Lesson 11

|  |  |
| --- | --- |
| Здравствуйте, уважаемые коллеги. Рад всех приветствовать на одиннадцатом уроке курса Основы языка С. Получив на предыдущих уроках представление об указателях и массивах, и вскользь упомянув строки, пришла пора изучить их подробнее. |  |
| Итак, что же такое строка. В повседневной жизни строка это набор или последовательность символов. Так вот в языке Си строка это тоже последовательность символов. А последовательности в языке Си представлены, вы наверное уже догадались, массивами и указателями. Никакого примитива Стринг в языке Си нет. Как бы нам этого не хотелось - его нет. Но есть одна хорошая новость, примитива Стринг не существует и в других языках (я имею ввиду, конечно, языки со строгой статической типизацией). Раз строка это массив или указатель - это всегда ссылочный тип данных. |  |
| Раз строка это набор символов, давайте немного вспомним что такое сиволы и как с ними работать. Как вам уже известно, символьная переменная это переменная типа char. В отличии от строки это примитивный тип данных, и к нему применимы все операции допустимые для примитивов, такие как присваивание, +, -, \*, / - хотя не все имеют смысл (я например не могу представить ситуацию в которой необходимо умножать или делить символы).  Здесь значением sym является 65 в кодовой таблице ASCII. Т.е. запись sym = 65 и sym = 'A' - абсолютно эквивалентны. Однако для улучшения читаемости кода лучше использовать вариант sym = 'A'. | char sym;  sym = 'A'; |
| Ну что, немного вспомнили, что такое символы, теперь переходим к строкам. Строками мы пользуемся с самого первого занятия: написав в двойных кавычках «Привет Мир», мы использовали строку. Строки принято называть типом данных, но мы-то знаем, что строка это массив, поэтому это не так. Строки в языке С можно описать двумя способами - как указатель и как массив из переменных типа char. *Давайте создадим строку в виде массива из char назовем ее string1 и запишем This is a string!" – это строка. Также создадим указатель, назовем его* string2 *и запишем This is also a string! – это тоже строка.*  У каждого из способов есть свои особенности. Так, например в массиве из переменных типа char мы можем записывать, удалять и изменять символы. *Давайте выведем наши строки в консоль и убедимся, что их можно так объявлять. Все работает. Теперь давайте изменим какой-нибудь символ в одной из строк.* Например, string1[5] = ‘X’; и символ с пятым индексом изменится. | int main (int argc, const char\* argv[]) {  char string1[256] = "This is a string!";  char\* string2 = "This is also a string!";  printf("%s \n", string1);  printf("%s \n", string2);  string1[5] = 'X';  printf("%s \n", string1); |
| Указатель на тип char нам такой возможности не даёт. Зато указатели на строки мы можем возвращать из функций. *Т.е. мы можем объявить функцию как указатель на char вернуть из нее строку и вывести в консоль.*  И это открывает перед нами широчайшие возможности по работе с текстами. | char\* helloFunction () {  return "Hello!";  }  printf("%s \n",\* helloFunction ()); |
| Параллельно с написанием нашей функции, приветствующей мир, давайте изучим некоторые стандартные возможности языка С для работы со строками. Например, специальную функцию, которая призвана выводить строки в консоль puts(); работает она очень похожим на printf() образом, но может выводить только строки, без каких-то других параметров, и добавляет символ конца строки. А также изучим специальную функцию gets() которая призвана считывать строки из консоли и записывать их в переменные.  Создадим изменяемую строку типа массив из char, назовём её name, передадим эту строку в функцию gets(), и выведем на экран результат, полученный из консоли. Запустим нашу программу, напишем имя и увидим его в консоли. | char name[255];  gets(name);  puts(name); |
| Теперь, казалось бы, мы можем поприветствовать пользователя нашей программы как следует, по имени. В нашей существовавшей функции приветствия внесём небольшие изменения. Создадим строку, в которой будем хранить приветственное слово, и в которую будет дописываться имя пользователя. Применим функцию склеивания строк. Поскольку склеивание - ненаучный термин, будем использовать слово конкатенация. И это слово подсказывает нам название функции, которую мы будем использовать : strcat(). Функция принимает на вход два параметра - строку, к которой нужно что-то прибавить, и строку, которую нужно прибавить. Логично предположить, что первая строка должна быть изменяемой (то есть являться массивом символов). Функция прибавит все символы второй строки в первую и вернёт указатель на изменённую строку. *Очень удобно. Запустим наш проект, введем имя и убедимся что все работает.* | char\* helloFunction(char\* name) {  char welcome[255] = "Hello, ";  return strcat(welcome, name);  }  int main(int argc, comst char\* argv[]) {  char name[256];  gets(name);  puts(helloFunction(name));  return 0;  } |
| Казалось бы, всё хорошо. Но, не тут-то было. Мы можем возвращать из функции только фиксированные строки, как в предыдущем примере. То есть, получается, нужно писать кейс, в котором содержатся все возможные имена, и оператором switch перебирать все возможные варианты ввода, иначе мы устраиваем утечку памяти, создавая болтающийся в воздухе указатель? Нет, нас это не устраивает. Что же делать? Какой бы мы ни создали указатель в функции - он перестанет существовать, как только мы выйдем из функции. Выход очень простой - раз указатель не идёт в мэйн, надо чтобы мэйн дал нам указатель.  *Добавим в аргументы функции указатель на выходную строку, заменим возвращаемое значение на void и напишем что out = strcat(welcome, name). Добавим в основную функцию массив char result [], который будет хранить результат и передадим в функцию helloFunction char name и char result. Запускаем, и не работает. А все дело в том, что мы не используем библиотечные функции, есть функция strcpy, которая не просто перекладывает указатель в определенную переменную, а копирует строку, теперь все работает.* | void helloFunction (char\* name, char\* out) {  char welcome[255] = “Hello, ”;  ~~out =~~ strcat(welcome, name);  strcpy(out, welcome);  }  int main(int argc, comst char\* argv[]) {  char name[256];  char result[256];  gets(name);  helloFunction(name, result)  puts(result);  return 0;  } |
| Если присмотреться - то все функции работающие со строками, именно так и делают - запрашивают источник данных и конечную точку, куда данные нужно положить. А кто мы такие, чтобы спорить с библиотечными функциями? Обратите внимание на то, что функции strcat() и strcpy() возвращает указатель на получившуюся строку, а наша функция возвращает void. Мы также можем возвращать указатель на строку, но тут встаёт вопрос о вызывающем функцию контексте, не потеряем ли мы получившийся указатель, и не будет ли он висеть в воздухе. Работа со строками до сих пор является очень и очень актуальной на программистских форумах, можете удостовериться. |  |
| Давайте рассмотрим ещё пару функций. Например, сравнение строк, функция strcmp, допустим, пусть именно меня программа приветствует как-то иначе. Функция возвращает отрицательные значения, если первая строка меньше второй, положительные, если первая больше второй, и 0 если строки равны. Это функция, которую удобно применять в условиях. И копирование строк, функция strcpy, также принимающую на вход две переменных - куда копировать, и что копировать.  *Если строки будут действительно равны, мы скопируем в строку с именем слово Master!. Скомпилируем, запустим, введем имя и все работает.* | void helloFunction (char\* name, char\* out) {  char welcome[255] = “Hello, ”;  if (strcmp("Ivan", name) == 0)  strcpy(name, "Master!");  strcat(welcome, name);  strcpy(out, welcome);  } |
| Из всех функций для работы со строками чрезвычайно часто используются atoi(); atof(); переводящие написанные в строке цифры в численные переменные внутри программы. atoi() – переводит в int, atof() – во float.  *Объявим переменную num, предложим пользователю ввести цифру, естественно в виде строки. Будем принимать ее при помощи функции gets Заведем переменную int number для хранения результата работы функции atoi. Затем давайте умножим результат сам на себя и выведем окончательное число в консоль. Запустим проект и убедимся, что все работает.*  Полный список функций для работы со строками можно посмотреть в заголовочном файле string.h. Описание и механика их работы легко гуглится, документации по языку очень много. | char num[64];  puts("Enter a number: ");  gets(num);  int number = atoi(num);  number \*= number;  printf("We powered your number to %d", number); |
| И напоследок еще пару слов об обработке символов. Функции для работы с символами содержатся в заголовочном файле stdlib.h. Естественно, наша программа может получить какие-то значения в виде строк. Не всегда же есть возможность использовать scanf(); - например, считывание из графических полей ввода даёт нашей программе значения в виде строк. В языке С есть функции для работы с каждым символом строки, например:  isalpha() – возвращает истину или ложь если символ является символом из алфавита;  isdigit() – если символ является цифрой;  isspace() – является ли символ пробелом;  isupper(), islower() – находится ли символ в верхнем или нижнем регистре; toupper(), tolower() – переводят символ в верхний или нижний регистр.  Можем использовать одну из них соответственно нашей задаче, допустим, пользователь может вводить своё имя как с заглавной буквы, так и всеми строчными. Уравняем оба варианта для нашей проверки одной строкой name[0] = tolower(name[0]); а после проверки вернём заглавную букву на место name[0] = toupper(name[0]); и удостоверимся что даже если мы напишем своё имя с маленькой буквы - программа напишет его с большой. Запустим и удостоверимся что все работает корректно. | // isalpha(), isdigit(), isspace(), isupper(), islower(), toupper(), tolower()  name[0] = tolower(name[0]);  name[0] = toupper(name[0]);  void helloFunction (char\* name, char\* out) {  char welcome[255] = “Hello, ”;  name[0] = tolower(name[0]); // то что добавлено  if (strcmp("Ivan", name) == 0)  strcpy(name, "Master!");  name[0] = toupper(name[0]); // то что добавлено  strcat(welcome, name);  strcpy(out, welcome);  } |
| На этом уроке мы познакомились со строками, способами работы с ними и немного улучшили понимание механизма работы программ. До встречи на следующем уроке. |  |