Конспект составлен с помощью:

## Бьерн Страуструп. Язык программирования С++

## Стандарт С++: перевод, комментарии, примеры (ISO IEC 14882\_2017 draft n4659)

Модель памяти С++  
The С++ memory model

* Базовой единицей хранения в модели памяти С++ является **байт (byte)**
* **Байт** должен быть по крайней мере настолько велик, чтобы вмещать любой элемент базового набора символов выполнения и восьмибитные кодовые единицы кода Unicode UTF-8
* **Байт** образуется непрерывной (смежной) последовательностью **бит**, число которых определяется **реализацией**
* **Наименьший** значащий бит называется младшим разрядом **(low-order bit)**
* **Наибольший** значащий бит называется старшим разрядом **(high-order bit)**
* **Память**, доступная для программ на С++, состоит **из одной или более** последовательностей смежных **байтов**
* Каждый **байт** обладает **уникальным** адресом

**Область памяти** - это либо объект скалярного типа, либо максимальная

последовательность соседних битовых полей с не нулевой длиной.

Замечание. Различные свойства языка С++, такие как ссылки или виртуальные функции, могут включать дополнительные области памяти, которые недоступны для программы, но находятся под управлением реализации.

Пример:

struct {

char а;

int b:5,

c:11,

:0,

d:8;

struct {int ее:8;} е;

}

содержит четыре отдельных области памяти : поле а и битовые поля d и е . ее представляют отдельные области памяти и могу модифицироваться одновременно, не создавая помех. Битовые поля b и с вместе образуют четвертую область памяти . Битовые поля b и c не могут модифицироваться одновременно, но, например, поля b и a могут модифицироваться.

Объектная модель С++

**Конструкции** программы на С++ **создают** объекты, **уничтожают** их, **ссылаются** на объекты, **получают доступ** к объектам, **манипулируют** с объектами.

**Объект (object)** есть область памяти

**Замечание.** Функция не считается объектом , независимо от того, занимает ли она память таким же образом, как объекты

Объект создается посредством его описания, посредством выполнения

**new**-выражения или самой реализацией, когда это необходимо

**Свойства** объекта определяются **при создании** объекта

**Объект** может иметь **имя**

**Объект** характеризуется **классом** памяти, который **влияет на его время жизни**

**Объект** имеет **тип**

**Термин тип объекта** (или объектный тип, object type) относится к типу, с которым данный объект был создан.

Некоторые объекты являются полиморфными (polymorphic);

Реализация генерирует информацию, связанную с каждым таким объектом, что делает возможным во время выполнения программы определять тип этого объекта.

Интерпретация значений других (не полиморфных) объектов определяется типом выражения, которые используются для доступа к ним

Объекты могут содержать другие объекты, называемые подобъектами (sub objects)

Подобъект может представлять собой подобъект-член (member sub object), подобъект базового класса (base class sub object) или элемент массива.

Объект, который не является подобъектом никакого другого объекта, называется полным объектом (complete object)

Для любого объекта х имеется некоторый объект, называемый полным объектом для х (complete object of х), который определяется следующим

образом:

* Если х - полный объект, то сам х считается полным объектом для х
* В противном случае, полный объект для х - это полный объект того (уникального) объекта, который содержит х.

Если полный объект, данное-член или элемент массива имеет классовый тип, то этот тип называется самым производным (most derived) классом, чтобы отличать его от классового типа любого подобъекта базового класса;

Объект самого производного классового типа называется самым производным объектом (most derived object).

**Самый производный** объект, если это не битовое поле, должен иметь **ненулевой размер** и должен **занимать один или более** байтов памяти.

**Подобъекты** базовых классов **могут** иметь **нулевой** размер.

Объект **тривиального** типа или типа со стандартной компоновкой должен занимать **непрерывную последовательность байтов** памяти.

Если объект **не представляет** собой **битовое поле** или подобъект базового класса с ненулевой длиной, то **адресом** этого объекта **служит** адрес **первого** байта памяти, которую он занимает.

Два различных объекта, которые не являются ни битовыми полями, ни подобъектами базового класса с нулевой длиной, должны иметь **различные** адреса.

Пример.

static const char testl = 'х';

static const char test2 = 'Х';

const bool b = &testl != &test2; //всегда false

Литералы

**Виды литералов:**

Имеется несколько видов литералов:

* Целый литерал
* Символьный литерал
* Плавающий литерал
* Строковый литерал
* Булевский литерал
* Указательный литерал
* Литерал определенный пользователем
* **Целый литерал** (integer literal) есть последовательность цифр, не содержащая точки или экспоненты, с необязательными разделительными одиночными кавычками, которые игнорируются при определении значения целого литерала.

Целый литерал может иметь префикс, который задает его основание, и суффикс, который задает его тип.

Лексически первая цифра в последовательности цифр считается самой старшей.

* **Двоичный** (binary) целый литерал (по основанию два) начинается с 0Ь или ОB и состоит из последовательности двоичных цифр.
* **Восьмеричный** (octal) целый литерал (по основанию восемь) начинается с цифры 0 и состоит из последовательности восьмеричных цифр.
* **Десятичный** (decimal) целый литерал (по основанию десять) начинается с цифры, отличной от О, и состоит из последовательности десятичных цифр.
* **Шестнадцатеричный** (hexadecimal) целый литерал (по основанию шестнадцать) начинается с Ох или ОХ и состоит из последовательности шестнадцатеричных цифр, то есть из десятичных цифр и букв от а до f и от А до F, которые обозначают десятичные значения от десяти (ten) до пятнадцати (fifteen).
* Символьный литерал представляет собой один или несколько символов, заключенных в одинарные кавычки, например, 'х', возможно, предшествуемые одним из префиксов u8, u, u или L

**Например:**

**u8 'w‘**

**u 'х‘**

**u 'у‘**

**L 'z‘**

* Символьный литерал, который не начинается с префиксов u8, u, u или L, считается обычным символьным литералом. Обычный символьный литерал, который содержит единственный с-символ, представимый множеством символов выполнения, имеет тип char, а значение этого литерала есть число, равное коду с-символа в множестве символов выполнения.
* Обычный символьный литерал, который содержит более одного с-символа, называется многосимвольным литералом (multicharacter literal). Многосимвольный литерал или обычный символьный литерал, содержащий одиночный с-символ, который не представим множеством символов выполнения, поддерживается условно, имеет тип **int**, а его значение определяется реализацией.
* Символьный литерал, который начинается с префикса u8, например, u8 'w', представляет собой символьный литерал типа char и называется символьным литералом UTF-8.
* Значение такого литерала эквивалентно значению кода символа (code point value) согласно стандарту ISO 10646, при условии, что этот код может быть представлен одним кодовым значением в кодировке UTF-8.
* Если значение кода символа не может быть представлено в кодировке UTF-8, программа считается неправильной.
* Символьный литерал UTF-8, который содержит несколько с-символов, считается неправильным.
* Символьный литерал, который начинается с буквы (префикса) u, например, u 'х', считается символьным литералом типа char16\_t. Значение литерала типа char16\_t, содержащего единственный с-символ, равно значению кода символа согласно ISO 10646, при условии, что этот код может быть представлен одним шестнадцатибитовым кодовым значением (то есть, при условии, что символ принадлежит множеству базовых многоязыковых кодовых символов.)
* Если значение кода символа не может быть представлено шестандцатибитовым значением, программа считается неправильной.
* Литерал типа char16\_t, который содержит несколько с­ символов, считается неправильным.
* Символьный литерал, который начинается с буквы (префикса) u, например, u 'у', считается символьным литералом типа char32\_t.
* Значение литерала типа char32\_t, содержащего единственный с-символ, равно значению кода символа согласно ISO 10646. Символьный литерал типа char32\_t, содержащий несколько с-символов, считается неправильным.

Символьный литерал, который начинается с буквы (префикса) L, например, L 'z', считается широкосимвольным литералом.

Широкосимвольный литерал имеет тип wchar\_t

Значением широкосимвольного литерала, содержащего единственный с­ символ, служит значение, равное коду этого с-символа в множестве широких символов выполнения, если с-символ имеет представление в множестве широких символов выполнения;

В случае отсутствия такого представления значение широкосимвольного литерала определяется реализацией.

* **Замечание.** Тип wchar\_t в состоянии представлять все члены множества широких символов выполнения
* Значение широкосимвольного литерала, который содержит несколько с­ символов, определяется реализацией.
* **Плавающий литерал** состоит из целочисленной части, десятичной точки, дробной части, символа е или Е, целочисленной экспоненты (возможно, со знаком) и необязательного суффикса типа.
* Целая и дробная части плавающего литерала обе состоят из последовательности десятичных цифр (по основанию десять). Необязательные разделительные одиночные кавычки в конструкции посл­-цифр при определении значения плавающего литерала игнорируются.
* **Пример.** Литералы 1.602’176’565е-19 и 1.602176565е-19 имеют одно и то же значение.
* Либо целая часть, либо дробная часть (но не обе части одновременно) могут быть опущены; десятичная точка или буква е (или Е), либо экспонента (но не обе конструкции одновременно) могут быть опущены.
* Целая часть, необязательная десятичная точка и необязательная дробная часть образуют значащую часть (significant part) плавающего литерала.
* **Комментарий.** Дробная часть действительного числа в форме с плавающей точкой называется также **мантиссой .**
* Строковый литерал (string literal) - последовательность символов, окруженная двойными кавычками, перед которой может находиться префикс:
* R, u8, u 8R, u, u R, u, UR, L или LR
* **Например:** “…”, R”(…)”, u8”…”, u8R”\*\*(…)\*\*”, u”…”, uR”\*~(…)\*~, и т.д.
* Строковый литерал с префиксом R называется необработанным строковым литералом (raw string literal). Конструкция посл-d-символов служит разделителем. Завершающая посл-d-символов из конструкции необработанная-строка является той же самой последовательностью символов, что и начальная посл-d-символов. Конструкция посл-d-символов должна состоять не более чем из 16 символов.

Булевский литерал:

* **False**
* **True**

Булевские литералы представляются служебными словами **true и false**.

Эти литералы считаются pr-значениями и имеют тип **bool**.

Указательный литерал:

* **nullptr**

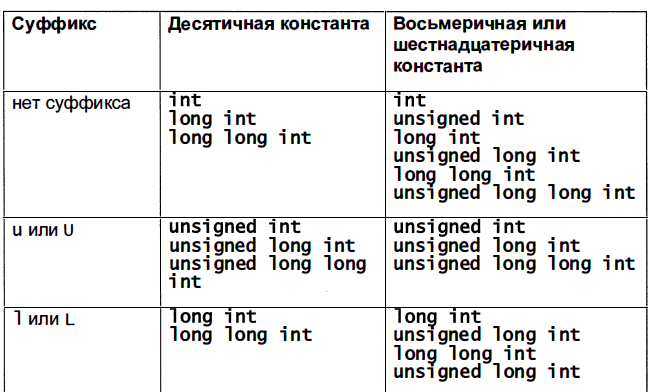
Указательный литерал представляется посредством служебного слова **nullptr**

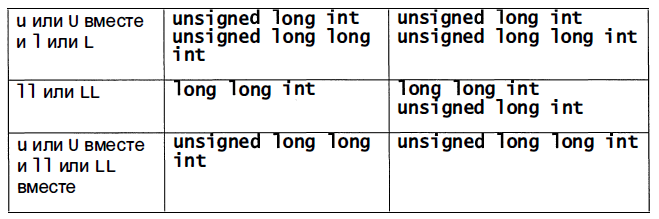
Он считается pr-значением типа **std**: :**nullptr**\_t.

**Замечание.** Тип **std**: :**nullptr**\_t - это отдельный тип, который не является ни указательным типом, ни типом указателя на член;

Скорее, pr-значение этого типа представляет собой нулевой константный указатель, и это значение может быть преобразовано в нулевое указательное значение или в нулевое значение типа указателя на член

**Типы целых констант**





Наследование конструкторов

Конструкция объявление-использования, которая именует конструктор, неявно объявляет множество наследующих конструкторов (inheriting constructors).

Множество наследуемых конструкторов-кандидатов (candidate set of inherited constructors) из класса Х, именумого в объявлении-использования, состоит из реальных конструкторов и «воображаемых» (notional) конструкторов

Это множество образуется путем трансформации параметров с умолчаниями и параметров, обозначенных многоточием, следующим образом:

* для каждого нешаблонного конструктора класса Х,- конструктор, который образуется удалением параметра-многоточия , а также
* для каждого нешаблон ного конструктора класса Х, который имеет хотя бы один параметр с аргументом по умолчанию,- множество конструкторов, образуемое путем удаления спецификации параметра­ многоточия и последовательного удаления параметров с аргументами по умолчанию из конца списка-типов-параметров, а также
* для каждого шаблона-конструктора класса Х,- шаблон-конструктор, который образуется удалением параметра-многоточия а также

Характеристики конструкторов для конструкторов и конструкторов-шаблонов

* Список параметров шаблонов, если он есть
* Список типов параметров
* Наличие или отсутствие спецификатора explicit

Каждый нешаблонный конструктор из множества наследуемых конструкторов-кандидатов, отличный от конструктора без параметров или от конструктора копирования/перемещения, имеющего единственный параметр, считается неявно-объявленным с теми же самыми характеристиками конструктора, если только в полном классе, где появляется объявление­ использования, не имеется конструктора, объявленного пользователем, с такой же сигнатурой, либо конструктор для этого класса был бы конструктором умолчания, конструктором копирования или перемещения.

Аналогично, каждый конструктор-шаблон из множества наследуемых конструкторов-кандидатов, считается неявно-объявленным с теми же самыми характеристиками конструктора, если только в полном классе, где появляется объявление-использования, не имеется эквивалентного объявленного пользователем конструктора-шаблона.

**Комментарий.**

Со списком-типов-параметров есть некоторая неясность. Это не синтаксическая категория (её нет в синтаксисе), и нигде, в том числе, нет определения этого понятия. Интуитивно ясно, о чем идет речь (это просто список типов, возможно, в виде идентификатора-типа), но явно это понятие нигде не вводится…

**Замечание.** Аргументы по умолчанию не наследуются.

Конструктор, объявленный таким образом, имеет тот же самый доступ, что и соответствующий конструктор в классе Х.

Он считается имеющим спецификатор constexpr, если пользовательский (user-written) конструктор мог бы удовлетворять требованиям на constexpr-конструкторы.

Он считается удаленным, если соответствующий конструктор из Х является удаленным, либо если умалчиваемый конструктор умолчания мог бы быть удаленным, за исключением того обстоятельства, что конструирование непосредственного базового класса Х в определении этого факта не рассматривается .

* Наследующий конструктор не должен быть явно настроенным или явно специализированным.

Замечание. Конструкторы умолчания и копирования/перемещения могут быть объявлены неявно.

Множество наследуемых конструкторов-кандидатов из dl для 81 состоит из следующих конструкторов:

* b1(const b1&)
* b1(b1&&)
* B1(int)

Множество конструкторов, присутствующих в dl, состоит из следующих:

* dl(), неявно-объявленный конструктор умолчания; в случае его оdг-использования программа считается неправильной
* dl (const dl&), неявно-объявленный конструктор копирования, не унаследованный
* dl (dl&&), неявно-объявленный конструктор перемещения, не унаследованный
* dl (int), неявно-объявленный наследующий конструктор

Множество наследуемых конструкторов-кандидатов из d2 для b2 состоит из следующих конструкторов:

* b2 (const b2&)
* b2 (b2&&)
* b2 (int = 13, int = 42)
* b2 (int = 13)
* B2()

Множество конструкторов, присутствующих в d2, состоит из следующих:

* d2 () , неявно-объявленный конструктор умолчания, не унаследованный
* d2 (const d2&), неявно-объявленный конструктор копирования, не унаследованный
* d2 (d2&&), неявно-объявленный конструктор перемещения, не унаследованный
* d2 (int , int), неявно-объявленный унаследованный конструктор
* d2 (int), неявно-объявленный унаследованный конструктор

Конец примера

Замечание. Если два объявления-использования объявляют унаследованные конструкторы с одной и той же сигнатурой, программа считается неправильной, так как неявно-объявленный конструктор, введенный первым объявлением-использования, не считается конструктором, объявленным-пользователем, и тем самым не устраняет другое объявление конструктора с такой же сигнатурой из второго объявления-использования.

Наследующий конструктор класса считается неявно-описанным, когда он оdr­ используется для создания объекта его классового типа.

Неявно-описанный наследующий конструктор выполняет множество инициализаций класса, которые выполнялись бы написанным пользователем подставляемым (inlinе) конструктором этого класса со списком-тет­ инициализаторов, единственный тет-инициализатор которого имеет идентификатор-тет-инициализатора, именующий базовый класс, указанный в спецификаторе-вложенного-имени из объявления-использования, и имеет список-выражений, структура которого описана ниже (в этом абзаце), и имеет пустой составной-оператор тела функции.

Если такой написанный пользователем конструктор оказался бы неправильным, программа считается неправильной. Каждое выражение из списка-выражений имеет вид static\_cast<T&&> (p) , где р - имя соответствующего параметра конструктора, а т - тип, с которым объявлен р.

Классы

Класс считается типом. Его имя становится именем-класса (9. 1 ) в пределах

его области действия .

Формирование имени-класса производится с помощью спецификатора­ класса и уточненного-спецификатора-типа (7. 1 .6.3). Объект класса состоит из (возможно пустой) последовательности членов и объектов базовых классов.

Имя-класса вводится в область действия, в пределах которой оно объявлено, непосредственно после того, как встретилось это имя-класса. Кроме того, имя-класса вводится в область действия самого класса; такое имя называется внедрённым-именем-класса (injected-class-name ).

Класс считается описанным после того, как встретилась заключительная фигурная скобка спецификатора-класса, даже если функции-члены ещё вообще не описаны. Необязательная посл-спецификаторов-атрибутов относится к классу в целом; атрибуты из посл-спецификаторов-атрибутов далее рассматриваются как атрибуты класса, как бы класс впоследствии ни именовался.

Если класс помечен вирт-спецификатором-класса final, и он появляется в спецификаторе-базового-типа в разделе-баз, то программа считается неправильной . Везде, где после конструкции служебное-слово-класса следует конструкция имя-заголовка-класса, идентификатор final и двоеточие или левая фигурная скобка, идентификатор интерпретируется как виртуальный спецификатор класса.

Пример.

struct А;

struct А final { };

//ОК: инициализация - значением переменной final struct X { };

struct C {constexpr operator int () { return 5; } };

struct B final : C { };

//ОК: описание вложенного класса в, но не объявление

//битового поля final Конец примера.

Объединение (union) есть класс, описанный со служебным-словом-класса union;

в каждый момент времени он содержит только один член.

**Комментарий.** Для новичка это звучит не слишком понятно. Если

говорить чуть проще, то в классе со служебным словом union может

быть задано несколько членов, как и в «обычных» классах, однако любой экземпляр объединения в каждый момент выполнения может содержать только один член из заданных.

Объединения - низкоуровневый механизм, оставшийся в С++ в

наследство от своего предшественника, языка С. Объединения могут

использоваться на нижних уровнях программной системы для реализации нестандартных схем выделения памяти.

Тривиально-копируемый класс (trivially copyable class) - это класс, который:

* не имеет нетривиальных конструкторов копирования
* не имеет нетривиальных конструкторов перемещения
* не имеет нетривиальных операций присваивания копированием
* не имеет нетривиальных операций присваивания перемещением
* имеет тривиальный деструктор

Тривиальным классом (trivial class) считается класс, который имеет конструктор умолчания, не имеет нетривиальных конструкторов умолчания и является тривиально-копируемым.

**Замечание.** В частности , тривиально-копируемый или тривиальный класс не имеет виртуальных функций и виртуальных базовых классов.

Классом со стандартной компоновкой (standard-layout class) считается класс S, для которого выполняются следующие условия:

* в классе нет нестатических данных-членов, тип которых - классовый тип с нестандартным расположением (или массив элементов такого типа), или ссылка,
* в классе нет виртуальных функций и виртуальных базовых классов
* все нестатические члены-данные класса имеют одни и те же права доступа
* класс не имеет базовых классов с нестандартной компоновкой
* класс имеет самое большее один подобъект базового класса любого данного типа
* все нестатические данные-члены класса, все битовые поля и все его базовые классы впервые объявлены в том же самом классе, и класс не имеет в качестве базового класса ни одного класса из множества типов M(S) - это множество определяется ниже
* Структурой со стандартной компоновкой (standard-layout struct) считается класс со стандартной компоновкой, описанный со служебным­ словом-класса struct или со служебным-словом-класса class.
* Объединением со стандартной компоновкой (standard-layout union) считается класс со стандартной компоновкой, описанный со служебным-словом-класса union.
* Замечание. Классы со стандартной компоновкой полезны для взаимодействия с программами на других языках программирования.
* РОD-структура - это класс, который не является объединением и который одновременно является тривиальным классом и классом со стандартной компоновкой, а также не содержит нестатических членов-данных с типом не-РОD-структуры, не-РОD-объединения (или массивов таких типов).
* Аналогично, РОD-объединение (POD union) - такое объединение, которое является одновременно тривиальным классом и классом со стандартной компоновкой, а также не содержит нестатических членов-данных с типом не­ РОD-структуры, не-РОD-объединения (или массивов таких типов).
* РОD-классом (POD class) называется класс, который является либо РОD­ структурой, либо РОD-объединением.

Структура программы С++

* Каждая подпрограмма имеет структуру, подобную функции main();
* Каждая программа содержит одну или несколько функций;
* Каждая функция содержит 4 основных элемента:
* 1. тип возвращаемого значения Int
* 2. имя функции main()
* 3. список параметров, заключённый в фигурные скобки {return 0;}\*
* 4. тело функции

\* эта строка значит "*вернуть операционной системе в качестве сигнала об успешном завершении программы значение 0*"

Организация консольного – ввода/вывода данных

#include <iostream>; *//директива процессора, предназначена для включения в исходный текст содержимое заголовочного файла<iostream>, содержащий описания функций стандартной библиотеки для работы с клавиатурой и экраном.*

using namespace std; *//директива означает, что все определённые ниже имена будут относиться к пространству имён std*

Int main() *//имя функции, которая не содержит параметров и должна возвращать значение типа Int*

{Int a,b; *//объявление двух переменных типа Int - целый тип*

cout <<”введите два целых числа”<<endl; *//оператор вывода данных на экран ,<< - операция помещения данных в выходной поток; endl**- манипулятор, переводит сообщение на новую сточку.*

cin>>a >>b; *//оператор ввода данных с клавиатуры,>> - операция для извлечения данных из выходного потока, читает значения из cin и сохраняет их в переменных.*

cout <<”их сумма равна”<<a+b; *//оператор вывода*

return 0;} *//возвращаемое значение для ОС*

Стандартные типы данных

* Целые типы данных – **short, int, long** и спецификаторы ***(signed,unsigned);***
* Вещественные типы – **float, double, long double;**
* Cсимвольные типы – **char, wchar\_t;**
* Логический тип – **bool**, принимающий значения (true-истина, false-ложь);

**Константы (const)**

* + *a=b+2,5* – неименованная константа;
  + *«1L»* – целочисленный литерал (тип long);
  + *«8»* – целочисленный литерал (тип Int);
  + *«f»* – символьный литерал
  + ***«\n»*** – конец строки
* **Формат описания именованной константы:**
* [<***класс памяти***>] const<тип><имя именованной константы>=<***выражение***>;
* const int l = -124;
* const float k1 = 2,345

***Класс памяти*** *-*это спецификатор, определяющий время жизни и область видимости данного объекта.

***Выражение*** - определяет значение именованной константы, т.е инициализирует её.

Переменные

Формат описания переменных:

**[<класс памяти>]<тип><имя>[=<выражение> | (<выражение>)];**

*Пример*:

int I,j;

double x;

Значение переменных должно быть определено с помощью:

1. оператора присваивания: int a; *//описание переменной*

int = a; *//определение значения переменной*

2. оператора ввода: int a; *//описание переменной*

cin >> a; *//определение значения переменной*

3. инициализация – определение значения переменной на этом этапе описания.

int i = 100; *//инициализация копией*

int i (100); *//прямая инициализация*

Управление форматом вещественных типов данных

Существует три аспекта оформления значения с плавающей запятой которыми можно управлять:

* + точность( кол-во отображаемых цифр), изменяется с помощью манипулятора setprecision
  + форма записи (десятичная или экспонециальная)
  + указание десятичной точки для значения с плавающей запятой, являющихся целыми числами

\*Для использования манипуляторов endl с аргументами требуется подключить заголовочный файл iomanip)

Пример:

#include <iostream>

#include <iomanip>\*

using namespace std;

int main()

{

double i=12345,6789;

cout << setprecision(3)<<i<<endl;

cout << setprecision(6)<<i<<endl;

cout << setprecision(9)<<i<<endl;

refurn 0;

}

//Результат работы программы:

//1.23e+004

//12345.7

//12345.6789

Управление размещение данных на экране

* **Используются манипуляторы**:

1. lief – выравнивает вывод по левому краю;

2. right – выравнивает вывод по правому краю;

3. internal – контролирует размещение отрицательного значения: выравнивает знак по левому краю, а значение по правому, заполняя пространство между ними пробелами;

4. setprecision(int w) – устанавливает max кол-во цифр в дробной части для вещественных чисел;

5. setw(int w) – устанавливает max ширину поля вывода;

Пример:

#include <iostream>

#include <iomanip>

using nanespace std;

int main()

{

cout <<“1” <<setw(10) <<“Ivanov” <<endl;

cout <<“2” <<setw(10) <<lief<<“Ivanov” <<endl;

cout <<“3” <<setw(10) <<right<<“Ivanov” <<endl;

return 0;

}

Операции. Унарные операции

* ***Операции увеличения (декремента) и уменьшения (инкремента)***

*на 1(++ и --);*

Записываются в двух формах:

* *Префикси*я – операция записывается перед операндом и увеличивает свой операнд на 1 и возвращает изменённое значение как результат
* *Постфиксия* – операция записывается после операндом, уменьшает свой операнд на 1 и возвращает изменённое значение как результат

Пример:

#include <iosteram>

int main()

using nanespace std;

{

int x=3, y=4;

cout <<++x<<“\t”<<--y<<endl;

cout <<x++<<“\t”<<y--<<endl;

cout <<x<<“\t”<<y<<endl;

return 0;

}

Операции отрицания (-,!)

* (-) - унарный минус – изменяет знак операнда целого или вещественного типа на противоположный;
* (!) – логическое отрицание, даёт в результате значение 0(ложь), если операнд отличен от 0(истина),если равен операнд 0 (ложь)
* тип операнда может быть любой

Пример:

#include <iosteram>

int main()

using nanespace std;

{

int x=3, y=0;

bool f=false, v=true;

cout <<-x<<“\t”<<!y<<endl;

cout <<-y<<“\t”<<!y<<endl;

cout <<v<<“\t”<<!v<<endl;

return 0;

}

Бинарные операции

* Арифметические операции:
  + умножение(\*),
  + деление(/),
  + остаток от деления(%),
  + сложение(+),
  + вычитание(-)

- Операции отрицания (-,!) унарный минус – изменяет знак операнда целого или вещественного типа на противоположный.

- Операции отношения: (<, <=, >, >=, == !=), меньше, меньше или равно, больше, больше или равно, равно, не равно, не равно соответственно)

Результатом операций являются значения true, false

Рассмотрим операции деления и остаток от деления:

*#include <iosteram>*

using nanespace std;

int main()

{

cout <<100/24<<“\t”<<100/24<<endl;

cout <<100/21<<“\t”<<100,0/24<<endl;

cout <<21%3<<“\t”<<21%6<<“-21%8”<<endl;

return 0;

Логические операции (&& и ||)

И (&&) - возвращает значение истина тогда и только тогда, когда оба операнда принимают значение истина, в противном случае операция возвращает значение ложь.

ИЛИ || - возвращает значение истина тогда и только тогда, когда хотя бы один операнд принимает значение истина, в противном случае –ложь.

* логические операции выполняются слева направо;
* приоритет операции && **выше** ||

Пример:

#include <iosteram>

using namespace std;

int main()

{

cout <<‘x\t y\t &&\t||’endl;

cout <<“0\t 1\t”<<(0 && 1)<<‘\t’<<(0||1)endl;

cout <<‘0\t 1\t’<<(0 && 1)<<\t’<< (0||1)endl;

cout <<‘1\t 0\t’<<(1 && 0)<<‘\t’<<(1||0)endl;

cout <<‘1\t 1\t’<<(1 && 1)<<‘\t’<<(1||1)endl;

return 0;

}

Операции присваивания

* формат операция простого присваивания (=):
* опреанд\_1 = операнд\_2

*пример*: a=b=c=100, это выражение выполняется справа налево, результатом выполнения с=100, является число 100, которое затем присвоиться переменной b, потом а.

* *Сложные операции присваивания*:
* (\*=) – умножение с присвоением
* (/=) – деление с присвоением
* (%=) – остаток от деления с присвоением
* (+=) – сложение с присвоением
* (-=) – вычитание с присвоением

ПРИМЕР: к операнду \_1 прибавляется операнд\_2 и результат записывается в операнд\_2

*т.е*. с = с + а, тогда **компактная запись с += а**

Операторы С++

Программа на языке С++ состоит из последовательности операторов, каждый из них определяет значение некоторого действия

Все операторы разделены на 4 группы:

* + операторы следования;
  + операторы ветвления;
  + операторы цикла;
  + операторы передачи управления.

Операторы ветвления

К ним относятся:

* *условный оператор* ***if***
* *оператор выбора* ***switch***

**Они** позволяют изменить порядок выполнения операторов в программе

Оператор выбора *switch*

Предназначен для разветвления процесса вычислений на несколько направлений

* **Выражение**, стоящее за ключевым словом ***switch****,* должно иметь арифметический тип или тип указатель.
* Все константные выражения должны иметь разные значения, но совпадать с типом выражения, стоящим после ***switch***
* Ключевое слово ***case***и расположенное после него константное выражение называют также меткой ***case***
* **Формат оператора:**

switch (<выражение>)

{

case <константное\_выражение\_1>: [<оператор 1>]

case <константное\_выражение\_2>: [<оператор 2>]

……………………………………………………………

case <константное\_выражение\_n>: [<оператор n>]

[default: <оператор> ]

}