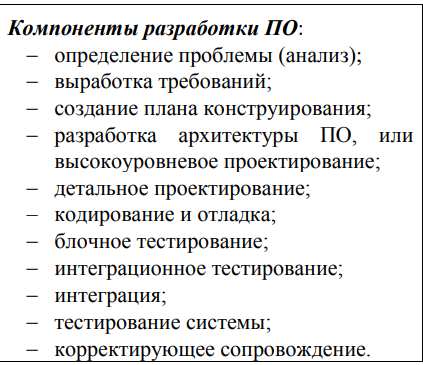
## **1. Современные подходы к разработке программного обеспечения (ПО): понятие конструирования ПО, основные этапы конструирования ПО.**

**Понятие конструирования программного обеспечения**

**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.**

**1 вопрос +**

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

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

* выбор языка программирования;
* конвенции программирования (НАБОР ПРАВИЛ ПРИ НАПИСАНИИ КОДА);
* выбор технологий;
* выбор основных методик конструирования (тестирование, кодирование, планирование, оценка, анализ, сопровождение, проектирование)

## **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** - программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке, **компоновщики** - программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль,
* **отладчики** - программа, позволяющая контролировать ход выполнения программы (приостанавливать, выполнять пошагово), просматривать и изменять области памяти,
* **профилировщики** - программа, позволяющая оптимизировать код программы (устранять утечки памяти, оптимизировать циклы, анализировать производительность),
* **программные библиотеки**,
* **редакторы кода**,
* **системы поддержки версий**.

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

**Классический жизненный цикл разработки программного обеспечения:**

**Жизненный цикл программного обеспечения** (ПО) — период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации. Этот цикл — процесс построения и развития ПО.

**Текстовый редактор:** компонента системы программирования - программа, позволяющая подготовить исходный код программы.

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

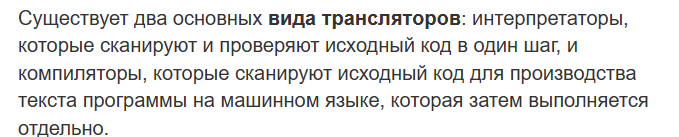
**Интегрированная среда разработки** (IDE, Integrated development environment): редакторы кода, транслятор, компоновщик, отладчик, система поддержки версий. Примеры: Visual Studio, NetBeans, Eclipse, Embarcadero Delphi и пр.

**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., используется программистами для разработки создания и тестирования программ или программного обеспечения. ( состоит из редактора, компилятора, сборщика, библиотекаря, отладчика, профайлера)

**Трансля́тор** — программа или техническое средство, выполняющее преобразование программы, представленной на одном из языков программирования, в программу на другом языке.



**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).

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

**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 код. Загрузчик: определение, назначение. Общая схема преобразования исходного кода в процесс операционной системы.**

**Компоновщик:**

**Определение:** программа, принимающая один или несколько объектных модулей и формирующая на их основе загрузочный модуль.

**Назначение:** если программа собирается из нескольких объектных файлов, компоновщик может собирать эти файлы в единый исполнимый модуль, вычисляя и подставляя [адреса](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D1%80%D0%B5%D1%81_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29) 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:**

**Определение:** программа, обычно входящая в состав операционной системы, предназначенная для запуска процесса операционной системы на основе загрузочного модуля.

**Назначение:** отвечает за загрузку исполнимых файлов и запуск соответствующих новых процессов.



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

***Иерархическая структура компонентов в 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++).**

Многофайловый проект в Visual C++ - это проект, состоящий из нескольких исходных файлов (.cpp), заголовочных файлов (.h) и, возможно, других типов файлов, таких как ресурсы (.rc) или файлы конфигурации (.xml).

Предварительно откомпилированные заголовки (Precompiled Headers) - это механизм оптимизации компиляции в C++, который позволяет ускорить время сборки проекта. Они создаются путем предварительной компиляции некоторого набора заголовочных файлов и сохранения их в специальном формате для последующего использования при компиляции других файлов проекта.

Создание предварительно откомпилированных заголовков в Visual C++:

1. Выберите заголовочные файлы, которые будут использоваться в большинстве файлов проекта и редко изменяются.
2. Откройте файл проекта (.vcxproj) в Visual Studio.
3. Выделите конкретный исходный файл, который будет использоваться для создания предварительно откомпилированных заголовков.
4. Щелкните правой кнопкой мыши и выберите "Свойства" (Properties).
5. В окне свойств выберите "C/C++" -> "Предварительно откомпилированные заголовки" (Precompiled Headers).
6. Установите опцию "Предварительно скомпилированный заголовок" (Precompiled Header) на "Создание (/Yc)".
7. Сохраните изменения.

Применение предварительно откомпилированных заголовков в C++ файле:

1. Откройте файл C++ (.cpp), для которого вы хотите использовать предварительно откомпилированные заголовки.
2. Щелкните правой кнопкой мыши на файле и выберите "Свойства" (Properties).
3. В окне свойств выберите "C/C++" -> "Предварительно откомпилированные заголовки" (Precompiled Headers).
4. Установите опцию "Использование предварительно скомпилированного заголовка" (Precompiled Header) на "Использование (/Yu)".
5. Сохраните изменения.

## **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: American Standard Code for Information Interchange —** 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 трансляции:** текст программы на языке программирования хранится в исходных файлах и основан на определенной кодировке символов

**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).

**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 - universal character set (универсальный набор символов);

UTF - Unicode transformation format (семейство кодировок).

Принято обозначение U+xxx, где xxx- число в шестнадцатеричном формате.

**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.

|  |  |
| --- | --- |
| **Кодировка** | **Представление (**[**hex**](https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D1%81%D1%82%D0%BD%D0%B0%D0%B4%D1%86%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F)**)** |
| [UTF-8](https://ru.wikipedia.org/wiki/UTF-8) | EF BB BF |
| [UTF-16](https://ru.wikipedia.org/wiki/UTF-16) ([BE](https://ru.wikipedia.org/w/index.php?title=Big_Endian&action=edit&redlink=1)) | FE FF |
| [UTF-16](https://ru.wikipedia.org/wiki/UTF-16) ([LE](https://ru.wikipedia.org/w/index.php?title=Little_Endian&action=edit&redlink=1)) | FF FE |
| [UTF-32](https://ru.wikipedia.org/wiki/UTF-32) (BE) | 00 00 FE FF |
| [UTF-32](https://ru.wikipedia.org/wiki/UTF-32) (LE) | FF FE 00 00 |

## **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.

С помощью символов алфавита записываются служебные слова, которые

составляют словарь 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

Алфавит языка программирования служит для построения слов в языке

программирования, которые называют лексемами.

Примеры лексем:

идентификаторы; ключевые (зарезервированные) слова; знаки операций;

константы; разделители (скобки, знаки операций, точка, запятая, пробельные символы).

Идентификатор C++:

- идентификаторы должны начинаться с буквы или подчеркивания;

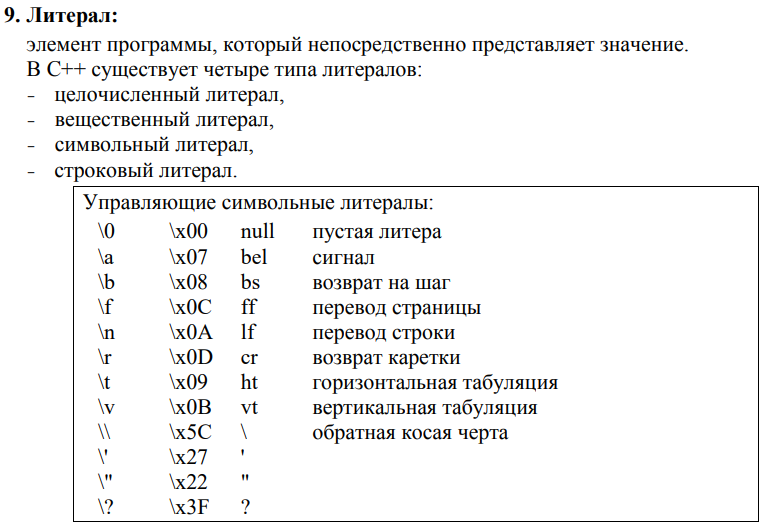
- идентификатор не может совпадать с ключевыми словами С++ или с

именами библиотечных функций;

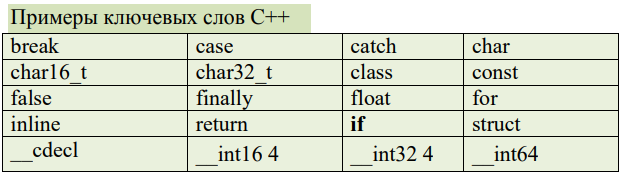
- идентификаторы могут состоять из любого количества символов, но компилятор гарантирует, что будет считать значащими:

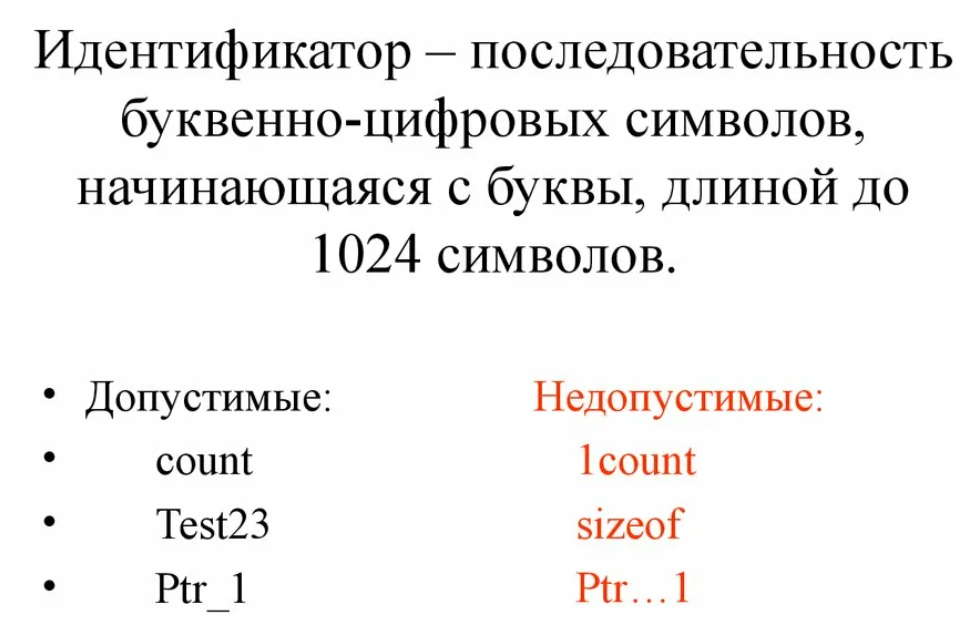
- идентификаторы чувствительны к регистру.

Зарезервированный идентификатор в программировании — это слово или символьная последовательность, которая имеет специальное значение и зарезервирована для использования в языке программирования или в его стандартной библиотеке. Примеры зарезервированных идентификаторов в языке программирования C++ включают такие ключевые слова, как "if", "while", "for", "int" и т.д

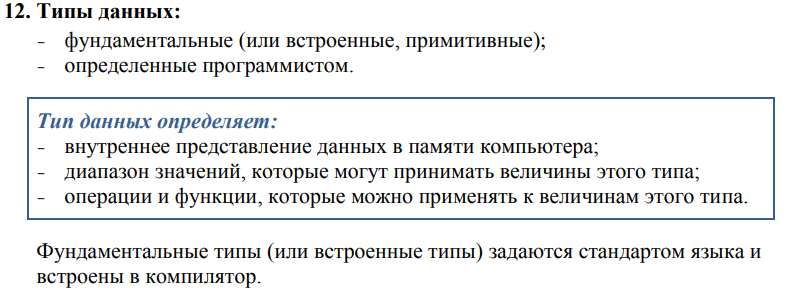


Ключевые слова: последовательности символов алфавита языка, имеющие специальное назначение. Ключевые слова зарезервированы компилятором для обозначения типов переменных, класса хранения, элементов операторов и т.д.





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



Фундаментальные типы C++

- int (целый);

- char (символьный);

- wchar\_t (расширенный символьный);

- bool (логический);

- float (вещественный);

- double (вещественный с двойной точностью);

- тип void.

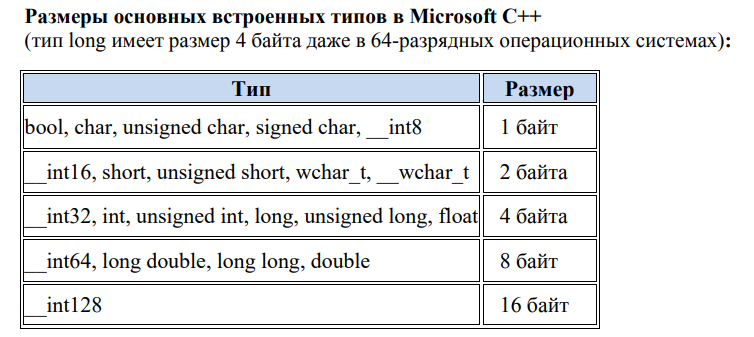
Модификаторы основных типов, уточняющие внутреннее представление и диапазон значений стандартных типов:

- short (короткий);

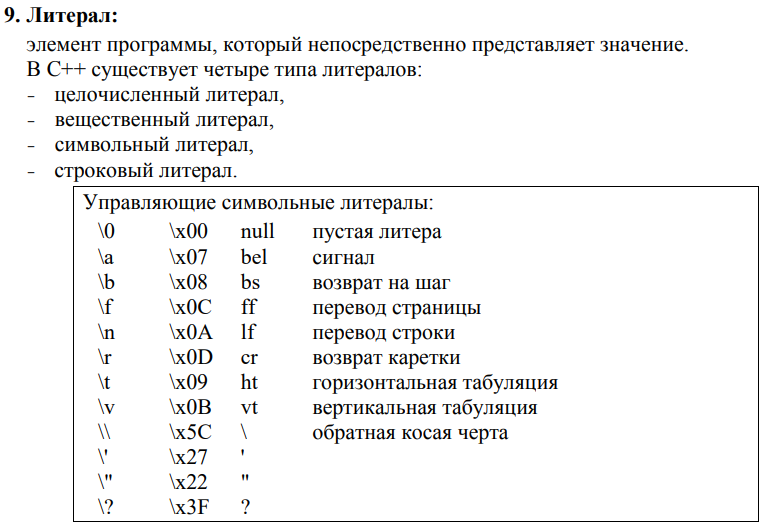
- long (длинный);

- signed (знаковый);

-unsigned(беззнаковый).



## **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

## **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);

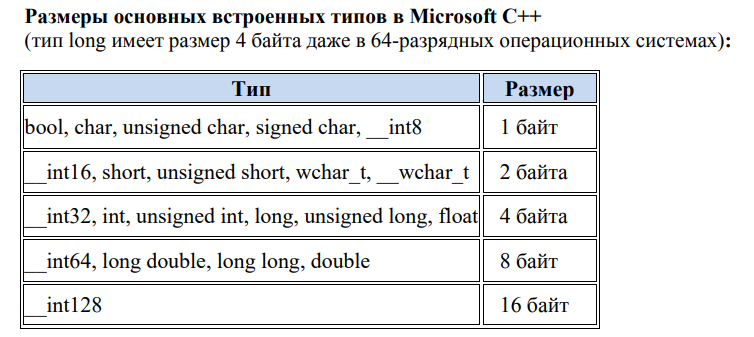
-методы, используемые для преобразования числа при выполнении

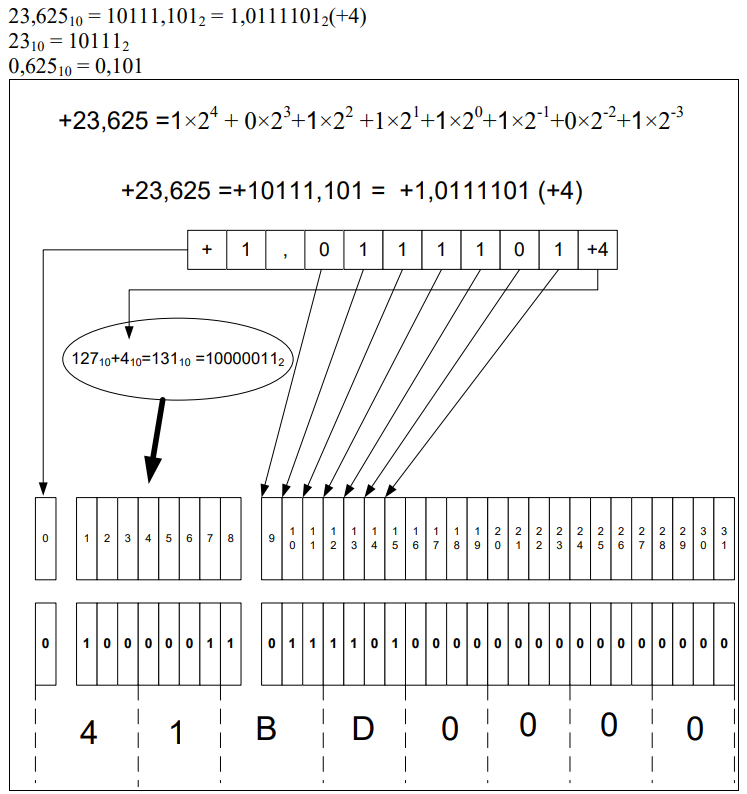
математических операций;

-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

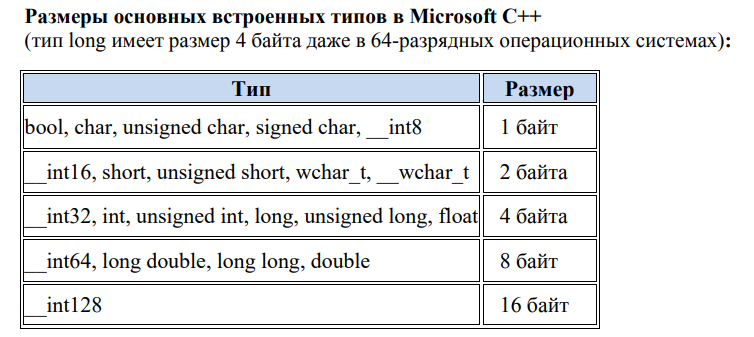
диапазон значений стандартных типов:

- 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).

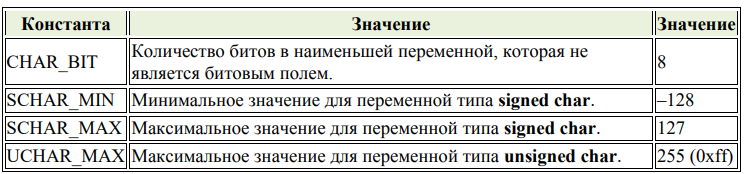
- спецификатор unsigned: старший разряд (бит) рассматривается как

значащий, позволяет представлять только положительные числа.

По умолчанию все целочисленные типы считаются знаковыми, то есть

спецификатор signed можно опускать. Диапазон значений зависит от

реализации.



* **int**: представляет целое число. В зависимости от архитектуры процессора может занимать 2 байта (16 бит) или 4 байта (32 бита). Диапазон предельных значений соответственно также может варьироваться от –32768 до 32767 (при 2 байтах) или от −2 147 483 648 до 2 147 483 647 (при 4 байтах). Но в любом случае размер должен быть больше или равен размеру типа short и меньше или равен размеру типа long

Данный тип имеет синонимы **signed int** и **signed**.

* **unsigned int**: представляет положительное целое число. В зависимости от архитектуры процессора может занимать 2 байта (16 бит) или 4 байта (32 бита), и из-за этого диапазон предельных значений может меняться: от 0 до 65535 (для 2 байт), либо от 0 до 4 294 967 295 (для 4 байт).

В качестве синонима этого типа может использоваться **unsigned**

* **long**: представляет целое число в диапазоне от −2 147 483 648 до 2 147 483 647. Занимает в памяти 4 байта (32 бита).

У данного типа также есть синонимы **long int**, **signed long int** и **signed long**

* **unsigned long**: представляет целое число в диапазоне от 0 до 4 294 967 295. Занимает в памяти 4 байта (32 бита).

Имеет синоним **unsigned long int**.

* **long long**: представляет целое число в диапазоне от −9 223 372 036 854 775 808 до +9 223 372 036 854 775 807. Занимает в памяти, как правило, 8 байт (64 бита).

Имеет синонимы **long long int**, **signed long long int** и **signed long long**.

* **unsigned long long**: представляет целое число в диапазоне от 0 до 18 446 744 073 709 551 615. Занимает в памяти, как правило, 8 байт (64 бита).

## **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.

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.

**Вывод типов**

В стандарте C++11 введено новое ключевое слово **auto** для определения ***явно***

инициализируемой переменной.

Создается переменная, тип которой выводится из инициализирующего значения:

auto variable1 = 5; auto variable2 = 2.5;

**Преобразование типов:**

* автоматическое преобразование;
* явное преобразование.

|  |  |
| --- | --- |
| ***явное*** | задается программистом в тексте программы с помощью:   * конструкции языка; * функции, принимающей значение одного типа и возвращающей значение другого типа. |
| ***неявное*** | выполняется автоматически транслятором (компилятором или  интерпретатором) по правилам, описанным в стандарте языка. |

Стандарты большинства языков запрещают неявные преобразования.

## *Пример:*

unsigned int u3 = 0 - 1;

cout << u3 << endl;

// вывод: 4294967295

Компилятор ***не предупреждает*** о неявных преобразованиях между целыми типами со знаком и без знака.

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

### Автоматическое (неявное) преобразование типов:

Для базовых типов преобразование типов выполняется без потери точности.

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

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 как ошибку.

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

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