Lesson 5

|  |  |
| --- | --- |
| Здравствуйте, коллеги, рад вас приветствовать на пятом уроке курса Основы языка С.  На этом занятии мы рассмотрим такие важные операторы, как break и continue, цикл for и оператор множественного выбора switch.  Сегодня мы напишем программу, которая будет определять, является-ли введённое  пользователем число простым, и пару функций из математической библиотеки, на случай если  не захочется искать её, а использовать нужно. |  |
| Что такое простое число? Это такое число, которое имеет ровно два делителя с целочисленным результатом - единицу и само себя. | #include<stdio.h>  int main(int argc, char \*argv[]) |
| *Наша программа будет спрашивать у пользователя число и определять простое оно или нет. Для этого заведём переменную number и привычными нам функциями попросим пользователя ввести число, которое положим в number.* | int number;  printf("Input number:");  scanf("%d", &number);  printf("\n"); |
| *Для подсчетов нам понадобятся дополнительные переменные, например переменная которая будет хранить количество делителей, назовем ее d и переменная итератор i значение которой будет увеличиваться от 1 до введенного пользователем числа.* | int d = 0, i = 1; |
| Напишем цикл while и пройдемся от числа *i до введенного пользователем числа.*  *После того как мы переберем все возможные варианты от 1 до введенного пользователем числа выведем в консоль получившийся результат. Напишем введенное число, а дальше предоставим программе выбор ставить ли частицу «не» при помощи заменителя %s и тернарного оператора. В случае истинности условия d==2 тернарный оператор вернет пустую строку, если же нет, вернет частицу «не»* | while (i<= number){  }  printf("Введенное число %d %s является простым", number, (d==2) ? ”” : ” не”); |
| *Итак, что же будет происходить в каждой итерации цикла? На каждой итерации цикла мы будем проверять делится ли введенное пользователем число на текущее значение итератора.*  *Если остаток от деления исходного числа на текущий делитель равен 0 то мы увеличим счетчик количества целочисленных делителей для данного числа.*  *Если количество целочисленных делителей не изменилось, то мы прекратим текущую итерацию цикла при помощи ключевого слова continue. Оператор continue передаст управление в логическую конструкцию цикла.*  *Если количество целочисленных делителей достигнет 3, то мы разорвем цикл при помощи ключевого слова break.* | while (i<= number){  if (number% i == 0){  d++;  else  continue;  if (d == 3) break;  } |
| *Итого должно получиться:* | int number;  printf("Input number:");  scanf("%d", &number);  printf("\n");  int d = 0, i = 1;  while (i<= number){  if (number% i == 0){  d++;  else  continue;  if (d == 3) break;  }  printf("Введенное число %d %s является простым", number, (d==2) ? ”” : ” не”); |
| *Запустим нашу программу и убедимся что работает она некорректно. Это произошло потому, что переменная итератор, находящаяся в управляющей конструкции цикла не изменяется в теле цикла. Если бы у нас не было условия выхода через оператор break, мы бы написали бесконечный цикл.*  *Увеличим переменную итератор сразу после проверки, и убедимся в корректной работе программы.* | if (number% i++ == 0){ |
| *Давайте подытожим:*  Оператор continue нужен для того, чтобы программа проигнорировала оставшиеся действия на текущей итерации цикла, часто используется для отбрасывания неподходящих значений в больших наборах данных, как в нашем случае, например, нам не нужно выполнять никаких действий и проверок, если переменная d не изменилась. Оператор continue просто напросто передаёт управление логической конструкции цикла.  Оператор break используется для того, чтобы выйти за пределы цикла, то есть мы сразу попадаем к следующему после цикла оператору, без передачи управления логической конструкции. То есть, очевидно, например, что когда делителей стало три - число точно не простое. |  |
| Зачастую, складываются ситуации, когда мы точно знаем, сколько итераций цикла нам понадобится, например, когда мы последовательно проверяем содержимое созданного нами числового ряда, или заполняем значениями таблицы, границы которых заранее известны. В конце концов, для подсчёта среднего арифметического. В этих ситуациях принято использовать  цикл for.  Это цикл с предусловием, где заранее отведено место для инициализации  переменной-счётчика, условия *захода в следующую итерацию цикла* и изменения переменной счетчика. *В более поздних версиях Си появилась возможность объявлять переменную счетчик прямо в управляющей конструкции.*  *В классическом Си необходимо объявить переменную счетчик заранее, а в управляющей конструкции задать ей начальное значение.*  *Условие захода в следующую итерацию цикла это логическое условие которое может принимать два значения true и false – если условие = true – идем на следующую итерацию, если false – выходим из цикла.*  *Например, такой цикл выведет нам числа от 0 до 4. На каждой итерации мы будем инкрементировать значение i, соответственно. пока логическое условие верно* i < 5 *мы будем заходить в тело цикла, а как только i станет равным 5, логическое условие вернет false и мы выйдем из цикла.* | int i;  for(i = 0;)  int i;  for(i = 0; /\*условие\*/; i++/i--){  }  for(int i = 0; /\*условие\*/; i++/i--){  }  int i;  for(i = 0; /\*условие\*/; i++/i--){  }  int i;  for(i = 0; i < 5; i++){  printf(“%d \n”, i)  } |
| Решим немного более сложную задачу возведения числа в степень.  Язык С не предоставляет оператора возведения в степень по умолчанию, как это делают языки высокого уровня, поэтому для этой математической операции нам нужно подключать специальную математическую библиотеку. Но я считаю это излишним, ведь не так уж и сложно написать собственную программу, которая-бы делала это. Как известно, возведение в степень это последовательное умножение основания на само себя указанное количество раз. А раз заранее известно сколько раз мы будем умножать основание само на себя, это работа для цикла for. |  |
| *Закомментируем предыдущий код.*  *Объявим переменную итератор i , переменную основание, переменную показатель и переменную результат, куда будем складывать результаты наших вычислений.* | int base;  int significative;  int result = 1;  int i; |
| *Логика работы следующая – результатом работы будет совокупность результатов предыдущих итераций умноженных на основание.*  *Запись вида* result = result\*base; можно сократить до result \*= base;  И выведем результаты работы цикла в консоль. | for(i = 0; i < significative; i++){  result = result\*base;  }  for(i = 0; i < significative; i++){  result\*= base;  }  printf(“%d powered by %d is %d \n”, base, significative, result); |
| *Конечно мы могли бы спросить у пользователя какое число и в какую степень он хотел бы его возвести, Для этого применим уже привычные нам конструкции.* | printf(“Enter base: “);  scanf(“%d”, &base);  printf(“Enter significative: “);  scanf(“%d”, &significative); |
| *Запустим нашу программу, введем для базы значение 2, для показателя 10, вообще степени двойки самые популярны в программировании числа. Запустим и убедимся, что наша программа работает корректно. 2 в степени 10 равно 1024.* |  |
| *Раз уж я упомянул математическую библиотеку, давайте посмотрим.* | Здесь показ мат. библиотеки. Может не нужно? |
| *Идем дальше.* В качестве бонуса сегодняшнего урока мы поговорим об операторе множественного выбора switch. Тем более, что теперь мы обладаем всеми необходимыми для этого знаниями. Оператор множественного выбора используется когда мы хотим описать действия для какого-то ограниченного количества условий. В отличие от оператора if который может использоваться также и для проверки диапазонов значений. Удобство применения того или иного оператора, естественно, зависит от задачи. Для примера, напишем свой собственный маленький калькулятор.  *Для работы над этим примером создадим новый файл, привычными движениями напишем #include <stdio.h>*  *int main( int argc, char \*argv[]),*  *return 0; и приступим*  *Заведем в нашей программе переменные типа float, которые будут хранить операнды – числа над которыми будут производиться действия. И переменную типа инт для хранения операции.*  *Спросим у пользователя, какие числа он хочет посчитать, для этого используем уже крайне привычную нам связку* printf – scanf.  *Далее таким же образом предложим пользователю ввести действие, действие мы закодируем в виде чисел 1 – сложение, 2 – вычитание, 3 – умножение и 4 – деление. Здесь вступает в силу тот факт, что на 0 делить нельзя, поэтому нам нужно не дать пользователю ввести в качестве второго операнда «0», если в качестве оператора он выбрал деление. Это можно оставить на усмотрение пользователя, но мы же сознательные программисты, поэтому не дадим пользователю сломать нашу программу. Совершенно очевидно, что просто скопировать ввод будет неправильным действием.*  *Для того чтобы не дать пользователю сделать неправильный ввод введем в прорамму условие:*  *Если if(operator == 4) – пользователь выбрал деление, используя цикл do – while будем просить его ввести второй операнд отличный от 0. И если (operator !=4), то тоже попросим пользователя ввести второе число.*  *Если мы воспользуемся нашими знаниями на текущий момент, то мы напишем примерно следующее: если оператор равен 1, то делать это, если 2 то делать вот это, и так далее описывали бы все возможные действия. Но существует гораздо более удобный оператор switch. Воспользуемся им относительно переменной* operator. Разделим действия оператора *switch на несколько так называемых кейсов.*  Оператор switch последовательно проверит входящую переменную на соответствие описанным в кейсах значениях. В случае, если значение совпадёт, будет выполнен блок кода до оператора break, если же значение переменной не совпадёт ни с одним из описанный в кейсах, выполнится блок по умолчанию default.  Важно помнить, что в случае отсутствия в case оператора break, программа будет выполнять последующие операторы, пока не найдёт break или пока не закончится конструкция switch, то есть пока не встретится закрывающая фигурная скобка. *Опишем присваивание результата в переменную result и последующий вывод результата в консоль.*  *Запустим наш калькулятор и убедимся что все работает. Сымитируем нерадивого пользователя и повводим на 4 оператор 0, программа естественно не даст нам этого сделать.* | #include <stdio.h>  int main( int argc, char \*argv[]){  float first;  float second;  float result;  int operator;  printf("Enter first operand: ");  scanf("%f", &first);  printf("/nEnter 1 for +, 2 for -, 3 for \*, 4 for / operation");  scanf("%d", &operator);  ~~printf("/nEnter second operand: ");~~  ~~scanf("%f", &first);~~  if(operator == 4){  do {  printf("/nEnter second operand: ");  scanf("%f", &second);  } while (second == 0);  } else {  printf("/nEnter second operand: ");  scanf("%f", &second);  }  ~~if(operator = 1){~~  ~~} else if (operator = 2){~~  ~~}~~  switch(operator){  case 1:  result = first + second;  printf("Result is: %f \n", result);  break;  case 2:  result = first - second;  printf("Result is: %f \n", result);  break;  case 3:  result = first \* second;  printf("Result is: %f \n", result);  break;  case 4:  result = first / second;  printf("Result is: %f \n", result);  break;  default:  printf("Unknown operator");  }  return 0;  } |
| На основе этого кода можно описать любые виды меню, описать поведение программ, которые должны опираться на получаемые извне команды или описывать конечные автоматы |  |
| На следующем уроке мы наконец-то подробно поговорим о функциях и заголовочных файлах, до новых встреч. |  |