# УТВЕРЖДАЮ

Зав. кафедрой программной инженерии БГТУ к.т.н., доцент Н.В. Пацей протокол № 9 от 06 мая 2022 г.

# Экзаменационные вопросы по дисциплине

**«Конструирование программного обеспечения», 1 курс ПОИТ, весенний семестр 2022**

**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;

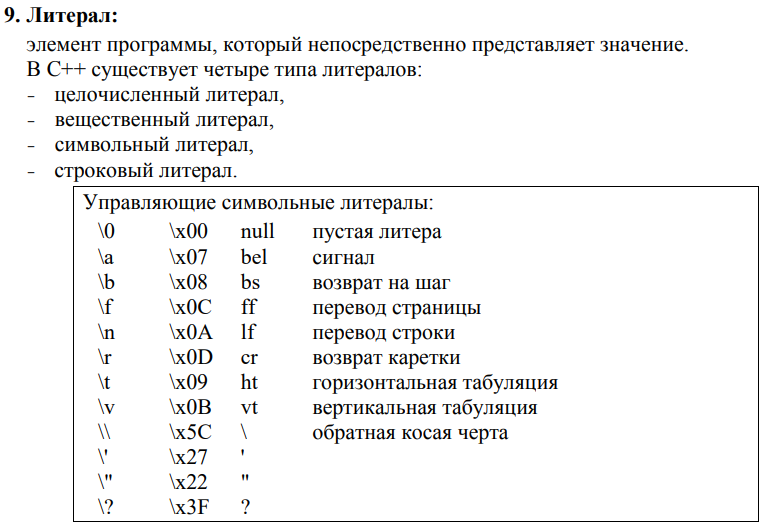
- идентификаторы чувствительны к регистру.

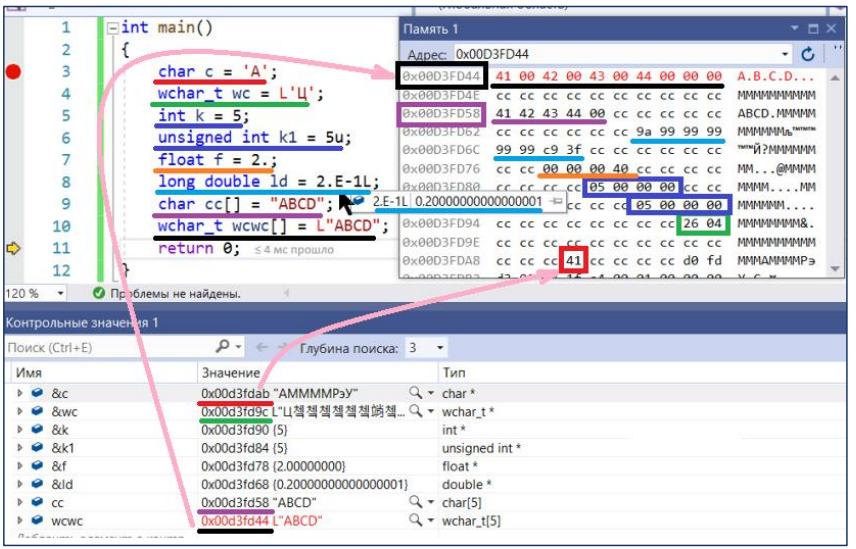
Длина идентификатора по стандарту не ограничена.

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

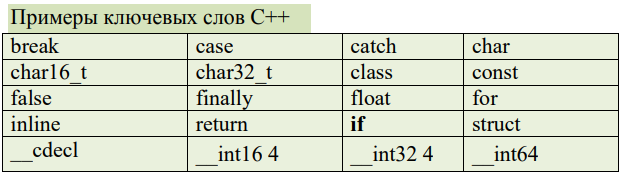
Зарезервированные идентификаторы: идентификаторы, которые предварительно определены в системе программирования.

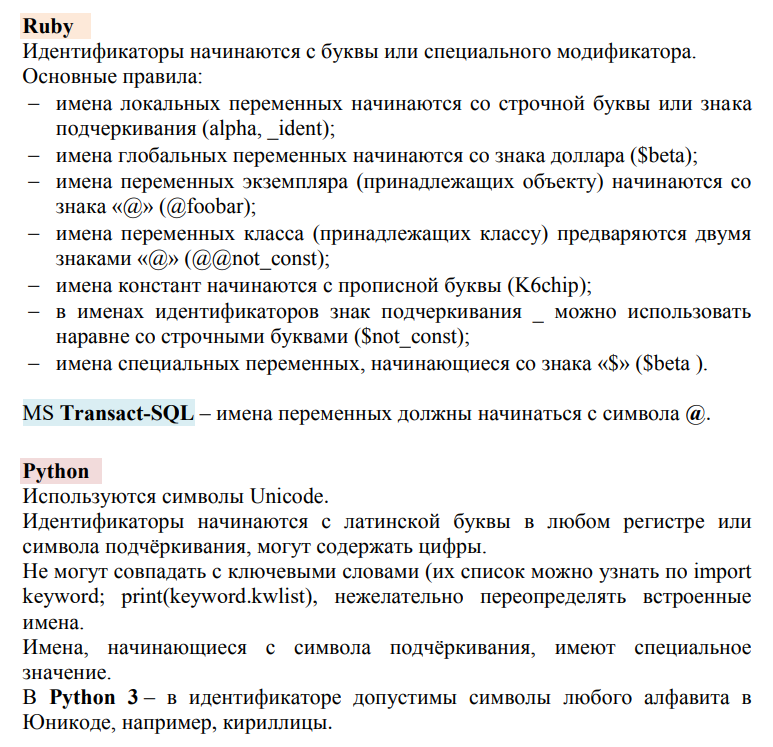
Зарезервированные идентификаторы С++: − все имена с двумя подчеркиваниями считаются зарезервированным; − Кроме того: isxxxx, memxxxx, strxxxx, toxxxx, wcsxxxx, Eцифраxxxx, LC\_Xxxx, SIGXxxx, SIG\_Xxxxx.



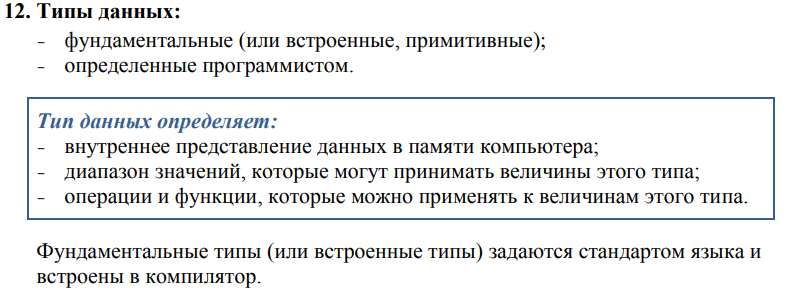


Ключевые слова: последовательности символов алфавита языка, имеющие специальное назначение. Ключевые слова зарезервированы компилятором для обозначения типов переменных, класса хранения, элементов операторов и т.д.





**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

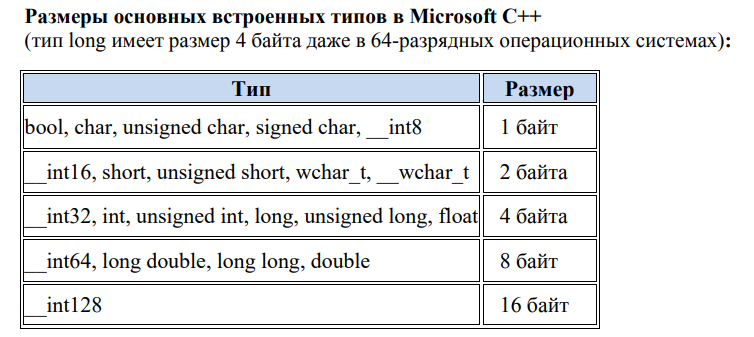
диапазон значений стандартных типов:

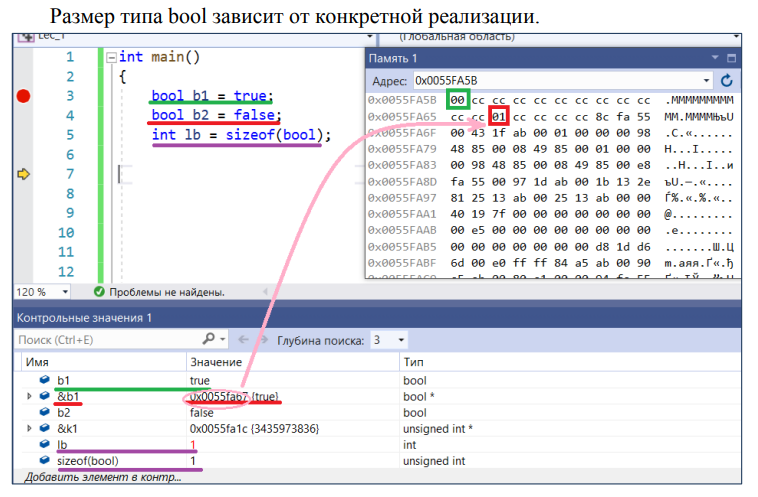
- short (короткий);

- 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.

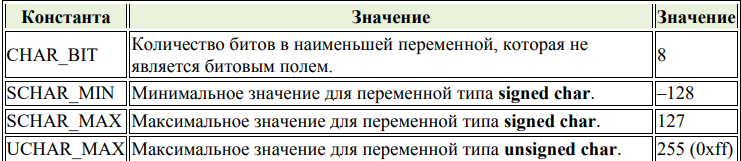
По умолчанию все целочисленные типы считаются знаковыми, то есть

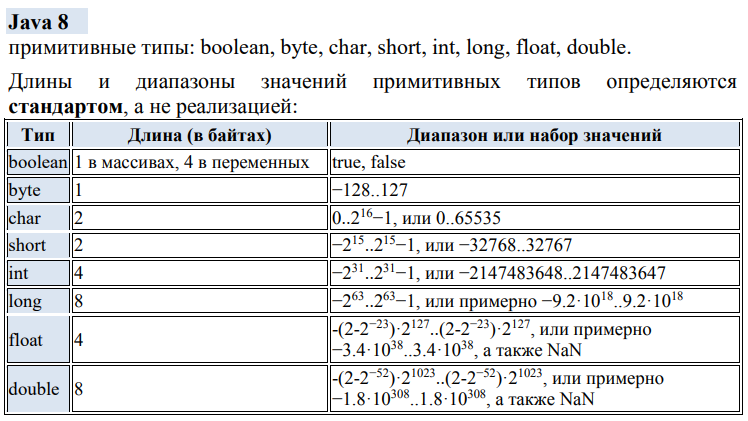
спецификатор signed можно опускать. Диапазон значений зависит от

реализации.

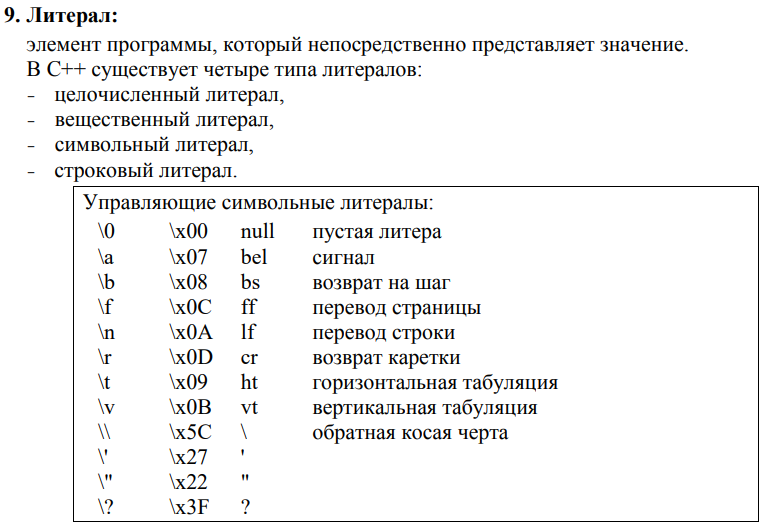
|  |  |
| --- | --- |
|  |  |
|  |  |

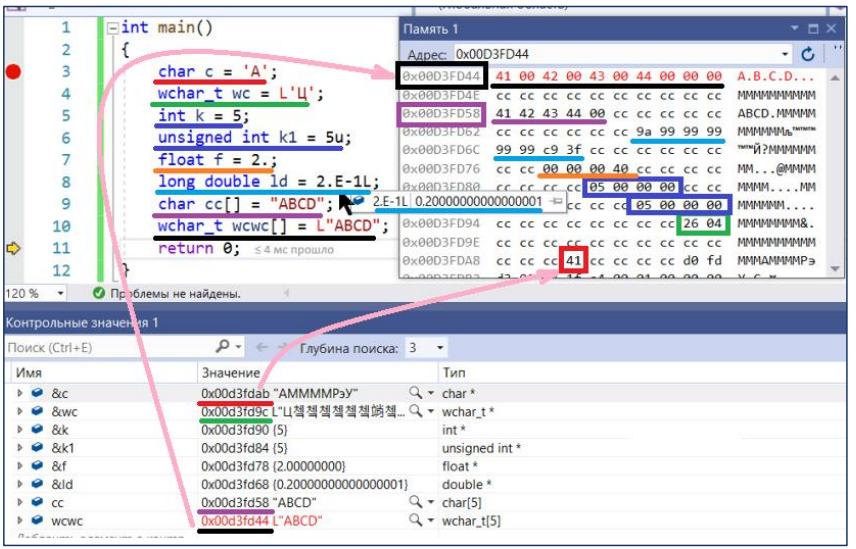
Два стандартных включаемых заголовочных файла, и , определяют числовые ограничения или минимальное и максимальное значения, которые может хранить переменная данного типа. Ограничения для некоторых целочисленных типов, заданные в стандартном файле заголовка , представлены в таблице:





**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)).

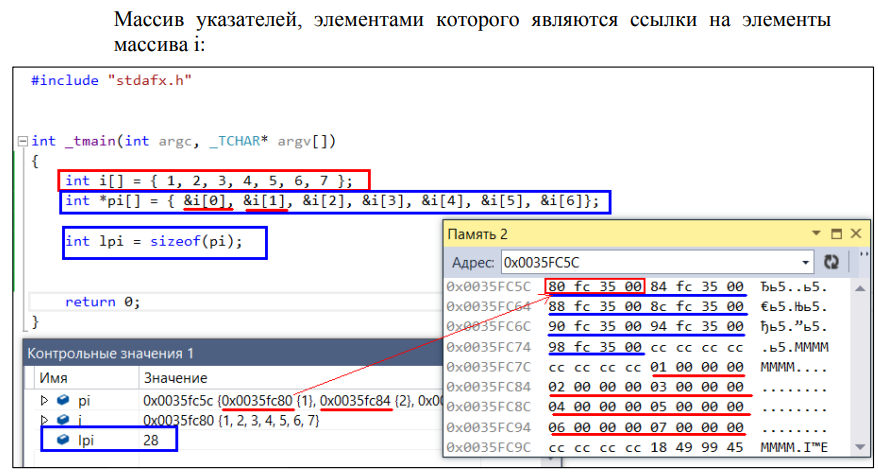
Массив (array) − это совокупность переменных, имеющих одинаковый тип и объединенных под одним именем. Доступ к отдельному элементу массива осуществляется с помощью индекса.

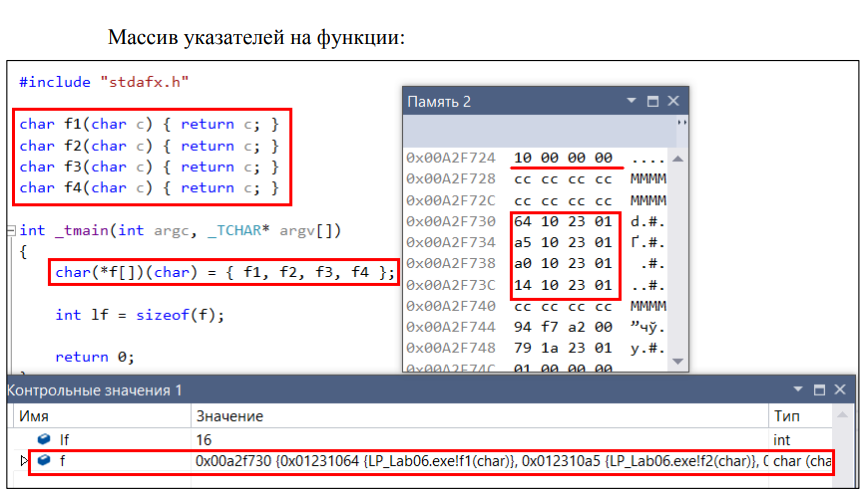
Объем памяти, необходимый для хранения массива, зависит от его типа и

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

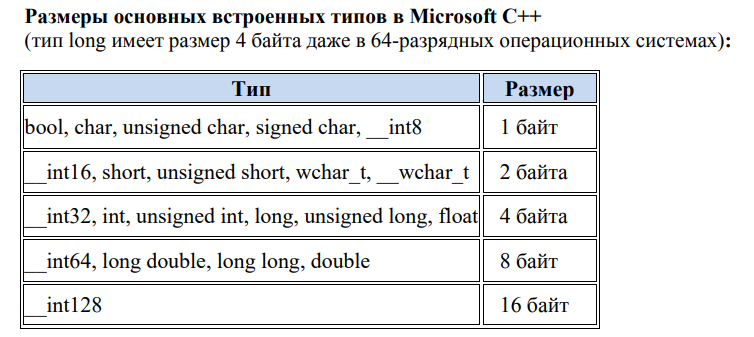
диапазон значений стандартных типов:

- 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);

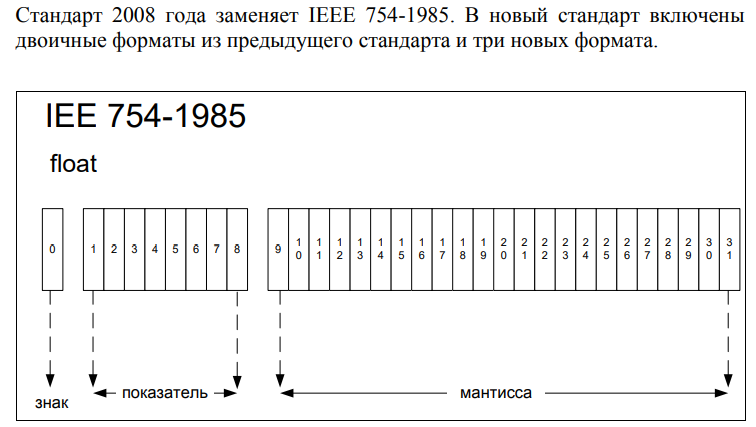
-методы, используемые для преобразования числа при выполнении

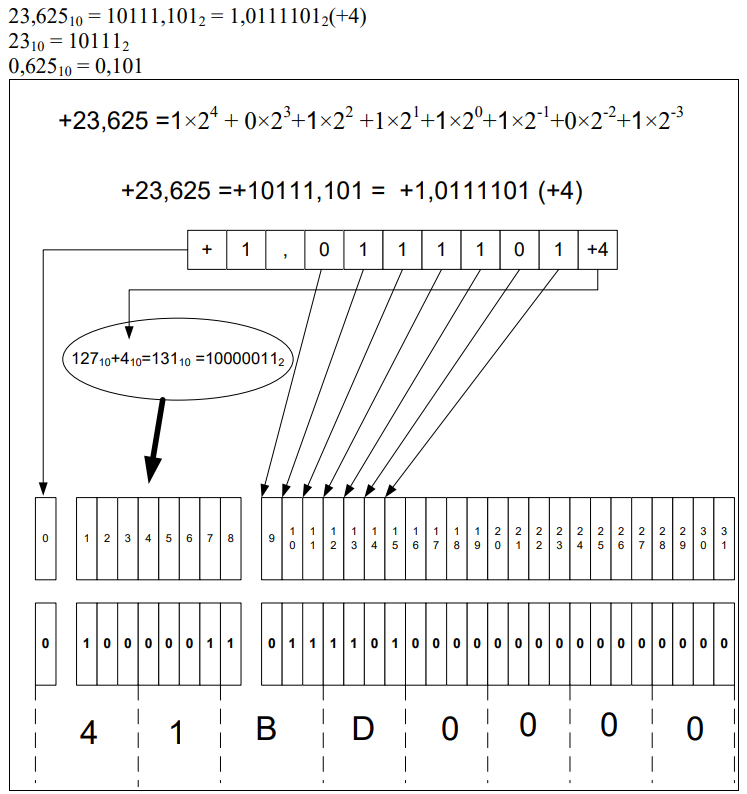
математических операций;

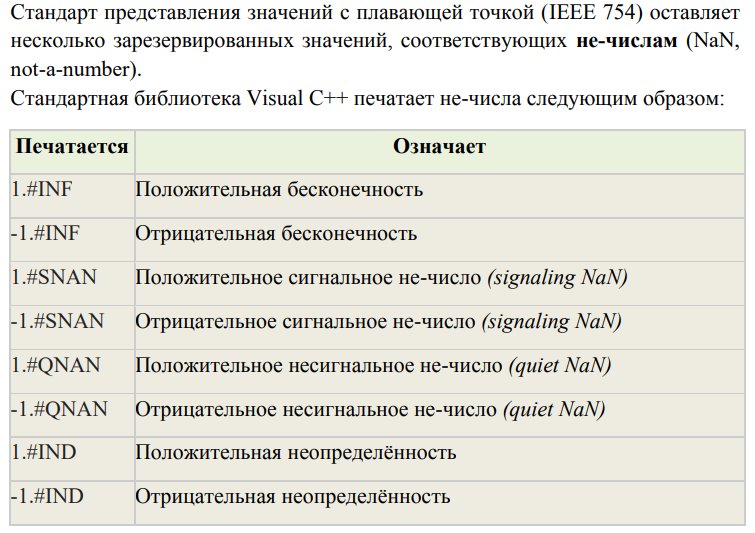
-исключительные ситуации: деление на ноль, переполнение, потеря

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

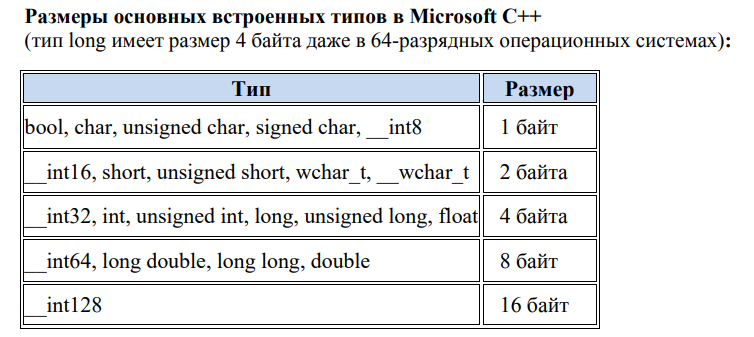
диапазон значений стандартных типов:

- 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.

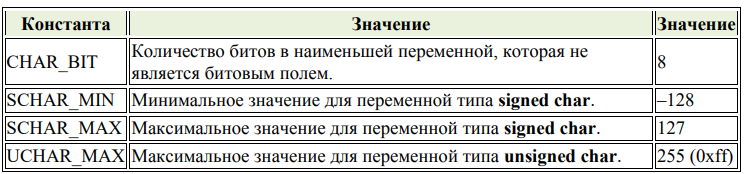
По умолчанию все целочисленные типы считаются знаковыми, то есть

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.

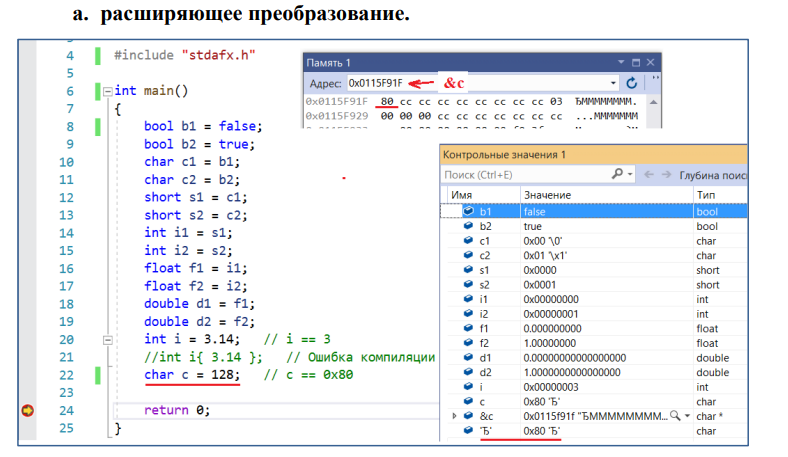
Если преобразование является сужающим, компилятор выдает предупреждение о

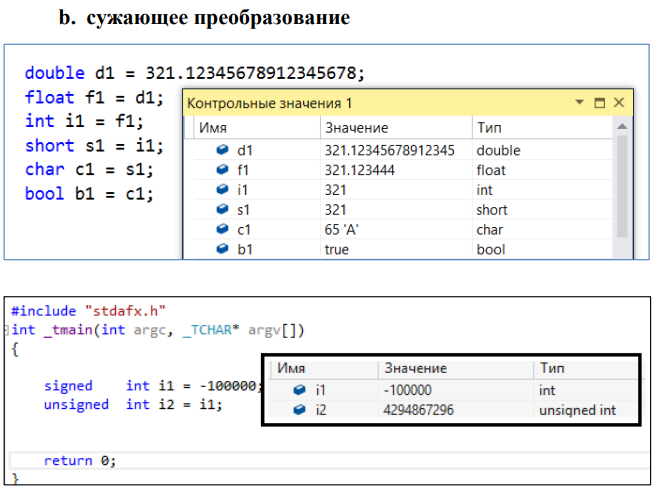
возможной потере данных.

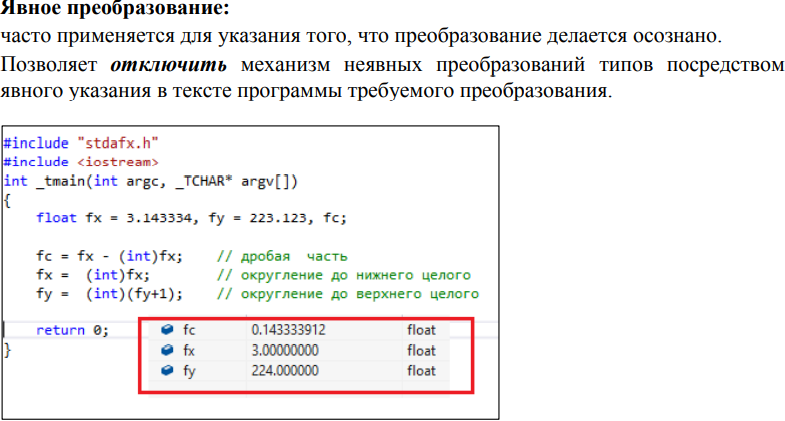
Происходит ли фактическая потеря данных, зависит от фактических значений.

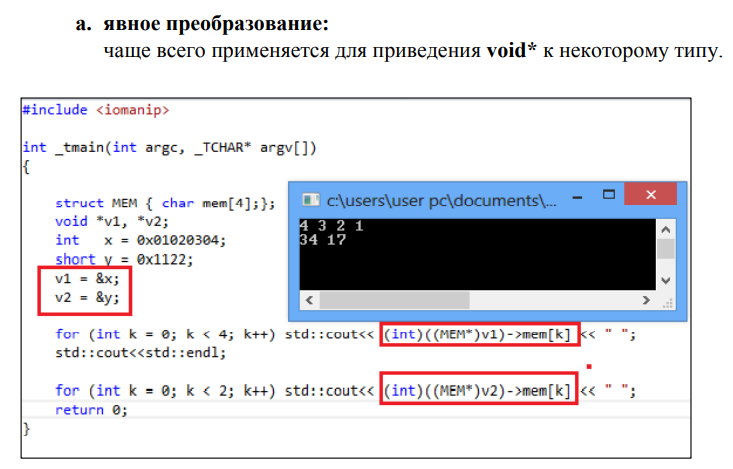
Рекомендуется рассматривать это предупреждение кок ошибку.

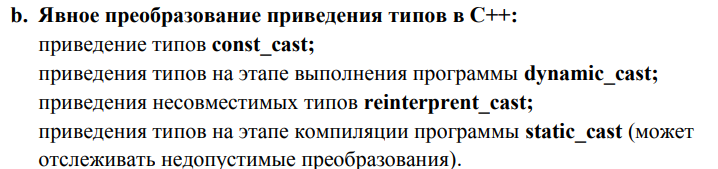
Если компилятору не удается найти допустимое преобразование, то выдается ошибка и объектный код не создается.











**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.

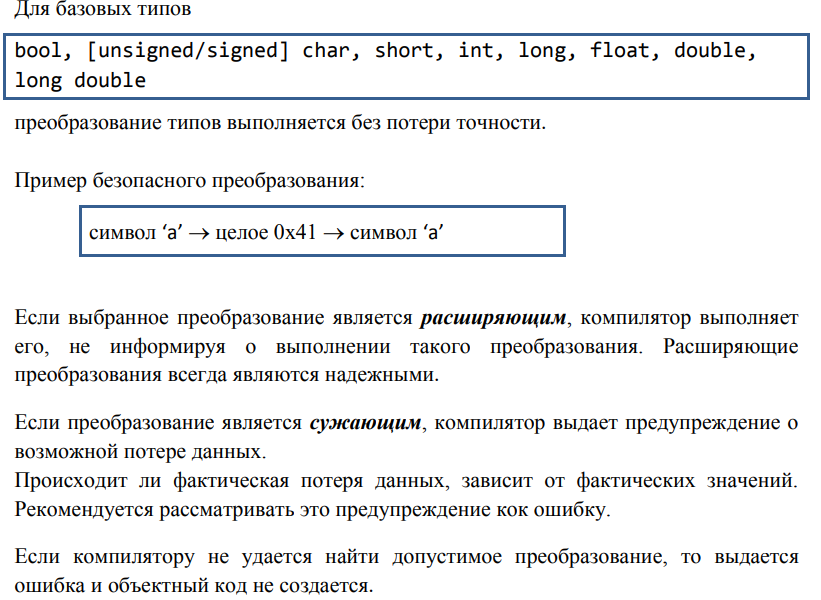
По умолчанию компилятор выравнивает элементы структуры, класса по значению размера:

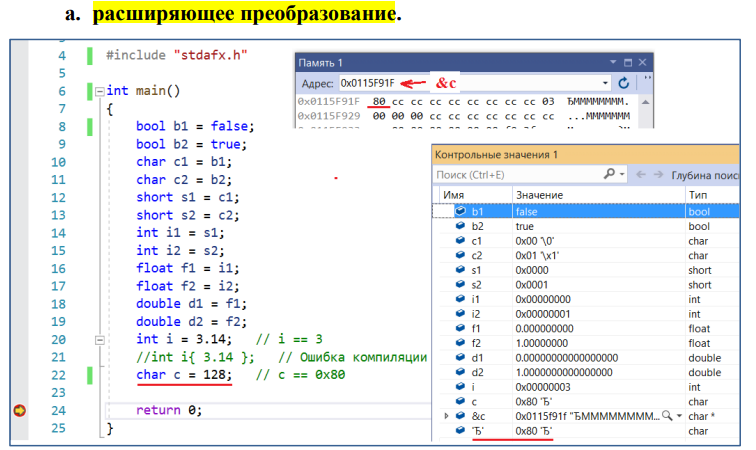
bool и char в однобайтовых границах;

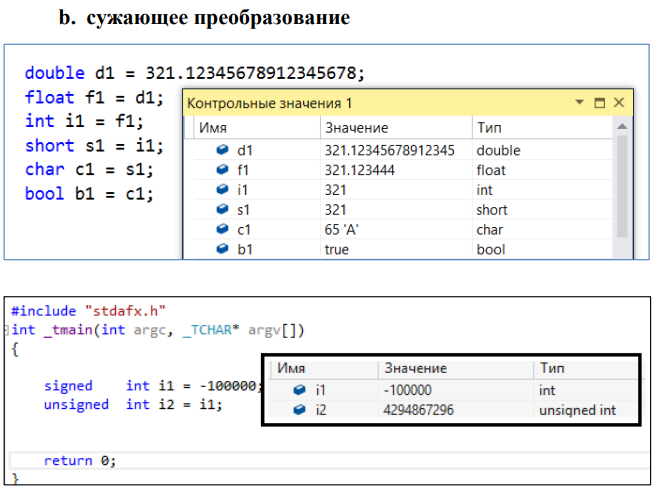
short на 2-байтовых границах;

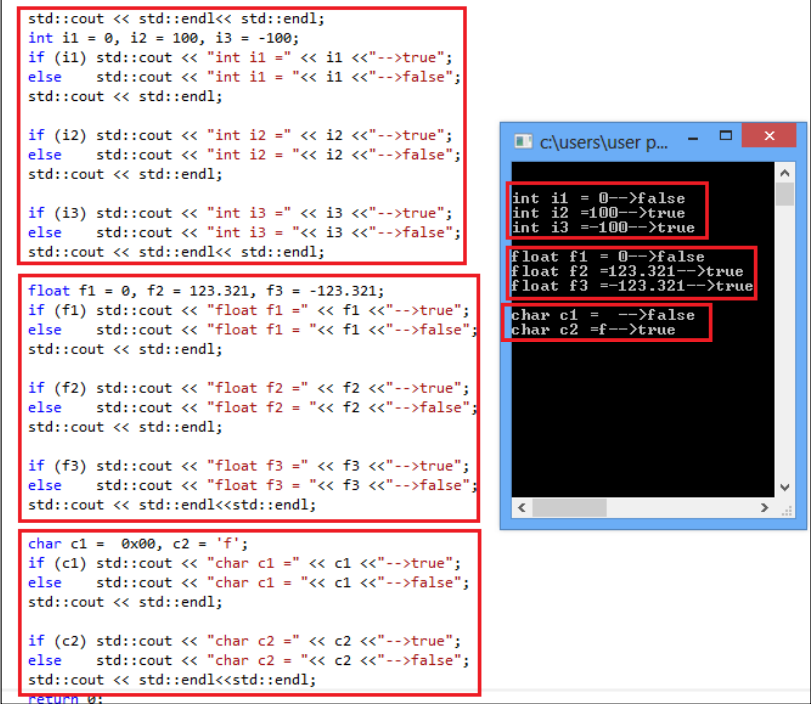
int, long и float в пределах 4-байтового диапазона;

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 «::».