ЛЕКЦИЯ №5

ЯЗЫК ПРОГРАММИРОВАНИЯ СИ

(ПРОДОЛЖЕНИЕ)

Представлен материал, который имеет отношение к основам программирования на алгоритмическом языке СИ (алфавит, операции, операторы, комментарии и другие элементы). Приводятся примеры программ, иллюстрирующие применение основных операторов СИ.

Функции языка СИ

Функции в языке СИ являются аналогом подпрограмм и (или) процедур в других языках программирования, например, FORTRAN, PASCAL и другие. С точки зрения структурного программирования функции позволяют решаемую задачу свести к набору достаточно мелких, но простых подзадач. Поэтому функции, как правило, прячут детали и особенности алгоритма. Однако в целом они позволяют писать наиболее простые и компактные программы с точки зрения логики.

Поэтому язык СИ устроен таким образом, что программа на нем представляет набор функций, который связан логически между собой. Среди данного набора функций в обязательном порядке должна присутствовать функция *main*, с которой собственно и начинается выполнение программы.

2 Функции в СИ устроены так, чтобы они являлись эффективными с точки зрения выполнения (затрат) на ЭВМ. Функции можно отдельно оттранслировать и собрать в единое целое – библиотеку, а потом на этапе сборки (*link*) загружать вместе с основной программой. В языке программирования существует целый набор библиотек функций: ввода-вывода; математическая, графическая и другие.

Целью данного раздела является изложение основных правил и соглашений, используемых для проектирования пользовательских функций, потребность в которых возникает по смыслу решаемой задачи.

Рассмотрим следующую задачу. Для заданного вещественного числа $a > 0$ вычислить значение величины

$$\frac{\sqrt[3]{a} - \sqrt[6]{a^2 + 1}}{1 + \sqrt[7]{a + 3}} \qquad (1)$$

Корень

$$y = \sqrt[k]{x}$$

вычислять с точностью $\varepsilon = 0.0001$ по следующей итерационной формуле:

$$y_0 = 1; \quad y_{n+1} = y_n + \frac{1}{k} \cdot \left(\frac{x}{y_n^{k-1}} - y_n \right), \qquad n = 0,1,2,\ldots$$

За ответ брать приближение y_{n+1}, для которого выполнено неравенство

$$\left| y_{n+1} - y_n \right| \le \varepsilon$$

Из анализа условия поставленной задачи следует, чтобы вычислить значение выражения (1) необходимо уметь вычислять *k*-ый корень из некоторого вещественного числа *x* с требуемой точностью *ε*. Данную вспомогательную задачу имеет смысл оформить в виде функции пользователя и затем ее применить при вычислении значения выражения (1). В качестве начальных данных для вспомогательной задачи служат значения величин: число *x*, степень *k* и точность вычислений *ε*. Предположим, что имя функции пользователя известно *sqr001 (k, x, eps)*. Она зависит от трех параметров, имена которых указаны в круглых скобках.

Построим блок-схему алгоритма для вычисления выражения (1) рис. 1.

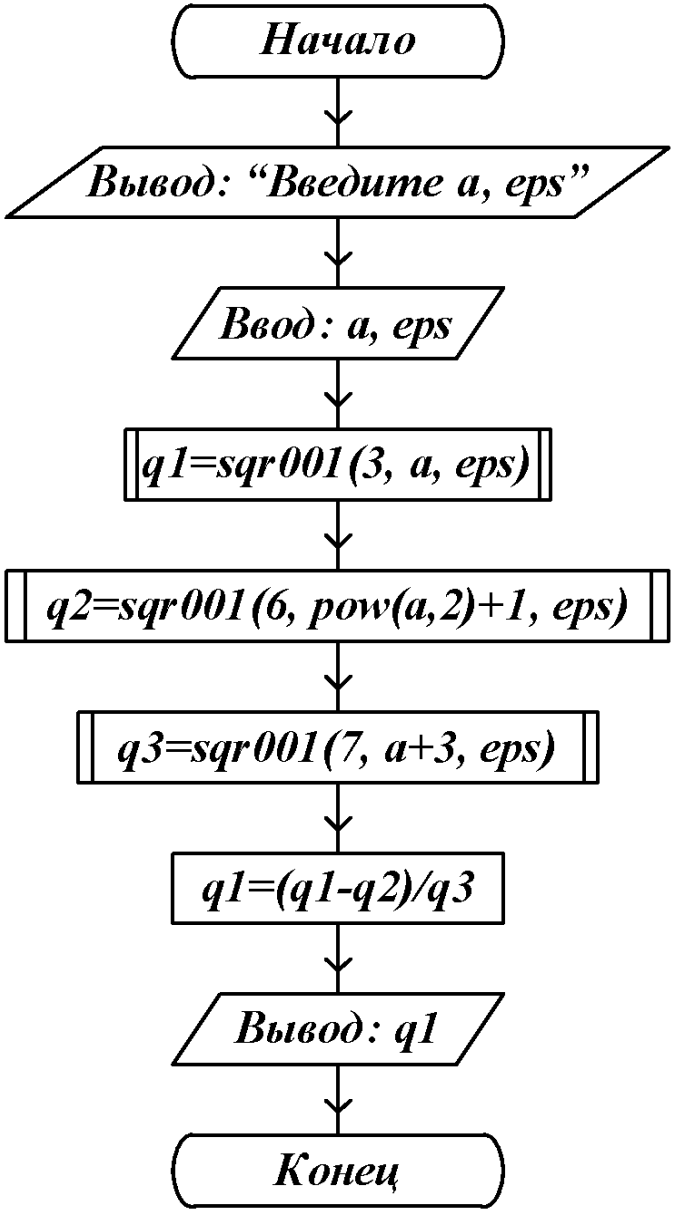

Рис. 1. Блок-схема алгоритма

5 **Теперь необходимо построить блок-схему алгоритма для вычисления значения корня *k*-ой степени числа *x* с точностью *ε*. Она имеет следующий вид:**

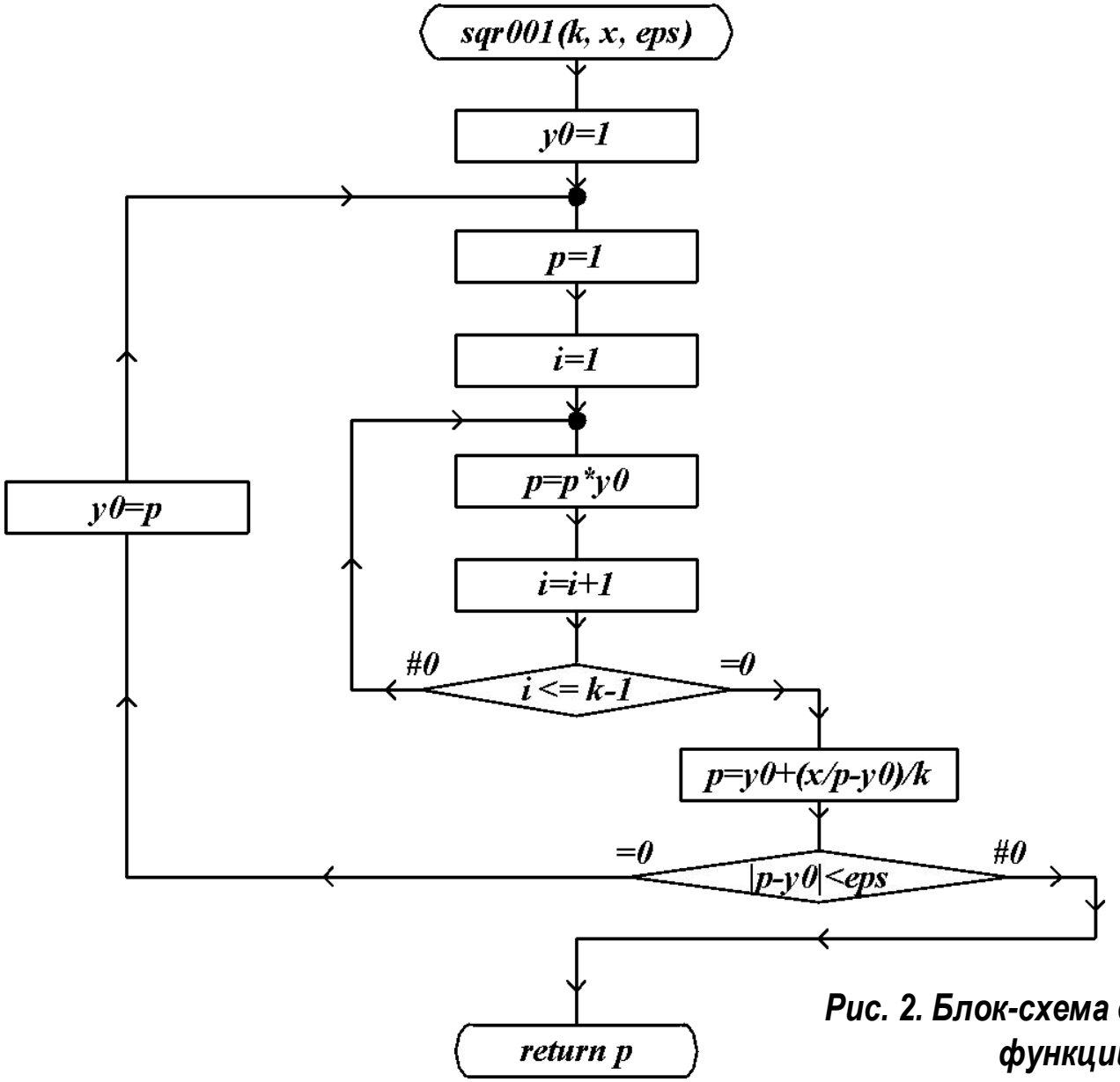

Рис. 2. Блок-схема вычисления функции

Объявление и описание функции. **Описание функции пользователя должно иметь следующий вид:**

```
[<тип>] <имя функции> (<список аргументов>)
  [<описание аргументов>]
{
  [<описание переменных>]
  <оператор>;
          .
          .
          .
   <оператор>;
}
```

По существу функция – это программа, которая оформлена специальным образом. Она имеет "заголовок" (первая строка), начинающийся с ключевого слова, который описывает тип данных возвращаемого ею значения. Затем следует имя функции, которое определяется пользователям по правилам задания идентификаторов переменных.

После имени в круглых скобках приводится список аргументов, от которых она зависит. Он представляет собой имена формальных параметров. Перед каждым параметром может стоять ключевое слово, которое должно указывать на то, какой тип данных оно может принимать. Если такие ключевые слова отсутствуют, то операторы объявления типов данных, принимаемые параметрами функции должны быть приведены после заголовка, что отражено второй строкой. Далее следует тело функции – набор операторов заключенных в фигурные скобки, который описывает дополнительные вспомогательные переменные, используемые в теле функции, и совокупность операторов языка СИ, реализующих вычислительный алгоритм решения некоторой вспомогательной задачи оформляемый пользователем в виде этой функции. Среди этого набора операторов может присутствовать оператор

return (<выражение>);

с помощью которого осуществляется передача значения некоторого выражения из функции в вызываемую программу.

Сейчас рассмотрим, каким образом транслятор определяет функции. Если ранее в описаниях не встречалось имя, за которым следуют круглые скобки (левая скобка), то транслятор воспринимает это имя как имя функции.

Кроме того, считается, что функция возвращает значение типа *int*, если перед ее именем явно не указан тип возвращаемого значения.

Аргументы функции. Аргументы (параметры) функции передаются двумя способами: по значению, когда функция получает свою собственную копию каждого аргумента, а не его адрес. Это означает, что функция не может воздействовать на оригинал, так как проводит все действия с копиями аргументов. Фактически каждый аргумент внутри функции является просто локальной переменной, инициализированной значением, с которым обратились к этой функции. Таким способом нельзя осуществлять передачу значений из функции в вызывающую ее программу.

Второй способ передачи аргументов основан на указании адреса (места ее хранения) и поэтому называется передачей параметров по ссылке. О нем речь пойдет несколько позднее, когда приступим к изложению материала об указателях.

Классы памяти. Вам уже известно. Что программа на СИ оперирует с множеством объектов: констант и переменных. Причем часть переменных может определяться вне функции и, по отношению к ней, будут являться внешними.
В языке СИ существует такое понятие как область действия переменных. Принято, что область действия переменной простирается от оператора описания и до конца программы.

Существуют также локальные переменные, область действия которых ограничена, например, функцией. Если программа велика и ее текст хранится в отдельных файлах, то для распространения влияния переменной на все файлы необходимо указать в них оператор описания внешней переменной (*extern*). В СИ имеется четыре класса памяти, которые определяют область действия переменных.

Локальные переменные (auto). Если при объявлении явно не указан класс памяти, то переменные считаются локальными и их область видимости определяется классом памяти *auto*. Область видимости локальных переменных простирается от оператора объявления переменной до момента ее последнего использования. Никаких действий по инициализации таких переменных транслятором не проводится. Поэтому пользователь сам должен позаботиться о присвоении им начальных значений, которые бы соответствовали смыслу решаемой задачи. В противном случае компилятор присвоит им то значение, которое хранилось в ячейке оперативной памяти выделенной для ее хранения. Такой процесс присвоения начального значения локальной переменной называется подбором мусора.

Регистровые переменные (register). Такой класс памяти может быть применен только для локальных переменных или формальных параметров функции (параметры, используемые при описании функции).

Спецификатор *register* означает, что пользователь размещает переменную не в оперативной памяти ЭВМ, а на одном из быстродействующих регистров компьютера. Такой спецификатор, как правило, применяется для переменных, к которым идет частое обращение, например, индексные переменные массивов, циклов и т. д. для регистровых переменных имеется существенное ограничение, состоящее в том, что нельзя обращаться к адресу ячейки, где хранится их значение. Поэтому, например, их нельзя передавать в функцию по ссылке.

Статические переменные (static). **Такой класс памяти определяет для статических переменных область видимости от точки их объявления до конца файла. Значения таких переменных сохраняется от одного обращения к ней до другого. При этом следует учитывать, что область их видимости по сравнению с локальными переменными расширяется, однако, тем не менее, они будут не видимы вне файла, где они определяются.**

Внешние переменные (extern). **Внешние переменные используются в тех случаях, когда программа хранится в нескольких файлах (об этом случае уже упоминалось выше). Для того чтобы распространить влияние некоторой переменной на другие файлы необходимо в них объявить эту переменную внешней путем использования спецификатора *extern*.**

Разработка программы. **Только теперь можем приступить к написанию программы решения поставленной ранее задачи, учитывая блок-схемы рис. 1 и рис. 2. Очевидно, что наша программа состоит из двух функций *main* и *sqr001*. Предположим, что сначала идет функция *main*, а затем функция *sqr001*. Также известно, что функция *sqr001* вызывается из функции *main*. Прежде чем писать программу следует напомнить, что всегда компилятор начинает выполнять программу с функции *main*, а поскольку функция *sqr001* следует за функцией *main*, то в ней ничего о функции *sqr001* неизвестно. Поэтому для таких ситуаций предусмотрен оператор объявления функции**

[<тип>] <имя функции> (<список типов аргументов>) ; ,

где <тип> – ключевое слово, описывающее тип возвращаемого значения, <имя функции> – идентификатор, описывающий имя функции пользователя, <список типов аргументов> – список ключевых слов через запятую, описывающих типы данных, принимаемые соответствующими аргументами функции. Для функции *sqr001* оператор объявления будет иметь вид

float sqr001 (int, float, float); **.**

Запишем текст программы.

Программа 1:

```
#include <stdio.h>
#include <math.h>
#include <conio.h>
void main (void)
{
   float sqr001 (int, float, float);
   float a, eps, q1, q2, q3;
   clrsrc ();
   printf ("Введите a, eps:");
   scanf ("%f %f", &a, &eps);
   q1 = sqr001 (3, a, eps);
   q2 = sqr001 (6, pow(a, 2)+1, eps);
   q3 = sqr001 (7, a+3, eps);
   q1 = (q1 - q2)/q3;
   printf ("\nЗначение выражения = %f", q1);
}
```

```
   /* Функция пользователя */
float sqr001 (int k, float x, float eps)
{
    int i;
    float y0,p;
    y0 = 1.0;
m1:p = 1.0;
    i = 1;
m2:p *= y0;
    i++;
    if (i < (k-1)) goto m2;
    p = y0 + (x/p - y0)/k;
    if (fabs(p-y0) >= eps)
       {
           y0 = p;
           goto m1;
       }
       return (p);
}
```

К О Н Е Ц