|  |  |
| --- | --- |
| Здравствуйте, коллеги. На этом видеоуроке мы рассмотрим базовые понятия языка С, организацию памяти и операторы ввода-вывода. |  |
| Шаблон программы:  - Как и программы на любом другом языке, программы на языке С имеют ряд обязательных элементов, также называемых шаблоном программы.  - Рассмотрим эти элементы подробнее. Среда Eclipse при создании файла оставила в первых строчках комментарий. Комментарий это любой текст, написанный для удобства программиста и игнорируемый компилятором. Комментарии бывают как однострочные, так и многострочные. В редких случаях можно встретить внутристрочные комментарии, они считаются дурным тоном, поскольку резко снижают читаемость кода. | /\*  \* FirstProgramm.c  \*  \* Created on: 27 мая 2017 г.  \* Author: ivanovcinnikov  \*/  // Это тоже комментарий |
| \*Директивы препроцессора:  - это такие команды, которые будут выполняться не просто до запуска программы, но даже до компиляции. В директивах препроцессора подключаются внешние заголовочные файлы, и определяются некоторые абсолютные значения проекта. Обратите внимание, что директивы препроцессора это достаточно сложный инструмент, и использовать его только для определения константных значений - не лучшее архитектурное решение  *Для нашего проекта нам понадобится директива #include <stdio.h> - эта директива подключит библиотеку стандартного ввода вывода в наш проект* | #include <stdio.h> |
| \*Функция main:  - Это точка входа в программу. Поскольку программа написанная на языке С работает на одном уровне с операционной системой, а не в среде виртуализации, она должна сообщить операционной системе, что она отработала нормально, это делается посредством возврата в качестве результата работы программы кода ноль. Любой другой возвратный код сообщает системе, что программа завершена некорректно и необходимо освобождать занятые ею ресурсы. Все операторы, кроме директив препроцессора, комментариев и описаний функций должны заканчиваться точкой с запятой. | int main(int argc, const char\* argv[]) {  return 0;  } |
| \*Форматированный вывод:  - Общение с пользователем на чистом Си происходит через консоль. Для того, чтобы выводить какую-либо информацию используется функция printf(); предназначенная для форматированного вывода некоторого текста в консоль. Функция описана в заголовке stdio.h, поэтому мы и включили данный заголовок в нашу программу. | printf("Hello world!"); |
| - Какого рода форматирование применяется при выводе строк в консоль? Существуют два основных инструмента придания выводу необходимого вида: экранированные последовательности (escape sequences) и заполнители (placeholders)  - "Экранированная последовательность" это буква или символ, написанные после знака обратного слэша, которые при выполнении программы будут на что-то заменены. Самые часто используемые это:  - символ новой строки - \n  - символ табуляции - \t  - символ обратного слэша - \\  - символ конца строки соответственно - \0 | printf("Hello world! \n");  // \n \t \\ \0 |
| *Чтобы убедится что это правильно работает выведем еще одну строку с надписью «Это новая строка» на следующую строку нашей консоли*  *Давайте добавим к нашей строке символ табуляции чтобы увидеть как он работает.*  *Как мы описали в тексте сначала идет пробел затем символ табуляции.*  *Отлично, теперь рассмотрим экранированную последовательность «\\» она делает ни что иное как добавляет символ обратного слэша в наш текст.*  *Обратите внимание что не поставив в конец строки последовательность \n мы заставим компилятор писать текст на текущей строке, не переходя на новую.*  *И наконец \0 – сообщает компилятору что строка закончилась. Даже если у нас есть еще какие-то символы до закрывающих кавычек компилятор их просто проигнорирует* | printf("This is a new row");  printf("This is a new \trow with tab");  printf("This is a new \trow with \\ symbol");  (добавить \n во все имеющиеся строки)  printf("Hello\0 world! \n"); |
| - "Заполнитель" это также специальная последовательность, но она говорит компилятору, что на место этой последовательности необходимо вставить некий аргумент, который будет передан после строки, через запятую, при вызове данной функции printf();  *Заполнитель начинается со знака процента и обозначает тип вставляемой переменной.*  Наиболее часто используются %d %s %p %f %lf %x для переменных типа:  - integer(decimal) - %i (%d)  - строк - %s  - char - %c  - указателей (о которых мы поговорим позже) - %p  - float - %f  - double(long float) - %lf  - беззнаковое целое в шестнадцатеричном виде - %x | int a = 50;  printf(“%d”, a);  // %d %s %p %f %lf %x |
| Для заполнителей %d %i %f часто используются дополнительные параметры, такие как количество знаков после запятой %.2f или общее количество знаков целого числа %5d. Это можно использовать для записи чисел с лидирующими нулями %05d; Для того чтобы вставить в текст символ процента нужно дважды написать процент.  *Добавим к предыдущему выводу новую строку и запустим проект (printf("This is a new \trow with \\ symbol"\n);) Как видим, все отлично работает.*  *Вместе с заполнителями мы можем использовать любой текст. Добавим какой-то текст, уберем лидирующие ноли чтобы посмотреть как работает заполнитель без 0. Как видим, добавилось три пробела перед цифрой* | // %.2f  //%5d  int a = 50;  printf("%05d%%", a);  printf(text and"%5d%%", a); |
| Переменные и типы данных:  - переменные в программе на языке С это некоторые именованные контейнеры, содержащие значение, строго описанное при создании контейнера. Переменные делятся на целочисленные, символьные, указатели и числа с плавающей точкой. Все, кроме указателей и символьных переменных бывают как знаковыми так и беззнаковыми. То есть самый старший бит в двоичной записи этих переменных отводится под определение, является-ли число отрицательным, или положительным. В классическом С нет булевого типа, вместо него используется целое число и значения 0 для лжи и любое другое число для истины, обычно это единица.  Об указателях мы будем подробно говорить на седьмом уроке. | int number = 50;  char symbol = ‘A’;  float real = 5,345f;  // no boolean type  int true = 1;  int false = 0; |
| *Тип Integer занимает в памяти 4 байта и может принимать значения от -2 млрд 147 млн до 2 млрд 147 млн* | int number = 50; // 4 byte -2.147.000000…2.147.000000 |
| - С символьным типом тоже не всё так просто. Если вкратце, то в переменной типа CHAR хранится число, которое можно интерпретировать как символ. По умолчанию тип знаковый, то есть может содержать значения от -128 до +127, но символы в таблице ASCII (American standard code for information interchange), что совершенно логично, имеют только положительные индексы, поэтому в читаемый текст в стандартном С можно превратить только латинский алфавит и некоторый набор знаков и символов, находящиеся на первых 128-ми местах в этой таблице. | Char symbol = 75; // 1 byte -128…127 |
| Также можно явно указать компилятору, что мы хотим использовать эту переменную как беззнаковую, для этого используется ключевое слово unsigned, что позволит нам хранить только положительные числа гораздо больших значений. *Например для переменной типа CHAR это будут значения от 0 до 255*, а для переменной типа int в unsigned int можно можно хранить значения от 0 до +4 млрд 294 миллионов. В более поздних редакциях были утверждены типы long long и другие, для хранения 64-х разрядных целых чисел. | unsigned Char symbol = 75; // 0…255  unsigned int = 50; // 0…4.294.000000 |
| Числа с плавающей точкой представлены двумя типами, 4-х байтный float и 8-ми байтный double (также называемый long float). Хранятся в памяти в неявном виде, а разделённые на мантиссу экспоненту и знак, что делает их одними из самых сложных в работе. При работе с числами с плавающей точкой нужно обращать особенное внимание на тип переменной, поскольку сравнение внешне одинаковых чисел разных типов с вероятностью 99% даст ложный результат.  *Совсем забыл, еще есть тип short, он редко используется но все же существует. Он занимает в памяти 2 байта и может принимать значения от -32768 до 32767* | float real = 5,345f; // 4 byte  double realdouble = 5,345; // 8 byte  short sh = 345; // 2 byte -32768…32767 |
| Как было сказано - переменная это именованный контейнер. У каждого такого контейнера есть свой собственный адрес в оперативной памяти. Язык С позволяет узнать этот адрес и работать с ним. Оператор взятия адреса это знак амперсанд, написанный перед именем переменной. *Т.е. у любой переменной всегда есть значение и адрес где это значение хранится.*  Для вывода в консоль адреса используется специальный заполнитель - %p.  *Соберем наш проект и запустим.*  Как видим, урок ведётся с 64-х разрядной операционной системы, поэтому ячейка памяти имеет 64-х разрядный адрес. | printf("Переменная а имеет значение: %d \n", a);  printf("Переменная а хранится по адресу: %p \n", &a); |
| Поговорив о переменных и о выводе в консоль нельзя не сказать о пользовательском вводе данных. Один из способов пользовательского ввода данных в программу - это использование функции scanf();. *Предложим пользователю ввести некоторое число*  Функция scanf() – это функция форматированного ввода. Принцип её работы очень похож на принцип работы функции printf(); В двойных кавычках мы указываем в виде заполнителя тип переменной, которую ожидаем от пользователя, а в качестве дополнительного аргумента указываем адрес той переменной, в которую хотим записать введённые пользователем данные. Получается процесс прямо противоположный выводу. В этой функции можно использовать все те же заполнители, что и при выводе, поэтому пользователь может ввести как целые числа, так и символы, строки и числа с плавающей точкой. Выведем в консоль изменённое введённое число, чтобы удостовериться, что всё работает.  *Скомпилируем нашу программу, введем некое число в консоль и увидим его удвоенным в нашем выводе* | int input;  printf("Введите, пожалуйста, число:");  scanf("%d", &input);  printf("Вы ввели число %d, мы удвоили его для Вас: %d\n", input, input\*2); |
| Здесь по видео про внутристрочный комментарий – по моему не к месту. Мы о них вначале говорили |  |
| *Раз уж мы коснулись арифметических выражений, поговорим немного о арифметике.*  В языке С поддерживаются все базовые арифметические операции, такие как сложение, вычитание, умножение, деление. Операции бинарной арифметики, такие как И, ИЛИ, НЕ, ИСКЛЮЧАЮЩЕЕ ИЛИ, СДВИГИ. А также все вышеперечисленные операции с последующим присваиванием в первую переменную. | // + - \* /  // & | ! ^ << >> |
| *Инициализируем переменную типа int и выведем ее в консоль*  *Выражение операции с последующим присваиванием в первую переменную означает запись вида variable = variable + 50; можно сократить до variable += 50; Выведем результат работы в консоль и увидим что к первоначальному числу прибавилась цифра 100*  *Рассмотрим оператор целочисленного деления. Если мы производим операции с целыми числами такими как int, short или char оператор целочисленного деления всегда будет возвращать только целые числа, отбросив дробную часть. И действительно результатом деления 170 на 2 будет 85, что мы и увидим в консоли. Но, поменяв делитель с 2 на 4 результатом будет деления будет 42,5, а мы увидим только 42 – дробная часть отбрасывается*  Также особенного внимания заслуживает оператор получения остатка от деления также называемый оператором взятия по модулю. Записывается как символ % и возвращает остаток от деления первого числа на второе. *Давайте посмотрим какой остаток будет у деления наших операндов* | int variable = 70;  printf("Переменная variable %d\n", variable);  variable = variable + 50;  variable += 50;  printf("К переменной было дважды прибавлено 50 и получилось %d \n", variable);  variable = variable / 2;  printf("Целочисленное деление переменной на 2: %d\n", variable);  variable = variable / 4; // дробная часть будет проигнорирована  printf("Целочисленное деление переменной на 4: %d\n", variable);  int reminder = variable % 4;  printf("Остаток от деления 170 на 4: %d\n", reminder); |
| *Рассмотрев арифметику в языке Си нельзя не упомянуть об операторах увеличения и уменьшения значения переменной на единицу. Они называются*  *операторы инкремента (++) и декремента (--).* | reminder++;  printf("Инкремент %d \n", reminder); |
| На сегодня все, остальные особенности языка мы рассмотрим на следующих уроках. До новых встреч. |  |