Структурная разработка программ

Артамонов Юрий Николаевич

Содержание

Алгоритмы

Решение любой задачи, связанной с вычислениями, включает в себя выполнение ряда действий в определенном порядке. Набор этих действий и сам порядок их выполнения часто называют *алгоритмом.*
Более строго:
Алгоритм — это строго определенная последовательность действий для решения задачи за конечное число шагов.
Любой алгоритм должен удовлетворять следующим требованиям:

- может применяться для решения некоторого класса задач - свойство массовости алгоритма;
- каждое действие алгоритма должно быть однозначно интерпретировано возможным исполнителем — свойство определенности алгоритма;
- исполнитель алгоритма должен получить решение или убедиться в невозможности его получения за ограниченное время — свойство результативности алгоритма;
- любой алгоритм можно разбить на последовательность элементарных шагов (операций) — свойство дискретности алгоритма.

Способы записи алгоритмов

Для записи алгоритмов используют следующие способы:

1. *словесное описание* (запись алгоритма на естественном языке);
2. *псевдокоды* (в отличие от словесного описания данный способ более формализован, в нем используются специально введенные обозначения отдельных команд, алгоритмических структур. Данный способ часто используется авторами книг по программированию без привязки к какому-то конкретному языку программирования);
3. *графическое описание* (используются специальные графические символы. Наибольшее распространение получил способ описания алгоритма с помощью *блок-схем*);
4. *программная реализация* (запись алгоритма на каком-либо конкретном языке программирования).

Пример записи алгоритма с использованием псевдокода

Программы на псевдокоде на самом деле не выполняются на компьютерах. Скорее они просто помогают программисту «продумать» программу перед попыткой написать ее на каком-либо языке программирования. Вот пример алгоритма выбора наибольшего числа из двух чисел на псевдокоде.

1: $n \leftarrow a, m \leftarrow b$
2: **if** $n > m$ **then**
3: $\quad max \leftarrow n$
4: **else**
5: $\quad max \leftarrow m$
6: **end if**

Псевдокод состоит только из операторов действия, в нем не используются объявления, предполагая, что программист сам знает, какая переменная, что означает и какого она типа.

Управляющие структуры

Обычно операторы в программе выполняются один за другим в порядке их записи. Это называется *последовательным выполнением*. Однако, довольно часто приходится менять этот естественный порядок, это называется *передачей управления*. В 1960-е годы стало ясно, что в основе большинства сложностей, испытываемых группами разработчиков ПО, лежит бесконтрольное использование передачи управления. Вина была возложена на оператор goto, который позволяет передавать управление в очень широком диапазоне. Понятие так называемого *струкурного программирования* стало почти синонимом «программирования без goto». Исследователи *Бом и Якопини* показали, что программирование возможно и при полном отсутствии операторов goto.

Управляющие структуры (продолжение)

В 70-е годы широкие круги профессиональных программистов начали принимать структурное программирование всерьез. Результаты оказались впечатляющими, среднее время на разработку программ существенно сократилось, ошибок стало меньше. Ключом к успеху стало попросту то, что программы, созданные на основе методов структурного программирования, более понятны, их проще отлаживать и модифицировать и, самое главное, более вероятно, что они написаны без ошибок. Работа Бома и Якопини показала, что все программы могут быть написаны с использованием всего трех управляющих структур:

- последовательной структуры (следование)
- структуры выбора
- структуры повторения

Блок-схемы

Для демонстрации этих трех управляющих структур удобно использовать графический способ записи алгоритмов в виде *блок-схем*. При рисовании блок-схем используются некоторые символы специального назначения, такие как прямоугольники, ромбы, овалы и т.д. Все графические символы, их размеры, а также правила построения блок-схем определены государственными стандартами:

1. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.
2. ГОСТ 19.002-80 ЕСПД. Схема алгоритмов и программ. Правила выполнения.
3. ГОСТ 19.003-80 ЕСПД. Схема алгоритмов и программ. Обозначения условные графические.

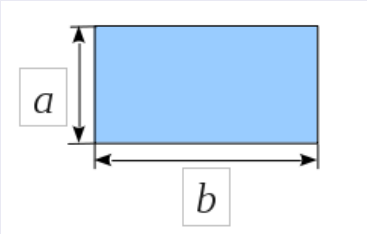
Последовательная структура

Последовательная структура, по существу, является встроенной в язык C. Если не указано иначе, компьютер автоматически выполняет операторы C один за другим в порядке их записи. Нарисуем блок-схему простейшей программы:

```
main()
{
   int a,b,c;
   scanf("%d%d", &a, &b);
   c = a+b;
   printf("%d\n", c);
   return 0;
}
```

Последовательная структура (продолжение)

Для обозначения выполнения операции или группы операций, в результате чего изменяется значение, форма представления или расположения данных используется символ блок схемы - *процесс*, который обозначается в виде обычного прямоугольника. Внутри символа или в виде комментария дается описание выполняемых действий на естественном языке.



Размеры символов должны удовлетворять соотношению $b = 1.5 \cdot a$, где a должен выбираться из ряда 10, 15, 20 мм. Допускается увеличивать размер a на число, кратное 5.

Последовательная структура (продолжение)

Для ввода-вывода данных в блок-схемах используются следующие символы:

- *Данные*. Символ отображает данные, носитель данных не определен.

- *Ручной ввод*. Ввод данных оператором в процессе обработки при помощи устройства, непосредственно сопряженного с компьютером (например, клавиатура).

- *Дисплей*. Ввод-вывод данных в случае, если устройство воспроизводит данные и позволяет оператору вносить изменения в процессе их обработки.

- *Документ*. Ввод-вывод данных, носителем которых служит бумага.

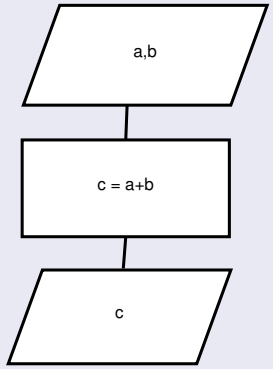
Последовательная структура (продолжение)

Для указания последовательности связей между символами используются *линии потока*. Для изображения линий потока существуют правила. Перечислим некоторые из них:

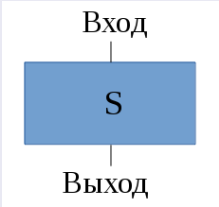
- линии потока должны быть параллельны линиям внешней рамки блок-схемы (границам листа, на котором изображена блок-схема);
- направление линии потока сверху вниз и слева направо принимается за основное и стрелками не обозначается, в остальных случаях направление линии потока обозначается стрелками;
- изменение направления линии потока производится под углом 90 градусов.

Последовательная структура (продолжение)

Таким образом, получаем следующую блок-схему нашей простейшей программы:



Как видно из рисунка, управление никуда не передается, команды выполняются строго последовательно. Это и есть последовательная структура. Данная структура описывает так называемые линейные алгоритмы — команды алгоритма выполняются последовательно друг за другом (линейно) без разветвлений и циклических повторений.



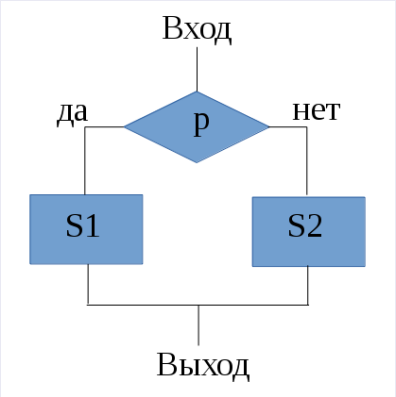
Структура выбора

Для графического изображения этой структуры используется символ *Решение* - выбор направления выполнения алгоритма в зависимости от некоторых условий.

Выделяют следующие разновидности структур выбора.

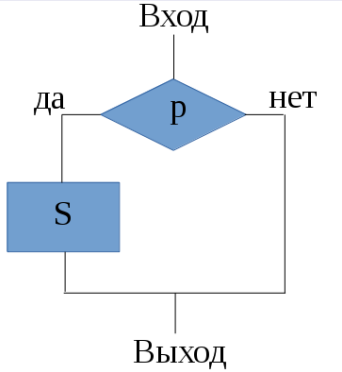
Структура выбора (продолжение)

Развилка полная. Используется в случае, когда выполнение программы может пойти двумя различными путями. Внутри символа «Решение» (или в виде комментария к этому символу) записывается логическое условие, по которому осуществляется выбор требуемого направления выполнения алгоритма. В зависимости от значения логического условия дальнейшее выполнение алгоритма идет либо по левой, либо по правой ветви. Символы «Процесс S1», «Процесс S2» могут обозначать унифицированные структуры, процедуры, функции и алгоритмы любой сложности.



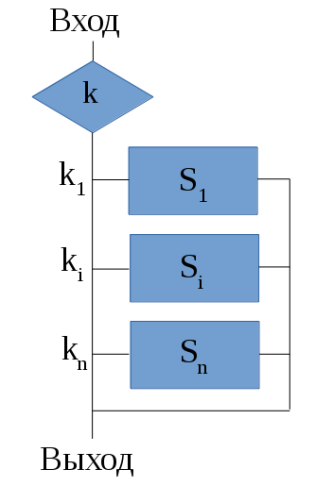
Структура выбора (продолжение)

Развилка неполная. Используется так же, как и «развилка полная» с тем отличием, что при выполнении одной из ветвей никаких изменений данных, поступающих на вход этой унифицированной структуры, не происходит.



Структура выбора (продолжение)

Выбор Предназначен для выбора из многих вариантов. Данную унифицированную структуру можно заменить несколькими вложенными друг в друга структурами «развилка полная». Однако при большом количестве условий использование данной структуры становится предпочтительным.



Структура выбора (продолжение)

Язык C предоставляет программисту все три типа структуры выбора:

- В структуре выбора if(развилка неполная) некоторые действия выполняются, если условие истинно, либо пропускаются, если это условие ложно.
- В структуре выбора if/else(развилка полная) некоторые действия выполняются, если условие истинно, и выполняется другое действие, если условие ложно.
- В структуре выбора switch(выбор, будем изучать позже) выполняется одно из набора различных действий в зависимости от значения некоторого выражения.

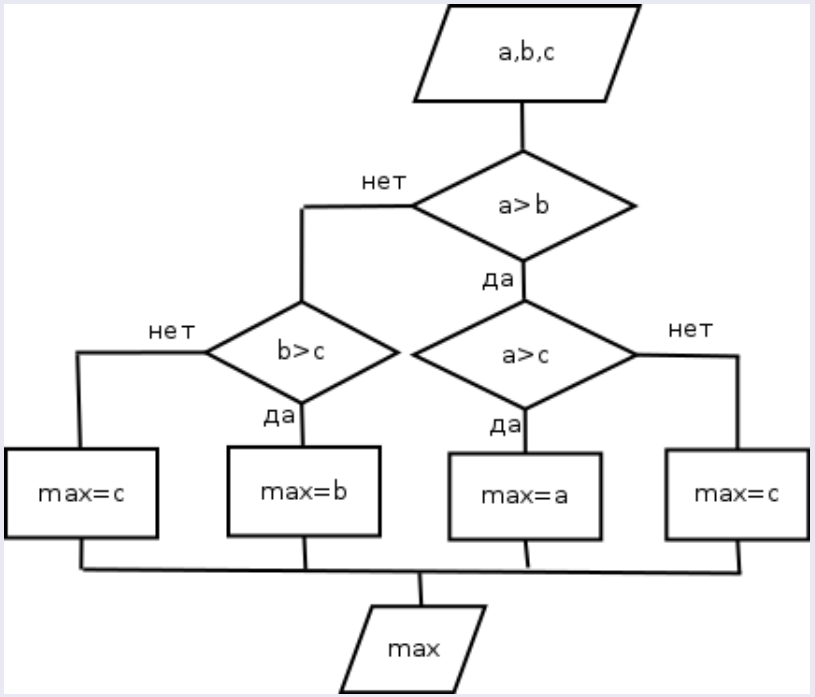
Структура выбора - пример

В качестве примера, рассмотрим алгоритм нахождения максимального из трех чисел.

```
main()
{
  int a, b, c, max;
  scanf("%d%d%d", &a, &b, &c);
  if (a>b)
    {
      if (a>c)
          max = a;
      else
        max = c;
    }
  else if (b>c)
    max = b;
  else
      max = c;
  printf("%d\n", max);
}
```

Структура выбора - пример

Рассмотрим блок-схему данного алгоритма.



Цикл while

Структура повторения позволяет программисту выполнить некоторые действия многократно, пока некоторое условие остается истинным. В виде блок-схемы такая структура выглядит следующим образом:

Цикл while в C

В языке C цикл while имеет следующую спецификацию:

```
while ( условие )
    {
      набор действий
     }
```

Если набор состоит из одного действия фигурные скобки можно не писать.

Язык C предусматривает три типа структур повторения: цикл while, циклы do/while, for (будут изучены позже).

Таким образом, в языке C имеется только 7 управляющих структур: последовательная, три типа выбора, три типа повторения. Любая программа может быть написана путем объединения этих структур.

Пример использования цикла while

Составим программу, находящую сумму всех чисел от 1 до 100.

```
main()
{
  int i, result;
  i = 1; result = 0;
  while (i<=100)
    {
      result = result + i;
      i++; /*Аналог i=i+1;*/
    }
  printf("%d\n",result);
  return 0;
}
```

Пояснения к коду

Перед началом цикла объявляются две переменные: i (выполняет роль счетчика, который указывает в каждый момент времени сколько еще раз требуется выполнить цикл), result (переменная, которая аккумулирует результат). Поскольку перед началом цикла переменная result еще не содержит результата, ее инициализируют нулевым значением (чтобы ее старое значение не смещало итоговое накопленное в цикле значение). Поскольку перед началом цикла он еще не выполнен ни разу переменную i устанавливают в 1. Внутри цикла выполняются два оператора: result = result+i : на каждой итерации цикла к старому значению result прибавляется текущее значение i; i++ : это краткая запись выражения i = i+1, в котором переменная счетчик увеличивается на единицу. После выполнения этих операторов цикл возвращается на начало и проверяет верность условия (i<=100). Если условие верно, цикл повторяется. Ясно, что поскольку на каждой итерации переменная счетчик увеличивается, то после определенного числа итераций условие (i<=100) нарушится и произойдет выход из цикла.

Пояснения к коду (продолжение)

Важно понять, что условие нарушится именно в тот момент, когда в переменной-аккумуляторе будет содержаться правильный результат - сумма всех чисел от 1 до 100. Например, можно добиться правильного ответа и таким кодом:

```
.....
  i = 0; result = 0;
  while (i<100)
    {
      result = result + i;
      i++;
    }
.....
```

Здесь мы начинаем считать с нуля, поэтому последнее значение, для которого условие должно быть верно - это i=99. Условие цикла модифицируется на (i<100). Мы также могли написать, например, условие (i<=99).

Пояснения к коду (продолжение)

В нашем случае условие цикла проверяется перед началом каждой итерации цикла - это пример *цикла с предусловием*. Кроме этого, в нашем случае количество итераций цикла уже известно до начала цикла, поэтому и потребовалось использовать счетчик. Такие циклы называются циклами с определенным числом повторений (*циклами модификации*). Для них обычно используются другие специальные операторы (хотя для реализации любых разновидностей циклов достаточно только цикла while).

Задание 3.1

Нарисуйте блок-схему программы нахождения суммы чисел от 1 до 100.

Задание 3.2

Написать программу, которая выводит все числа от 1 до 99 по три числа в одной строке:

```
1 2 3
4 5 6
7 8 9
...
```

Задание 3.3

Написать программу, которая считывает размер стороны квадрата и затем выводит этот квадрат в виде звездочек. Например, если сторона квадрата равна 4, то программа должна вывести:

```
****
****
****
****
```

Задание 3.4

Напишите программу, которая для заданного числа находит сумму его цифр. Например, если число 2345, то программа должна посчитать $2 + 3 + 4 + 5 = 14$

Операции присваивания

В языке C предусмотрено несколько операций присваивания для более короткой записи выражений. Например, оператор
c = c +3;
может быть сокращен с помощью операции сложения с присвоением +=
c+=3
Операция += прибавляет значение выражения справа от операции к значению переменной слева от нее и сохраняет результат в данной переменной. Аналогично такой трюк можно использовать и с двухместными операциями -, *, /, %. Примеры их использования приведены в таблице.

Операции присваивания (продолжение)

Пусть
int c=3, d=5, e=4, f=6, g=12;

Операция присваивания	Пример выражения	Пояснение	Присваивает
+=	c += 7	c = c+7	10 переменной c
-=	d -= 4	d = d-4	1 переменной d
*=	e *= 5	e = e*5	20 переменной e
/=	f /= 3	f = f/3	2 переменной f
%=	g %= 9	g = g%9	3 переменной g

Выражение c+=3 компилируется быстрее, чем раскрытое c = c+3, т.к. в сокращенной форме переменная c оценивается только один раз, а в раскрытой форме два раза.

Операции инкремента, декремента

В языке C предусмотрены операции инкремента ++ и декремента --. При увеличении переменной c на 1 вместо выражений c = c+1 или c+=1 может использоваться операция инкремента ++.
Если операция ++ идет перед переменной, например, ++c, то она называется *преинкрементной*, если операция ++ идет после переменной, например, c++, то она называется *постинкрементной*. Аналогично: - - c *предекрементная операция*; c - - *постдекрементная операция*.
Операция преинкремента (предекремента) над переменной вызывает увеличение (уменьшение) этой переменной на 1, затем новое значение переменной используется в выражении, в котором она появляется. Операция постинкремента (постдекремена) над переменной вызывает использование текущего значения этой переменной в выражении, в котором она появляется, затем значение переменной увеличивается (уменьшается) на 1.

Операции инкремента, декремента (пояснения)

```
// Так оформляется однострочный оператор
#include <stdio.h>
main()
  {
      int c;
      c = 5;
      printf("%d\n", c);  //Выводится значение 5
      printf("%d\n", c++); // На экран выводится значение 5, переменная
  еще не изменилась внутри оператора
      printf("%d\n", c); //Переменная уже приняла  новое значение 6
      c = 5;
      printf("%d\n", c);  //Выводится значение 5
      printf("%d\n", ++c); // На экран выводится значение 6, переменная
  изменилась внутри оператора
      printf("%d\n", c); //Опять выводится 6
      printf("%d\n", c++ + ++c); //Подумайте, что будет выведено?
      printf("%d\n", c); //А теперь?
  }
```

Более сложный пример

Разработать программу для подсчета средней оценки в группе, которая при каждом запуске будет обрабатывать произвольное число оценок.

Программный код примера

```
#include <stdio.h>
main()
  {
    float avarage; //Новый тип данных
    int counter, grade, total;
    total = 0; counter = 0;
    printf("Введите оценку, или -1 для окончания ");
    scanf("%d", &grade);
    while ( grade != -1)
      {
        total += grade;
       ++counter;
        printf("Введите оценку, или -1 для окончания ");
     scanf("%d", &grade);
      }
```

Программный код примера (продолжение)

```
    if (counter != 0)
      {
        //Пример преобразования типа данных
           avarage = (float) total/counter;
        //Пример форматированного вывода
          printf("Средняя оценка в классе равна %.2f\n", avarage);
      }
    else
      printf("Оценки не введены.\n");
    return 0;
}
```

Пояснения к программному коду

Средние величины не всегда выражаются целочисленными значениями. Часто среднее является значением 3.5, т.е. содержит дробную часть. Эти значения называются числами с плавающей точкой и представляются типом данных float. Переменная avarage объявлена как float, чтобы не потерят дробную часть результата вычисления.
Поскольку переменные total, counter являются целыми числами, то вычисление total/counter также будет целым числом (вспомним, что в C осуществляется целочисленное деление). Поэтому в коде осуществляется операция приведения типов: avarage = (float) total/counter;
Операция (float) создает для своего операнда total временную копию с плавающей точкой. Использование операции приведения типов подобным образом называется *явным преобразованием.* Значение, хранимое в total, по-прежнему является целым числом, однако в вычислениях используется его временная копия с плавающей точкой.

Пояснения к программному коду (продолжение)

Для вывода значения переменной avarage используетя спецификатор преобразования %.2f функции printf. Символ f указывает, что будет выведено значение с плавающей точкой. Символ .2 представляет собой точность, с которой будет отображено это значение. Она устанавливает, что значение будет отображено с 2 десятичными знаками справа от десятичной точки. Когда значения с плавающей точкой выводятся с указанием точности, выводимое значение округляется до заданного числа десятичных знаков. Значение в пямяти остается неизменным.

По умолчанию спецификатор без указания точности %f выдает число с точность до 6 знаков. Это эквивалентно % .6f.

Задание 3.5

Палиндромом называется число, которое читается одинаково как слева направо, так и в обратном порядке. Например, 12321, 66666, 23432. Написать программу, которая считывает пятизначное число и определяет, является ли оно палиндромом.

Задание 3.6

Написать программу, которая считывает целое число и определяет, сколько цифр в этом числе равно 7.

Задание 3.7

Разработать программу перевода числа из десятичной системы счисления в число двоичной системы счисления.

Задание 3.8

Написать программу, которая последовательно выводит числа, кратные числу 2: 2, 4, 8, 16, 32... Ваш цикл должен быть бесконечным.

Задание 3.9

Написать программу, которая считывает три ненулевых целых числа и определяет, могут ли они быть сторонами прямоугольного треугольника, выводя результат на экран.

Задание 3.10

Факториалом неотрицательного целого числа n называется выражение $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$. Это выражение обозначается $n!$. Написать программу, которая вычисляет факториал числа.

Задание 3.11

Написать программу, которая оценивает значение математической константы e по формуле:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$