Lesson 7

|  |  |
| --- | --- |
| Коллеги, здравствуйте. Вот и пришла пора поговорить о серьёзном низкоуровневом программировании. О том, от чего стараются оградить программистов языки высокого уровня и современные фреймворки. Об указателях.  На этом уроке мы поговорим о том, что такое указатели и как они соотносятся с остальными переменными, что такое передача аргумента по значению и по ссылке. |  |
| Как мы помним из второго занятия, все переменные и константы, используемые в программе, хранятся в оперативной памяти. У каждой переменной и константы в памяти есть свой собственный адрес. Этот адрес выдаётся нашей программе операционной системой и мы можем использовать его по-своему усмотрению.  В языке С есть возможность получить доступ к переменной не только по имени, но и по адресу. Получение доступа к переменной по адресу называется разыменовыванием. *Давайте выведем в консоль всю имеющуюся информацию о переменной «а». Мы знаем, что это целочисленная переменная значением 50, которая хранится по какому-то адресу.*  Адрес переменной может храниться в специальной переменной, которая называется указатель.  Для объявления указателя пишут тип переменной, адрес которой будет храниться в указателе, знак звёздочки и имя указателя. Такому указателю можно присвоить значение адреса существующей переменной, также как мы делали это раньше с другими типами данных  Для наглядности выведем всю имеющуюся у нас на данный момент информацию на экран. Напомню, для вывода адреса используется заполнитель %p. *Выведем в консоль значение переменной pointer и адрес переменной pointer*  *Увидим, что значение переменной pointer является совершенно случайным числом.*  *Если мы представим это в виде адреса то все станет на свои места. Адрес «а» - это значение переменной pointer.* | #include <stdio.h>  int main (int argc, const char\* argv[]) {  int a = 50;  printf("value of a is %d \n", a);  printf("address of a is %p \n", &a);  int \* pointer;  pointer = &a;  printf("value of pointer is %d \n", pointer);  printf("address of pointer is %p \n", &pointer);  printf("value of pointer is ~~%d~~ %p \n", pointer); |
| Пока что ничего необычного, все эти операции мы выполняли на предыдущих уроках. Но поскольку *pointer* это необычная переменная, а указатель, то мы можем получить не только её значение, но и значение переменной, на которую она указывает*. Давайте запишем, вывести в консоль - переменная pointer указывает на такое-то значение и разименуем pointer. Т.е. получим доступ к переменной, на которую ссылается указатель pointer.*  Таким образом, в указателе хранится ссылка на значение некоторой переменной. То есть указатель - это примитивный ссылочный тип данных. Без указателей невозможно себе представить создание классов, интерфейсов и всеми любимого объектно-ориентированного программирования. | printf(“variable ‘pointer’ points at: %d”, \* pointer); |
| *Давайте изменим значение переменной «а», не на прямую, а с использованием указателя. Как видим, значение переменной изменилось.* | \*pointer = 70;  printf("value of a is %d \n", a); |
| Теперь, когда мы знаем об указателях, и умеем получать значения переменных, на которые они указывают, а также изменять их, перед нами открываются невообразимые ранее перспективы. Мы можем писать функции не создавая в них копии переменных, а передавать в них ссылки на уже существующие переменные, тем самым экономя память, и ускоряя выполнение программы. |  |
| Например, не составит труда написать программу, которая бы меняла местами значения двух переменных. Но написать функцию, которая бы проделывала тоже самое невозможно без применения указателей. Почему? очень просто - внутри функции создаются свои собственные переменные, значения которых меняются местами, и даже если мы вернём одну из них – как быть со второй? А получить доступ ко второй переменной мы не можем, поскольку, помним, она находится в области видимости функции. Такая передача аргументов называется передачей по значению (мы берём значение некоторой переменной и передаём внутрь функции). Т.е. мы берем значения некоторых переменных в функции Мэйн и передаем их в функцию, как значения новой переменной внутри функции.  Как решить эту проблему? Передавать не значения переменных, а их адрес, тем самым сообщив функции, что нужно не создавать новые копии переменных, а сделать что-то с уже существующими, и, естественно указать адрес, с какими именно. Передача в качестве аргумента адреса, и создание в теле функции нового указателя называется передачей по  ссылке. То есть функция будет ссылаться на переменные, на которые мы укажем.  *Давайте немного модифицируем нашу функцию – передадим ей адрес переменных. Внутри функции у нас будут создаваться не переменные, а указатели на переменные, т.е. мы будем ссылаться на те же самые значения – т.е. мы будем изменять переменные, которые создали в функции Мэйн. Перепишем логику из предыдущего урока:*  \*x ^= \*y;  \*y ^= \*x;  \*x ^= \*y;  *Теперь нам не нужно ничего возвращать, потому что в функции ничего не создавалось, поэтому заменим int на void.* | int swap\_variables(int x, int y) {  return x;  }  int first = 50;  int second = 40;  swap\_variables(40, 50);  swap\_variables(&first, &second);  int swap\_variables(int \*x, int \*y) {  //\*x = 50;  //\*y = 40;  \*x ^= \*y;  \*y ^= \*x;  \*x ^= \*y;  return x;  }  void swap\_variables(int \*x, int \*y) {  \*x ^= \*y;  \*y ^= \*x;  \*x ^= \*y;  } |
| *Удалим из функции все лишнее и выведем в консоль переменные* first и second *до их смены и после. Передадим в функцию* swap *переменные first и second и убедимся, что все работает корректно.* | int first = 50;  int second = 40;  printf("first = %d, second = %d \n", first, second);  swap\_variables(&first, &second);  printf("first = %d, second = %d \n", first, second); |
| Применение такого подхода открывает перед нами широкие возможности, которые мы рассмотрим на следующих уроках.  До новых встреч. |  |