1.1

Определение функции содержит четыре элемента:

* Тип возвращаемого значения (return type)
* Имя функции (function name)
* Список параметров (parametr list)
* Тело функции (function body)

Прим: функция main должна иметь тип возвращаемого значения int

Тело функции представляет собой блок операторов (block of statement)

Когда return получает значение, его тип должен быть совместим с типом возвращаемого значения функции

**Тип данных** определяет содержимое элемента данных и какие операции возможны с ним.

return 0 в main() свидетельствует об успехе; любое другое число, возвращаемое функцией означает отказ, а само значение указывает на его причину.

1.2

Основу **iostream** составляют два типа данных: istream и ostream (input & output)

* cin – стандратный ввод
* cout – стандартный вывод
* cerr – стандартная ошибка (для создания предупреждений и ошибок)
* clog – для создания информационных сообщений (логирование)

**Поток** – это последовательность символов, записываемая или читаемая из устройства ввода-вывода некоторым способом.

#include <iostream> :

* #include - директива препроцессору
* <iostream> **-** заголовок
* Заголовок и директива должны находится на одной строке
* Располагается вне тела функции

Выражение содержит один или несколько **операндов** и, как правило, **оператора**.

std::cout << "Enter thought numbers: "<< std::endl;

<< **-** оператор. Он получает два операнда:

1. левый операнд должен быть объектов ostream – std::cout
2. Правый операнд **–** это подлежащее отображению значение

Поскольку оператор возвращает свой левый операнд, результат первого оператора становится левым операндом второго оператора (<<)

endl – специальный оператор (также называемый манипулятором), который нужен для сброса буффера и перехода на новую строну. Сброс буффера немедленно записывает весь вывод, который программа сформировала, в поток вывода 🡪 данные не будут ожидать записи, занимая лишнюю память.

# Пространство имен

Пространтсва имен нужны для того, чтобы избежать вероятных конфликтов, когда возникают случаи, в которых в двух разных библиотеках используется одни и те же названия.

Все имена, определенные в стандартной библиотеке, принадлежат пространству имен std

:: - оператор области видимости

std::cout – используется имя *cout* пространства имен *std*

# Классы

Класс определяет тип данных и набор операций, связанных с этим типом.

Каждый класс является определением типа. Имя типа совпадает с именем класса, т.е. класс Class определен как тип Class.

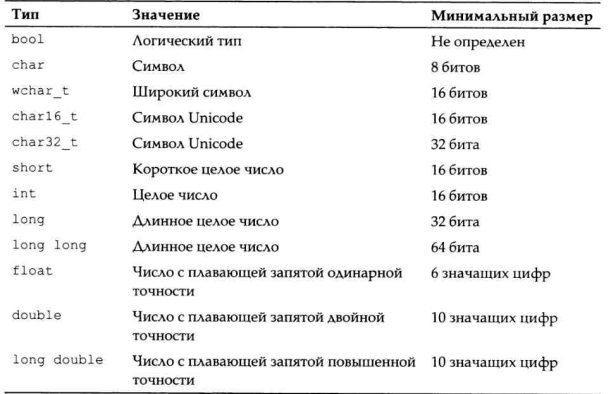
Можно создать переменную класса, например, Class item;

Функция-член – это функция, определенная в составе класса. Функции-члены называют также методами класса.

item1.isbn():

* item1 – объект класса
* isbn – метод класса или член-функция
* () – оператор обращения

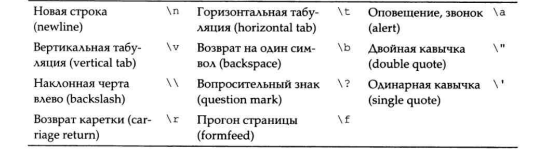
# Переменные и базовые типы



Какой тип использовать?

* Используйте беззнаковый тип, когда знаете, что значения не могут быть отрицательными
* Использовать int для целочисленной арифметики (или long long, когда места мало)
* Не использовать char и bool в арифмитических выражениях. Только хранение
* Использовать double, а не float. У float обычно недостает точности, а различия в затраттах на вычисления нехначительны. (long double не нужен)

Управляющая последовательность - ?



Переменная – именованное хранилище, которым могут манипулировать программы.

У каждой переменной есть тип. Тип определяет размер и расположение переменной в памяти. а также диапазон значений, которые моугут храниться в ней, и набор применымых к переменной операций. Переменная == объект

Определение переменной состоит из спецификатора типа и одного или нескольких имен переменных, отделенных запятыми.

Объект – область памяти, для которой указан тип

Инициализация - это выделение памяти для переменной при её создании, отталкиваясь от заданного типа (т.е. как бы выделяет контейнер)

присвоение - это замена содержимого в контейнере

4 способа инициализации переменной:

1. int var = 0;
2. int var = {0};
3. int var(0);
4. int var{0};

long double id = 3.141592

int a{id},b={id}; // не скомпилируется

int c(id), d = id; // скомпилируется

Компилятор откажет в инициализации переменных а и б, поскольку использование значения типа long double для инициализации переменной int может привести к потере данных (дробная часть будет усечена).

std::string empty; // неявно инициализируется пустой строкой

Sales\_item item; // Объект item инициализируется значением по умолчанию

Некоторые классы требуют, чтобы объект был инициализирован явно (иначе ошибка)

Если инициализировать переменные неявно глобально, то их значение по умолчанию будет 0 (или пустая строка в случае с string), а если в теле функции, то они будут неинициализированными.

**Указатели**

Указатель – это составной тип, переменная которого указывает на объект другого типа.

Используется для косвенного доступа к другим объектами.

В отличии от ссылок, указатель – это настоящий объект.

Указатели могут быть присвоены и скопированы.

Один указатель за время своего существования может указывать на несколько разных объектов.

Указатель можно не инициализировать в момент определения. Его значение будет неопределенно.

int \*p1, \*p2; // p1,p2 – указатели на тип int

double dp, \*dp2 // dp,\*dp2 – переменная и указатель на тип инт

Получение адреса объекта

Указатель содержит адрес другого объекта. Для получения адреса объекта используется оператор обращения к адресу (address-of operator), или оператор &.

int val = 45;

int \*p =&val; // p содержит адрес переменной val, p – указатель на пер.val

Второй оператор определеяет p как указатель на тип int и инициализирует его адресом объекта val типа int.

Указатель на ссылку невозможен, поскольку у вторых нет адреса.

Тип указателя и тип объекта, на который он ссылается, должны совпадать.

double dval;

double \*pd=&dbal; // ок

double \*pd2=pd; // ок

int \*pi = pd; // тип pi и pd отличаются

pi = &dval; // присвоение адреса типа double указателю на тип int

Значение указателя

Хранимое в указателе значение (адрес) может находится в одном из четырех состояний:

1. Оно может указывать на объект
2. Оно может указывать на область непосредственно за концом объекта
3. Это может быть нулевое значение, означающее, что данный указатель не связан ни с одним объектом
4. Оно может быть недопустимо. Любое иное значение, кроме приведенного выше, является недопустимым

Копирование или иная попытка доступа к значению п недопустимому указателю является серьезной ошибкой.

Когда указатель указывает на обхект, для доступа к этому объекту можно использовать оператор обращения к значению (deference opertor) или оператор \*

int val = 34;

int \*p = &val; // p содержит адрес val

std::cout<< \*p; // \* возвращает объект, на который указывает p

\*p = 0; // \* возвращает объект; присваивается новое значение val через указатель p

cout << \*p; // выводит 0. Если вывести без \*, то выдаст адрес объекта val

При присвоении значения \*p оно присвается **объекту**, на кторый указывает указатель p.

Обратиться к значению можно только по допустимому указателю, который указывает на объект.

int i = 42;

int &r = i;

int \*p;

p = &i;

\*p = i

int &r2 = \*p;

Нулевые указатели

Нулевой указатель (null pointer) не указывает ни на какой объект.

Есть несколько способов получить нулевой указатель

1. int \*p1 = nullptr; // эквивалентно int \*p1 = 0; (чаще всего используют)
2. int \*p2 = 0; //
3. int \*p3 = NULL; // ~ int \*p3 = 0; (необходимо #include cstdlib)

NULL – переменная препроцессора

Препроцессор – это программа. которая выполняется перед компилятором (при нажатии кнопки «RUN» сперва выполнятся команды препроцессора, а потом уже отредактированный текст будет подан в компилятор: удаление комментариев, поиск и выполнение директив (команд))

int zero = 0;

pi = zero; // ошибка: нельзя присвоить переменную типа int указателю

Присвоение заставляет указатель указывать на другой объект

int i = 42;

int \*pi = 0;

int\* pi2 = &i; // указывает на i

int \*pi3; // неинициализирвоан

pi3 = pi2; // указывают оба на i

pi2 = 0; // теперь не содержит адрес

pi = &i – изменяем значение **pi** – теперь оно указывает на i

\*pi = 0; изменяем **ОБЪЕКТ**, на который ссылается pi, т.е. наш i

Другие операции с указателями

Пока значение указателя допустимо, его можно использовать в условии.

int ival = 1024;

int \*pi=0;

int\*pi2=&ival;

if (pi){ // ложь

…

}

if (pi2){ // истина

…

}

Два указателя **равны**, если

1. имеют **один и тот же адрес**.
2. оба нулевые
3. если указывают на тот же объект
4. или на область за концом этого объекта

прим: указатель на объект и указатель на область за концом объекта могут содержать одинаковый адрес

void \* является специальным типом указателя, способного содержать адрес **любого** объекта. Содержит адрес, но тип неизвестен.

double obj = 3.14, \*pd = &obj;

void \*pv = &objl

pv = pd;

Действия с указателем void

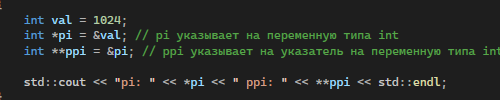
* Сравнить
* Передать функции или возратить из неё
* Присовить другому указателю типа void

Нельзя использовать для работы с объктом, адрес которого он содержит, т.к. неизвестен тип, а следовательно неизвестны и операции. которые можно выполнять с ним

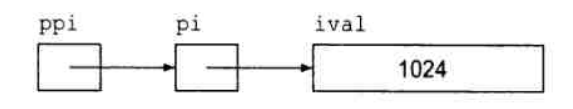
Используется для работы **с памятью как с областью памяти**, а не для доступа к объекту, хранящемуся в этой области**.**

**Указатели на указатели**

Пример реализации двойного указателя

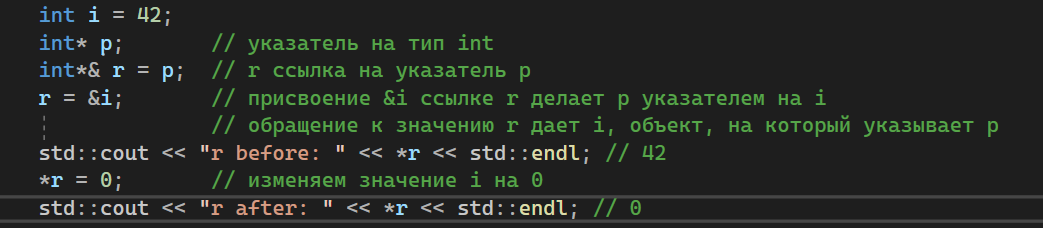


Эти объекты можно было бы представить таким образом



При обращении к значению указателя на переменную типа int возвращается значение типа int 🡪 при обращении к значению указателя мы будем получать указатель 🡪 нам необходимо дважды обратиться к значению указателя.

**Ссылки на указатели**



**Спецификатор const**

Значение переменной можно сделать неизменым, используя в её определении спецификатор const. (!!! БОЛЬШИМИ БУКВАМИ ИМЯ КОНСТАНТЫ !!!)

const int bufSize = 512; // размер буфера ввода

bufSize = 200; // Выдаст ошибку

Необходимо сразу инициализировать константу

const int i = get\_size(); // ok

const int j = 42; // ok

const int k; // выдаст ошибку

Можно использовать операции, которые не изменяют сам объект. Например, можно использовать в арифм.выражениях (const int to bool equal int to bool)

(!) По умолчанию константы локальны для файла

При использовании ссылки на константу, следует задать ссылке тоже константный тип

const int ci = 1024;

const int &ref = ci; // ok

int &ref2 = ci; //Ошибка: неконст ссылка на конст объект

r1 = 42; // Ошибка r1 ссылка на контснту

Исключение 1: мы можем инициализировать ссылку на константу результатом выражения, тип которого может быть преобразован в тип ссылки (мы можем связать ссылку и указатель на константу с неконстантным объектом, литералом или выражением)

или

int i = 42;

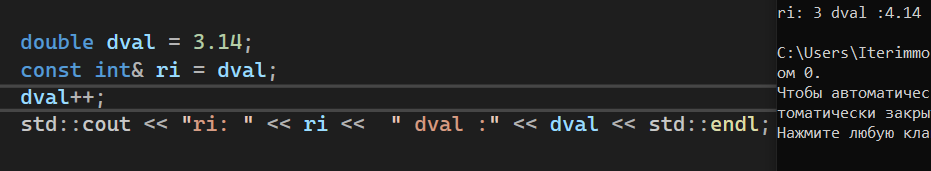
const int &r1 = i; // связываем ссылку констн инт с обычным объектом инт

const int &r2 = 42; // ok

const int &r3 = r1\*2; // ok

int &r4 = r1\*2; // ошибка. r4 неконстанта.

Ссылка инт то флоат



т.е. хоть ri является ссылкой, при изменении объекта dval, ri остается тем же. К тому же, если не использовать const, то компилятор вообще не позволит выполнить эту процедуру, т.к. у ссылкок и объекта, на который ссылаются, должен быть одинаковый тип данных.

Компилятор преобразует код в следующий вид

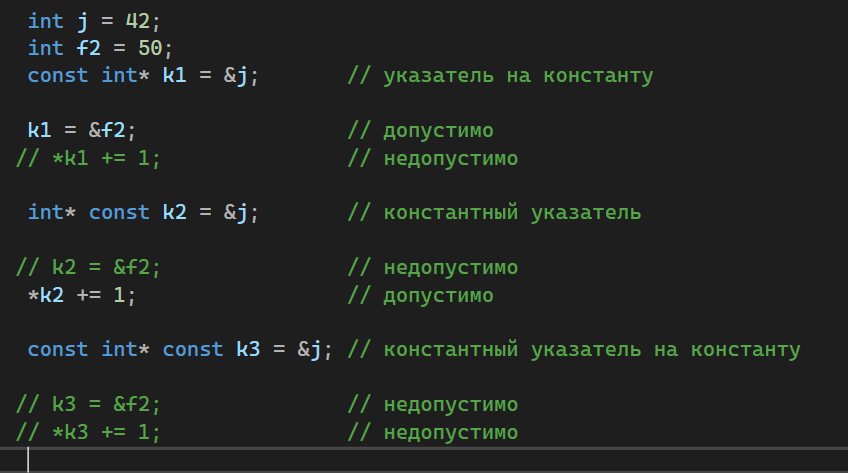
const int temp = dval; // создать временную константу типа int из перем.типа double

const int &ri = temp; // связать ссылку с временной константой

В данном случае ссылка ri связана с временным объектом (temporary).

Временный объект – это безымянный объект, создаваемый компилятором для хранения промежуточного результата вычисления.

**Указатель на константу**



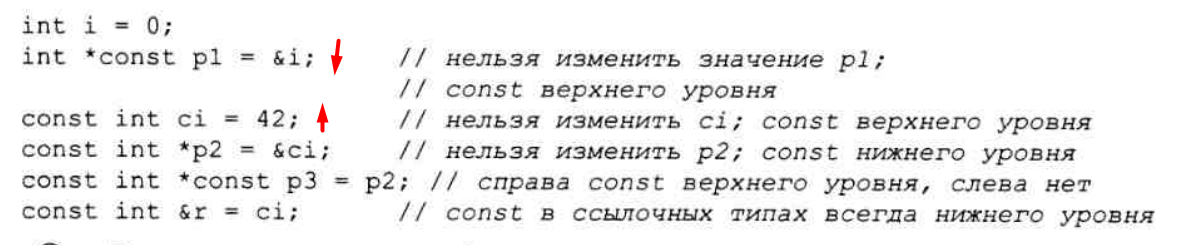
У указателя на константу можно изменить саму константу, из-за чего изменится наш указатель или заставить ссылаться на другой элемент.

У констанстантного указателя можно изменять значение через значение указателя, однако нельзя поменять объект, к которому привязан наш указатель

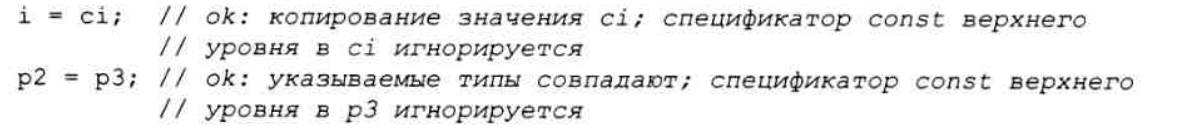
У константного указателя на константу нельзя изменить ни объект, к которому привязан наш указатель, ни значение объекта через указатель. Истинная константа.

Спецификатор const верхнего и нижнего уровня:

* Верхнего уровня (top-level const) используется для обозначения того ключевого слова const, который объявляет константой сам указатель (объект сам константа – не позволяет изменять сам объект)
* Нижнего уровня (low-level const) – когда указатель способен указывать на константный объект (находится в начале – не позволяет менять связь )

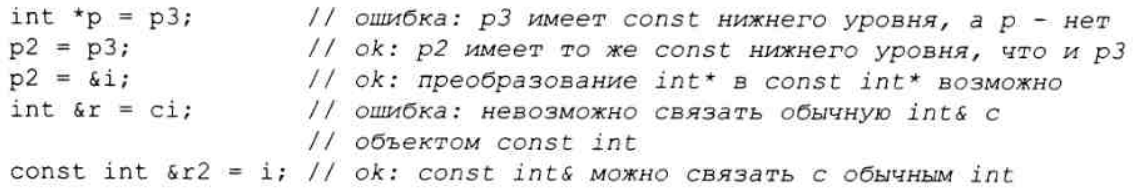


Различие между спецификаторами const верхнего и нижнего уровней проявляются при копировании объекта. При копировании объекта спеицифактор внешнего уровня игнорируется, т.к. копирование не изменяет сам объект.



Спецификатор const нижнего уровня никогда не игнорируется. При копировании объектов у них обоих должны быть одинаковые спецификаторы const нижнего уровня, или должно быть возможно преобразование между типами этих двух объектов.

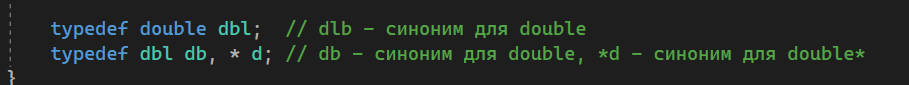
Как правило, преобразование неконстанты в константу возможно, но не наоборот.



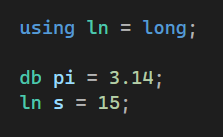
# Работа с типами

Псевдоним типа (type alias) – это имя, являющееся синонимом имени другого типа. Нужны для упрощения определения типов, облегчая их использование

Определяется следующим образом:



Также существует второй способ



Псевдоним типа – это имя типа. Оно может присуствовать везде, где присутствует имя типа.

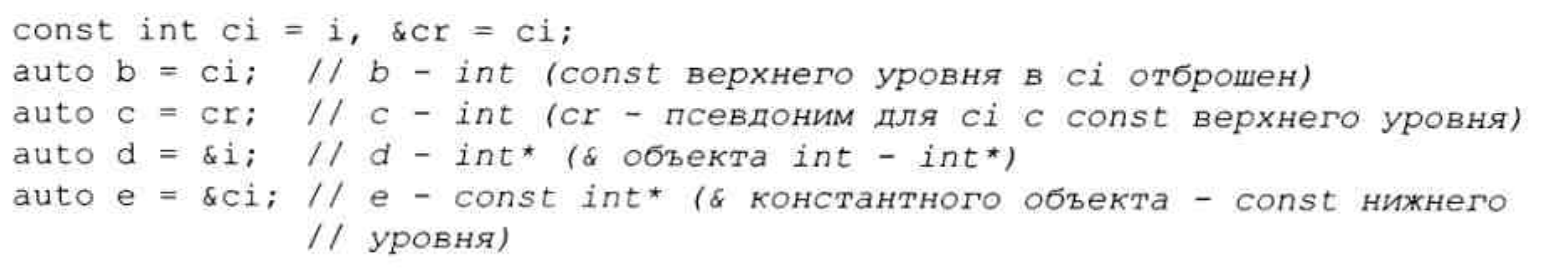
Спецификатор типа **auto**

Тип определяется самим компилятором.

Переменная должна быть инициализирована.

auto item = val\_1 + val\_2;





Cпецификатор const верхнего уровня отбрасывается

Спецификатор const нижнего уровня учитывается в случае, когда инициализатор является указателем на константу

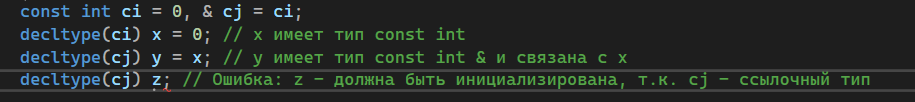
Если необходимо, чтобы у выведенного типа был специффикатор верхнего уровня, его нужно указать явно.

Спецификатор типа **decltype**

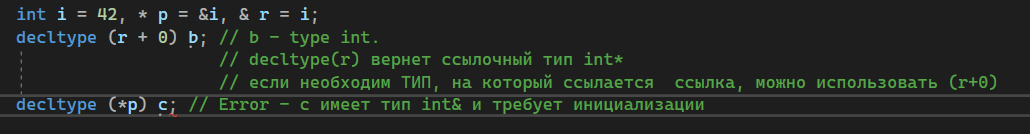
Используется, когда необходимо узнать тип выражения, но нет необходимости вычислять его.

decltype ( f() ) sum = x;

Пример



При работе с указателями



decltype((var)) – всегда возвращает ссылочный тип

decltype(var) – возвращает ссылочный тип, только если var – ссылка.

Отличие **auto** от **decltype**

auto – автоматический вывод типов

decltype - возвращает тип указанного выражения

В отличии от auto, который выводит тип по значению переменной, decltype выводит тип выражения, переданного ей.

# Структуры данных

Структура данных – это способ группировки взаимосвязных данных и стратегии их использования.

(sales\_item группирует ISBN книги, количество проданных экземпляров и выручку от этой продажи + набор операций: функция isbn() и операторы <<, >>, + , - )

Определяя класс мы создаем свои типы данных.

# Классы

Фундаментальные идеи, лежащие в основе концепции классов – абстракция данных и инкапсуляция.

Абстракция данных – это программный подход, полагающийся на разделение интерфейса и реализации.

Инкапсуляция обеспечивает разделение интерфейса и реализации класса.

Интерфейс класса состоит из операций, которые пользователь класса может выполнить с его объектом.

Реализация включает переменные-члены класса, тела функций, составляющих интерфейс, а также любые функции, которые нужны для определения класса, но не предназначены для общего использования

Инкапсулируемый класс скрывает свою реализацию от пользователей, которые могут использовать интерфейс, но не имеют доступа к реализации.

Класс, использующий абстракцию данных и инкапсуляцию, называют **абстрактным типом данных**.

# Терминология

Переменная – именованное хранилище, которым могут манипулировать программы.

Буфер – это область данных, используемая для хранения данных.

Директива #include – Делает код в указанном заголовке доступным в программе

Заголовок – Механизм, позволяющий сделать определения классов или других имен доступными в нескольких программах. включается в код программы при помощи директивы #include

Манипулятор – объект, непостредственно манипулирующий потоком ввода-ввывода

Пространство имен – механизм применения имен, определенных в библиотеках. Позволяет избежать случайных конфликтов имени.