# Семинар 1. Модульное программирование. Средства создания подпрограмм.

Для улучшения эффективности программ в языках высокого уровня были разработаны средства модульного программирования, предусматривающие использование подпрограмм и модулей.

***Подпрограмма* –** это относительно самостоятельный фрагмент программы, соответствующим образом оформленный и снабженный именем.

В зависимости от способов описания, формирования результата и вызова различают подпрограммы двух видов: процедуры и функции.

*Процедуры* предназначены для выполнения некоторых действий, например, печати строки, а *функция* – позволяет получить задаваемую скалярную величину, которую возвращает в качестве результата.

С\С++ декларированы как языки *функционального* программирования. Поэтому в С\С++ нет понятия процедуры, как средства языка. Однако существует возможность создания функций, которые не возвращают значения и реализуют действия, свойственные процедурам.

### **Подпрограммы в С/С++**

Каждая программа на С/С++ обязательно должна включать единственную главную функцию с именем **main.** Кроме нее в программу может входить произвольное количество функций, выполнение которых прямо или косвенно инициируется функцией main.

Функции можно описывать в любом месте программы и даже в другом файле, но только не внутри другой функции.

Различают ***объявление*** и ***описание*** функций.

***Описание функции*** состоит из заголовка, при необходимости используемого в качестве прототипа, и собственно тела функции.

**<Тип результата> <Имя подпрограммы> ([<Список параметров>])**

**{ [< Объявление локальных переменных и констант >]**

**<Операторы>**

**}**

***Объявление функций*** предполагает включение в начало программы ***прототипов*** (заголовков) всех используемых функций, тогда описания функций могут быть приведены в любом порядке. Если прототипы функций в начале программы не помещать, то все функции должны быть обязательно описаны до своего явного вызова.

**Пример объявления и описания подпрограммы:**

int max(int a, int b); // Объявление подпрограммы (прототип)

int max(int a, int b) // Описание подпрограммы

{

if (a>b) return a;

else return b;

}

Функция может получать данные двумя способами:

а) **неявно** – с использованием видимых в других функциях переменных;

б) **явно** – через параметры.

**Неявная передача:**

• приводит к большому количеству ошибок, вызванных возможной «порчей» глобальных данных подпрограммами;

• жестко связывает подпрограмму и данные, существенно снижая степень универсальности подпрограмм.

При отсутствии параметров список можно заменить служебным словом void или опустить, например: int print(void); int print();

***Пример неявной передачи данных.***

описание функции:

float prog(){return sin(x)\*x;}

вызов функции:

y=prog();

при этом в основной программе необходимо описать глобальную переменную x, через которую будут передаваться данные в функцию:

float x,y;

**Явная передача данных.**

Существует два способа передачи данных в подпрограммы через список параметров.

**Первый** из них известен как «передача параметров по значению». В этом случае в подпрограмму передаются копии фактических параметров, и никакие изменения этих копий не возвращаются в вызывающую программу.

**Второй** способ называют «передачей параметров по адресу». При использовании этого метода в подпрограмму передаются адреса фактических параметров, соответственно, все изменения этих параметров в подпрограмме происходят с переменными основной программы.

По умолчанию в языках Си и С++ применяется передача **по значению.** Если подпрограмма меняет передаваемое значение, и это измененное значение надо вернуть в вызывающую программу, то используют один из двух способов:

а) в качестве параметра применяют *указатель* и при вызове передают адрес;

б) в качестве параметра применяют *ссылку* и при вызове просто указывают имя параметра.

В обоих вариантах подпрограмма получает копию адреса, по которому надо записать возвращаемое значение.

Например:

а) возврат значения с использованием параметров-указателей:

***описание функции:***

void prog(int a,int \*b){\*b=a;}

***вызов функции:***

prog(c,&d);

б) возврат значения с использованием параметров-ссылок:

***описание функции:***

void prog(int a, int&b){b=a;}

***вызов функции:***

prog(c,d);

Если наоборот надо запретить изменение параметра, переданного адресом, то его объявляют с описателем const, например:

int prog2(const int \*a) {…} .

**Формальные и фактические параметры.**

**Формальными** называются параметры, определенные в заголовке функции при ее описании.

Каждый формальный параметр не только перечисляется (именуется), но и специфицируется (для него задается тип). Спецификация формальных параметров — это либо пусто, либо void либо список отдельных параметров.

Синтаксис описания списка формальных параметров:

**<Список формальных параметров>:**

**<тип пар.><имя пар.1>,<тип пар.2 ><имя пар.3>,<тип><имя пар.4>,…..<тип пар.><тимя пар.n>|| <пусто>|| void**

Совокупность формальных параметров определяет ***сигнатуру функции***. Сигнатура функции зависит от количества параметров, их типа и порядка размещения в спецификации формальных параметров.

Пример функции со списком формальных параметров.

float max (**float** a, **char** b) {…}

**Фактическими** называются параметры, задаваемые при вызове функции.

Синтаксис описания списка фактических параметров:

**<Список фактических параметров>:**

**<имя пар.1>,<имя пар.3>,<имя пар.4>,…..<тимя пар.n>|| <пусто>|| void**

Пример вызова функции со списком формальных параметров.

Max1= max (x,y);

***Формальные и фактические*** параметры должны совпадать:

* по количеству;
* по типу;
* по порядку следования.

Однако, имена формальных и фактических параметров могут не совпадать.

Пример:

**int k, l, n=6; float d=567.5, m=90.45;**

**void fun2(int a,float c,float b){….}//** *формальныепараметры*

**fun2(n,d,m);//** *фактические параметры*

Совокупность формальных параметров определяет *сигнатуру функции.* Сигнатура функциизависит от количества параметров, их типа и порядка размещения в спецификации формальных параметров. Сигнатура является важной характеристикой функции.

Это понятие очень важно при использовании указателей на функции и для переопределения функций, что широко используются при разработке универсальных программ.

**Возврат результатов через параметры функции (процедуры).**

Существует два способа передачи данных в подпрограммы через *параметры.* Первый из них известен как «передача параметров по значению». В этом случае в подпрограмму передаются *копии фактических* *параметров*, и никакие изменения этих копий не возвращаются в вызывающую программу.

Второй способ называют «передачей параметров по адресу». При использовании этого метода в подпрограмму передаются *адреса фактических параметров*, соответственно, все изменения этих параметров в подпрограмме происходят с переменными основной программы.

*По умолчанию в языках С/С++ применяется передача по значению.*

**Оператор возврата значения функции.**

Если описывается подпрограмма-функция, которая возвращает в вызывающую программу формируемое значение, то в теле функции обязательно наличие оператора возврата return, формирующего и передающего это значение, например:

int max(int a,int b)

{

if (a>b) return a;

else return b;

}

Вызовфункции может выглядеть, например, так:

k=max(i,j); **.**

В соответствии со стандартом, если функция, возвращающая значение, не использует оператора return**,** то в вызывающую программу возвращается ***случайное значение*** (называемое *мусором*).

При необходимости функция может содержать несколько операторов возврата return, как в примере выше.Встретив любой из них, функция формирует возвращаемое значение и передает управление в конец подпрограммы.

Если функция не объявлена со спецификацией void, то ее можно использовать в качестве операнда в любом выражении, например:

K=max(a,b)/max(c,e).

Следует отметить, что оператор return в схеме алгоритма изображается как обычный блок *процесс*, так как он только формирует результат по правилам С/С++ и передает управление на конец подпрограммы. А уже после возврата управления в точку завершения подпрограммы, последняя выполняет стандартные действия по возврату управления вызвавшей ее программе или подпрограмме.

### **Модули С/С++**

Среды Visual Studio и Qt Creator позволяют создавать и отлаживать программы, использующие не только стандартные, но и пользовательские библиотеки подпрограмм – *модули*.

Модуль C/C++ представляет из себя коллекцию ресурсов программы: константы, переменные, типы и подпрограммы и оформляется как **многофайловый проект.** Обычно сам модуль С/С++ включает два исходных файла: **заголовочный файл с расширением «.h»** и **файл реализации** (исполняемый файл) **с расширением «.cpp»**.

*Заголовочный файл* играет роль интерфейсной секции модуля. В него помещают объявление экспортируемых ресурсов модуля:

* прототипы (заголовки) процедур и функций,
* объявление переменных, типов и констант.

Заголовочный файл подключают командой **#include "<Имя модуля>.h"**, записываемой в файле реализации программы или другого модуля, если они используют ресурсы описываемого модуля.

*Файл реализации* представляет собой секцию реализации модуля. Он должен содержать команды подключения используемых модулей, описания экспортируемых процедур и функций, а также объявление внутренних ресурсов модуля.

При создании первый (главный) файл многофайлового проекта уже содержит заготовку основной функции программы – функции main(). Для создания файлов модуля и добавления их к проекту можно использовать два пути:

1. Вновь вызвать многошаговый Мастер заготовок. Это делается с использованием команды меню File/New. Выполнение этой команды при открытом проекте вызовет открытие окна Мастера заготовок на вкладке Files, на которой необходимо выбрать тип файла, добавляемого к проекту и задать его имя.
2. Использовать команду меню Project/Add New Items. Выполнение этой команды также вызовет открытие окна Мастера заготовок, с помощью которого необходимо выбрать тип файла, добавляемого к проекту и задать его имя.

Следует отметить, что модули С/С++, в отличие от модулей некоторых других языков программирования, хранятся не в откомпилированном виде, а в исходных текстах. Именно поэтому их подключают специальными директивами препроцессора, чтобы при компиляции программы исходные тексты всех файлов модуля были включены в исходный текст программы и обработаны компилятором вместе с программой.

**Пример 1.** Разработать модуль для реализации нахождения НОД двух целых чисел.

*Файл mod.h:*

#ifndef modh\_20180810

#define modh\_20180810

int nod(int a,int b);

#endif

Для оформления файлов модуля используются дополнительные команды препроцессора

*#ifndef <Идентификатор>,*

*#define <Идентификатор>,*

*#endif*

Так, команды препроцессора в примере

#*ifndef modh\_20180810,*

*#define modh\_20180810,*

*#endif****,***

в файле mod.h позволяют исключить повторную компиляцию текста программы при многократном подключении заголовочного файла за счет определения для него уникального идентификатора.

Чтобы определить уникальный идентификатор, который точно не встречается в других библиотеках, можно использовать следующую схему, которая и применена в примере для mod.h:

**<ИМЯФАЙЛА><РАСШИРЕНИЕ>\_<ГОД><МЕСЯЦ><ДЕНЬ (создания)>.**

В нашей программе это modh\_20180810, который включает как имя модуля, так и дату его создания.

*Файл mod.cpp: // файл реализации*

#include "mod.h"

int nod(int a,int b)

{

while (a!=b)

if (a>b) a=a-b;

else b=b-a;

return a;

}

*Файл main.cpp: // файл основной программы*

#include <locale.h>

#include <stdio.h>

#include <conio.h>

#include "mod.h"

int main()

{

setlocale(0,"russian");

int a=18,b=24,c;

c=nod(a,b);

printf("Наибольший общий делитель=%d\n",c);

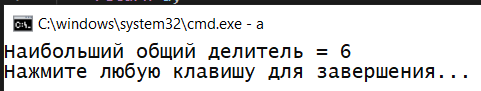
puts("Нажмите любую клавишу для завершения...");

\_getch();

return 0;

}

**Результат работы программы:**



На рисунке 1 приведена схема взаимодействия заголовочного и исполнительного файлов модуля mod.



Рисунок 1 – Схема взаимодействия файлов программы при модульном программировании.

* 1. **Применение подпрограмм при решении задач вычислительной математики**

**Задание 1.** Написать программу определения интеграла заданной функции на заданном отрезке методом прямоугольников c указанной точностью. Реализовать расчет с помощью подпрограммы.

Проверить программу для 4 функций

**Y=x;**

**Y=x^4**

**Y=sin(22\*x)**

**Y=tg(x)**

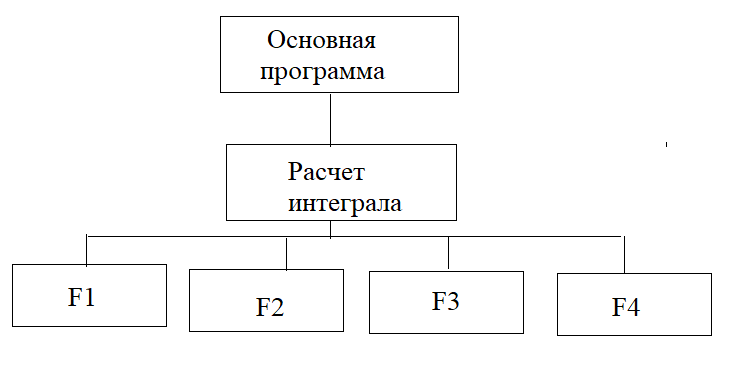
На отрезке a,b

**a=0;b=1.57;**

С точностью

**eps1=0.1;eps2=0.01;eps3=0.001;eps4=0.0001;**

**Структурная схема программы**

****

**Текст программы**

**#include <stdio.h>**

**#include <math.h>**

**float f1(float x)**

**{return x;}**

**float f2(float x)**

**{return x\*x\*x\*x;**

**}**

**float f3(float x)**

**{return sin(22\*x);}**

**float f4(float x)**

**{return tan(x);}**

**float integral(float a,float b,float eps,int &n)**

**{int i,k;**

**float s1,s2,x,d;**

**n=5;**

**d=(b-a)/n;**

**s2=1.0e+10;**

**k=0;**

**do**

**{s1=s2;**

**s2=0;n=n\*2;**

**d=d/2;**

**x=a;k++;**

**for(i=1;i<=n;i++)**

**{ s2=s2+f3(x);**

**x=x+d;**

**}**

**s2=s2\*d;**

**} while(fabs(s2-s1)>eps);**

**return s2;**

**}**

**int main()**

**{float a,b,eps1,eps2,eps3,eps4;**

**int raz=0;float integ;**

**a=0;b=1.57;**

**eps1=0.1;eps2=0.01;eps3=0.001;eps4=0.0001;**

**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");**

**puts("| Function | EPSILON | INTEGRAL | N | ");**

**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ");**

**integ=integral(a,b,eps1,*raz*);**

**printf("| Y=sin(22\*x) | %7.5f | %10.7f | %5d |\n",eps1,integ,raz);**

**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ");**

**integ=integral(a,b,eps2,*raz*);**

**printf("| Y=sin(22\*x) | %7.5f | %10.7f | %5d |\n",eps2,integ,raz);**

**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ");**

**integ=integral(a,b,eps3,*raz*);**

**printf("| Y=sin(22\*x) | %7.5f | %10.7f | %5d |\n",eps3,integ,raz);**

**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ");**

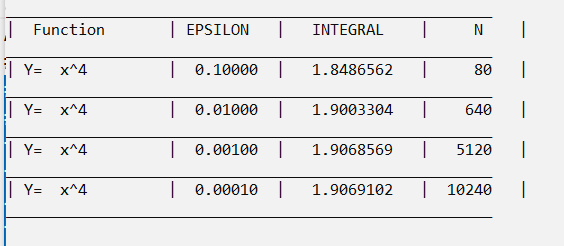
**integ=integral(a,b,eps4,*raz*);**

**printf("| Y=sin(22\*x) | %7.5f | %10.7f | %5d |\n",eps4,integ,raz);**

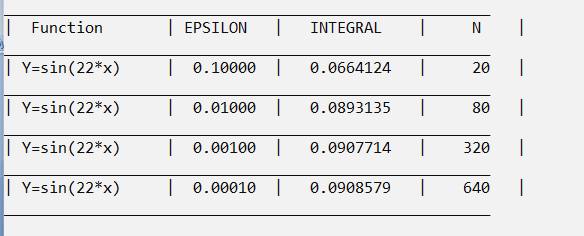
**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ");**

**return 0;**

**}**

**Пример работы для**

**y=x^4**

**Y=sin(22\*x)**

Такая реализация имеет недостаток. Хотя программа решает поставленную задачу для любой функции, которая на отрезке непрерывна, при каждом запуске программы можно определить интеграл только для одной функции.

Чтобы вычислить интеграл для другой функции, в теле подпрограммы нужно поменять **имя вызываемой функции**, что требует доступа к исходному тексту подпрограммы. Это уменьшает универсальность программы.

Однако есть возможность реализовать эту программу более универсально, используя указатель на функцию, что будет рассмотрено позже.

**Задание 2.** Написать программу табуляции заданной функции на отрезке a,b с шагом h. Для вывода таблицы значений использовать подпрограмму.

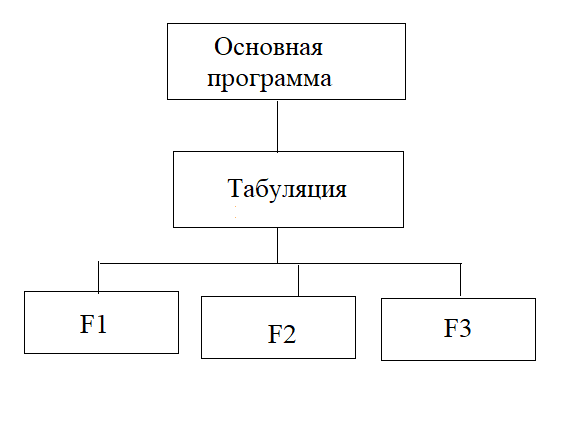
Проверить программу на нескольких функциях

**Y=x^2-1**

**Y=x-3**

**Y=x ^2\*sin(x)-x+1**

**Структурная схема программы**



**Текст программы**

**#include <stdio.h>**

**#include <math.h>**

**#include <locale.h>**

**float F1(float x)**

**{ return x\*x\*sin(x)-x+1;}**

**float F2(float x)**

**{ return x\*x-1;**

**}**

**float F3(float x)**

**{ return x-3;**

**}**

**void TablicaF(float a,float b, float h)**

**{ float fx,x;**

**x=a;**

**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");**

**puts("| Tablica funkcii |");**

**//printf("| %10s |\n",s);**

**printf("|on diapaxon %5.2f -%5.2f with step %5.3f |\n",a,b,h);**

**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");**

**puts("| X | Y |");**

**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");**

**while(x<=b)**

**{ fx=F1(x);// замена имени функции**

**printf("| %6.2f | %8.4f |\n",x,fx);**

**puts("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");**

**x=x+h;**

**}**

**}**

**int main()**

**{ float xn,xk,h;**

**setlocale(0,"russian");**

**puts("Input Xn,Xk,h for y=x^2\*sin(x)-x+1");// замена подсказки**

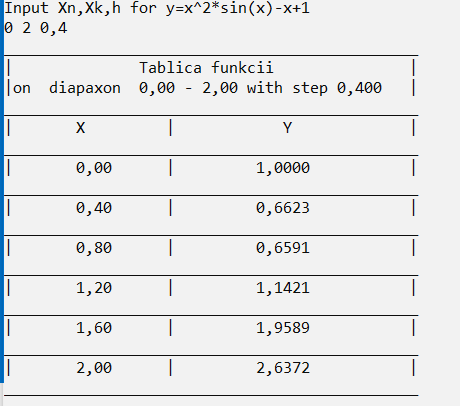
**scanf("%f %f %f",&xn,&xk,&h);**

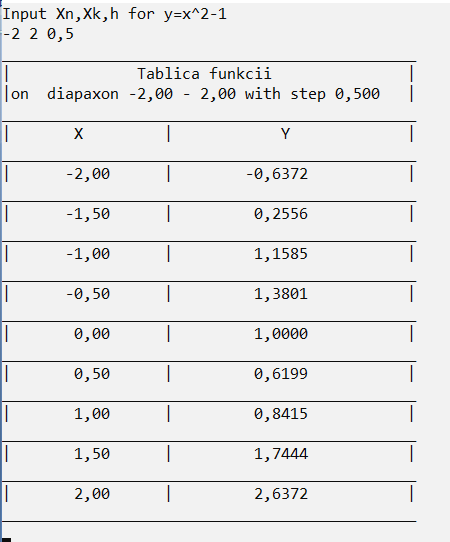
**TablicaF(xn,xk,h);**

**return 0;**

**}**

**Пример работы программы**

** Для функции F1**

 **Для функции F2**

Эта программа имеет тот же недостаток. Чтобы табулировать другую функцию, в теле подпрограммы нужно поменять **имя вызываемой функции**, что так же требует доступа к исходному тексту подпрограммы.

**Задание 3.** Написать программу нахождения корня заданной функции на заданном отрезке a,b с точностью eps **методом хорд**. Оформить вычисление корня в виде подпрограммы.

Проверить программу на нескольких функциях.

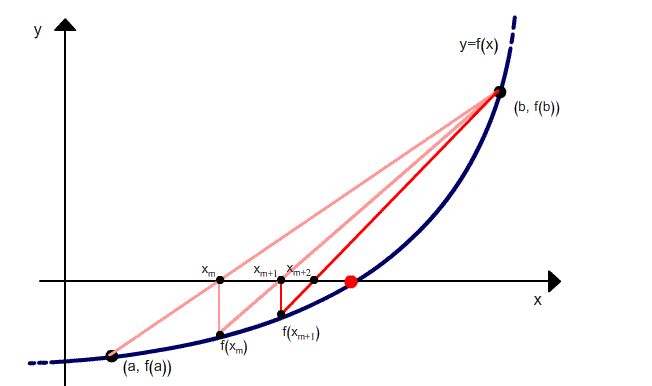
**Y = x^3+x^2-3 ; на отрезке 0.6 - 1.4**

**Y= x^2-sin(5\*x); на отрезке 0.5 - 0.6**

**Y=x^2 -1; на отрезке -2 - 2**

**Уравнение хорды:  
https://latex.codecogs.com/gif.latex?y=f(a)%2B\frac%7bf(b)-f(a)%7d%7bb-a%7d\cdot%20(x-a)**

**Алгоритм**

****

В точке x=x1, y=0, в результате получим первое приближение корня

x1=https://latex.codecogs.com/gif.latex?a-\frac%7bf(a)%7d%7bf(b)-f(a)%7d\cdot%20(b-a)

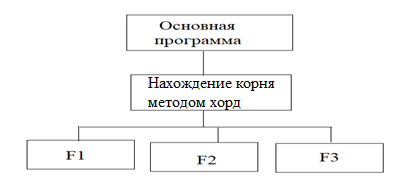
Проверяем условия:  
1. f(x1)\*f(b)<0,  
2. f(x1)\*f(a)<0.  
Если выполняется условие (1), то в формуле точку *a* заменяем на x1, получим:  
https://latex.codecogs.com/gif.latex?x_%7b2%7d%20=%20x_%7b1%7d%20-%20\frac%7bf(x_%7b1%7d)%7d%7bf(b)-f(x_%7b1%7d)%7d(b-x_%7b1%7d)  
Продолжая этот процесс, получим для n-го приближения.

Приближение хn считаем по формуле:

https://latex.codecogs.com/gif.latex?x_%7bn%7d=x_%7bn-1%7d-\frac%7bf(x_%7bn-1%7d)%7d%7bf(b)-f(x_%7bn-1%7d)%7d\cdot%20(b-x_%7bn-1%7d)

Продолжаем процесс, пока |f(x)|>eps

**Структурная схема**

****

**Текст программы**

**#include <stdio.h>**

**#include <math.h>**

**float F1(float x)**

**{ return x\*x\*x+x\*x-3; //0.6 1.4 отрезок, на котором есть корень**

**//return x\*x-sin(5\*x); // 0.5-0.6 отрезок, на котором есть корень**

**// return 5\*x\*x\*x-x-1; // 0.6 – 0.8 отрезок, на котором есть корень**

**// return x\*x-1; // -2 0, 0 2 отрезки, на которых есть корень**

**// return x\*x\*x-2\*x-5 // 1.9 - 2.93 отрезок, на котором есть корень**

**}**

**float rootHord(float a,float b, float eps)**

**{ float fx,fa,fb,x;**

**fa=F1(a);**

**fb=F1(b);**

**x=a-fa/(fb-fa)\*(b-a); // формула расчета очередного приближения х**

**fx=F1(x);**

**while(fabs(fx)>=eps)**

**{**

**if (fx\*fa<0)**

**{ fb=fx; b=x;}**

**else**

**{fa=fx; a=x;}**

**x=a-fa/(fb-fa)\*(b-a); // формула расчета очередного приближения х**

**fx=F1(x);**

**}**

**return x;**

**}**

**int main()**

**{float xn,xk,eps;**

**puts("Input Xn,Xk,eps");**

**scanf("%f %f %f",&xn,&xk,&eps);**

**printf("xn=%7.2f xk=%7.2f eps=%10.7f\n",xn,xk,eps);**

**if (F1(xn)\*F1(xk)<0) // Корень есть**

**{**

**printf("Root F1 on %7.3f - %7.3f raven ",xn,xk);**

**printf("%8.6f\n",rootHord(xn,xk,eps)); // вызов подпрограммы для F1(x)**

**}**

**else // Корень отсутствует**

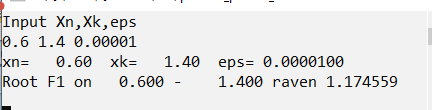
**{ printf("Root F1 on %7.3f - %7.3f ",xn,xk);**

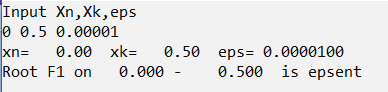
**printf("is epsent\n");**

**}**

**return 0;**

**}**

****

****

Эта программа тоже имеет тот же недостаток, что и две предыдущих. Чтобы найти корень другой функции, в теле подпрограммы нужно поменять формулу вычисляемой функции, что так же требует доступа к исходному тексту подпрограммы.

**Задание на дом. Переделать программы в многофайловые проекты (модуль)**