double z = divide(x, y);, будет сгенерировано исключение, поэтому последующие инструкции из блока try выполняться не будут, а управление перейдет в блок catch, в котором на консоль просто выводится сообщение об ошибке. После выполнения блока catch программа аварийно не завершится, а продолжит свою работу, выполняя операторы после бллока catch:

**** ****

## 35. **Препроцессор: определение, назначение, применение, директивы, выражения, макросы,** **директивы условной компиляции. Примеры на C++.**

Препроцессор С++:  
директивы: #include, #define, #if, #else, #elif, #endif, #ifdef, #ifndef, #error, #line,  
#pragma, #undef; операторы: defined, #, ##.  
Препроцессор C/С++:  
текстовый процессор, который обрабатывает текст исходного файла на первой  
фазе компиляции.

Инструкции, регламентирующие работу компилятора, называются директивами  
препроцессора.  
Назначение.  
директивы препроцессора могут:  
− заменить какие-то лексемы в исходном тексте;  
− вставить содержимое других файлов в указанном месте;  
− подавить компиляцию части файла.  
Директивы препроцессора могут появляться в произвольном месте исходного  
текста, при этом они будут воздействовать только на оставшуюся часть исходного  
файла.

Препроцессор – программа для обработки текста.  
Препроцессор может быть отдельной программой, или же интегрирован в  
компилятор.  
Входные и выходные данные для препроцессора имеют текстовый формат.  
Препроцессор преобразует текст в соответствии с директивами препроцессора.  
В случае если текст не содержит директив препроцессора, то он остаётся без  
изменений.

Препроцессор можно вызвать отдельно для обработки текста программы без ее  
компиляции.  
Основные директивы препроцессора:  
#include – вставляет текст из указанного файла;  
#define – задаёт макроопределение (макрос) или символическую  
константу;  
#undef – отменяет предыдущее определение;  
#if – осуществляет условную компиляцию при истинности  
константного выражения;  
#ifdef – осуществляет условную компиляцию при определённости  
символической константы;  
#ifndef – осуществляет условную компиляцию при неопределённости  
символической константы;  
#else – ветка условной компиляции при ложности выражения;  
#elif – ветка условной компиляции, образуемая слиянием else и if;  
#endif – конец ветки условной компиляции;  
#line – препроцессор изменяет номер текущей строки и имя  
компилируемого файла;  
#error – выдача диагностического сообщения;  
#pragma – действие, зависящее от конкретной реализации  
компилятора. директивa #include (включение файла).  
Директива #include вставляет содержимое заданного файла в место расположения  
этой директивы в исходном тексте программы.  
Синтаксис:  
#include "спецификация\_пути"  
#include <спецификация\_пути>  
где "спецификация\_пути" — это имя файла, с необязательным указанием  
его местоположения.  
Кавычки и угловые скобки определяют способ поиска включаемых файлов:  
– если спецификация файла заключена в угловые скобки, то он должен  
находиться в каталоге, указанном компилятором (обычно это каталог INCLUDE,  
в котором хранятся заголовочные файлы).  
– если имя файла заключено в кавычки, то поиск выполняется в рабочем  
каталоге.  
– если указан полный путь, то предпроцессор использует его для поиска файла.  
4. Управление процессом компиляции:  
«Прагма» – это инструкция компилятору С/С++, используется для указания опций  
компилятору для управления его работой.  
Синтаксис:  
#pragma параметры  
Так #pragma once контролирует, чтобы конкретный включаемый файл при  
компиляции подключался строго один раз.

директивы: #if, #else  
Директива условной

компиляции #if позволяет подавить компиляцию части  
исходного файла.  
Каждая директива #if в исходном файле должна иметь соответствующую  
закрывающую директиву #endif.

директива #error  
Синтаксис:  
#error текст  
С помощью директивы #error можно определить текстовую строку (обратите  
внимание, что строка записывается без кавычек), которая будет выведена как  
сообщение об ошибке при компиляции. предопределенные макросы  
Компилятор C++ автоматически определяет некоторые макросы, например:  
\_\_LINE\_\_  
Этот макрос заменяется номером текущей строки в форме десятичной целой  
константы.  
Несмотря на то, что он называется предопределенным макросом, значение его  
меняется динамически в зависимости от местоположения макроса.  
Этот макрос в сочетании с макросом \_\_FILE\_\_ можно использовать при генерации  
сообщения об ошибке для вывода несоответствия, обнаруженного программой. В  
этом случае сообщение будет содержать номер строки с именем исходного файла,  
в котором была обнаружена ошибка. опрераторы препроцессора # и ##.  
Стрингификация (#) – это преобразование фрагмента кода в строковую  
константу, т.е. преобразование аргумента в строку. Например, в результате  
стрингификации STR(hello) аргумент преобразуется в символьную строку "hello".  
Канкатенация (##) (оператор конкатенации) – это конкатенация двух строковых  
констант. При работе с макросами, это означает объединение двух лексических  
единиц в одну более длинную. Например, один аргумент макроса может быть  
объединен с другим аргументом или с каким-либо текстом.

## **36. Статическая библиотека: определение, назначение, применение. Реализация статической** **библиотеки в Visual C++. Утилита LIB. Создание статической библиотеки. Применение**

Библиотека – файл, содержащий либо код на интерпретируемом языке, либо  
байт-код для виртуальной машины.  
Например, библиотеки для языка Python могут распространяться либо в виде  
файлов с исходным кодом (расширение «py»), либо в виде файлов с байт-  
кодом (расширение «pyc», (пайк) «py» + буква «c» от англ. compiled).

Статическая библиотека:  
файл (обычно с расширением lib), содержащий объектные модули;  
входной файл для компоновщика (linker).  
Достоинства:  
− просто использовать;  
− не требуется наличие самой библиотеки;  
− исполняемый файл один (расширение .exe).  
Недостатки:  
− платформенно зависима;  
− загружается в память с каждым экземпляром запущенного приложения;  
− при изменении кода библиотеки необходима компоновка всех  
приложений, которые используют библиотеку.  
Статическая библиотека – файл с объектными модулями, который  
подключается к программе на этапе компоновки.  
Компоновщик (linker, редактор связей) – программа, принимающая один или  
несколько объектных модулей и формирующая на их основе загрузочный  
модуль:

Если программа состоит из нескольких объектных файлов, компоновщик  
собирает эти файлы в единый исполнимый модуль, вычисляя и подставляя  
адреса вместо неопределенных внешних имен, в течение времени компоновки  
(статическая компоновка) или во время исполнения (динамическая  
компоновка)

Статическая библиотека Microsoft:  
файл с расширением lib.  
Для работы с библиотекой предназначена утилита LIB.  
7.1. Создание статической библиотеки с помощью «Мастера классических  
приложений Windows». Шаг 1:

Определяем имя решения, проекта и выбираем место размещения на диске.  
7.2. Создание статической библиотеки с помощью «Мастера классических  
приложений Windows». Шаг 2.  
Выбираем тип приложения «Статическая библиотека» (снимаем флажок  
«Предварительно скомпилированные заголовки» при необходимости):

В проект добавляем один или несколько файлов, содержащих реализации  
функций библиотеки.  
7.3. Запускаем проект на выполнение

В пункте раздела «Библиотекарь»  
→ «Командная строка» отображается  
текущее значение параметра /OUT.  
Расширение выходных файлов определено как .lib.  
Директорий – папка проекта Debug.

После построения проекта в папке решения Debug размещен файл статической  
библиотеки (.lib).

В журнале проекта зафиксировано выполнение сборки проекта.  
Видим, что файл статической библиотеки создан утилитой LIB.

ежимы использования утилиты LIB:  
− построение или изменение библиотеки;  
− извлечение элемента-объекта библиотеки в файл;  
− создание файла экспорта и библиотеки импорта.  
Эти режимы взаимоисключающие, LIB можно использовать только в одном  
режиме.  
LIB принимает те или иные входные файлы в зависимости от режима  
использования.

Параметры LIB:  
https://docs.microsoft.com/ru-ru/cpp/build/reference/overview-of-lib?view=vs-2019  
/DEF  
Создание библиотеки импорта и файла экспорта  
Дополнительно в Построение библиотеки импорта и файла экспорта.  
/ERRORREPORT  
Передача Майкрософт сведений о внутренних ошибках.  
Дополнительно в Запуск программы LIB.  
/EXPORT  
Экспорт функции из программы.  
Смори в разделе Построение библиотеки импорта и файла экспорта.  
/EXTRACT  
Создание объектного файла (OBJ-файла), содержащего копию  
элемента существующей библиотеки.  
Дополнительно в разделе Извлечение члена библиотеки.  
/INCLUDE  
Добавление символа в таблицу символов.  
Дополнительные сведения см. в разделе Построение библиотеки  
импорта и файла экспорта.  
/LIBPATH  
Переопределяет путь к библиотеке среды.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/LIST  
Отображает информацию о выходной библиотеке в стандартном виде.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/LTCG  
Инициирует построение библиотеки с помощью создания кода времени  
компоновки.  
Дополнительные сведения см. в разделе Запуск программы LIB.  
/MACHINE  
Задание целевой платформы для программы.

8  
Дополнительные сведения см. в разделе Запуск программы LIB.  
/NAME  
При построении библиотеки импорта указывает имя библиотеки DLL,  
для которой была создана библиотека импорта.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/NODEFAULTLIB  
Удаляет одну или несколько библиотек по умолчанию из списка  
искомых библиотек при разрешении внешних ссылок.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/NOLOGO  
Отключает вывод программой LIB уведомления об авторских правах и  
номере версии, а также отображение команд командного файла.  
Дополнительные сведения см. в разделе Запуск программы LIB.  
/OUT  
Переопределяет имя выходного файла, используемое по умолчанию.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/REMOVE  
Пропуск объекта из выходной библиотеки.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/SUBSYSTEM  
Сообщает операционной системе способ запуска программы,  
созданной путем привязки к выходной библиотеке.  
Дополнительные сведения см. в разделе Управление библиотекой.  
/VERBOSE  
Отображает подробные сведения о ходе сеанса, включая имена  
добавляемых OBJ-файлов.  
Дополнительные сведения см. в разделе Запуск программы LIB.  
/WX  
Обработка предупреждений, как ошибок.  
Дополнительные сведения см. в разделе Запуск программы LIB.

Заголовочные файлы. Правила создания.  
В заголовочном файле задают:  
− прототипы всех функций, которые входят в библиотеку (интерфейс  
библиотеки), например: void Print(Inctance d);  
− типы пользователя;  
− константы, например: #define DICTNAMEMAXSIZE 20

9.2. Главная программа для использования статической библиотеки.  
Добавляем в решение новый пустой проект SE\_Lab06Ex – консольное  
приложение, которое будет использовать созданную нами библиотеку.

Назначаем проект SE\_Lab06Ex запускаемым и выполняем его.  
На странице свойств проекта в разделе «Компоновщик»  
→ Командная  
строка в окне «Дополнительные параметры» добавляем параметр,  
указывающий местоположение и имя статической библиотеки.

Подключаем в main заголовочный файл библиотеки:

Получаем исполнимый файл SE\_Lab06Ex:

**37. Теория формальных языков. Определения: алфавит языка, цепочка, пустая цепочка, длина  
цепочки, равенство (эквивалентность) цепочек, конкатенация цепочек, итерация цепочки,  
операторы + и \*. Примеры.**

Формальный язык – это множество конечных слов над конечным алфавитом,  
например, язык программирования.

Алфавитом V называется конечное множество символов, допустимых в  
языке.  
Пример:},,{ cbaV = , I = {anbn},  
где n — натуральное число.  
Алфавит задает язык I, состоящий из цепочек вида ab, aabb, aaabbb и т.д.  
Язык I представляет собой бесконечное множество цепочек, но его описание  
состоит всего из 8 символов, т.е. конечно.Цепочкой  
α (альфа) в алфавите I называется любая конечная  
последовательность символов этого алфавита.  
Цепочка, которая не содержит ни одного символа, называется пустой  
цепочкой и обычно обозначается  
ε или  
λ.

Длиной цепочки  
α (обозначается |  
α|) называется число составляющих ее  
символов.  
Пример: |  
α| = 3, |  
β| = 4, |  
γ| = 7. Длина пустой цепочки |  
ε|=0.  
4). Конкатенацией (сцеплением) цепочек  
α и  
β называется цепочка  
γ=  
αβ, в  
которой символы данных цепочек записаны друг за другом.  
Пример:  
α=abc,  
β=aaaa,  
αβ= abcaaaa.  
5). Для любой цепочки  
α справедливо утверждение  
αε=  
εα.  
6).  
αn называется итерацией цепочки  
α.  
Справедливы следующие утверждения:  
α0 =  
ε  
αn =  
αn-1  
α =  
ααn-1 для n≥1.  
Пример:ab=α ,abababab ==α 33 )( ,ε  
α  
=0  
7). Цепочки  
α и  
β равны (  
α =  
β), если они имеют один и тот же состав  
символов, одно и тоже количество символовβ=α и тот же порядок  
следования символов.  
8). Реверсом (обращением) цепочки  
α называется цепочка  
αR, составленная  
из символов цепочки  
α, записанных в обратном порядке.  
Пример. Пусть алфавит I = {a, b, c, d}, тогда для цепочек этого алфавита  
α=ab и  
β=bcd будет справедливо |  
α|=2, |  
β|=3,  
αβ= abbcd,  
α2=abab,  
βR=dcb.  
9). ПустьI – алфавит, тогда+  
I – множество всех цепочек, состоящих из  
символов алфавита I, исключая пустую цепочку (  
λ).+  
∉ I  
λ  
10). ПустьI – алфавит, тогда\*  
I – множество всех цепочек, состоящих из  
символов алфавита I, включая пустую цепочку.\*  
I∈λλ∪= +  
II \*

4  
11). Языком L(I) над алфавитомI называется произвольное подмножество  
цепочек из\*  
I ,\*  
)( IIL ⊆ .  
Пример:},,{ cbaI = ,},,{)(1 bcacabIL =  
,},,{)(2 cbaIL = ,},,{)(3 ccbbaaIL = , ...  
12). Язык)(1 IL является подмножеством языка)(2 IL , если каждая цепочка,  
входящая в язык1L , входит в язык2L ; язык2L включает язык1L .))()(()()( 2121 ILILILIL ∈α⇒∈α∀⇔⊆  
13). Языки)(1 IL и)(2 IL совпадают (эквивалентны), если язык)(1 IL  
включает язык)(2 IL и язык)(2 IL включает язык)(1 IL .))()()()(()()( 122121 ILILILILILIL ⊆∧⊆⇔=.

Алфавит будем обозначать заглавными буквами латинского алфавита:  
I, V, G, ...  
Символы алфавита будем обозначать строчными буквами латинского  
алфавита:  
a, b, c, ...  
Цепочки будем обозначать символами греческого алфавита:  
α,  
β,  
γ, δ,  
λ,  
ε, ω,...  
Пример: α = a 1... a n – цепочка из n символов.  
Пустую цепочку символов будем обозначать  
λ,  
ε.

**38. Теория формальных языков. Определение формального языка, эквивалентность двух языков, способы задания формального языка. Лексика, синтаксис и семантика языка.  
Примеры.**

Теория формальных языков (формальных грамматик) занимается описанием, распознаванием и переработкой языков. Описание любого языка должно быть конечным, хотя сам язык мо­жет содержать бесконечное множество цепочек. Полезно иметь возможность описания отдельных типов языков, имеющих те или иные свойства, т. е. иметь различные типы конечных описаний. Синтаксис языка можно задать с использованием синтаксических диаграмм или другим альтернативным способом, воспользовавшись нотацией Бэкуса-Наура (НБН). При ис­пользовании нотации Бэкуса-Наура нетерминальные символы (подлежащие дальнейшему определению) языка заклю­чаются в угловые скобки вида < ... > .

Последовательность символов : : = означает «определя­ется как», а символ | означает "или". Пример описания в нотации Бэкуса-Наура арифметических выражений, содержащих переменные *a, b,*приведен ниже.

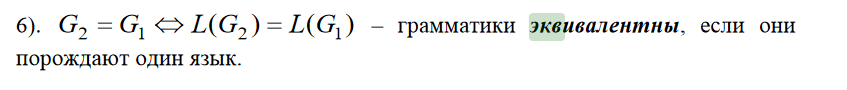
< выражение > : : = < терм > | < терм > + < выражение > | < терм > - < выражение >

< терм > : : = < множитель > | < множитель > \* < терм > | < терм > / < множитель >

< множитель > : : = a|b| ( < выражение > )

+ Существуют два универсальных способа описания отдельных классов языков. Первый способ – *грамматики*как механизмы, порождающие цепочки символов. Второй способ определяет язык в терминах множества цепочек, допускаемых некоторым распознающим устройством, называемым*автоматом*.

Формальный язык – это множество конечных слов над конечным алфавитом,  
например, язык программирования.

****

Основными объектами грамматики являются базовые элементы языка или терминальные символы, а также цепочки, построенные из этих элементов.

При определении грамматики будем придерживаться следующих соглашений:

* прописными буквами обозначаются нетерминальные символы языка;
* строчными буквами латинского алфавита обозначаются терминальные символы языка;
* цепочки обозначаются либо прописными буквами латинского алфавита, либо греческими буквами.
* символ «→»используется для обозначения отношения "определяется как".

Введем в рассмотрение пустую цепочку ε, не содержащую ни одного символа.

*Длиной*цепочки будем называть число символов, входящих в эту цепочку, например:

*B = abab, |B| = |abab*| = 4,

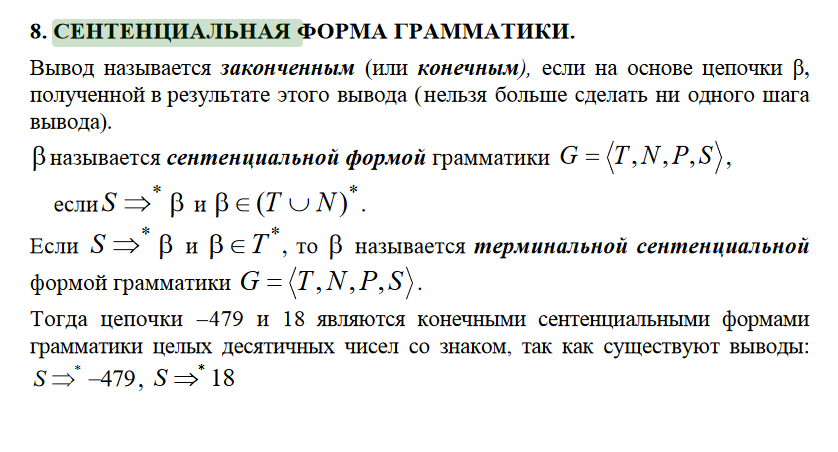
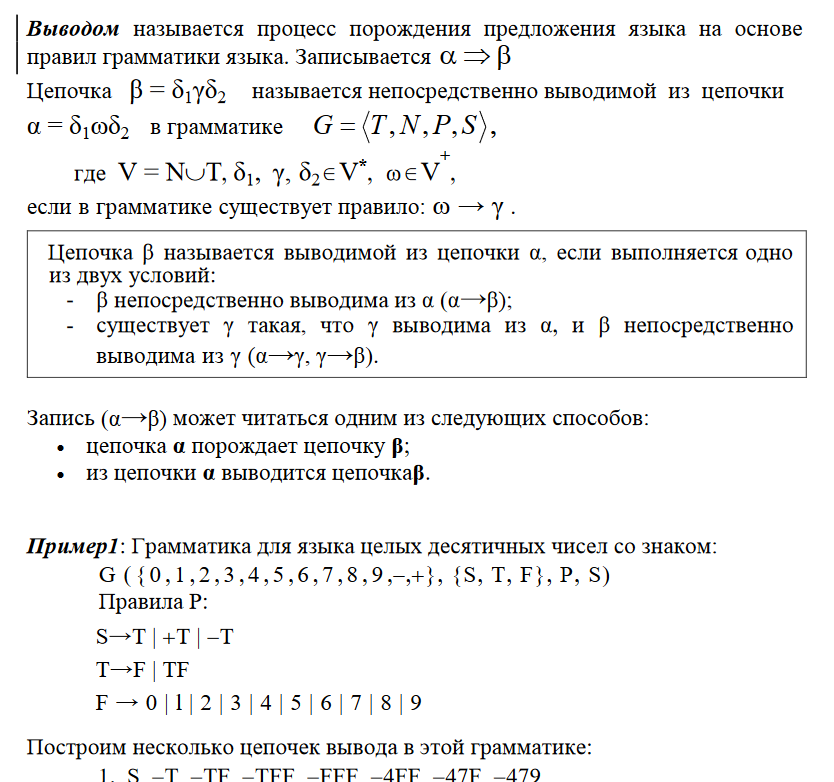
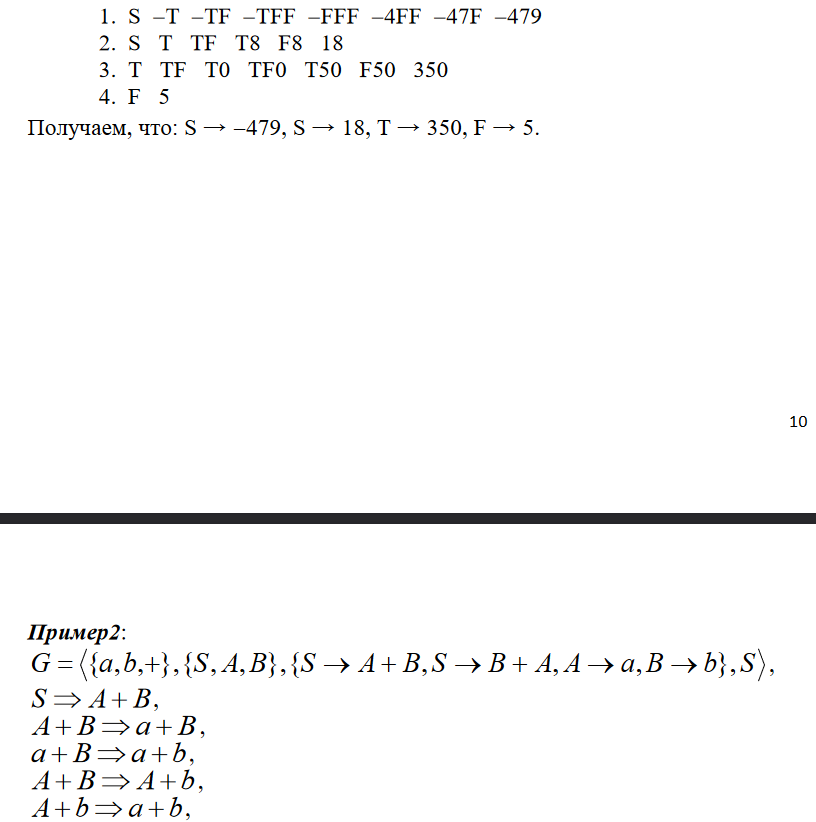
| ε| = 0.

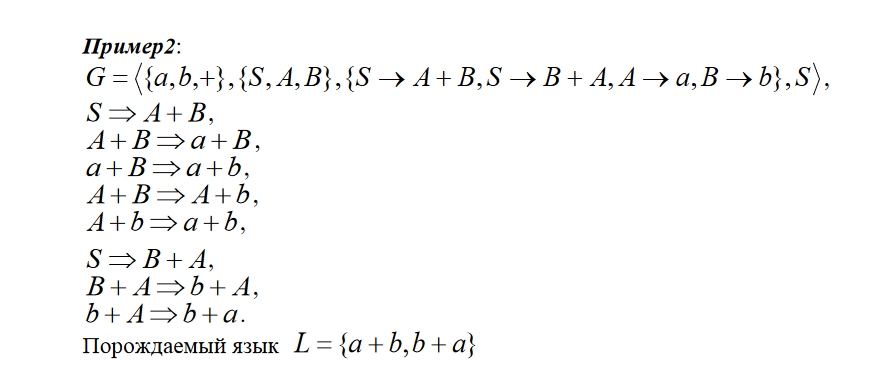
*Конкатенацией*двух цепочек*X*и*Y*называется такая цепочка*Z*, которая получается непосредственным слиянием цепочки*X*, стоящей слева, и цепочки*Y*, стоящей справа. Например, если*X = ffg*,*Y = ghh*, то конкатенация*X*и*Y*– это цепочка*Z*=*ffgghh*.

+ Будем обозначать через А\*множество всех возможных цепочек конечного множества А базовых элементов (символов), включая пустую цепочкуε; Множество А называют также*алфавитом*. Любое множество цепочек L ⊆A\*называется*формальным языком*, определенным на алфавите А.

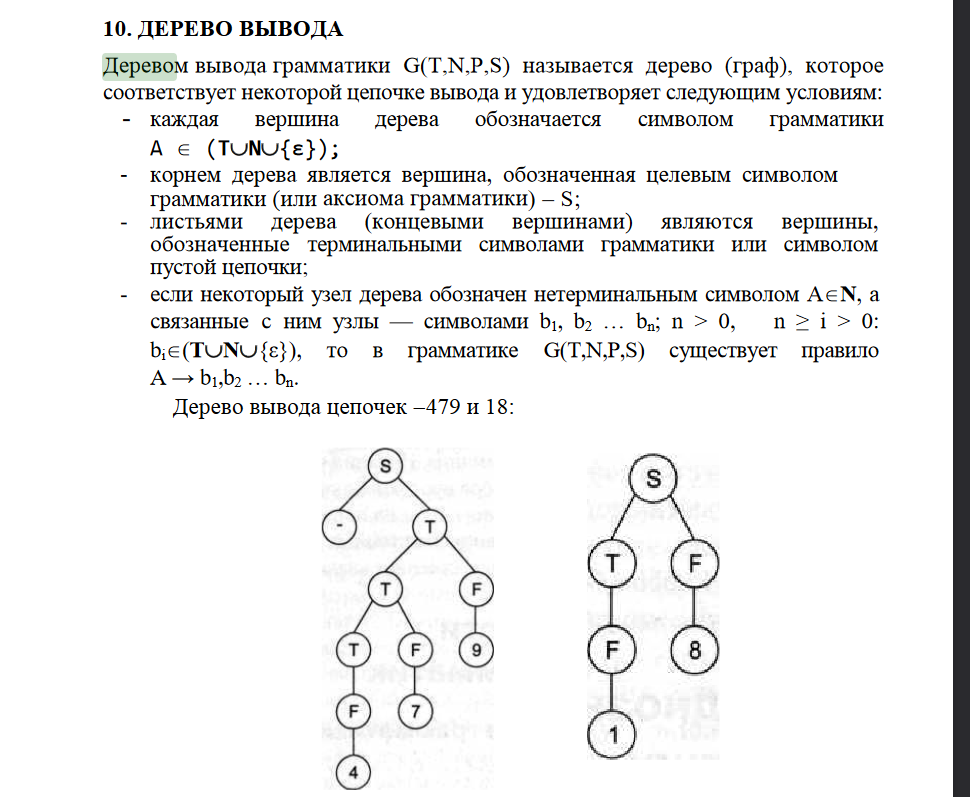
**39. Теория формальных языков. Формальная грамматика (определение, назначение),  
выводимость цепочки символов в грамматике, сентенциальная форма грамматики, язык,  
порождаемый грамматикой, способы задания грамматик. Примеры.**

****

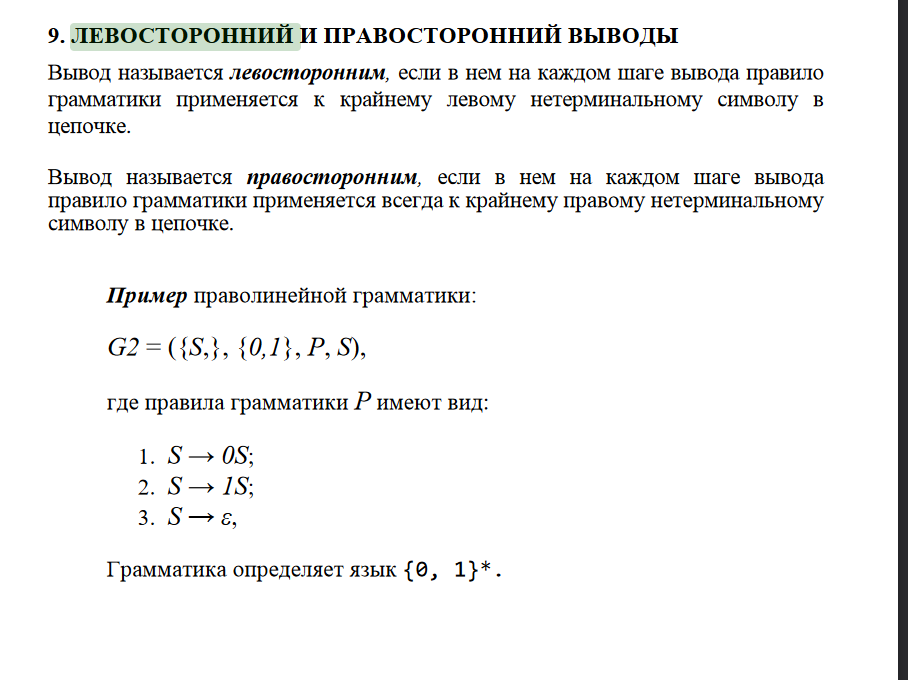
**** **** 

****

## **40. Теория формальных языков. Определение порождающей грамматики. Определение** **вывода, левосторонний и правосторонний выводы, дерево вывода. Примеры.**

****

13  
10. ДЕРЕВО ВЫВОДА  
Деревом вывода грамматики G(T,N,P,S) называется дерево (граф), которое  
соответствует некоторой цепочке вывода и удовлетворяет следующим условиям:  
- каждая вершина дерева обозначается символом грамматики  
А ∈ (T∪N∪{ε});  
- корнем дерева является вершина, обозначенная целевым символом  
грамматики (или аксиома грамматики) – S;  
- листьями дерева (концевыми вершинами) являются вершины,  
обозначенные терминальными символами грамматики или символом  
пустой цепочки;  
- если некоторый узел дерева обозначен нетерминальным символом А∈N, а  
связанные с ним узлы — символами b1, b2 ... bn; n > 0, n ≥ i > 0:  
bi∈(T∪N∪{ε}), то в грамматике G(T,N,P,S) существует правило  
A → b1,b2 ... bn.  
Дерево вывода цепочек −479 и 18:  
1). Цепочка\*  
)( TN ∪∈β выводима из цепочки+  
∪∈α )( TN в  
грамматикеSPNTG ,,,= , если существуют цепочки, такие чтоβ⇒γ⇒⇒γ⇒γ⇒γ⇒α n...210  
. Записывается какβ⇒α \* .  
Тогда последовательностьβγγγγα ,,,,, 210 n называется выводом длиныn .  
Пример:SAAAASASG },,1000,10{},,{},1,0{ λ→→→=  
Вывод100010010 AAAS ⇒⇒⇒  
Длина вывода цепочки1000A из стартового символаS равна 2.  
(Запись второго правила рекурсивна).

14  
2). Записьβ⇒α \* предполагает0≥n шагов выводаβ изα .  
В том случае, еслиβ⇒α , то число шагов вывода0=n .  
Записьβ⇒α + предполагает0>n шагов выводаβ изα .  
3). Для записи правил вывода с одинаковыми левыми частями  
α → β1 α → β2 ... α → βn  
будем пользоваться сокращенной записью  
α → β1 | β2 |...| βn.  
Каждое βi , i = 1, 2, ... ,n – альтернатива правила вывода из цепочки α.  
4).)(GL – язык, порождаемый грамматикойG .  
Язык)(GL содержит все терминальные цепочки, выводимые изS :}\*|{)( \* α⇒∈α= STGL  
.  
Язык, порождаемый грамматикой – это множество всех выводимых из  
аксиомы грамматики терминальных цепочек  
5).)(GL – множество терминальных сентенциальных форм грамматикиG .  
Пример:SbBaAABSBASBASbaG },,,,{},,,{},,,{ →→+→+→+=  
,},{)( abbaGL ++=  
6).)()( 1212 GLGLGG =⇔= – грамматики эквивалентны, если они  
порождают один язык.  
Пример:  
Язык L в алфавите}0,1{=V , состоящий из пустой строки и строк, каждая из  
которых содержит строку, состоящую из нулей и такого же количества  
единиц, можно также описать с помощью формальной системы определения  
множеств как L={0n1n | n ≥0}. ****