**Семинар 2. Программы разветвленной и циклической структуры**

|  |  |  |  |
| --- | --- | --- | --- |
| **Структура** | **Псевдокод** | **Структура** | **Псевдокод** |
| **1. Следование** | **Действие 1**  **Действие 2** | **4. Выбор** | **Выбор Код**  **Код1: Действие 1**  **Код2: Действие 2**  **...**  **Все-выбор** |
| **2. Ветвление** | **Если Условие**  **то Действие 1**  **иначе Действие 2**  **Все-если** | **5. Цикл с заданным количеством повторений** | **Для Индекс =**  **Начало, Конец, Шаг**  **Действие**  **Все-цикл** |
| **3. Цикл-пока** | **Цикл-пока Условие**  **Действие**  **Все-цикл** | **6. Цикл-до** | **Выполнять**  **Действие**  **До Условие**  **Все-цикл** |

**Задача 1.** Табулировать функцию

при *xn* ≤ *x* ≤ *xk* c шагом *h*

**Задача 2.** Разработать программу, которая с заданной точностью ε находит значение аргумента x по заданному значению функции y при известном значении n



где *n*>1, *x*>0

**Задача 3.** Определить



при x ≥ 0

**Разбор решения задачи 1**

Для разработки алгоритма используем ***метод пошаговой детализации***. Его целесообразно использовать для разработки алгоритмов сложных программ.

При использовании этого метода сначала алгоритм представляют в виде нескольких очевидных действий (последовательности подзадач), например: ввод данных, вызов подпрограммы решения задачи и вывод результатов. Затем последовательно раскрывают каждый блок, стараясь, чтобы на каждом шаге выявилось не более одной-двух конструкций, записывая остальные действия в виде вызова подпрограмм. Проектирование завершают, когда в записи остаются действия, реализация которых для программиста очевидна.

***Шаг 1****.* Определяем общую структуру программы.

**Программа:**

Ввести a, b, xn, xk, h.

Определить количество точек *n*, в которых необходимо рассчитать значение функции.

Рассчитать и напечатать значения функции *y* в *n* точках.

**Конец.**

Ввод в детализации не нуждается. Детализируем вторую операцию.

***Шаг 2****.* Определяем количество точек *n* в заданном интервале с заданным шагом.

**Определить количество точек *n*:**

Рассчитать **.**

**Все.**

***Шаг 3****.* Детализируем расчет и вывод значений функции.

**Рассчитать и напечатать значения функции *y* в *n* точках:**

**Для** *i*=1,*n*,1

*x = xn +* (*i -*1)*\*h*.

Определить *z.*



Вывести на экран *x*, *y*.

**все-цикл.**

**Все.**

***Шаг 4****.* Детализируем определение *z*.

**Определить *z*:**

**Если** *x* ≤ a

**то z = sin x**

**иначе**

**если** *x* ≥ *b*

то *z* = 1+*x*

иначе z = x3

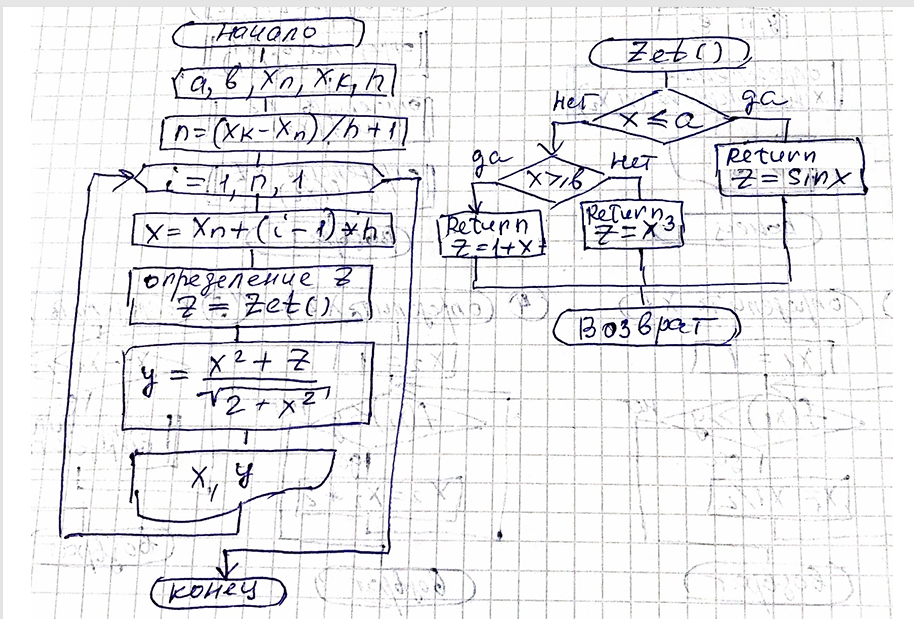
**все-если**

**все-если.**

**Все.**

Таким образом, мы получили полный алгоритм решения задачи.

**Схема алгоритма основной программы и подпрограммы**



**Задача 1\_1.** Вычислить значение функции, заданной следующим образом.

Y= где: z= sin(x), при x<=a;

z= x^3, при a<x<b;

z= 1+x, при x>=b.

x,a,b вводятся с клавиатуры и a<b.

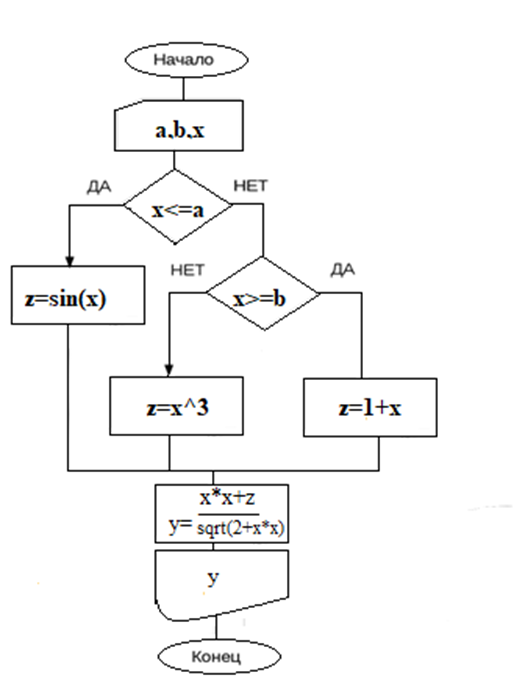


Схема алгоритма к решению Задачи 1\_1

**Текст программы**

**#include <stdio.h> // подключение процедур ввода вывода**

**#include <math.h> // подключение математических функций**

**int main()**

**{ float a,b,x,z,y;**

**puts("Input a,b, a<b :");**

**scanf("%f %f",&a,&b);**

**puts("Input x :");**

**scanf("%f",&x);**

**printf("inputed x=%7.2f, a=%7.2f, b=%7.2f.\n",x,a,b);**

**if(x<=a)**

**{z=sin(x);**

**printf("z=sin(x)= %10.5f\n",z);**

**}**

**else**

**if(x>=b)**

**{z=1+x;**

**printf("z=1+x= %10.5f\n",z);**

**}**

**else**

**{z=x\*x\*x;**

**printf("z=x\*x\*x= %10.5f\n",z);**

**}**

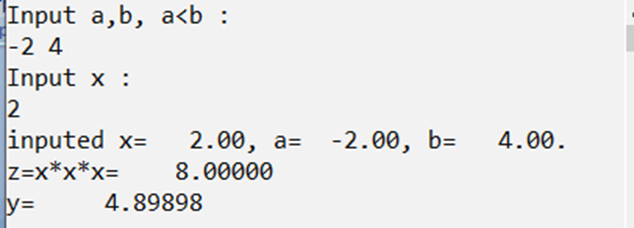
**y=(x\*x+z)/sqrt(2.0+x\*x);**

**printf("y=%12.5f\n",y);**

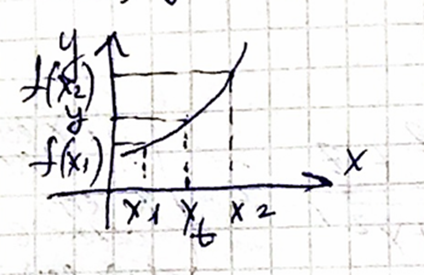
**return 0;**

**}**

**Пример работы программы**

****

**Разбор решения задачи 2**

****

При *n*>1 данная функция является монотонно возрастающей. Для нахождения значения x можно применить **метод половинного деления**. Суть данного метода заключается в следующем. Вначале определяют отрезок [*x*1,*x*2], такой что *f*(*x*1) ≤ *y* ≤*f*(*x*2). Затем делят его пополам *x*t = (*x*1 + *x*2) / 2 и определяют, в какой половине отрезка находится *x*. Для этого сравнивают *f*(*x*t) и *y*. Полученный отрезок опять делят пополам и так до тех пор, пока разность *x*1 и *x*2 не станет меньше заданного значения ε.

***Шаг 1.*** Определяем общую структуру программы.

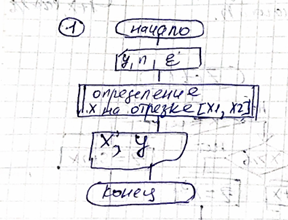
**Программа:**

Ввести *y*, *n*, eps.

Определить *x*.

Вывести *x*, *y*.

**Конец.**



***Шаг 2****.* Детализируем операцию определения *x*.

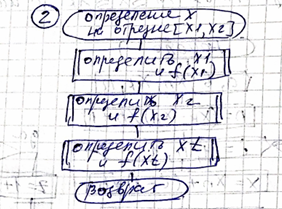
**Определить *x*:**

Определить *x*1, такое что *f*(*x*1) ≤ *y*.

Определить *x*2, такое что *f*(*x*2) ≥ *y*.

Определить *x* на интервале [*x*1, *x*2].

**Все.**



***Шаг 3.*** Детализируем операцию определения *x*1. Значение *x*1 должно быть подобрано так, чтобы выполнялось условие *f*(*x*1) ≤ *y*. Известно, что *x*>0, следовательно, можно взять некоторое значение x, например, *x*1=1, и последовательно уменьшая его, например, в два раза определить значение *x*1, удовлетворяющее данному условию.

**Определить *x*1:**

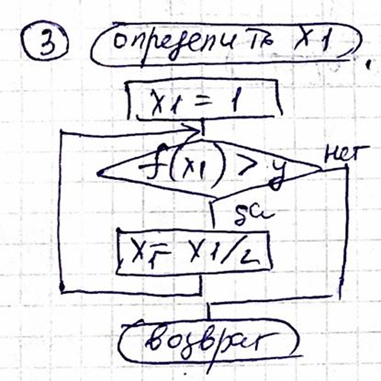
*x*1:=1

**цикл-пока** *f*(*x*1) > *y*

*x*1:=*x*1/2

**все-цикл**

**Все.**

****

***Шаг 4.*** Детализируем операцию определения *x*2. Значение *x*2 определяем аналогично *x*1, но исходное значение будем увеличивать в два раза.

**Определить *x*2:**

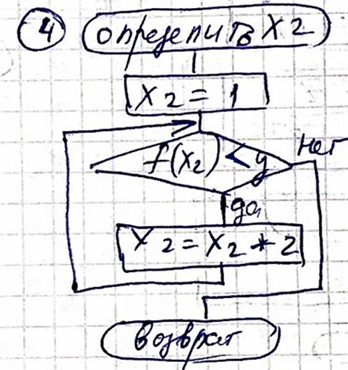
*x*2:=1

**цикл-пока** *f*(*x*2) < y

*x*2:=*x*2\*2

**все-цикл**

**Все.**

****

***Шаг 5.*** Детализируем операцию определения x. Определение x выполняется последовательным сокращением отрезка [*x*1, *x*2].

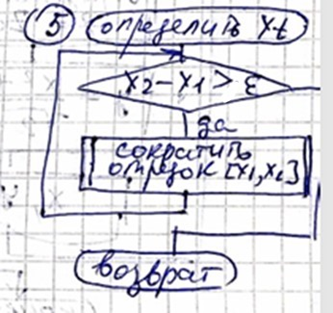
**Определить *x*:**

**цикл-пока** *x*2-x1>eps

Сократить отрезок [x1,x2].

**все-цикл**

**Все.**

****

***Шаг 6****.* Детализируем операцию сокращения интервала определения x. Сокращение отрезка достигается делением пополам и отбрасываем половины, не удовлетворяющей условию *f*(*x*1) ≤ y ≤ *f*(*x*2)

**Сократить интервал определения x:**

*xt*:=(*x*1+*x*2)/2

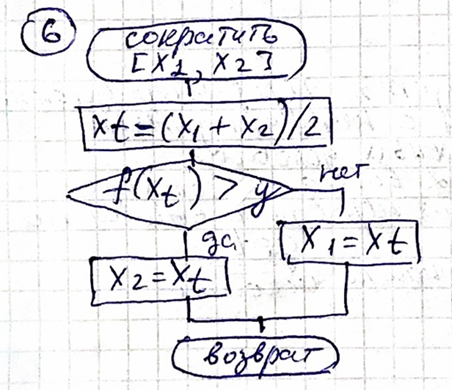
**если** *f*(*xt*)>y

**то** *x*2:=*xt*

**иначе** *x*1:=*xt*

**все-если**

**Все.**

****

Таким образом, за шесть шагов мы разработали весь алгоритм, который выглядит следующим образом.

**Программа:**

Ввести y,n,eps.

x1:=1

**цикл-пока** f(x1)>y

x1:=x1/2

**все-цикл**

x2:=1

**цикл-пока** f(x2)<y

x2:=x2\*2

**все-цикл**

**цикл-пока** x2-x1>eps

xt:=(x1+x2)/2

**если** f(xt)>y

**то** x2:=xt

**иначе** x1:=xt

**все-если**

**все-цикл**

Вывести xt,y.

**Конец.**

При разработке алгоритма методом пошаговой детализации мы использовали псевдокод, но можно было использовать и схемы алгоритма.

Достоинством метода пошаговой детализации является то, что в процессе разработки на каждом шаге решается одна достаточно простая задача. Использование метода, таким образом, существенно облегчает разработку алгоритмов.

**Разбор решения задачи 3**

Для решения этой задачи можно использовать рекурсивный алгоритм. Но рекурсия, которая при этом получается, является линейной, а потому целесообразно построить обычный цикл.

Поскольку слева направо по приведенной формуле считать не получается, попробуем организовать вычисление справа налево.

***Шаг 1****.* Определяем общую структуру программы. По условию значение *x* должно быть неотрицательным. Откуда общая структура программы определяется двумя возможными способами.

**Программа 1:**

Вводим *x*.

**Если** *x* ≥ 0

**то**

Рассчитать *s*.

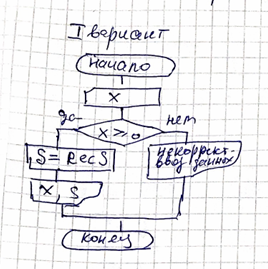
Вывести *x* и *s* на экран.

**иначе**

Вывести на экран сообщение о некорректном вводе.

**Все-если.**

**Конец.**



Или второй вариант, при котором пользователь получает возможность исправить ошибку ввода без повторного запуска программы.

**Программа 2:**

Ввести *x*.

**Цикл-пока** *x* < 0

ввести *x*

**Все-цикл.**

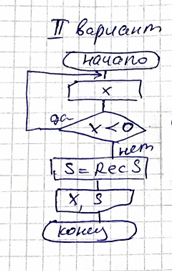
Рассчитать *s*.

Вывести *x* и *s* на экран.

**Конец.**

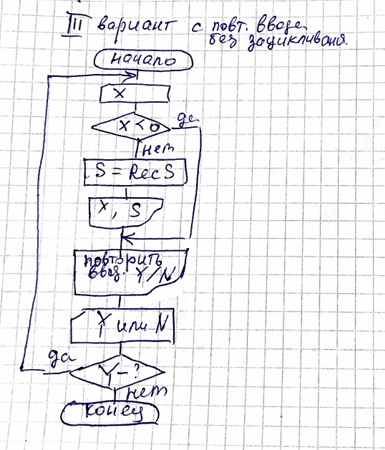
**Обратите внимание** на то, как реализованы циклы ввода данных.

В реализации алгоритма на псевдокоде – это **Цикл-пока**, а в схеме алгоритма ниже – это **Цикл-до**.



При работе со вторым вариантом, если пользователь не знает, какие значения можно вводить, ему придется программу завершать аварийно. Поэтому желательно предусмотреть для пользователя возможность выхода при вводе специальных данных, например специального слова.

**Программа 3:**



***Шаг 2****.* Теперь детализируем вычисление s. Несложно определить, что цикл, который мы будем строить, является счетным. Тогда

**Рассчитать s:**

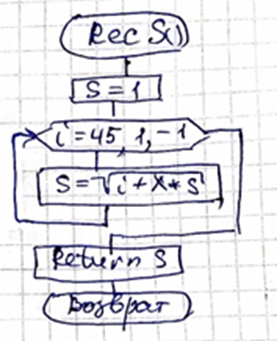
s = 1

**Для** *i* = *n*,1,-1



**все-цикл**

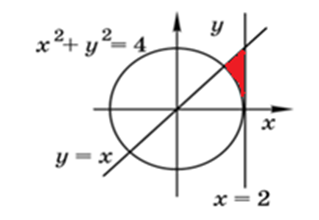
**Все.**

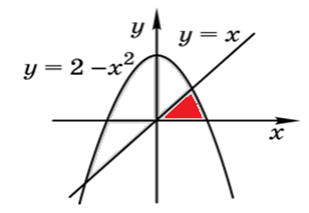


Вывод: при использовании псевдокода в принципе невозможно получить описание неструктурного алгоритма.

**Задания по желанию для выполнения дома:**

Составить схему алгоритма и написать программу проверки принадлежности точки заштрихованной области на процедурном С/С++.

******

******