<https://www.youtube.com/watch?v=nIY1fBkGQq4>

*Динамическим программированием* (англ. *dynamic programming*, ДП, динамика) называется подход, обычно заключающийся в следующих общих шагах:

* Свести задачу для *n* к задаче для чисел, меньших, чем *n* — то есть найти какую-то рекурсивную формулу.
* Создать массив (или какую-нибудь другую структуру данных) для хранения ответов на подзадачи.
* Заполнить начало массива вручную (базовые случае, для которых формула не работает).
* Обойти массив от известных значений в неизвестные и заполнить ответы по формуле.
* Вывести ответ — обычно просто записанный в последней ячейке массиве.

Чтобы решить задачу динамическим программированием, вы должны ответить на 5 вопросов:

* Что лежит в массиве? (чаще всего самый важный вопрос)
* Как инициализировать начало массива?
* Как обходить массив? (чаще всего слева направо, но не всегда)
* Какой формулой считать элементы массива?
* Где в массиве лежит ответ?

Этим приемом можно решать большое количество важных задач, и в этом разделе мы в этом убедимся.

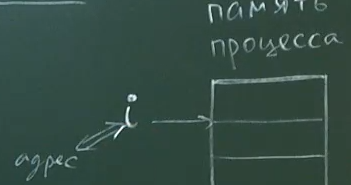
**Адреса и указатели:**

вот если я заведу какую-нибудь переменную и у нее есть какое-то место в памяти процесса. Это место характеризуется адресом. Соответственно у этой ячейки есть адрес.

Адрес этих нескольких байт, которые занимает переменная, является числом.

Т.е. вот это имя соответствует некоторому адресу, другими словами – эквивалентно.

В то же время этот адрес нам неизвестен. Когда мы пишем программу мы пользуемся символичным именем. Компилятор же он использует и то и другое. Он связывает это друг с другом, т.к. в машинных кодах имен у переменных не существует.

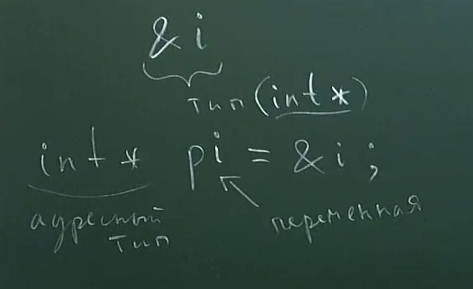
т**.е. не имя указывает на ячейку – а адрес**

В C++ переменные представляют собой тип l-значения **(l-value**). l-значение (l-value) **– это значение, имеющее адрес (в памяти).**

мы можем узнать этот адрес с помощью унарной операции **&**

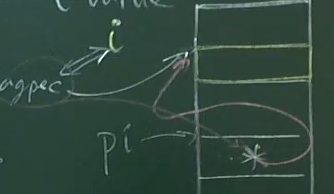
Эта штуковина имеет особенный тип. **&i** если i есть int , то она будет иметь тип **int\***

если адрес — это число, то его можно сохранить (положить) в соответствующую переменную. Для этого мне достаточно создать таковую переменную.

Вот такая переменная **адресного типа называется указателем.**

раз у меня есть переменная это что значит?

Значит есть ячейка в которой хранится этот адрес – тоже какое то число! т.е. я взал адрес I и положил его сюда.

но если мы продолжим эту историю.

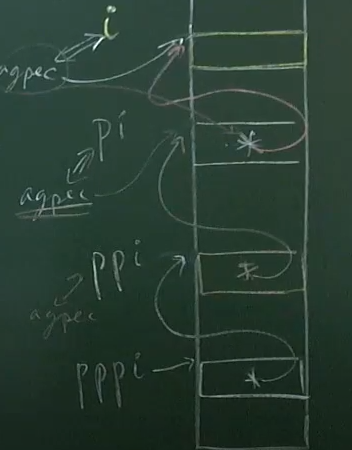
указатель являясь переменный сам имеет соответствующий ему адрес.

и тогда мы можем записать

**int\*\* ppi=&pi**

естественно, как у ячейки памяти да у нее тоже есть собственный адрес

соответствует если я захочу я могу создать ячейку PPPI и так далее. На

самом деле да я могу сколько угодно создавать таких ячеек.

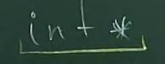
**int\*\*\* pppi=&ppi**

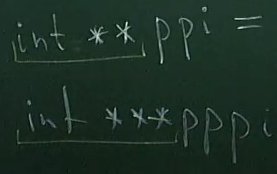
Вопрос – как нам достучаться до реального значения переменной.

Т.е. нам нужна кроме операции взятия адреса какая-то обратная операция. Мы должны иметь возможность воспользоваться вот этим хранимым адреса

Это операция **разыменования**!

и обозначается \*.

вы должны уметь отличать вот эту звёздочку и вот эту звёздочку Почему? Да потому, что вот эта звездочка — это часть имени типа и она используется там где описывается тип переменной или параметр передаваемый в функцию.

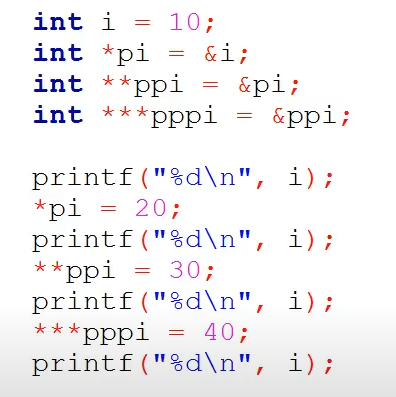
и хотя принято при описании указателя часто вот тут ставят пробел, а вот тут пробел не ставят - тем не менее эти звездочки это не разыменование, это просто часть **имени типа**, которая говорит о том, что здесь описывается **адресный тип указателя**

А \* которая применяется как разыменование – **это операция!!!**

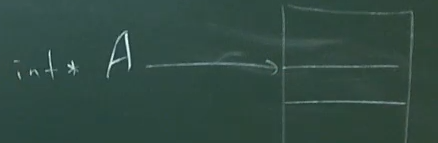
\*pi=20.

То у меня слева – это вычисляемое **l-value выражение.**

но естественно, что в процессе попытки пройти по адресу бывают ошибки когда люди пытаются пройти по какому-то адресу они могут обнаружить, что этого адреса не существует в реальности. Дело в том, что память процесса операционной системы виртуализирована, а это значит что физические адреса не соответствуют тем адресам которые видят наша программа.



**разыменование** это не единственное, что можно делать с адресами. Оказывается, при помощи арифметических операций мы можем вычислять адреса соседних ячеек.

вот этот адрес у меня указывает на какую-то ячейку какого-то типа, ну допустим, что а имеет тип int

это означает что он указывая сюда и будет пытаться интерпретировать лежащие здесь как целое число

Адрес следующей ячейки – А+1

при этом естественно, что тип int который подразумевается хранимым вот здесь - он занимает не один байт, а несколько - вероятнее всего четыре.

т.е. следующая ячейка находиться на расстоянии+4 байта.

Для разных типов адресов операция плюс 1 будет приводить к смещению адреса на разное количество байт.

адресная арифметика позволяющая смещаться указателем вперед назад на сколько-то ячеек - она именно гуляет по вот таким ячейкам.

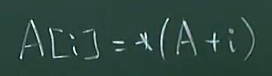
И вот с такой позиции интересно узнать, что такое массив. Чистых массивов в языке С как бы и нет.

**на самом деле массив это всего лишь константный адрес и выделенная память**



я могу взять \*(А+1) или \*(А+2)…..

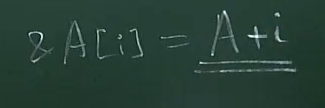
Можно написать вот такую формулу

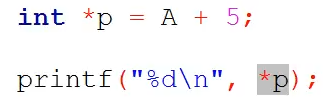


это дает очень интересный эффект – если мне нужен адрес I- той ячейки массива Я могу записать **&A[i]**

**и тут наступает интересный момент , что &\*P =P**

т.е. эти операции обратны друг другу.

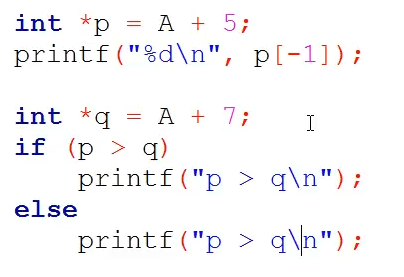




а можно записать p[0].?

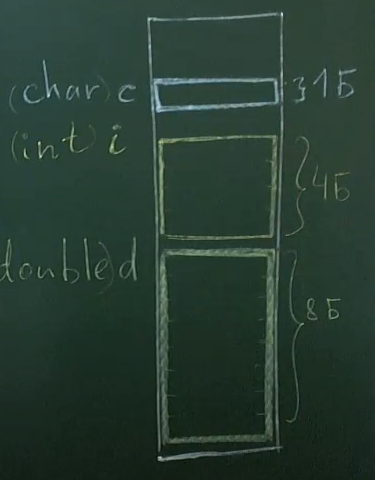
можно - по тому адресу в котором ты находишься без смещения и он не достает значение

а если **p[-1**].?



Указатели можно и вычитать! (однотипные конечно). Это целое число.

Какого размера ячейки для разных типов?

чар в моем компиляторе занимает один байт, int четыре байта, дабл - он самый стабильный, т.к. он подвержен стандарту.

скажите мне пожалуйста: адрес каждой из этих переменных одинакового размера или разного? адрес мухи и адрес слона в зоопарке имеет абсолютно одинаковую ячейку для хранения.

Это просто некоторое количество байт необходимо в рамках данной архитектуру чтобы показать какую-то конкретную ячейку, а уж какого она размера определяет тип указателя

Среди типов языка С есть один очень интересный - **VOID**

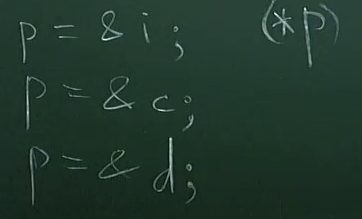
Невозможно создать переменную данного типа – **~~VOID X~~**

Но при этом можно создать переменную **VOID \*Р – это указатель.**

Его еще называют безтиповый указатель.

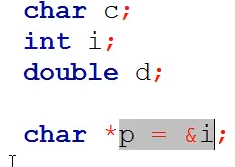
Мы можем положить в Р – любой адрес. – адрес целого, чара, дабла и т.д.

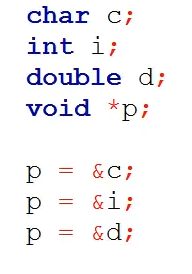
Вопрос - а что будет если я попытаюсь интерпретировать данные по без типовому указателю **\*Р**?

чисто формально это должен быть VOID, но он в переменных быть не может.

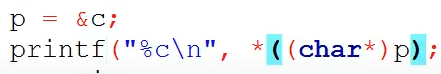
а раз быть не может, значит компилятор не разрешит нам разыменования без типового указателя - **~~\*Р~~**

Для чего он тогда нужен ?

тут нам выдаст предупрежедние!!!

 в этом случае – все ГУД!

Что бы его применить – существует операция **явного приведения типа**

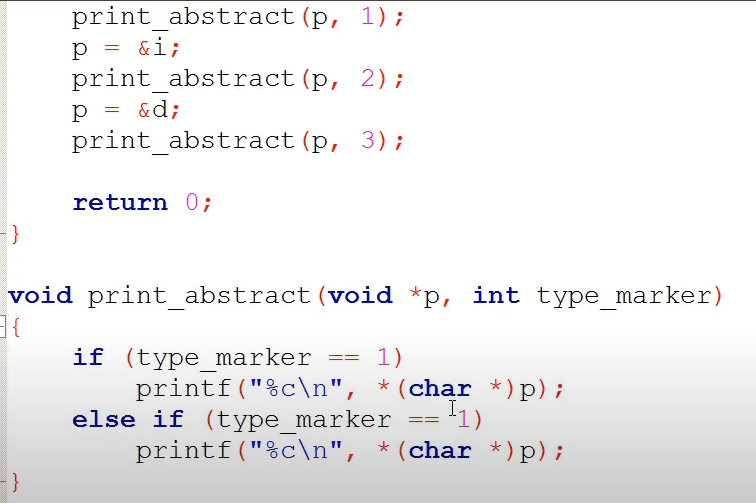


звездочка надо - что это адрес, пусть и без типовый и его надо интерпретировать как адрес, но после этого его необходимо разыменовывать.

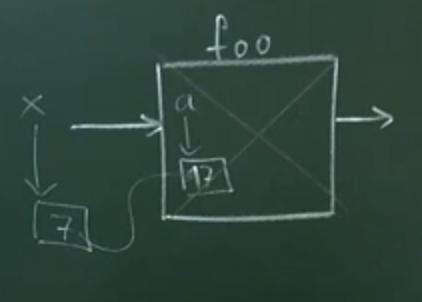
**Но странно- зачем мне это все локально вот здесь вот гонять, почти бессмысленно?**

Но сделаем некий **use case!**

Создадим некоторую функция и передадим параметр void \*p. Но при этом нам понадобиться передать маркер типа.

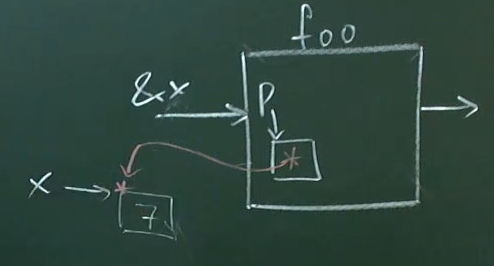


**Передача параметра в функцию**



как изменить переменную, которую мы передали?

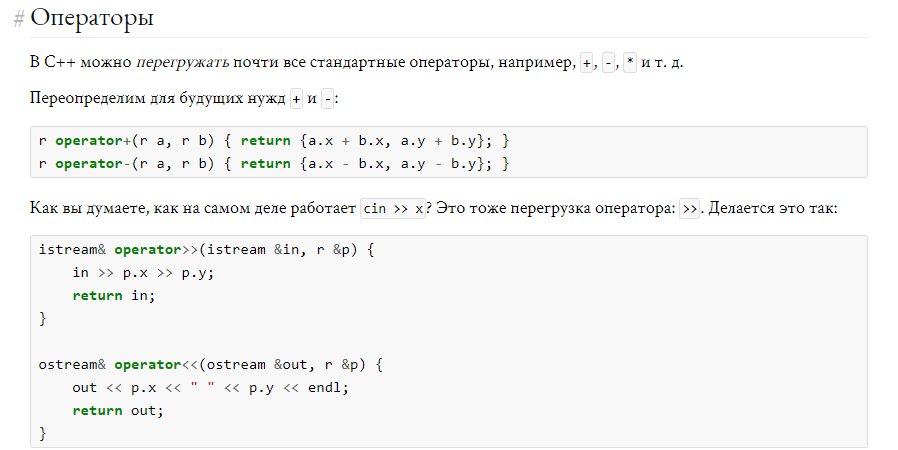
оказывается, мы можем в качестве параметра локального существующего временно передать не саму переменную x точнее не значение переменной x, а адрес этой ячейки



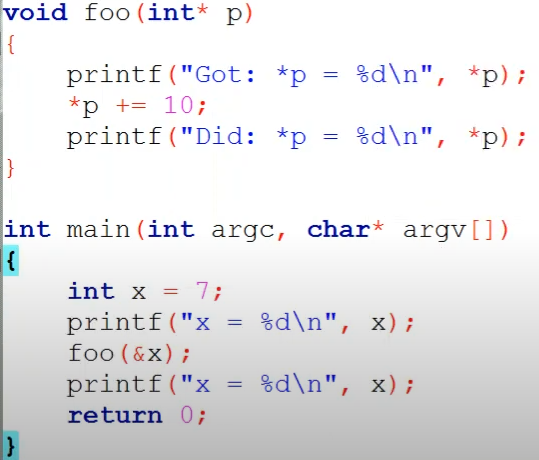
я, пользуясь этим адресам могу внутри функции разыменовывая свой указатель дотягиваюсь до самой переменной.

естественно, что в этом случае типы должен жестко совпадать иначе компилятор будет бить тревогу.

это дает вам понимание, почему при передаче в различные функции мы пишем &

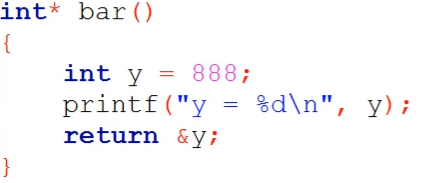


это функция универсальная и она должна уметь принимать любой указатель: на один тип или на другой.

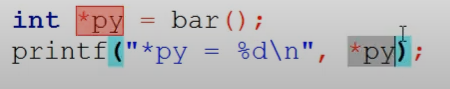


есть некоторые неудобства в таких изменяемых параметров в том, что мне приходится все время в течение функции писать звездочка р ( т.е. разыменовывать)

а если создать функцию – которая возвращает указатель



а в main запишем



Будет работать?

но она выдаст предупреждение , **что нельзя возвращать адреса локальных переменных!**