Lesson 9

|  |  |
| --- | --- |
| Здравствуйте, коллеги. На предыдущем занятии мы начали говорить о массивах, давайте продолжим. Это будет непросто, но мы поговорим о том, что из себя представляет идентификатор массива, чем чреват выход за пределы массива, затронем тему арифметики указателей и научимся передавать массивы в функции. |  |
| Как упоминалось на предыдущем занятии - массив - это ссылочный тип данных. То есть в идентификаторе хранится ссылка на первый байт первого элемента массива, дальнейший доступ к элементам осуществляется посредством смещения относительно этого байта. Таким образом понятно, что запись вида array[0] говорит нам о том, что нужно взять адрес массива и сместить указатель на 0 элементов того типа, из которых состоит массив. Отсюда становится ясно, почему индексирование массивов начинается с нуля. | int arr[ARRAY\_LENGTH];  arr[0] = 20; |
| *Давайте я попробую в комментариях к коду визуализировать положение вещей в массивах. Вот в квадратных скобках наш массив, квадратики – это элементы нашего массива. Соответственно, если мы берем элемент с нулевым смещением, то есть по нулевому индексу, это будет вот этот вот индекс, квадратными скобочками я его нарисую. Все остальные индексы мы просто игнорируем. Далее, берем следующий элемент массива. Для этого возьмем смещение в единицу, допустим, это будет число 50. Соответственно в комментариях опять рисуем наш массив, и уже смещение будет на единицу. То есть мы игнорируем нулевой элемент и берем первый. Ну, надеюсь, немножко стало понятнее.* | int arr[ARRAY\_LENGTH]; // [ [][][][][][][][][][] ]  arr[0] = 20; // [ [] ]  arr[1] = 50; // [ [] ] |
| Относительно выхода за пределы массива надо сказать, что ни компилятор, ни тем более операционная система никаких проверок не делают, поэтому это полностью ложится на плечи программиста. Никаких сред виртуализации, никаких исключений, только случайные данные, которые могут попасться нашей программе. Надо сказать, что всё-таки большая часть значений за пределами массива будет равна нулю, но всё равно лишний раз экспериментировать не стоит. | arr[10] = 60; // [ ] [] |
| Как мы уже знаем в идентификаторе массива хранится ссылка на первый байт первого элемента массива, т.е. идентификатор является по сути - указателем. Но существует несколько отличий: указатель - это переменная, к ней применимы например операции инкремента и декремента, чего конечно нельзя делать с идентификатором массива. Тем не менее, с помощью идентификатора массива мы можем получить доступ к элементам массива не только при помощи записи индекса в квадратных скобках, но и при помощи так называемой арифметики указателей. Мы знаем, что массив - это единая область памяти, и значения в нём располагаются подряд по очереди, значит, отсчитав от указателя на первый индекс нужное количество байт - мы получим указатель на второй байт. Давайте для примера подсчитаем среднее арифметическое всех чисел в массиве, с использованием арифметики указателей. Заполним этот массив с клавиатуры.  *Создадим вспомогательную переменную float result, для хранения результата и в цикле будем запрашивать у пользователя цифры. Количество введенных цифр должно соответствовать количеству элементов массива, поэтому условием выхода из цикла будет равенство итератора и длины массива – 1, т.к. индексация массива начинается с 0, длина массива на 1 больше последнего индекса. Выведем в консоль надпись «введите значение», при помощи функции scanf считаем его и сразу положим в массив. Здесь мы к указателю на первый элемент массива прибавляем значение итератора и получаем индекс массива, в который будет положено значение.* Как вы видите, некоторые подсчеты программа выполняет за нас - мы прибавляем к указателю единицу, двойку, тройку и т.д, а программа понимает, что надо взять не следующий по счёту байт, а следующий указатель, т.к. в данном примере мы используем массив в котором хранятся значения типа int, а как вы помните int = 4 байт, то при увеличении указателя на 1 мы обратимся к области памяти находящейся на 4 байт дальше идентификатора, при увеличении на 2 на 8 байт и т.д. Подсчитать среднее арифметическое не составит труда.  *Выведем в консоль получившийся массив при помощи цикла for и привычной нам функции printf и напишем что среднее арифметическое будет, и посчитаем его. Для этого к результату будем прибавлять существующий результат и значение массива на которую указывает конструкция \*(average + i). Далее при помощи* функции printf выведем в консоль среднее арифметическое. *Запустим, повводим цифры и убедимся что все работает.* | int i = 0;  float result = 0;  while (i < ARRAY\_LENGTH){  printf("Enter value %d:", i);  scanf("%d", arr + i);  i++;  }  printf("Your array is: ");  for (i = 0; i < ARRAY\_LENGTH; i++)  printf("%d ", arr[i]);  printf("\nAnd the average is: ");  for (i = 0; i < ARRAY\_LENGTH; i++)  result += \*(arr + i);  printf("%f \n", result/ARRAY\_LENGTH); |
| Как мы уже говорили, идентификатор массива - это не обычный указатель. Обычный указатель хранит в себе адрес какой-то другой переменной, и сам где-то хранится. Указатель на начало массива хранит в себе адрес массива, то есть адрес его нулевого элемента, и сам этот указатель находится в этом самом месте. На первый взгляд сложновато? Но пусть Вас это не сбивает с толку, на деле всё не так жутко. На деле это означает, что при передаче массива в функцию мы не должны использовать оператор взятия адреса, поскольку идентификатор массива сам по себе является указателем на собственное начало. |  |
| В только что написанной нами программе оформим вывод массива на экран и поиск среднего арифметического в виде функции.  *Опишем функции printArray и* average *в которые передадим указатель на массив и его длину, т.к. в массиве не содержится сведений о его размере. Т.к. мы передаем в функцию указатель, то все действия которые описаны в этой функции будут происходить с массивом который мы создали в основной части программы и указатель на который передали в функцию, никакого копирования значений. Для корректной работы наших функций объявим в них итератор и изменим названия переменных на название аргументов.* | void printArray(int\* array, int length){  int i;  for (i = 0; i < length; i++)  printf("%d ", array[i]);  }  float average(int\* array, int length){  float result=0;  int i;  for (i = 0; i < length; i++)  result += \*(array + i);  return result/length;  } |
| Напишем вызов наших функций в основной части программы. *Запустим и убедимся, что все также работает корректно. Вот собственно и вся магия передачи массивов в функции.* | printf(“Our array is: ”);  printArray(arr, ARRAY\_LENGTH);  printf(“\n And the average is: ”);;  printf(“%f \n”, average(arr, ARRAY\_LENGTH)); |
| *На этом будем прощаться, н*а следующем уроке познакомимся с двумерными массивами и массивами указателей.  До новых встреч! |  |