Этапы разработки программного обеспечения

Весь процесс разработки компьютерных программ стандартно разбивается на следующие этапы:

1. Анализ постановки задачи. Выяснить при этом входную и выходную информацию, определить идею решения, полноту входной информации, сформулировать, накладываемые на входную информацию ограничения (то есть описать спецификации).

2. Построение математической модели. Определить какие математические структуры больше подходят для задачи, существуют ли аналоги решения этой задачи. После чего проверить полноту математической модели, удобство работы с ней, реализации.

3. Разработка алгоритма. Необходимо тщательно обдумать алгоритм, уяснить его достоинства и недостатки, постараться избавиться от последних и усилить первые (при разработке алгоритма нахождения НОД можно было бы использовать переборный алгоритм, но он требует намного больше вычислений, чем алгоритм Евклида).

4. Анализ алгоритма. Необходимо оценить его сложность по памяти (объему памяти для хранения самой программы, исходных данных и промежуточных переменных) и/или быстродействию (количеству вычислительных операций).

5. Доказательство правильности алгоритма. Этот этап предусматривает как доказательство конечности алгоритма, так и анализ его работы на всех ветвях на разных типах входных данных (см. доказательство для алгоритма Евклида нахождения НОД).

6. Реализация алгоритма. Это написание программы на некотором языке программирования. Напишем программу для нахождения НОД двух натуральных чисел на языке программирования Паскаль (стандарт ISO).

```
 Program NOD;
var a,b,m,n  : word;
 begin
       read(a,b);
```

```
    m:=a; n:=b;
    while ( a <> b ) do
     begin
      if a>b then
        a:=a-b
      else
        b:=b-a;
      end;
    writeln('NOD(', m, ',', n, ') = ', a)
end.
```

7. Тестирование и отладка программы. Тестирование и отладка два перекрывающихся процесса. Тестирование устанавливает факт наличия ошибки, а отладка выявляет причину ошибки.

Типичная ошибка начинающих программистов: программа работает на нескольких наборах данных, а на следующем – нет.

Различают две стратегии тестирования: по белому ящику и по черному.

В первом случае проверяется выполнение всех операторов на всех ветвях программы. Во втором случае тестируется программа без рассмотрения ее внутренней структуры, т.е. идет проверка ее работоспособности на различных вариантах входных и выходных данных. (Подробнее об этом будет рассказано на практике).

8. Оформление документации. Необходимо описать технические характеристики, структуру входных и выходных данных, правила использования программы, при необходимости - реализуемые алгоритмы и возможности модификации.

9. Внедрение.

Типы ошибок

При разработке алгоритма и составлении программы мы можем столкнуться с тремя типами ошибок:

1) Синтаксическими - ошибка в записи текста; например, пропущена закрывающая скобка в арифметическом выражении: (a*(b*(c+d))+a. Здесь три открывающиеся скобки и только 2 - закрывающиеся.

2) Семантическими - например, пусть дана символьная строка Str1 и число Num. Запись вида Str1/Num будет семантически неверна, так как типы числителя и знаменателя несовместимы.

3) Логическими - например, при вычислении расстояния S, которое пройдет автомобиль, движущийся со скоростью v за время t, мы вместо формулы S=v*t запишем S=v+t.

Выполним все основные этапы разработки алгоритмов на примере следующей задачи:

Даны три числа a, b, c. Необходимо определить, можно ли построить треугольник, длинами которого являются числа a, b, c?

Этап 1. Анализ постановки задачи.

Что нам дано? Три числа a, b, c.

Какие могут быть эти числа? Числа могут быть действительные или целые, но положительные. Так как длина стороны треугольника не может быть отрицательным числом или символьной строкой (условие 1).

Идея решения? Проверить, удовлетворяют ли числа условиям, накладываемым на длины сторон треугольника.

Что должно получиться на выходе? Одно из сообщений:

а) Числа a,b,c могут выступать в качестве длин сторон треугольника (True - Истина);

б) Числа a, b, c не могут выступать в качестве длин сторон треугольника (False - Ложь).

При желании можно уточнить, по какой причине мы получили ответ (б).

Этап 2. Построение математической модели.

Математическая модель? Можно использовать следующие правила геометрии, то есть, условие, что длины сторон треугольника должны удовлетворять условию (2):

a+b>c - для любых трех сторон треугольника.

Какие числа мы будем использовать? Чтобы не увеличивать ограничения, накладываемые на задачу, лучше использовать вещественные числа (double).

Что будет, если все числа отрицательные? Это вырожденный случай, треугольник превращается в точку. В этом случае будем считать, что треугольник построить невозможно. Аналогично, будем считать, что треугольник построить невозможно, если хотя бы одно из чисел будет меньше или равно 0.

Этап 3. Разработка алгоритма.

Запишем алгоритм решения этой задачи на псевдокоде:

```
Начало
1. Ввести a, b, c
2. Проверить, является ли введенная информация вещественными числами?
Если введенная информация нечисловая,
   то Ответ: "Числа a,b,c не могут выступать в качестве
                  длин сторон треугольника - нарушается условие 1 -
                  Нечисловая информация"
   иначе                  // a, b, c - содержат числовую информацию,
Если (a>0) и (b>0) и (c>0)
      то Если (a+b>c) и (b+c>a) и (c+a)>b
```

то Ответ: "Числа a,b,c могут выступать в качестве
длин сторон треугольника"
иначе Ответ: "Числа a,b,c не могут выступать в качестве
длин сторон треугольника - нарушается условие 2"
иначе Ответ: "Числа a,b,c не могут выступать в качестве
длин сторон треугольника - нарушается условие 1 -
среди a, b, c есть отрицательное ил нулевое число"

Конец

Замечание: Решение задачи можно упростить, если не указывать причину, по которой нельзя построить треугольник. В этом случае ответ будет: True (Истина) - если можно построить треугольник; False (Ложь) - если нельзя построить треугольник по любой из указанных причин.

Этап 4. Анализ алгоритма.

Этот алгоритм линейный с разветвлениями. Сложность его по быстродействию - O(n), где n - максимальная длина пути при выполнении алгоритма (грубо говоря - количество операторов в самой длинной ветви).

Этап 5. Доказательство правильности алгоритма.

Повторяющихся участков в алгоритме нет. Если проанализировать все ветви алгоритма, то мы всегда закончим выполнение программы.

Все условия, накладываемые на входные данные, проверяются. Следовательно, данный алгоритм конечен и работает правильно.

Этап 6. Реализация алгоритма.

Запишем программу на Delphi, реализующую этот алгоритм (здесь мы берем упрощенный вариант, то есть определяем только - треугольник или не треугольник):

```
function Triangle (a, b, c:real):boolean;
begin
   result:= (a>0) and (b>0) and (c>0) and
   ((a<(b+c)) and (b<(a+c)) and (c<(a+b)));
end;
```

Этап 7. Тестирование программы.

Проведем тестирование и отладку нашей программы.

Необходимо подготовить следующие тесты:

В таблице 2.1 приводится набор тестов для нашей задачи. Пусть числа a, b, c, должны принадлежать диапазону [t1, t2], где t1 - нижняя, а t2 - верхняя граница допустимого диапазона для вводимых чисел.

Табл. 2.1. Множества тестов для задачи с треугольниками

N	a	b	c	Ожидаемый результат
1	3	5	4	"Можно построить треугольник"
2	3	-	-	"Нельзя построить треугольник. Только 1 число"
3	3	5	-	"Нельзя построить треугольник. Только 2 числа"
4	3	3	4	"Нельзя построить треугольник. Условие 2"

5	3.5	5.5	4.5	"Можно построить треугольник""
6	!?	600	400	"Нельзя построить треугольник. Условие 1 - не числовые данные"
7	0	5	4	одно число <= A
8	3	0	4	
9	3	5	4	
10	0	0	4	
11	0	5	0	
12	3	0	0	
13	-3	5	4	
14	3	-5	4	
15	3	5	-4	
16	0	0	0	
17	-1	-2	-3	
18	-3	-5	-4	
19	3	-5	-4	
20	-3	5	-4	
21	0.001	5	5.5	приближение к граничному условию
22	B	5	4	результат - число >=B
23	3	B	4	
24	3	4	B	
25	B	B	4	
26	B	5	B	
27	3	B	B	
28	B	B	B	
29	3	-5	-4	
30	1	2	1	числа не являются сторонами треугольника
31	1	1	2	
32	3	1	1	
33	B-1	B/2	B/2	граничное условие
34	B-1	B-1/2	B-1/2	

Заметим, что в данной таблице приводится далеко не полный набор тестов. Однако, при реальном тестировании программ рекомендуется выбрать минимальное подмножество, покрывающее все случаи (если это, конечно, возможно).