1. Введение

Зачем нужно программирование?

Иногда создается впечатление, что все существующие задачи могут быть решены с помощью готовых программ для компьютеров. Во многом это действительно так, но опыт показывает, что всегда находятся задачи, которые не решаются (или плохо решаются) стандартными средствами. В этих случаях приходится писать собственную программу, которая делает все так, как вы этого хотите.

Языки программирования

Процесс работы компьютера заключается в выполнении некоторого набора команд, которые выполняются в определенной последовательности. Машинный код команд, реализованный в двоичной системе счисления, указывает, какую именно действие должен выполнять центральный процессор. Итак, чтобы заставить компьютер решать ту или иную задачу, надо задать последовательность двоичных кодов соответствующих команд.

Писать такие последовательности очень сложно, потому что, кроме необходимости помнить двоичные коды команд, программист должен держать и памяти и двоичные коды адреса данных, которые использовались во время выполнения программы. Для облегчения этого процесса и предназначены языки программирования — способы записи алгоритмов, которые бы в более наглядном виде подавали последовательность действий компьютера.

Язык программирования Си создан в 1972 г. Деннисом Ритчи для написания операционной системы UNIX. Сам язык, однако, не связан с какой-либо одной операционной системой или машиной; и хотя его называют языком системного программирования, так как он удобен для написания операционных систем, он может использоваться для написания любых больших вычислительных программ, программ для обработки текстов и баз данных.

Компиляторы языка Си работают почти на всех типах современных ЭВМ в операционных системах UNIX, Linux, MS-DOS, OS/2, Windows, Windows NT и т. д. Язык программирования C оказал существенное влияние на развитие индустрии программного обеспечения, а его синтаксис стал основой для таких языков программирования, как C++, C#, Java, D и др.

Как запустить программу, написанную на языке программирования

Текст программы, написанный на каком-то языке программирования, например на C, называется *исходным кодом*. Исходный код пишется в текстовом редакторе и сохраняется с нужным расширением. Обычно файлы, написанные на языке C, имеют расширение .c.

Для запуска программы на языке C сначала необходимо перевести ее из текстовой формы, понятной человеку, в форму, понятную машине.

Последовательность символов, понятная машине, называется *машинным* или *исполняемым кодом*. Например, файл с расширением .exe представляет собой исполняемый код.

Программа, которая осуществляет перевод из исходного в исполняемый код, называется *компилятором*.

Первая программа на Си

Единственный способ выучить новый язык программирования — это писать на нем программы. При изучении любого языка первой, как правило, предлагают написать приблизительно следующую программу:

```
Напечатать слова Hello, world!
```

Си-программа, печатающая "Здравствуй, мир!", выглядит так:

```
#include <stdio.h>

main()

{

    printf ("Hello, world!\n");

}
```

Программа на Си, каких бы размеров она ни была, состоит из *функций* и *переменных*. Функции содержат *инструкции*, описывающие вычисления, которые необходимо выполнить, а переменные хранят значения, используемые в процессе этих вычислений. Функции в Си похожи на подпрограммы и функции Фортрана или на процедуры и функции Паскаля. Приведенная программа — это функция с именем main. Обычно вы вольны придумывать любые имена для своих функций, но "main" —

особое имя: любая программа начинает свои вычисления с первой инструкции функции main.

Обычно main для выполнения своей работы пользуется услугами других функций; одни из них пишутся самим программистом, а другие берутся готовыми из имеющихся в его распоряжении библиотек. Первая строка программы:

```
#include <stdio.h>
```

сообщает компилятору, что он должен включить информацию о стандартной библиотеке ввода-вывода. Эта строка встречается в начале многих исходных файлов Си-программ.

Один из способов передачи данных между функциями состоит в том, что функция при обращении к другой функции передает ей список значений, называемых аргументами. Этот список берется в скобки и помещается после имени функции. В нашем примере main определена как функция, которая не ждет никаких аргументов, что отмечено пустым списком ().

Инструкции функции заключаются в фигурные скобки {}. Функция main содержит только одну инструкцию

```
printf ("Hello, world!\n");
```

Функция вызывается по имени, после которого, в скобках, указывается список аргументов. Таким образом, приведенная выше строка — это вызов функции printf с аргументом "Hello, world!\n". Функция printf — это библиотечная функция, которая в данном случае напечатает последовательность символов, заключенную в двойные кавычки.

В Си комбинация \n внутри строки символов обозначает символ новой строки и при печати вызывает переход к левому краю следующей строки. Если вы удалите \n (стоит поэкспериментировать), то обнаружите, что, закончив печать, машина не переходит на новую строку. Символ новой строки в текстовый аргумент printf следует включать явным образом. Если вы попробуете выполнить, например,

```
printf ("Hello, world!
");
```

компилятор выдаст сообщение об ошибке.

`#include <stdio.h>`	Включение информации о стандартной библиотеке.
`main()`	Определение функции с именем main, не получающей никаких аргументов.
`{`	Инструкции main заключаются в фигурные скобки.
`printf ("Hello world!\n");`	Функция main вызывает библиотечную функцию printf для печати заданной последовательности символов; \n — символ новой строки.
`}`	

Интегрированная среда разработки

Чтобы получить из исходного кода выполняемую программу, необходимо использовать компилятор. На современном уровне все этапы создания, компиляции (она проходит в два этапа — трансляция и компоновка), отладки и проверки программы объединены и выполняются внутри специальной программы-оболочки, которую называют *интегрированная среда разработки* (*IDE – integrated development environment*).

В нее входят текстовый редактор, транслятор, компоновщик и отладчик.
В этой среде вам достаточно набрать текст программы и нажать на одну клавишу, чтобы она выполнилась (если нет ошибок).

Компиляция и запуск программы в среде Dev C++

Dev-C++ — бесплатная интегрированная среда разработки приложений для языков программирования C/C++.

Это программа с открытым исходным кодом.

Из-за своего небольшого размера и простоты установки является идеальным средством для людей, которые только делают первые шаги в программировании. В тоже время, программа обладает всеми необходимыми функциями для разработки небольших проектов.

Скачать программу можно здесь:
http://www.bloodshed.net/dev/devcpp.html

Для компиляции и выполнения программы в Dev-C++ используется следующие команды:

Скомпилировать и выполнить (F11) — при написании кода программы удобнее всего пользоваться этой командой. Если в коде содержаться ошибки, то выведутся сообщения об ошибках, помогающие их исправить, если ошибок не содержится, то программа сразу запустится и можно будет проверить ее работу.

Скомпилировать (F9) — только компиляция.

Выполнить (F10) — эта команда позволяет многократно запускать программу без повторной компиляции кода.

Если в программе есть ошибки, вы увидите в нижней части экрана оболочки сообщения об этих ошибках (к сожалению, на английском языке). Если щелкнуть по одной из этих строчек, в тексте программы выделяется строка, в которой транслятору что-то не понравилось.

При поиске ошибок надо помнить, что часто ошибка сделана не в выделенной строке, а в предыдущей — проверяйте и ее тоже; часто одна ошибка вызывает еще несколько, и появляются так называемые наведенные ошибки.

После удачный компиляции в той же директории, куда вы сохранили исходный код программы, появится файл с тем же именем и расширением .exe.

Его можно запускать как из среды разработки командой «Выполнить», так и сам по себе.

Если запускать, например, рассмотренную выше программу "Hello, world!", то обнаружится, что программа компилируется и выполняется, но сразу заканчивает работу и возвращается обратно в оболочку, не дав нам посмотреть результат ее работы на экране. Бороться с этим можно так — нужно добавить строку `getchar()`, которая заставит программу ожидать нажатия клавиши.

Таким образом, получаем программу:

```
#include <stdio.h>
main()
{
    printf("Hello, world!");
    getchar();
}
```

2. Объекты данных, типы данных

Переменные и константы являются основными объектами данных, с которыми имеет дело программа.

Имена переменных

Переменная — это участок памяти, имеющий имя, в котором хранится значение, которое может быть изменено программой.

Имена составляются из букв и цифр; первым символом должна быть буква. Символ подчеркивания "_" считается буквой; его иногда удобно использовать, чтобы улучшить восприятие длинных имен переменных. Не начинайте имена переменных с подчеркивания, так как многие переменные библиотечных программ начинаются именно с этого знака. Большие (прописные) и малые (строчные) буквы различаются, так что x и X — это два разных имени. Обычно в программах на Си малыми буквами набирают переменные, а большими — именованные константы.

Разумно давать переменным осмысленные имена в соответствии с их назначением, причем такие, чтобы их было трудно спутать друг с другом.

Типы и размеры данных

Каждая переменная или константа принадлежит определенному типу данных. Тип объекта данных определяет множество значений, которые этот объект может принимать, и операций, которые над ними могут выполняться.

В Си существует всего лишь несколько базовых типов:

`char` — единичный байт, который может содержать один символ из допустимого символьного набора;
`int` — целое, обычно отображающее естественное представление целых в машине;
`float` — число с плавающей точкой (вещественное число) одинарной точности;
`double` — число с плавающей точкой (вещественное число) двойной точности.

Имеется также несколько квалификаторов, которые можно использовать вместе с указанными базовыми типами.

Например, квалификаторы short (короткий) и long (длинный) применяются к целым:

```
short int sh;
long int counter;
```

В таких объявлениях слово int можно опускать, что обычно и делается.

Квалификаторы `signed` (со знаком) или `unsigned` (без знака) можно применять к типу char и любому целочисленному типу. Значения unsigned всегда положительны или равны нулю и подчиняются законам арифметики по модулю 2^n , где n — количество битов в представлении типа. Так, если значению char отводится 8 битов, то `unsigned int` имеет значения в диапазоне от 0 до 255, а `signed int` — от -128 до 127 (в машине с двоичным дополнительным кодом).

Тип	Размер в байтах (битах)	Интервал изменения
char	1 (8)	от -128 до 127
unsigned char	1 (8)	от 0 до 255
signed char	1 (8)	от -128 до 127
int	2 (16)	от -32768 до 32767
unsigned int	2 (16)	от 0 до 65535
signed int	2 (16)	от -32768 до 32767
short int	2 (16)	от -32768 до 32767
unsigned short int	2 (16)	от 0 до 65535

signed short int	2 (16)	от -32768 до 32767
long int	4 (32)	от -2147483648 до 2147483647
unsigned long int	4 (32)	от 0 до 4294967295
signed long int	4 (32)	от -2147483648 до 2147483647
float	4 (32)	от 3.4E-38 до 3.4E+38
double	8 (64)	от 1.7E-308 до 1.7E+308
long double	10 (80)	от 3.4E-4932 до 3.4E+4932

Константы

Константа — это объект данных, значение которого в течении выполнении программы не может быть изменено.

Каждая константа относится к какому-то определенному типу данных.

Целая константа, например 1234, имеет тип int. Константа типа long завершается буквой l или L, например 123456789L; слишком большое целое, которое невозможно представить как int, будет представлено как long. Беззнаковые константы заканчиваются буквой u или U, а окончание ul или UL говорит о том, что тип константы — unsigned long.

Константы с плавающей точкой имеют десятичную точку (123.4), или экспоненциальную часть (1e-2), или же и то и другое. Если у них нет окончания, считается, что они принадлежат к типу double. Окончание f или F указывает на тип float, а l или L — на тип long double.

Целое значение помимо десятичного может иметь восьмеричное или шестнадцатеричное представление. Если константа начинается с нуля, то она представлена в восьмеричном виде, если с 0x или с 0X, то — в шестнадцатеричном. Например, десятичное целое 31 можно записать как 037 или как 0X1F. Записи восьмеричной и шестнадцатеричной констант могут завершаться буквой L (для указания на тип long) и U (если нужно показать, что константа беззнаковая). Например, константа 0XFUL имеет значение 15 и тип unsigned long.

Символьная константа есть целое, записанное в виде символа, обрамленного одиночными кавычками, например 'x'. Значением символьной константы является числовой код символа из набора символов на данной машине. Например, символьная константа '0' в кодировке ASCII

имеет значение 48, которое никакого отношения к числовому значению 0 не имеет.

Строковая константа — это нуль или более символов, заключенных в двойные кавычки, как, например, "Я строковая константа" или "". Кавычки не входят в строку, а служат только ее ограничителями.

Объявления переменных и констант

Объявление — это описание элемента программы (переменной, функции и др.), которое используется, чтобы уведомить компилятор о существовании элемента.

Объявление переменных специфицирует тип и содержит список из одной или нескольких переменных этого типа. Например,

```
int lower, upper, step;
char c, line [1000];
int results[20];
```

Обратим внимание на последнюю строку примера.

Массив — это пронумерованная последовательность величин одинакового типа, обозначаемая одним именем. Элементы массива располагаются в последовательных ячейках памяти, обозначаются именем массива и индексом. Каждое из значений, составляющих массив, называется его компонентой (или элементом массива).

В строке

```
int results[20];
```

объявляется массив типа int размера 20, т. е. эта переменная хранит 20 элементов типа int.

В своем объявлении переменная может быть инициализирована, т. е. может быть задано ее начальное значение. Например:

```
char letter= 'a';
int i = 0;
float eps = 1.0e-5;
```

К любой переменной в объявлении может быть применен квалификатор const для указания того, что ее значение далее не будет изменяться.

```
const double e = 2.71828182845905;
```

Применительно к массиву квалификатор `const` указывает на то, что ни один из его элементов не будет меняться.

Реакция на попытку изменить переменную, помеченную квалификатором const, зависит от реализации компилятора.

3. Операции, выражения, операторы

Под *операцией* понимают некоторое действие, которое может быть выполнено над одним или несколькими объектами данных для получения результата.

Например, арифметические операции в языках программирования аналогичны по записи арифметическим операциям в математике.

Объекты данных, над которыми выполняются операции, называются *операндами.*

Унарные операции — операции с одним операндом, бинарные операции — с двумя, тернарные — с тремя операндами.

Арифметические операции

Бинарными (т. е. с двумя операндами) арифметическими операциями являются +, -, *, /, а также операция деления по модулю %. Деление целых сопровождается отбрасыванием дробной части, какой бы она ни была.

Операция % к операндам типов float и double не применяется.

Приоритет операций — очередность выполнения операций в выражении, при условии, что в выражении нет явного указания порядка следования выполнения операций (с помощью круглых скобок).

Бинарные операции + и - (сложение и вычитание) имеют одинаковый приоритет, который ниже приоритета операций *, / и %, который в свою очередь ниже приоритета унарных операций + и - (операция минус преобразует положительное значение в отрицательное значение и наоборот). Арифметические операции одного приоритетного уровня

выполняются слева направо.

Примеры выражений с арифметическими операциями:

```
a + b%c
(a + b)/3*5
```

Операции отношения и логические операции

Операциями отношения являются:
> (больше)
>= (больше или равно)
< (меньше)
<= (меньше или равно)
== (равно)
!= (не равно).

Действие операций отношения заключается в сравнении первого и второго операнда. Если операнды не удовлетворяют условию отношения, то результат операции равен нулю, т. е. условие ложно. Если операнды удовлетворяют условию отношения, то результат отношения не равен нулю, т. е. условие истинно.

Например, результатом операции 3>1 является 1 (истина), т. к. 3>1 — истина, результатом операции 2==1 является 0 (ложь), т. к. 2=1 — ложь.

Операции отношения выполняются слева направо. При нечетком понимании их действия возможно получение, вообще говоря, неверного результата. Например, с точки зрения синтаксиса языка Си выражение a<x<b записано совершенно правильно, но действия, выполняемые в соответствии с ним будут отличаться от принятых в математике: сначала будет вычислено выражение a<x, которое даст результат 0 или 1, а затем этот результат будет сравниваться с b.

Чтобы это выражение соответствовало математическому смыслу, его нужно разбить на две части, a < x и x < b, и связать его логической операцией && («и»), т. е. (a < x) && (x < b). Такая запись читается так: если a меньше x и x меньше b, то результат — истина.

Логические операции служат для построения сложных условий, подобных приведенному в предыдущем параграфе. В языке Си

определены следующие логические операции:

!	логическое отрицание	(унарная),
&&	логическое "и"	(бинарная),
||	логическое "или"	(бинарная).

Результаты логических операций:

A	B	!A	A && B	A | | B
0	0	не 0 (1)	0	0
0	не 0 (1)	не 0 (1)	0	не 0 (1)
не 0 (1)	0	0	0	не 0 (1)
не 0 (1)	не 0 (1)	0	не 0 (1)	не 0 (1)

Например, если значение переменной a равно 5, то значение выражения ((6>a)&&(a==5)) равно 1.

Операция присваивания, выражения

Выражение — последовательность символов, имеющая некоторое значение определенного типа.

Например, A — переменная типа int, выражениями будут являться следующие последовательности символов:

`A` (значение этого выражения равно значению переменной A, тип значения — int)
`A*2` (значение этого выражения — значение переменной A, умноженное на 2, тип значения — int)
`7.219` (значение этого выражения равно 7.219, тип значения — double)
`3+50` (значение этого выражения равно 53, тип значения — int)

Операция присваивания — это операция, позволяющая изменять значения переменных, в результате ее выполнения переменная слева изменяет свое значение на значение выражения справа.

Знак операции присваивания в языке Си — это знак `=`.

Например, после выполнения

```
A=5+2;
```

переменная A будет иметь значение 7.

После выполнения

```
A=5;
B=6;
A=B;
```

переменная A будет иметь значение 6.

Результат операции присваивания — значение присвоенной величины.
Т. е. после выполнения

```
A=(B=5);
```

переменная A будет иметь значение 5.

Любое выражение, как было сказано, имеет значение. Но выражения могут быть предназначены не только для вычисления значений, но и для изменения значений переменных, вывода строк на экран и т. п.

Например,

`A=45` — это выражение изменяет значение переменной A, при этом само выражение A=45 имеет значение, равное 45.

`printf("Hello, world!")` — выражение выводит на экран строку Hello, world!, при этом само выражение `printf("Hello, world!")` имеет значение, равное числу реально выведенных символов.

Операторы

Инструкция или *оператор* — наименьшая автономная часть языка программирования; команда. Программа обычно представляет собой последовательность операторов. Каждый оператор вызывает выполнение некоторых действий на соответствующем шаге выполнения программы.

Любой оператор имеет вид:
`выражение;` — выражение и точка с запятой.

Например, операторами являются строки:

```
x = 0;
A = b+x%2;
printf ("Hello, world!");
```

5. Функции вывода и ввода

Функция вывода printf

Вызов функции вывода printf имеет следующий вид:

```
    printf("форматная строка", выражение1,
выражение2, ...);
```

Форматная строка — это строка, которую функция printf выводит на экран. Она может содержать специальные сочетания символов — спецификации преобразования. При выводе строки такие сочетания символов заменяются соответственно значениями выражений (например, значений переменных), которые мы указали при вызове функции. Спецификаций преобразования в форматной строке должно быть столько же, сколько и указанных выражений.

Например, %d — спецификация преобразования для вывода целых чисел, результатом выполнения

```
A=6;
printf("A = %d   B = %d   C = %d", A, 4, 2+3);
```

будет вывод на экран строки:
`A = 6   B = 4   C = 5`

Для выражений различных типов используются разные спецификации преобразования:

%d - для вывода целых чисел;
%c - для вывода образа символа, соответствующий аргумент должен содержать код символа;
%f - для вывода вещественного числа в виде целой и дробной части;

%e - для вывода вещественного числа в виде мантиссы и порядка;
%g - для вывода вещественного числа в виде %f или %e в зависимости от значения числа;
%u - для вывода беззнакового целого числа в десятичной системе счисления;
%o - для вывода беззнакового целого числа в восмеричной системе счисления;
%x - для вывода беззнакового целого числа в шестнадцатеричной системе счисления;
%s - для вывода на экран символьной строки, соответствующий аргумент должен быть адресом строки (т. е. именем символьного массива или строковой константой).

Для дополнительного управления преобразованием данных используются модификаторы преобразования. Они записываются между символом % и спецификатором преобразования.

Таким образом, полный вид специкации преобразования:

%[флаг][ширина][.точность][l][спецификатор типа преобразования]

[флаг] : “-“ или “+” ; ”-“ выравнивает текст вывода по левому краю; “+” выводит знак для положительных и отрицательных значений;
[ширина] : минимальный размер поля вывода;
[.точность] : для вещественных чисел количество знаков после десятичной точки;
[l] : для вывода целых чисел типа long;

Примеры спецификаций преобразований с модификаторами:

```
%-20.6lf , %6d, %8.4f
```

Пример:

```
double a=-78.98;
int c=24;
printf("a=%8.4lf, c=%6d", a, c);
```

На экране:

```
a=-78.9800, c=    24
```

Также форматная строка может содержать так называемые управляющие

символы:

\n – перевод строки;
\t – табуляция;
\v – вертикальная табуляция;
\b – возврат на 1 символ;
\r – возврат на начало строки;
\a – звуковой сигнал.

Пример:

```
printf("Hello\nworld!");
```

На экране:

```
Hello
world!
```

Функция ввода scanf

Вызов функции scanf имеет следующий вид:

```
scanf(форматная строка, адрес_переменной_1,
адрес_переменной_2, ...);
```

Функцию scanf рекомендуется использовать без лишних символов в форматной строке, только спецификации преобразования, иначе может возникнуть непредсказуемая ситуация. Также лучше использовать один вызов функции scanf для ввода только одной переменной. Спецификации преобразования при вводе переменных различных типов такие же, как и при выводе.

Обратите внимание, что при вызове функции указываются не имена самих переменных, а адреса этих переменных. Нужно сообщить функции адрес ячейки, в которую необходимо занести данные. Адреса переменных большинства типов обозначаются знаком &.

Пример:

```
int n;
scanf( "%d", &n );
```

После выполнения этих операторов программа будет ждать ввода числа с клавиатуры. После введения числа переменная n получит значение, равное этому числу.

Перед именами строк и массивов операция взятия адреса & не ставится, т. к. имя строки или массива само определяет их адрес.

Пример ввода строки с помощью scanf:

```
char name[41];
scanf( "%s", name );
```

6. Стандартные математические функции

Для использования операций не нужно подключать никакие дополнительные библиотеки, они содержаться в самом языке.

Для математических вычислений могут понадобиться не только операции, но и некоторые математические функции. Стандартные математические функции содержатся в библиотеке и доступны при подключении заголовочного файла math.h строкой `#include <math.h>`.

sin (x) — sin x;
cos (x) — cos x;
tan(x) — tg x ;
log (x) — ln x;
log10(x) – lg x;
exp(x) — e^x ;
sqrt (x) — $\sqrt{x}$;
pow(x, y) — x^y;
abs(x) — |x|;
acos(x) — arccos x;
asin(x) – arcsin x;
atan(x) — arctg x;
sinh(x) — sh x;
cosh (x) — ch x;
tanh(x) — th x.

Для тригонометрических функций аргумент x измеряется в радианах и имеет тип double, как и значения функций.

Пример:
выражение `pow(x, cos(y)+2))` имеет значение, равное $x^{\cos y + 2}$.