Введение в программирование

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

23 марта 2017 г.

Версия: 0.16

Аннотация

Что есть программирование. Откуда что берется.

Цель: Научится писать простейшие программы, которые могут вводить числа в консоли, выполнять арифметические действия, в частности, над введенными числами, сохранять промежуточный результат в переменные и выводить что-то в консоль, например, результат вычисления.

Внимание! Данная заметка является предварительной!

1 Введение

Раньше каждое устройство делалось с нуля... С увеличением количества входов увеличивалось в размерах само устройство. С точки зрения пользователя и раньше существовали по отдельности: калькулятор, портативный музыкальный проигрыватель, портативный (по размеру как крупный сотовый телефон) телевизор, устройство спутниковой навигации GPS и тому подобное. Сейчас за счет унификации узлов их эквивалент обеспечивается одним устройством – сотовым телефоном.

Как вычислить 2*3+5*7 на калькуляторе? Нужна память, а именно – сначала вычисляем 2*3 и сохраняем в памяти. Далее вычисляем 5*7, а потом прибавляем к этой величине сохраненное в памяти. Без памяти это выражение не вычислить.

Упражнение: придумайте арифметическое выражение, для вычисления которого потребуется две ячейки памяти.

Существует парадигма функционального программирования. Мы же фактически рассматриваем императивное программирование. В данном подходе необходимо указать последовательность действий, приводящих к достижению цели. Как следствие, необходима память для хранения промежуточных результатов.

1.1 Первая программа

Работающая программа – это программа, которая как-то о себе дала знать: чем-то мигнула, что-то включила или выключила и тому подобное. В противном случае отсутствовал бы смысл в запуске программы. Так, если она что-то и вычислила и уж тем более вычислила правильно, то пока она это не «покажет» пользователю, бессмысленно говорить о работе программы.

Первая программа В связи с выше описанным, простейшая программа – это программа, которая что-то выводит на «экран», а точнее в консоль. Поэтому традиционно программировать начинают со следующего примера программы:

```
1  //Программа с которой принято начинать программирование.
2  //Предыдущая строчка как и эта являются комментариями.
3  //Комментарий - текст идущий после двух косых черт (//).
4  //Они не влияют на результат работы самой программы.
5  //Они пишутся для человека, читающего твой код.
6  //В тексте программы допустимы также и пустые строки:
7
8  #include <stdio.h>//Ссылка на вспомогательный код,
9  //отвечающий за ввод и вывод.
10 //stdio - расшифровывается как STandarD Input/Output.
11 //Есть и другие, например: math.h, stdlib.h и так далее.
12
13 //Программа состоит из одного единственного "блока".
14 int main()
15 {
16   //Следующая строчка выполнит печать текста.
17   printf("Hello world!\n");//Выполняем печать.
18   //В консоли будет напечатана строчка: "Hello world!"
19   //и будет курсор перемещен в начало следующей строчки.
20   return 0;//Завершаем работу программы.
21   //Отмечу, что после всех действий используется
22   //точка с запятой (;).
23 }
```

Отмечу, что числа, расположенные слева от текста, не являются частью программы, а необходимы для удобства пояснения её устройства. Так, всюду далее при необходимости будет даваться ссылка на номера соответствующих строчек. Иногда возможно применение сокращения «стр.».

Пробелы, форматирование текста, в частности, отступы не влияют на работоспособность программы. Тем не менее, в случае нарушения принятого форматирования текста программы оная не будет зачтена.

Строчки 1-6 являются комментарием к программе. Они применяются для пояснения происходящего в программе человеку, читающего код твоей программы. Смысл комментария строчки 3 заключается в том, что текст комментария начинается с двойного символа косая черта («//») и не обязательно с начала строки (см. строчка 17).

Простейший вывод в консоль Написать программу на чистом Си, которая бы чего-то выводила, например, на экран невозможно. Необходим вспомогательный код, который обеспечивает связь программы с "реальным миром"(с операционной системой). Базовый набор такого рода вспомогательного кода находится в библиотеке `stdio.h`, название которой расшифровывается как **st**andar**d** **i**nput/**o**utput, т.е. стандартный ввод/вывод. Данная библиотека содержит вспомогательный код, отвечающий за ввод данных (например, чисел) в программу и вывод из программы текста, численного результата (тех же чисел). В частности, в строчке 17 вызывается вспомогательный код *printf*, который обеспечивает печать в консоль строчки указанной в соответствующих кавычках, например, как в "Hello world!".

Помимо ввода и вывода, при написании программ необходимо вызвать функции, которые в принципе можно написать самому, но на это может уйти много времени и, главное, усилий, что может привести к её провалу. В таких случаях также прибегают к вспомогательному коду из различных библиотек, например, для вычисления тригонометрических функций. Последнее означает, что в программах чуть посложнее строчек на подобии 8ой становится все больше и больше.

Блок действий Любая программа начинает свою работу с «блока» кода под названием `main`. Отмечу, что именно такие именные блоки кода (как в обсуждаемой программе) в языке Си называются *функциями*, так как в общем случае такие блоки имеют входные аргументы, которые указываются в скобках, и возвращают некий результат работы, где возвращаемый тип указывается перед названием функции. В общем случае данное обсуждение является предметом отдельной заметки.

В простейшем случае (как в рассматриваемой программе), функция состоит только из списка подряд идущих действий (в нашей первой программе см. строчки 17 и 20). Подчеркну, что каждое из действий в Си завершается точкой с запятой (;). При этом блок кода (состоит из списка действий) сам по себе не является действием, поэтому и не завершается этим символом.

Результат выполнения программы (в данном случае, единственной функции `main`) заключается в выполнении перечисленных действий в *теле* блока (см. стр.

16–22). Завершением (выходом из) функции считается выполнение действия `return` (по сути, возврат из функции её значения), которое возвращает вычисленное значения функции, т.е., в данном случае, результат выполненной работы.

В сложных функциях (т.е. речь идет не от `sin`, `cos` и тому подобное) принято возвращать отрицательное число в случае ошибки и неотрицательное число в случае успеха. Бывает, что величина самого значение характеризует дополнительную информацию. Например, объем успешно обработанных данных или причину возникновения ошибки. Ноль (0) возвращается в случае успеха, но без каких либо подробностей.

Контрольные упражнения Упражнение. Запусти указанную программу. Поиграйся с выводимым текстом. Размножь строчку 17, поизменяй текст, находящийся в кавычках.

Упражнение. Что произойдет, если убрать комбинацию «\n»? Что произойдет, если добавить «\n», «\t» или «\r» внутри/в конце текстовой строчки?

1.2 Язык Си в качестве калькулятора

Выводить просто текст не интересно, тем более, что он фактически в точности повторяет то, что написано в самой программе. Нужно заставить компьютер «поработать», ведь именно в этом заключается его основной смысл. Например, сделать то, что человеку не всегда просто: выполнить некоторые арифметические вычисления.

Тип числа С точки зрения компьютера (точнее, процессора входящего в его состав) числа бывают целыми и вещественными. Указанные типы чисел преследуют разные цели: целые для счетчиков, т.е. целочисленного подсчета чего либо, например, степень многочлена, а вещественные для численных вычислений, т.е., в согласии с предыдущем, поиск корней многочлена.

Последнее оказало влияние и на язык Си, так, в нем вещественные числа имеют десятичную точку (именно точку, а не запятую, язык Си же не русифицирован): например,

$$10.1, -15.78, 43., .2$$

и тому подобное, а целые – не имеют оную: например,

$$10, -15$$

и так далее. Как будет ясно из ниже написанного, крайне важно отмеченные типы чисел не путать.

Над указанными числами доступны (встроены в сам язык Си, т.е. являются его частью) как минимум следующие стандартные арифметические операции: +, -, * и /. Сразу отмечу, что в отличии других языков программирования, в СИ отсутствует встроенная операция возведению числа в степень.

Если тип хотя бы одного из операндов (у каждой операции их два) – вещественное число, то второе преобразуется в вещественное. Последнее крайне важно, так как смысл отмеченных вычислительных операций зависит от типа операндов, при этом, численный результат может существенно различаться. Так, для вещественных чисел вычислительные операции +, -, * и / действуют в соответствие с общепринятыми математическими законами, а для целых имеется своя специфика.

В текущих заметках для обозначения операции вычисления выражения (т.е. приведения его к числовому значению) будет использоваться символ ~. Подчеркну, что данная операция не имеет отношение к языку Си. Она будет использоваться исключительно для пояснения сути происходящего в программе, а именно – для рассмотрения внутреннего устройства программы. В виду последнего, для следующей программы будет уместно прояснить, что (7. + 4.)/5.~2.2, ровно как это и сделано в строчке 8 самого текста программы.

Вывод чисел в консоль Приведем пример программы, которая выводит (печатает) числа в консоль:

```
1  //Программа-калькулятор.
2  #include <stdio.h>
3  #include <math.h>//Добавлено ради математических функций:
4  //sqrt, pow, sin, cos, asin, acos, tan, atan и других.
5
6  int main()
7  {
8    //Печать значения выражения (7.+4.)/5.~2.2:
9    printf("Value ~ %f\n", (7.+4.)/5.);//Вещественное число
10   //В консоль напечатается:
11   //Value ~ 2.2
12   printf("Value ~ %f\n",//Команды можно разбивать.
13     3*sqrt(1.8+.5) );//Вызов функции sqrt.
14   //В консоль напечатается 4.5497, а именно:
15   //Value ~4.5497
16   //Печать значения выражения (7+4)/5~2:
17   printf("Value ~ %d\n", (7+4)/5);//Числа целые!
18   //В консоль напечатает число 2.
19   return 0;//!!Все команды завершаются символом ;!
```

```
20 }
```

Печать вещественных чисел осуществляется комбинацией символов `%f` (см. 9), где `%f` от слова **f**loat (плавающая точка). Причем, число фактически вставляется в то место, где и находится эта самая комбинация символов (`%f`). Поясню, что при выполнении действия строчки 9, сначала вычисляется значение выражения (7. + 4.)/5., а уже потом вычисленное значение (2.2) вместе с самой строчкой («"Value %d\n"») передается в функцию printf, т.е. передается её управление, что приводит к печать указанной информации в консоли.

1.3 Арифметические выражения

В выражениях вместо чисел можно использовать и математически функции (берущиеся, например, из библиотеки math.h, для этого нам и нужна строчка 3). Например, в выражении $3 * sqrt(1.8 + .5)$ (см. строчку 13), сначала будет вычислено значение $1.8 + .5 \sim 2.3$, далее оно будет подставлено в *sqrt* и вычислено $sqrt(2.3)$ (квадратный корень из числа 2.3), а потом и оно будет использовано (подставлено) в самом выражении. Система далее продолжит приводить выражение к числу (как в точности как в математике) ровно как и в прошлом (см. строчку 9) варианте. Такой способ вызова вспомогательного кода (функций) называется *вызов по значению*, т.е. сначала вычисляются значения всех аргументов функции, а уже потом к ним производится применение вспомогательного кода (вызов функции).

Вещественные числа Другие функции указаны в комментарии под строчкой 4. Отмечу, что, во-первых, все указанные функции выполняют действия над вещественными числами, т.е. и аргументы и возвращаемое значение имеет вещественный тип. Во-вторых, функции acos, atan и тому подобные являются обратными соответственно к функциям cos, tan, и, в-третьех, у всех тригонометрических функций (sin, asin, cos, tan и тому подобных) угол измеряется в радианах.

Функция $\mathrm{pow}(a, b)$ возводит число a в степень b, т.е. $\mathrm{pow}(a, b) = a^b$. log, ln.

Целые числа При вычислениях с целыми числами имеется определенная тонкость, а именно – хоть первые 3 операции действуют ровно также как и для вещественных – в соответствие с ожиданием – (точнее они производятся в соответствующем кольце вычетов), последняя имеет особенность. Так, деление выполняется в целочисленной арифметике, т.е., например, $11/5 \sim 2$, а $(-11)/5 \sim -2$. Для целых чисел доступно также и крайне важная и отличительная операция % – деление с остатком. Приведем примеры для прояснения сути:

$$11\%5 \sim 11 - 2 * 5 \sim 1, 11\%(-5) \sim 11 + 2 * (-5) \sim 1,$$

$$(-11)\%5\sim(-11)+2*5\sim-1, (-11)\%(-5)\sim(-11)-2*(-5)\sim-1.$$

В конце концов отметим, что печать целых чисел осуществляется комбинацией `%d` (см. строчку 17), где `%d` от слова **d**ecimal (число).

Контрольные упражнения Упражнение 1. Поиграйся с вещественным арифметическим выражением, а именно – замени выражение $(7.+4.)/5.$, используемое в программе, на собственное.

Упражнение 2. Поиграйся с целочисленным арифметическим выражением. Убедись в правильном понимании операции взятия остатка (%).

Упражнение 3. Воспользуйся отмеченными ранее математическими функциями. Вычисли длину гипотенузы, зная катет и угол.

Упражнение 4. Какую ошибку выдаст компилятор, если перепутать `%d` с `%f` и наоборот? Запомни выводимую ошибку на будущее.

1.4 Переменные

Обсудим конструкцию языка Си, соответствующую памяти (как это было в примере с калькулятором). В языке Си памяти соответствует понятие *переменной*, причем, *именной*, дабы можно было указывать, о какой именно переменной идет речь. В отличии от некоторых других языков программирования, в Си перед тем как переменную можно будет использовать её нужно сначала создать (*объявить*). Замечу, что с точки зрения компьютера/процессора имен нет, есть адреса/индексы в памяти. Имена у ячеек памяти появляются благодаря языкам программирования.

Создание переменных Начнем с переменных по своему типу соответствующих обсуждаемым ранее вещественным и целым числам. Вещественным числам соответствует тип double (или float, имеющих меньший диапазон/точность), а целым – тип int.

Переменная объявляется (создается, т.е. выделяется память) следующей конструкцией (фрагмент кода):

```
1 //Объявление переменной:
2 //тип данных       имя переменной;
3 //например:
4 int a;//Объявление переменной, a, целочисленного типа.
5 double b;//Объявление переменной, b, вещественного типа.
6 int c;//Объявления можно чередовать.
7 int d, e, f;//Объявлять можно сразу несколько переменных.
```

После создания переменной её можно использовать, а именно – главное, что для переменных вводится операция присвоения, т.е. сохранение некой величины в переменную, и неявная операция считывания/извлечения значения из переменной. Для присвоения в языке Си используется операция =.

Отмечу, что первоначального значения переменные не имеют. В конкретных реализациях (т.е. в компиляторе) обычно содержат *мусор*(произвольное числовое значение, но могут и 0). На конкретное значение рассчитывать не стоит. При вычислениях нарушение последнего может привести к странным итоговым значениям, а то и к принудительному (иначе, странному) завершению программы операционной системой.

Присвоение Для присвоения используется конструкция:

```
1  int a;//Объявление переменной a целочисленного типа.
2  //переменной a ещё не было ничего присвоено, поэтому
3  //a~мусор, т.е. a содержит мусор!
4  a=3;//Переменной a присвоено значение 3.
5  //a~3, т.е. содержит значение 3.
6  double b = 7.2;//Значение можно присвоить при объявлении.
7  //b~7.2.
8  a=9;//В переменную можно сохранить новое значение!
9  //Теперь, a~9. Значение, которое ранее было в a, затерто.
10 b+2=4;//Так нельзя. Это будет ошибкой. Уравнение не будет
11 //решено! Слева должна быть именно переменная.
12 5=a;//Аналогично, и это будет ошибкой, то куда
13 //присваиваем, должно быть с лево от знака =.
14 a+b=7;//Уж тем более это ошибочно.
```

На любом шаге программы в переменную можно сохранить новое значение. Следовательно, в отличии от выкладок из математики, механики и др., бессмысленно говорить о значении переменной без указания номера шага хода выполнения программы. В принципе, можно построить график из значений переменных, где по вертикали значение, а по горизонтали номер шага. Забегая вперед можно уточнить, что в таком случае также нужно указывать и вычислительный блок, о котором идет речь. Просто в данной программе он пока один. Но подобные графики почти никогда явно не строятся.

Извлечение значения Если переменная используется в математическом выражении, то при вычислении используется значение ранее сохраненное в этой переменной. При применении оператора присвоение (=), сначала вычисляется значение правой части, а потом результат присваивается левой. Поэтому одна и та же

переменная может фигурировать и справа и с лева от оператора =. При применении переменных в выражениях справа от оператора присвоения производится только извлечение самих значений.

```
1  double a, b, c, d;//Объявление переменных.
2  //a, b, c, d ~ мусор.
3  a=2.1;//Присвоили переменной a значение 2.1
4  //a ~ 2.1
5  b=a;//Присвоили переменной b значение переменной a.
6  //b ~ 2.1
7  //Переменный a и b никак с друг другом не связаны:
8  a=1.5;//Присвоили новое значение переменной a.
9  //a ~ 1.5, но
10 //b ~ 2.1
11 //С правой стороны равенства можно использовать выражения:
12 c=a+1.1;
13 //c ~ 2.6
14 //в том числе сложные арифметические выражения:
15 c=a+1.1*c-b;
16 //c ~ 2.26.
17 //В арифметическом выражение справа от знака присвоения
18 //может быть применена и сама переменная из левой части.
19 c=c+a;//Сначала вычисляется значение правой части ~ 3.76.
20 //c ~ 3.76, и потом присваивается левой части.
```

1.5 Ввод/вывод

Ввод Помимо вывода программа должна уметь вводить данные, т.е. в нашем случае числа. В противном случае, результат работы программы фактически неизменяемый и поэтому неинтересен. Более того, иначе результат можно было бы подтасовать...

Как выполнить ввод, а фактически – как присвоить значение переменной уже после запуска программы? Делается это посредством функции scanf:

```
1  double a;
2  //Считываем число введенное пользователем,
3  //обращаем внимание на &.
4  scanf("%lf", &a);//%lf используется для double.
5  //a ~ введенное пользователем значение.
6  int b;
7  scanf("%d", &b);//%d используется для int.
```

```
8 //b ~ введенное пользователем второе значение.
```

Символ амперсант (&) перед переменной обязателен. Пока пускай это будет как элемент магии. Обращаю внимание на отличие от функции `printf` – комбинация «%d» используется как и ранее для целых чисел, а вот для вещественных теперь уже – «%lf».

Вывод Вывод можно делать достаточно сложным. Частично это уже было показано ранее. В частности, одним вызовом функции printf можно вывести не только значение больше одной переменной, но и ещё и сопутствующий текст. Следующий код показывает эквивалентность.

```
1 double a=1.1;
2 int b=4;
3 printf("a ravno %lf, ",a);
4 printf("b ravno %d\n",b);
5 //Эквивалентно можно переписать как
6 printf("a ravno %lf, b ravno %d \n", a, b);
```

Первый аргумент функции printf – текстовая строка – называется *форматной*. Она указывает, какой текст необходимо вывести. Символ процент (%) является экскейпным, т.е. он не выводится на печать. Вместо этого он указывает на дополнительное действие, а именно – сюда нужно вставить значение соответствующей переменной из списка аргументов (после форматной строки). Последовательно идущие процентные комбинации символов заменяются на значения в соответствие с их порядком в списке аргументов.

Контрольные упражнения Упражнение. Напиши программу, которая по известной формуле что-либо считает, а именно – программа приветствует пользователя, запрашивает необходимые числа, вводит числа, выполняет некие вычисления и выводит результат.

1.6 Итого

Выжимка По итогам данной заметки теперь ты имеешь/знаешь как:

- выводить текст, числа и значение переменных в консоль
- (не)перемещать курсор на следующую строку консоли
- вводить численные данные путем присвоения их переменным
- выполнять арифметические действия как над числами, так и над переменными

Эталонная программа Приведем в согласии с данной заметкой фрагмент эталонной программы:

```
1 //При запуске программа должна написать в чем её цель.
2 //В данном случае, вычислить периметр и площадь круга.
3 printf("Calculate the perimiter and area of a circle\n");
4 printf("Please enter the radius:");//Приглашение на ввод.
5 double r;
6 scanf("%lf", &r);//Считываем введенный радиус. см. &!
7 int area = r*r*M_PI;//Результат можно сохранить в
8 //переменную, а можно сразу подать на печать.
9 printf("area ~ %f, perimiter ~ %f\n", area, 2*M_PI*r);
```