Структурное программирование на языке Си

1. Теория
2. Проект
3. Графики функций
4. Точки пересечения
5. Штриховка
6. Вычисление площади
7. Оформление отчета

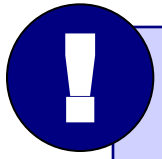
Структурное программирование на языке Си

Тема 1. Теория

Этапы разработки программ

1. Постановка задачи

- определить **цель** и **категорию** программы (системная, прикладная)
- определить **исходные данные** и требуемый **результат**
- проверить, является ли задача **хорошо поставленной** (должны быть определены все связи между исходными данными и результатом)



Плохо поставленные задачи:

- не хватает исходных данных
- заданы не все связи между исходными данными и результатом
- задача не имеет решения
- задача имеет множество решений

- Зафиксировать требования к программе **в письменной форме**

Этапы разработки программ

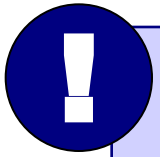
2. **Разработка модели данных**
   - формальная модель
   - типы данных (массивы, структуры, …)
   - взаимосвязь между данными
3. **Разработка алгоритма**
   - выбор существующего или разработка нового
   - возможен возврат к шагу 2
4. **Разработка программы**

   Языки: C, C++, Visual Basic, Delphi (Паскаль), `…
5. **Отладка программы** (поиск и исправление ошибок)

   debug – извлечение жучков (*bug*), 1945, МАРК-I
   - **отладчик** (точки останова, пошаговый режим, просмотр переменных)
   - **профайлер** (сколько выполняется каждая из процедур)

Этапы разработки программ

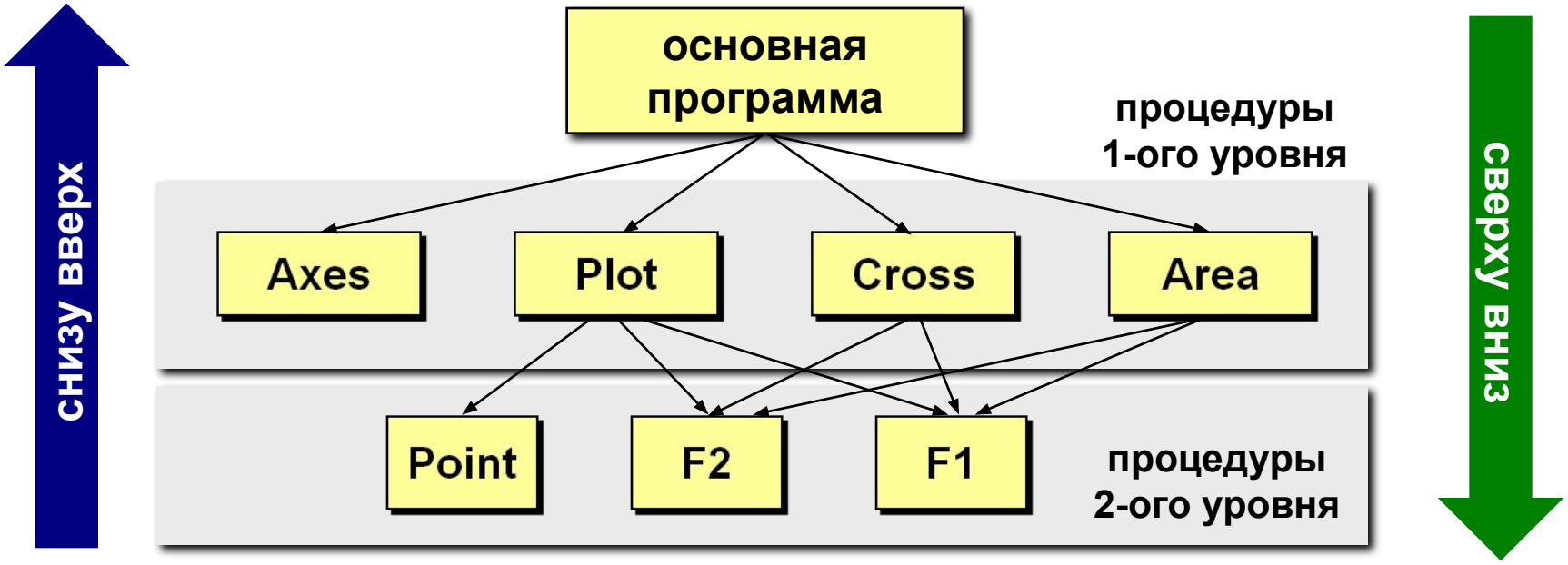
6. **Тестирование программы** (проверка на исходных данных, для которых известен результат)
   - **альфа**-тестирование: внутри фирмы (тестеры)
   - **бета**-тестирование: в других организациях, распространение через Интернет



Тестирование может показать наличие ошибок, но не их отсутствие.

7. **Разработка документации**
   - справочная система
   - руководство пользователя (*User Manual*)
   - руководство разработчика
8. **Сопровождение (техническая поддержка)**
   - исправление ошибок, найденных заказчиком
   - обучение и консультирование заказчика
   - новые версии по льготной цене

Методы проектирования программ



Проектирование «снизу вверх»

сначала составляются процедуры нижнего уровня, из которых затем «собираются» процедуры более высокого уровня.

- **легче начать** программировать
- более **эффективные** процедуры

- процедуры необходимо связывать с основной задачей («**держать в голове**»)
- при окончательной сборке может **не хватить** «кубиков»
- часто программа получается **запутанной**
- сложно распределить работу **в команде**

Проектирование «сверху вниз»

метод последовательного уточнения:

1) начинаем **с основной программы**;
2) она разбивается на подзадачи, для каждой из которых пишется процедура-«**заглушка**»;
3) реализуем каждую из процедур тем же способом.

- меньше вероятность **принципиальной ошибки** (начали с главного)
- проще **структура** программы
- удобно **распределять** работу в команде

- в разных блоках могут быть реализованы похожие операции (можно было решить одной **общей процедурой**), особенно в команде

Структурное программирование

Существовавшие проблемы:

- увеличилась **сложность** программ
- сократилось **время** на разработку

Цели:

- повысить **надежность**
- уменьшить **время** и **стоимость** разработки
- облегчить тестирование и **отладку**
- возможность **переделки** одного модуля
- улучшить **читабельность**
  - без переходов на другую страницу
  - избегать трюков и запутанных приемов

Структурное программирование

Принципы:

- **абстракции**: программу можно рассматривать на любом уровне без лишних подробностей
- **модульности**: программа разбивается на отдельные модули, которые могут отлаживаться независимо друг от друга
- **подчиненности**: связь между модулями «сверху вниз»
- **локальности**: каждый модуль использует только свои локальные переменные, глобальные переменные только в крайних случаях

Модуль

Модуль – это программный блок (процедура или функция), отделенный от кода других модулей, который полностью решает самостоятельную задачу своего уровня.

- работа модуля не зависит от того, **откуда** он вызывается, и от того, сколько раз он вызывался **до этого**
- размер модуля не более **50-60 строк** (1 страница)
- модуль имеет **один вход** и один **выход**
- модуль начинается с «шапки»-**комментария** (входные данные, результаты, какие модули использует)
- имена переменных – **смысловые**
- в одной строке – один оператор
- «трюки» – долой

Оформление текста программы

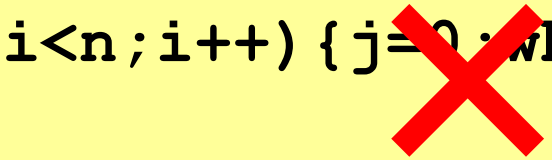
Шапка – комментарий в начале процедур и функций.

```
//-----------------------------------------
// Sum сумма элементов массива
// Вход: A[] - массив целых чисел
//         n  - размер массива
// Выход: S = A[0]+A[1]+...+A[n-1]
// Вызывает: -
//-----------------------------------------
int Sum ( int A[], float n )
{
...
}
```

Оформление текста программы

Отступы – тело цикла, условного оператора, оператора выбора и т.п. сдвигается вправо на 2-3 символа.

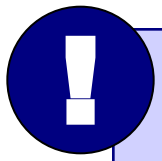
```
for(i=0;i<n;i++){j=0;while(j<i){j++;k=k%N;}
k++;}
```



лесенка

```
for ( i=0; i<n; i++)
  {
  j=0;
  while ( j < i )
    {
    j++;
    k = k % N;
    }
  k++;
  }
```

- легче читать текст программы
- видны блоки
- просто найти ошибки со скобками (лишняя, не хватает)



Скорость работы программы не меняется!

Оформление текста программы

- «говорящие» имена функций, процедур, переменных: **Sum**, **ShowMenu**, **count**, **speed**.
- пробелы в операторах

```
for(i=0;i<n;i++)a+=10;
```

```
for ( i=0; i<n; i++ )
  a += 10;
```

- выделение пустыми строками и комментариями важных блоков

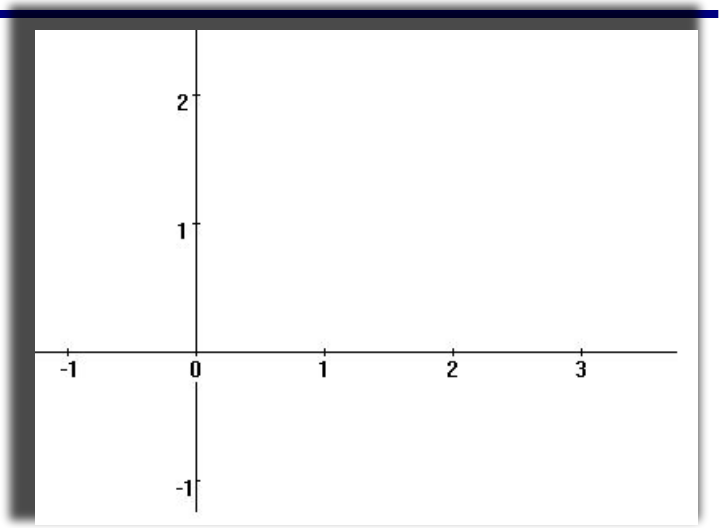
```
  // ввод данных
printf( "Введите число\n" );
scanf ( "%d", &n );
  //вычисления
n2 = n*n;
  // вывод результата
printf ( "Его квадрат %d", n2 );
```

Структурное программирование на языке Си

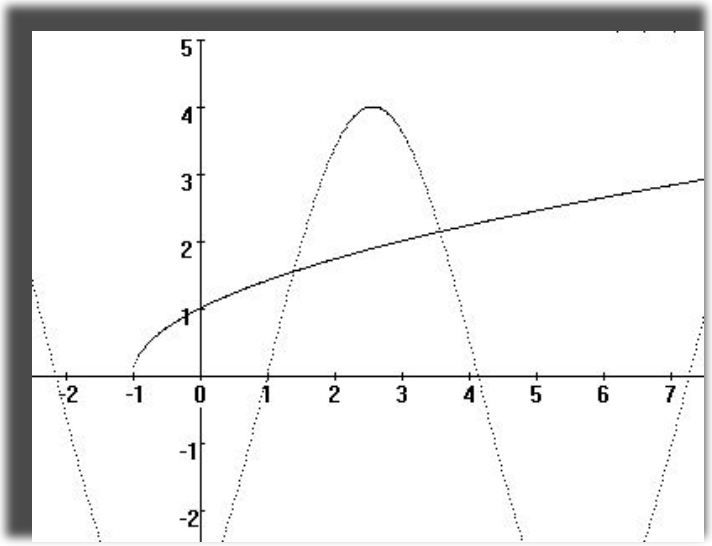
Тема 2. Проект

Проект «Графики функций»

- построить координатные **оси** и сделать их разметку

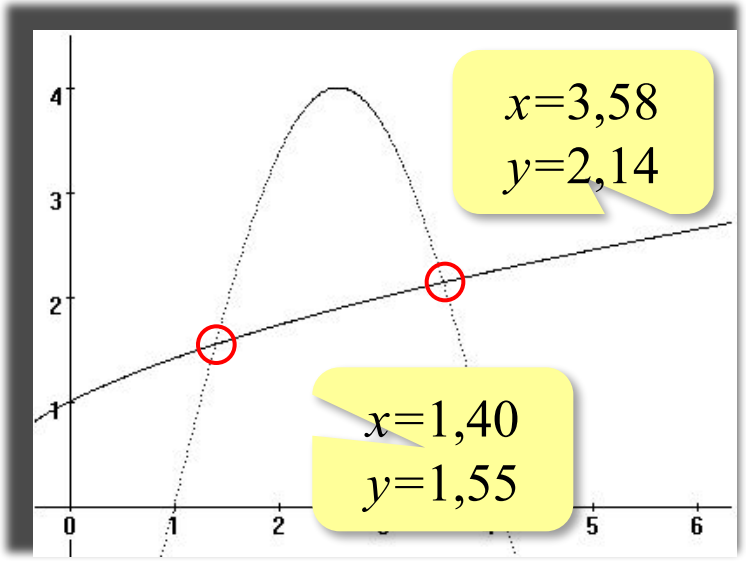


- построить **графики** заданных функций (по вариантам)

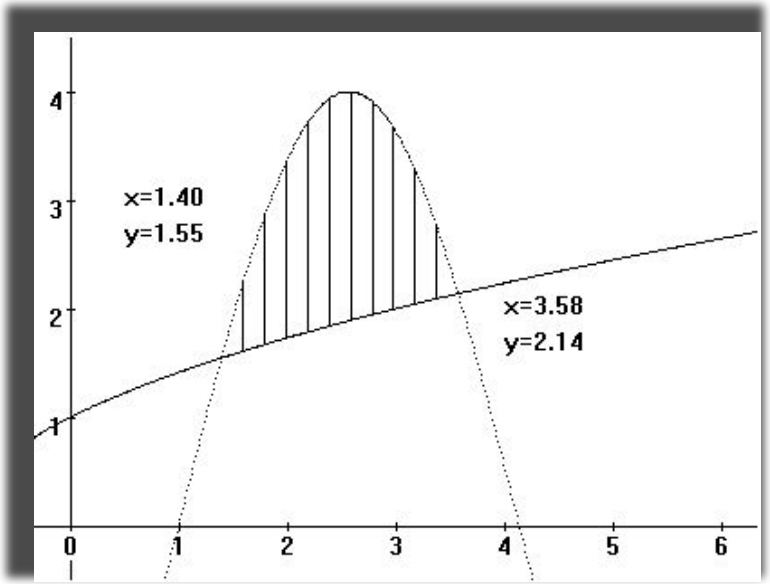


Проект «Графики функций»

- найти **точки пересечения** графиков, используя численные методы

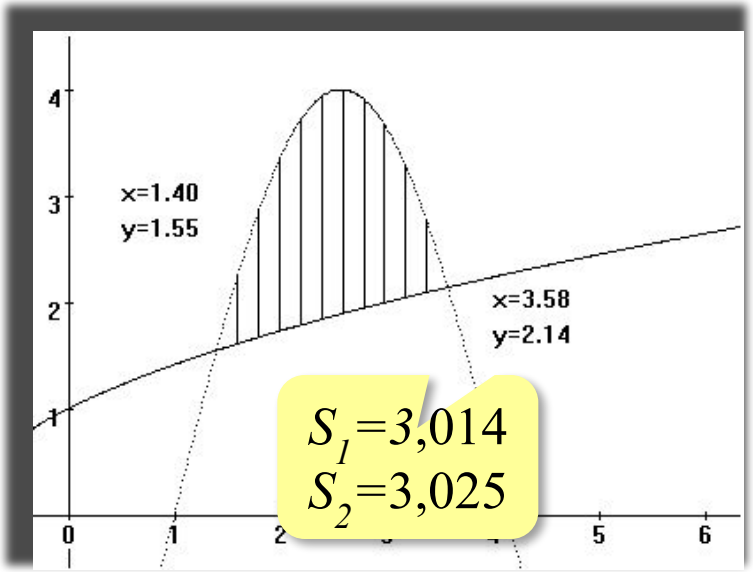


- **заштриховать** образованную замкнутую область

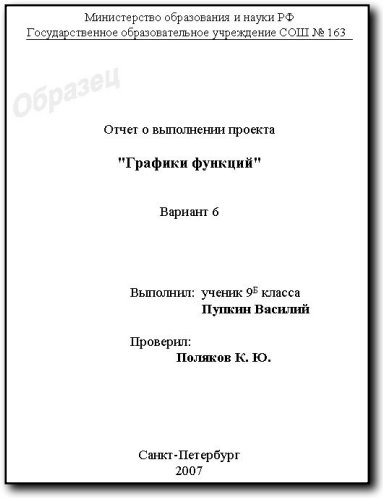


Проект «Графики функций»

- вычислить **площадь** этой области двумя способами



- оформить **отчет** по работе


Министерство образования и науки РФ
Государственное образовательное учреждение СОШ № 163

Образец

Отчет о выполнении проекта

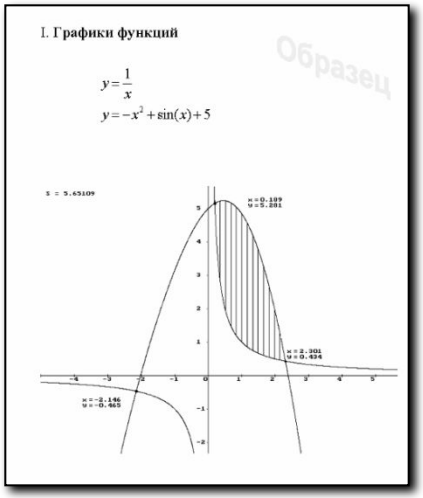
"Графики функций"

Вариант 6

Выполнил: ученик 9Б класса
Пупкин Василий

Проверил:
Поляков К. Ю.

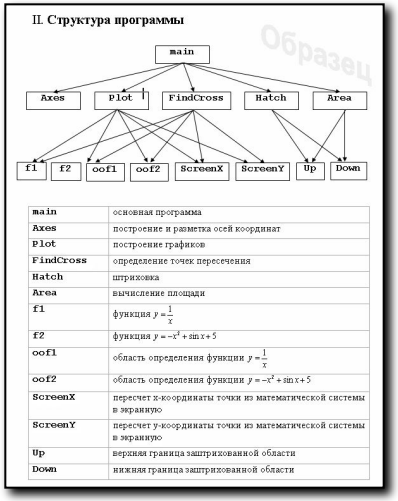
Санкт-Петербург
2007


I. **Графики функций**

Образец

$y=\frac{1}{x}$

$y=-x^2+\sin(x)+5$


II. **Структура программы**

Образец

main	основная программа
Axes	построение и разметка осей координат
Plot	построение графиков
FindCross	определение точек пересечения
Hatch	штриховка
Area	вычисление площади
f1	функция $y=\frac{1}{x}$
f2	функция $y=-x^2+\sin x+5$
oof1	область определения функции $y=\frac{1}{x}$
oof2	область определения функции $y=-x^2+\sin x+5$
ScreenX	пересчет x-координаты точки из математической системы в экранную
ScreenY	пересчет y-координаты точки из математической системы в экранную
Up	верхняя граница заштрихованной области
Down	нижняя граница заштрихованной области


III. **Текст программы**

Образец

Структура программы

```
#include<graphics.h>
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

глобальные константы и переменные

процедуры и функции

```
main()
{
 initwindow(800,600);
```

основная программа

```
 getch();
 closegraph();
}
```

Разбивка программы на этапы

Основная программа

```
Axes();  // оси координат
Plot();  // графики функций
Cross(); // точки пересечения графиков
Hatch(); // штриховка
Area();  // площадь (способ 1)
Area2(); // площадь (способ 2)
```

Процедуры-заглушки

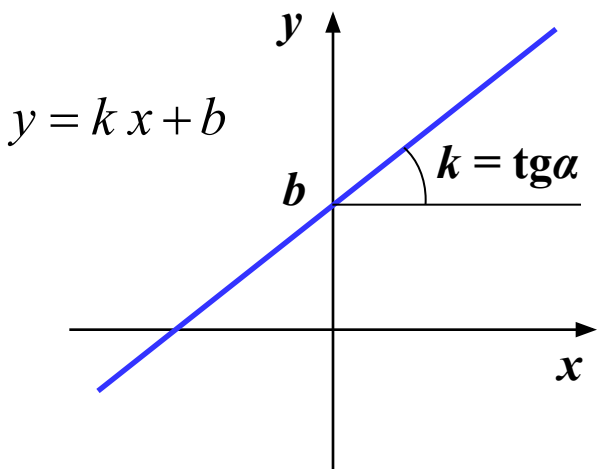
```
//-------------------------------------
// Axes  оси координат
//-------------------------------------
void Axes()
{
}
```

Структурное программирование на языке Си

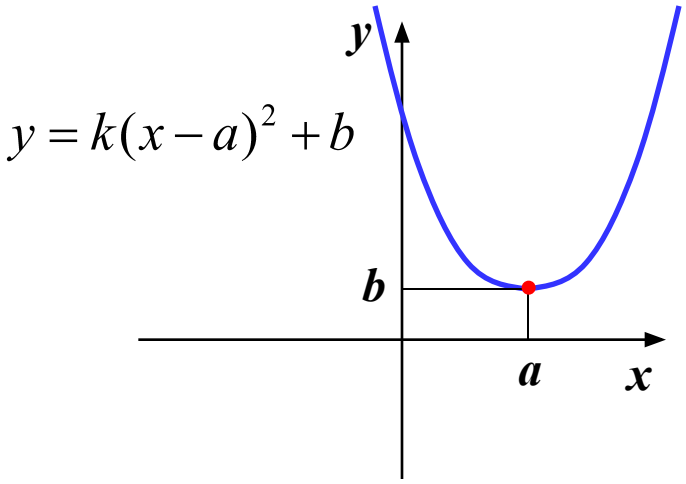
Тема 3. Графики функций

Функции, заданные в явном виде $y = f(x)$

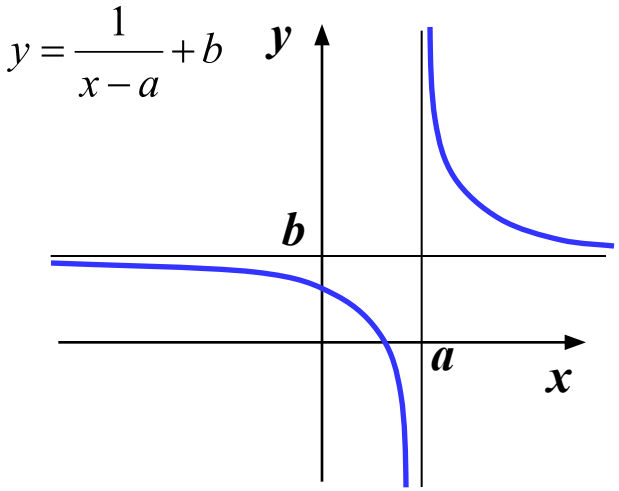
прямая



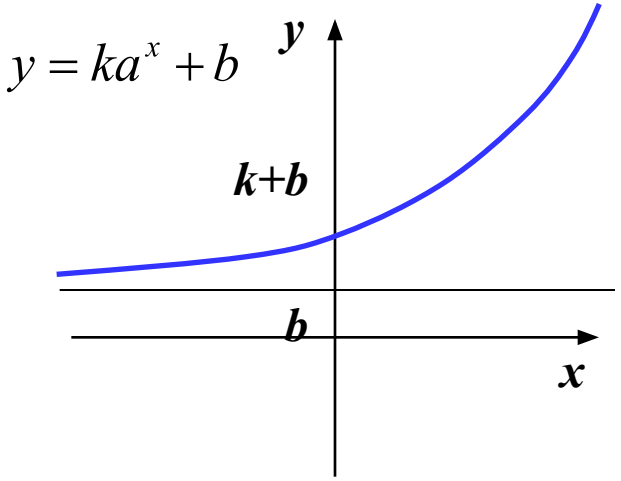
парабола



гипербола



степенная функция

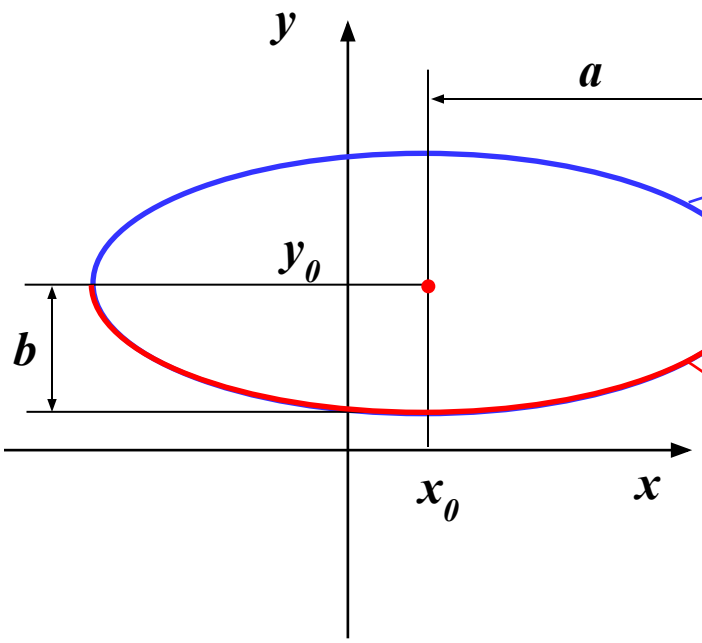


Функции, заданные в неявном виде

$f(x, y) = 0$

пример: уравнение **эллипса**

$$\frac{(x-x_0)^2}{a^2}+\frac{(y-y_0)^2}{b^2}=1 \quad \Leftrightarrow \quad \frac{(x-x_0)^2}{a^2}+\frac{(y-y_0)^2}{b^2}-1=0$$



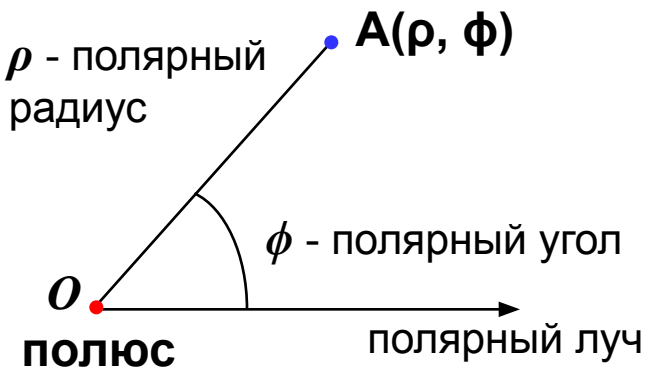
$$\frac{(y-y_0)^2}{b^2}=1-\frac{(x-x_0)^2}{a^2}$$

$$(y-y_0)^2=b^2\left(1-\frac{(x-x_0)^2}{a^2}\right)$$

$$y-y_0=\pm\, b\sqrt{1-\frac{(x-x_0)^2}{a^2}}$$

$$y=y_0\pm\, b\sqrt{1-\frac{(x-x_0)^2}{a^2}}$$

Полярные координаты

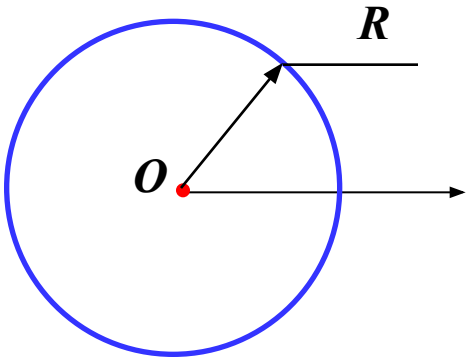


$$\rho = f(\phi)$$

Описание фигур, полученных при **вращении** объектов.

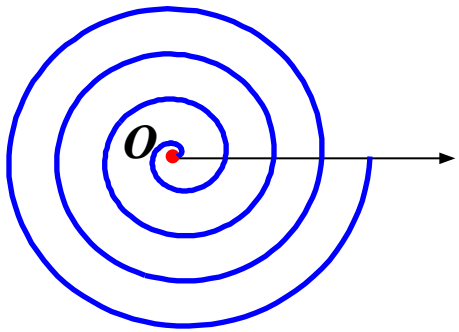
Примеры:

$$\rho = R$$



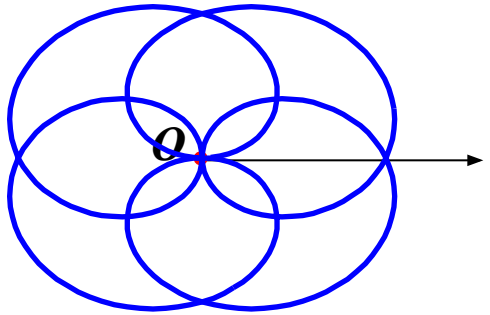
окружность

$$\rho = a \cdot \phi$$



спираль Архимеда

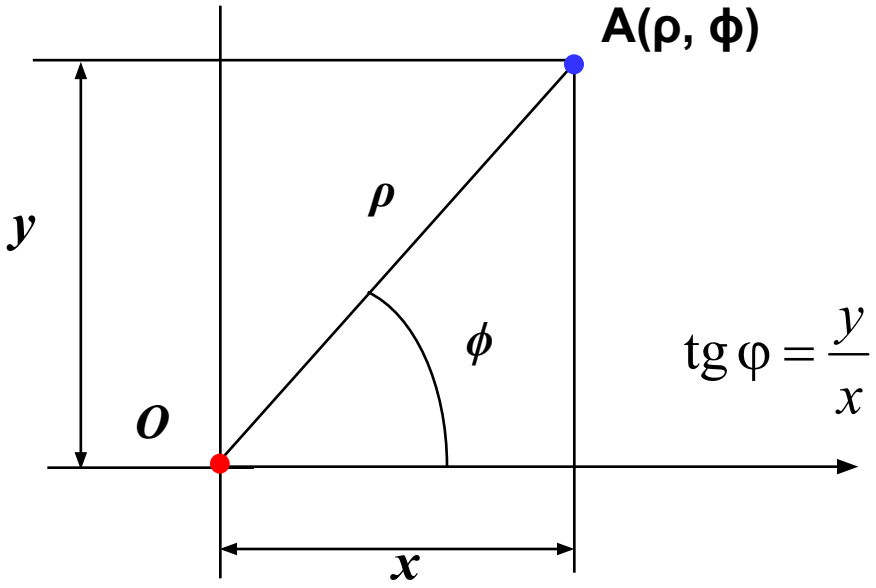
$$\rho = a \cdot \sin(2\phi/3)$$



«роза»

Полярные координаты

Переход к декартовым координатам



$$x = \rho \cdot \cos(\varphi) \qquad \rho = \sqrt{x^2 + y^2}$$

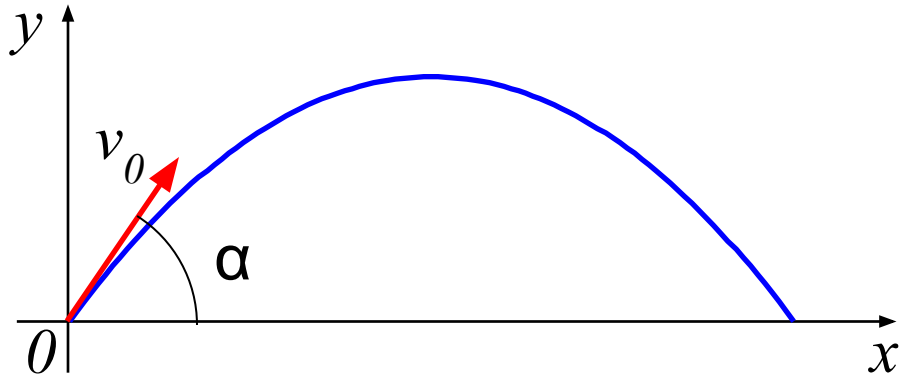
$$y = \rho \cdot \sin(\varphi) \qquad \varphi = \operatorname{arctg}\frac{y}{x}$$

Описание в параметрической форме

$$x = f_1(t)$$
$$y = f_2(t)$$

t – независимый параметр («время»)

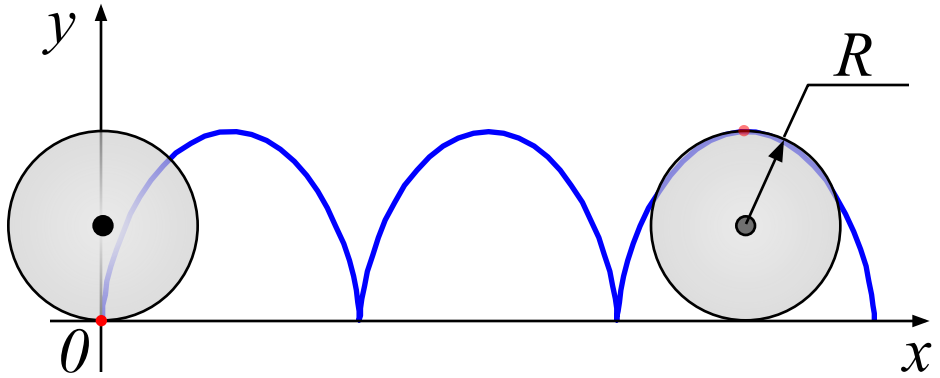
Описание фигур, полученных при сложном **движении** объектов.



$$x = v_0 \cdot \cos(\alpha) \cdot t$$

$$y = v_0 \cdot \sin(\alpha) \cdot t - \frac{gt^2}{2}$$

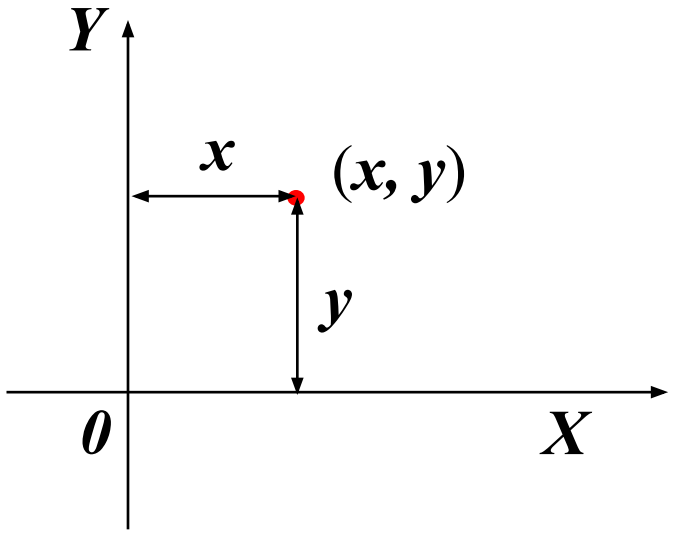
Циклоида – траектория точки на ободе колеса при вращении



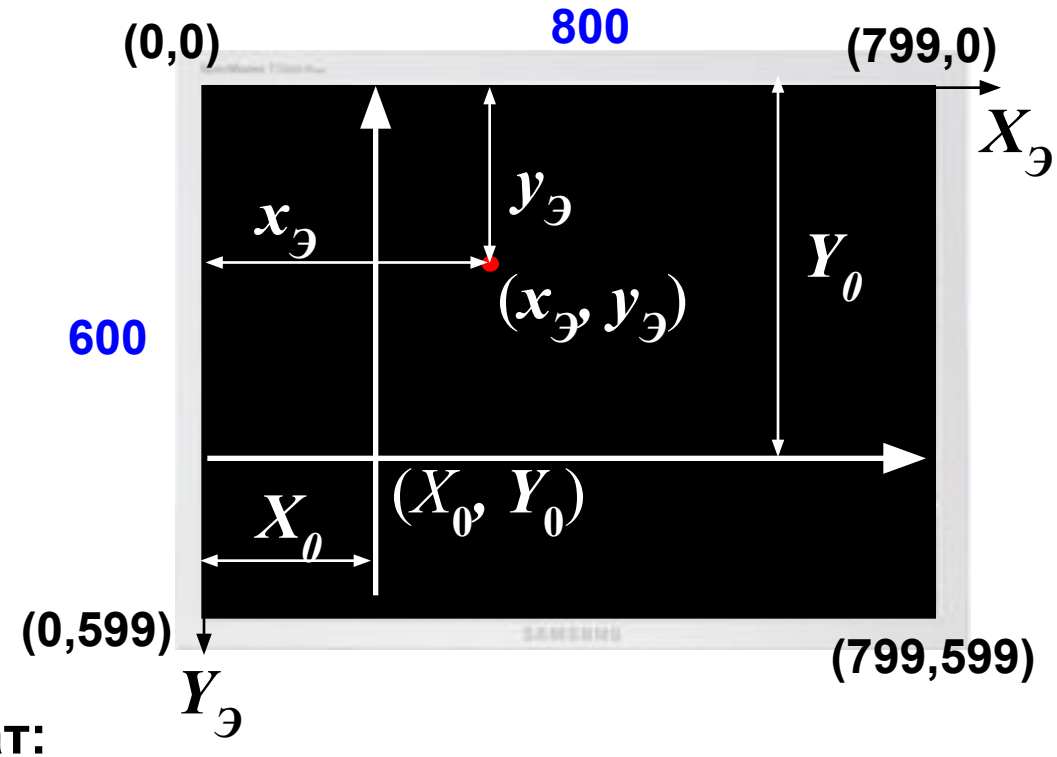
$$x = R \cdot (t - \sin t)$$
$$y = R \cdot (1 - \cos t)$$

Системы координат

Математическая



Экранная



Преобразование координат:

X_0, Y_0 – экранные координаты точки (0,0)

k – **масштаб** (во сколько раз растягивается единичный отрезок)

$$x_Э = X_0 + k \cdot x$$

$$y_Э = Y_0 - k \cdot y$$

? **Почему «минус»?**

Структура программы

```
#include<graphics.h>
#include<stdio.h>
#include<conio.h>
#include<math.h>
const int X0 = 100, Y0 = 400, // начало координат
      k = 80;                 // масштаб
```

глобальные переменные

процедуры и функции

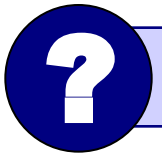
```
main()
{
 initwindow(800, 600);
```

основная часть

```
 getch();
 closegraph();
}
```

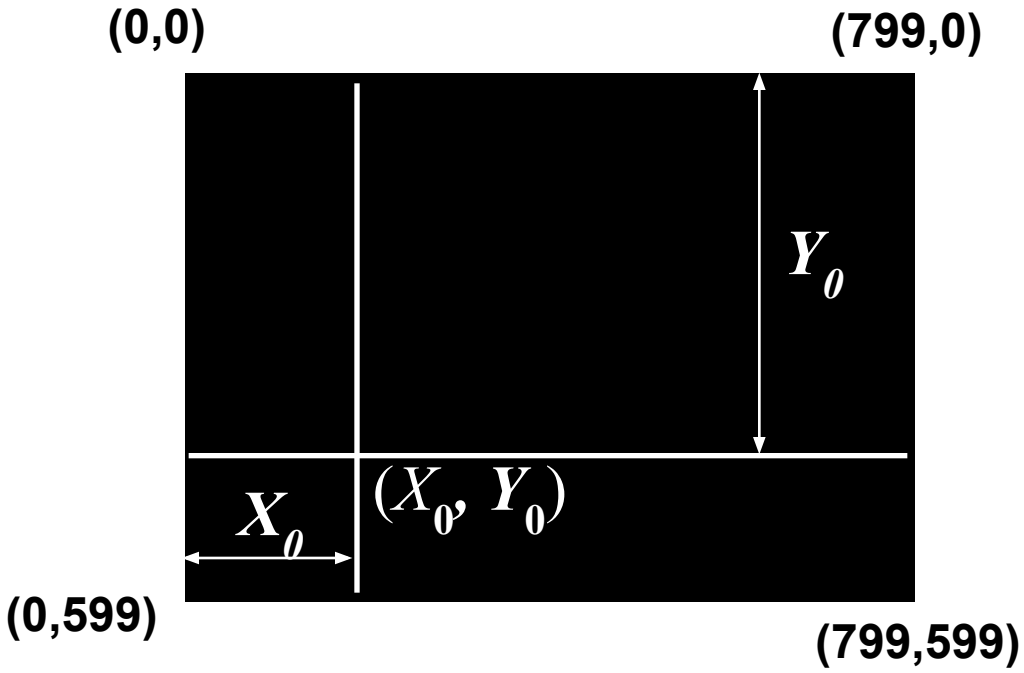
Перевод в экранные координаты

```
//-----------------------------------------
// SCREENX – перевод X в координаты экрана
//-----------------------------------------
int ScreenX (float x)
{
 return  X0+k*x;
}
//-----------------------------------------
// SCREENY – перевод Y в координаты экрана
//-----------------------------------------
int ScreenY (float y)
{
 return  Y0-k*y;
}
```



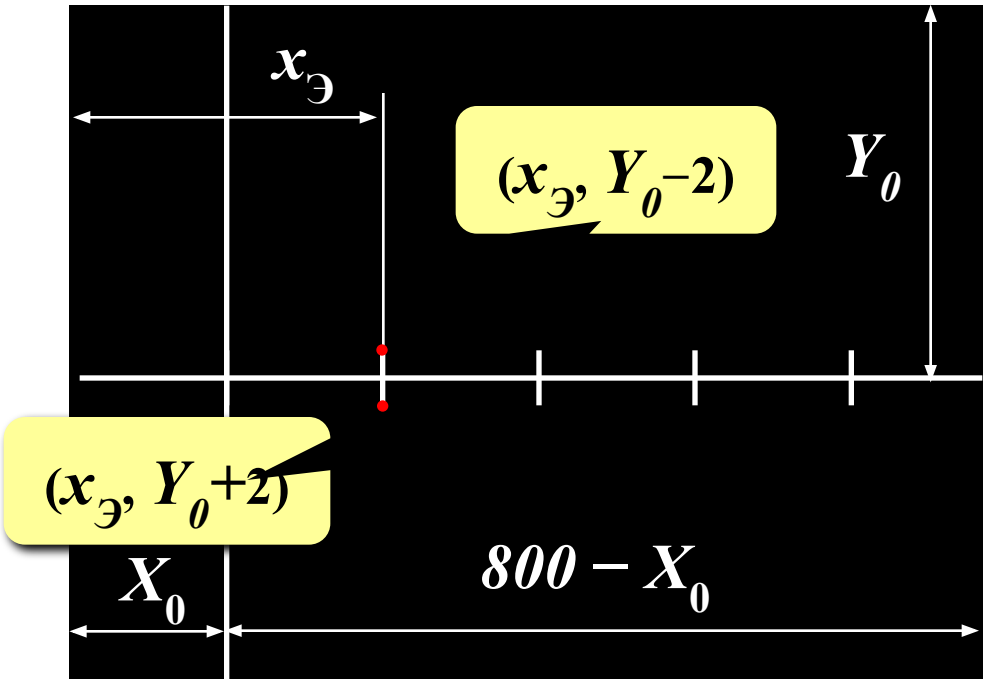
Откуда берутся X_0, Y_0 и k?

Оси координат




```
void Axes()
{
 line ( X0, 0, X0, 599 );
 line ( 0, Y0, 799, Y0 );
}
```

Разметка оси X («черточки»)



Число меток на $[0, x_{max}]$:

длина $800 - X_0$

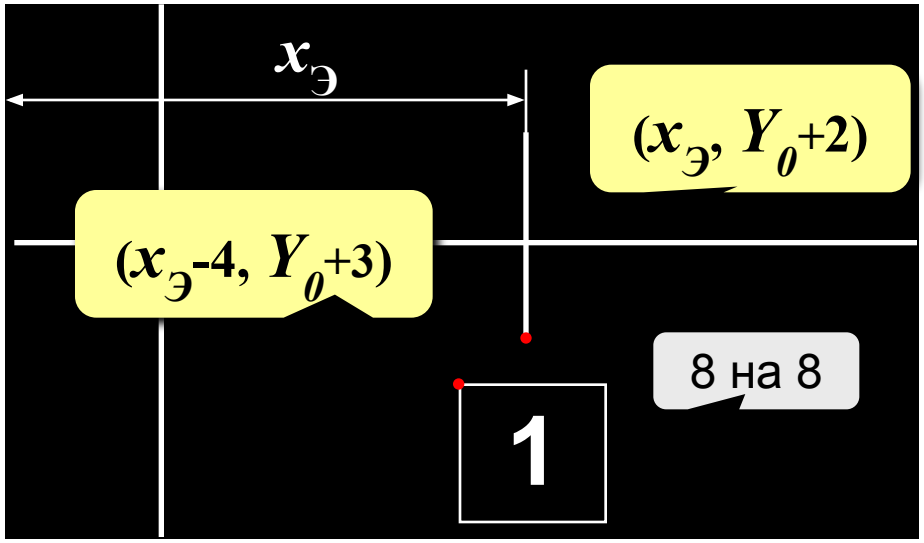
единичный отрезок k

$$N = \frac{800 - X_0}{k}$$

```
void Axes()
{
  int i, xe;
  ...
  for (i=1; i<=(800-X0)/k; i++) {
    xe = ScreenX(i);
    line ( xe, Y0-2, xe, Y0+2 );
    }
}
```

переходим к экранным координатам

Разметка оси X (числа)



Вывод символьной строки в графическом режиме:

```
outtextxy(x, y, s);
```

координаты левого верхнего угла

строка:
```
char s[5];
```

```
void Axes()
{
  char s[5];
  ...
  for (i=1; i<=(800-X0)/k; i++) {
    ...
    sprintf ( s, "%d", i );
    outtextxy ( xe-4, Y0+3, s );
    }
}
```

с запасом…

перевести целое число **i** в строку **s**

вывести строку **s** на экран

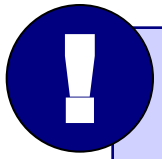
Оси с разметкой (полностью)

```
void Axes()
{
  int i, xe, ye;
  char s[5];

  line ( X0, 0, X0, 599 );
  line ( 0, Y0, 799, Y0 );

  for (i=1; i<=(800-X0)/k; i++)
    {
    xe = ScreenX(i);
    line ( xe, Y0-2, xe, Y0+2 );
    sprintf ( s, "%d", i );
    outtextxy ( xe-4, Y0+3, s );
    }
  ...
}
```

Задания

«4»: Сделать разметку осей полностью (не только положительной части оси X).

«5»: Сделать задание на «4», использовав только 2 цикла (1 цикл для каждой оси).

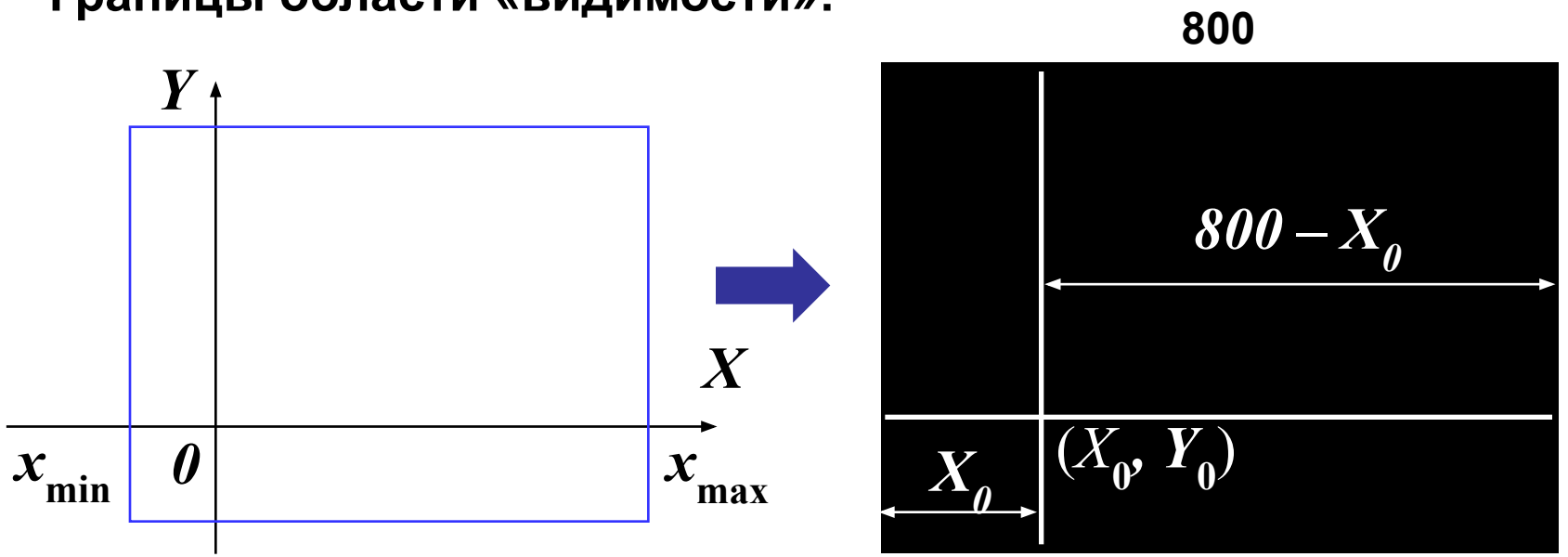


Разметка должна работать правильно при любых значениях X_0 и Y_0.

Построение графика по точкам

- Нельзя рисовать за границами экрана (зависание!).
- Область определения функции (деление на ноль, корень из отрицательного числа, …)!

Границы области «видимости»:



$$x_{\min} = -\frac{X_0}{k}, \qquad x_{\max} = \frac{800 - X_0}{k}$$

Вывод точки с проверкой

```
//-------------------------------------
// POINT вывод пикселя с проверкой и
//       пересчетом координат
//-------------------------------------
void Point ( float x, float y, int color )
{
  int xe, ye;
  xe = ScreenX(x);
  ye = ScreenY(y);
  if ( xe >= 0  &&  xe < 800  &&
       ye >= 0  &&  ye < 600)
    putpixel(xe, ye, color);
}
```

координаты в математической системе

цвет точки

если точка $(x_э, y_э)$ в пределах экрана…

Описание функций

```
//-------------------------------------
// F1, F2
// Вход: x
// Выход: y = f(x)
//-------------------------------------
float f1 ( float x )
{
  return sqrt(x+1);
}
float f2 ( float x )
{
  return 4*sin(x-1);
}
```

$$f_1(x) = \sqrt{x+1}$$

$$f_2(x) = 4\sin(x-1)$$

Области определения

Для $f_1(x) = \sqrt{x+1}$

```
//-------------------------------------
// ODZ1 - область определения f1(x)
// Вход:  x
// Выход: 1, если x входит в ОДЗ
//        0, если x не входит в ОДЗ
//-------------------------------------
int odz1 ( float x )
{
  return ( x >= -1 );
}
```

```
if ( x >= -1 ) return 1;
else           return 0;
```

Для $f_2(x) = 4\sin(x-1)$ **не нужно!**

Вывод графика функции

```
//-------------------------------------
// PLOT вывод графиков функций
//-------------------------------------
void Plot ()
{
  float xmin = - 1.* X0 / k,
     xmax = (800. - X0) / k;
  float x,
      h = (xmax - xmin) / 1000;
  for ( x = xmin; x <= xmax; x += h)
      if ( odz1(x) )
         Point(x, f1(x), LIGHTRED);
}
```

чтобы не отбрасывать остаток

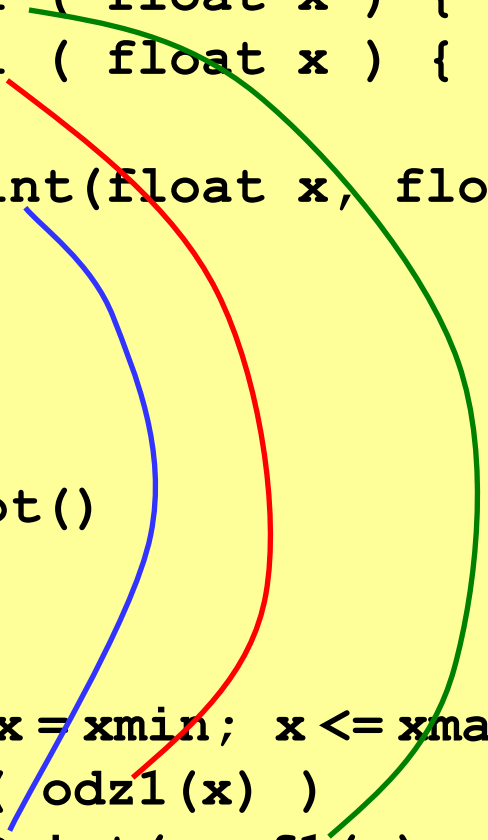
границы видимой части

шаг по x

Что плохо?

Общее расположение

```
float f1 ( float x ) { return sqrt( x + 1 ); }
int odz1 ( float x ) { return x >= -1; }
...
void Point(float x, float y, int color)
{
...
}
...
void Plot()
{
  ...
  for ( x = xmin; x <= xmax; x += h)
    if ( odz1(x) )
       Point(x, f1(x), LIGHTRED);
}
```



Задания

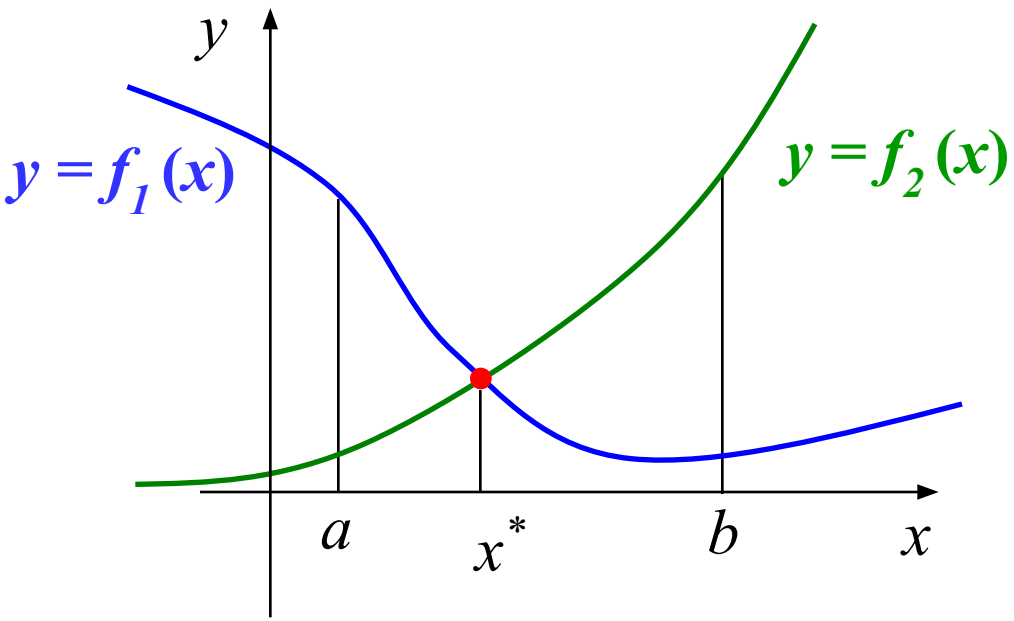
«4»: Построить графики в соответствии с заданием.

«5»: Построить графики, соединив точки линиями.

Структурное программирование на языке Си

Тема 4. Точки пересечения

Точки пересечения



Точка пересечения:

$$f_1(x^*) = f_2(x^*)$$

$$f_1(x^*) - f_2(x^*) = 0$$

$$f(x^*) = 0$$

Пример:

$$f_1(x) = \sqrt{x+1}$$

$$f_2(x) = 4\sin(x-1)$$

$$\sqrt{x+1} - 4\sin(x-1) = 0$$

Проблема:

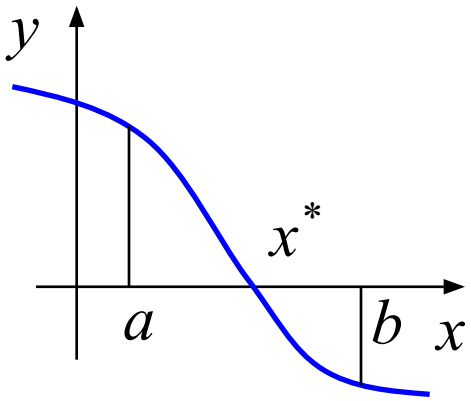
уравнение сложно (или невозможно) решить аналитически (получить формулу для x^*)

Методы решения уравнений $f(x) = 0$

- **Точные (аналитические)**

$$\sin x = 0 \quad \Rightarrow \quad x = \pi k,\ k \in Z$$

- **Приближенные**
  - **графические**



  - **численные**

(методы последовательного приближения):

1) по графику найти интервал $[a, b]$, в котором находится x^* (или одно **начальное приближение** x_0)
2) по некоторому алгоритму уточнить решение, сужая интервал, в котором находится x^*
3) повторять шаг 2, пока не достигнута требуемая точность:

$$b - a < \varepsilon$$

Численные методы

Применение: используются тогда, когда точное (аналитическое) решение неизвестно или очень трудоемко.

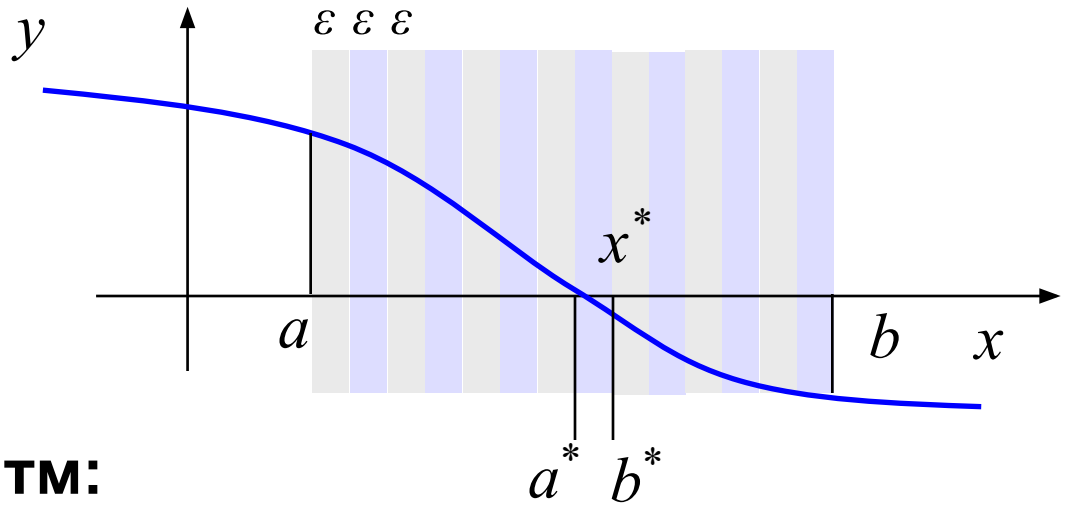
- дают хотя бы какое-то **решение**
- во многих случаях можно оценить ошибку и найти решение **с заданной точностью**

- решение всегда приближенное, **неточное**

$$\sqrt{x+1} - 4\sin(x-1) = 0 \quad \Rightarrow \quad x = 1,3974 \qquad x \approx 1,3974$$

Метод прямого («тупого») перебора

Задача: найти решение уравнения $f(x) = 0$ на интервале $[a, b]$ с заданной точностью ε (чтобы найденное решение отличалось от истинного не более, чем на ε).



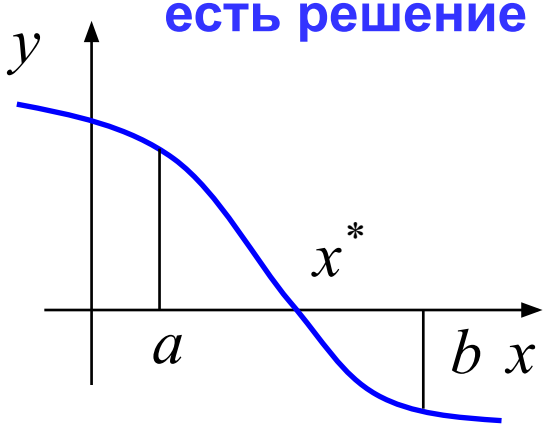
Алгоритм:

- разбить интервал $[a, b]$ на полосы шириной ε
- найти полосу $[a^*, b^*]$, в которой находится x^*
- решение – a^* или b^*

Как улучшить решение?

Есть ли решение на $[a, b]$?

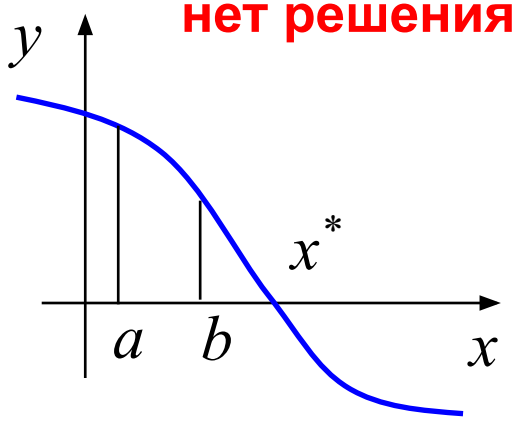
есть решение



$$f(a) > 0$$
$$f(b) < 0$$

$$f(a) f(b) < 0$$

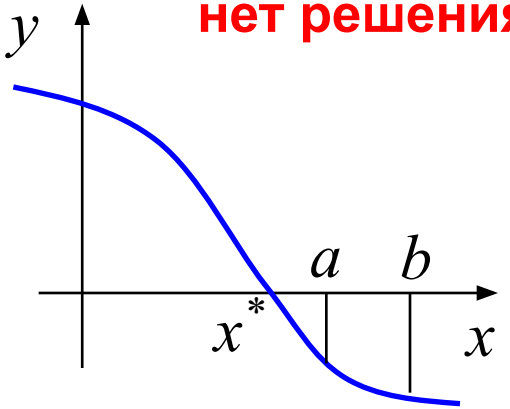
нет решения



$$f(a) > 0$$
$$f(b) > 0$$

$$f(a) f(b) > 0$$

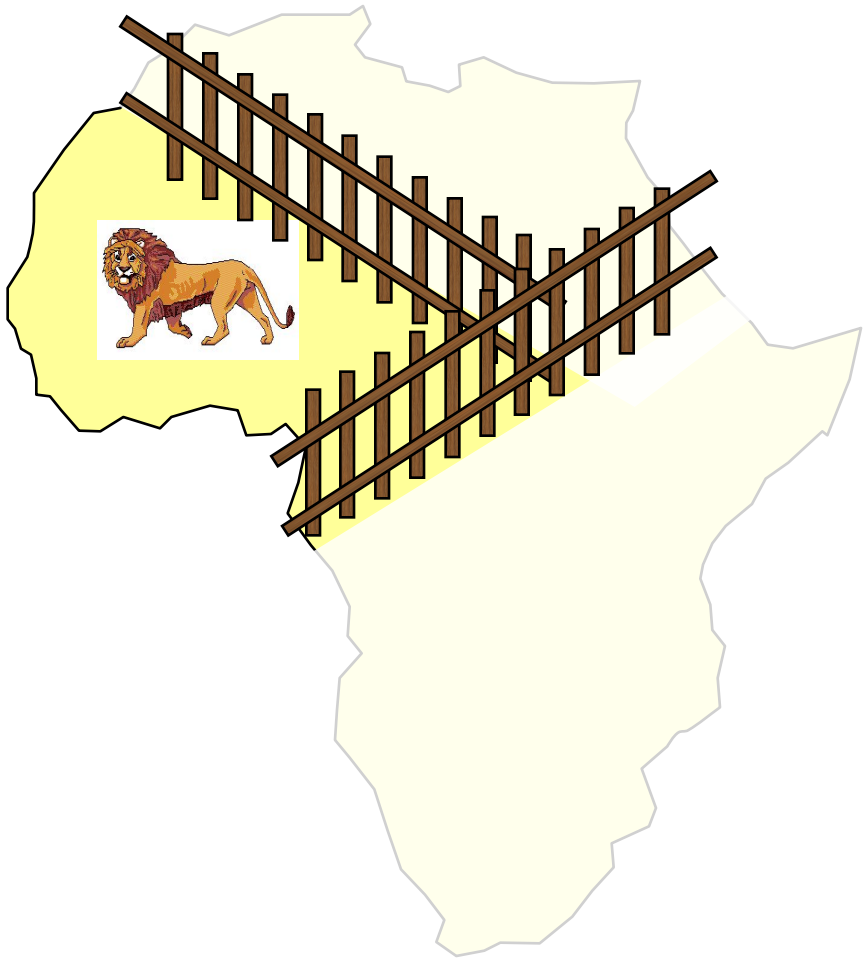
нет решения

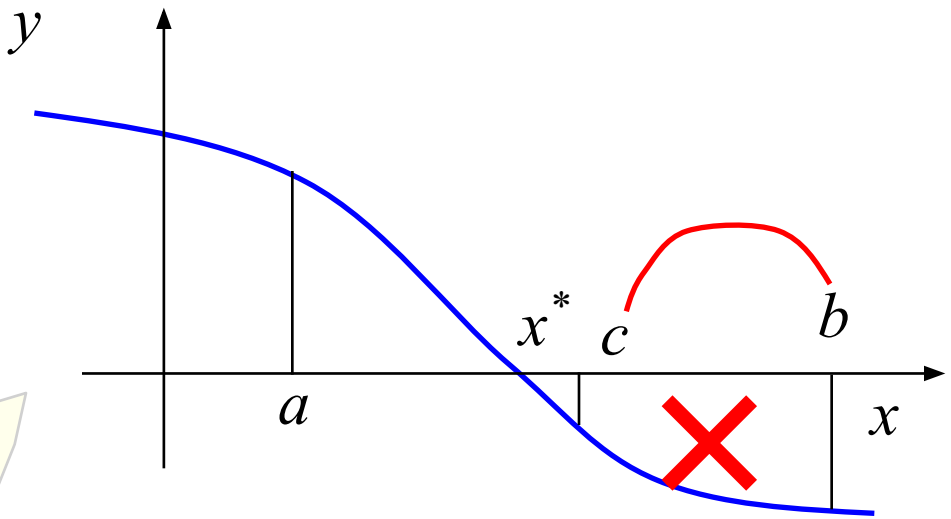


$$f(a) < 0$$
$$f(b) < 0$$

Если *непрерывная* функция $f(x)$ имеет разные знаки на концах интервала $[a, b]$, то в некоторой точке x^* внутри $[a, b]$ она равна 0, то есть $f(x^*) = 0$!

Метод дихотомии (деление пополам)





1. Найти середину отрезка `[a,b]`:
   ```
   c = (a + b) / 2;
   ```
2. Если `f(c)*f(a)<0`, сдвинуть правую границу интервала
   ```
   b = c;
   ```
3. Если `f(c)*f(a)≥0`, сдвинуть левую границу интервала
   ```
   a = c;
   ```
4. Повторять шаги 1-3, пока не будет `b – a ≤ ε`.

Метод дихотомии (деления пополам)

- **простота**
- можно получить решение с **любой** заданной **точностью**

- нужно знать **интервал** $[a, b]$
- на интервале $[a, b]$ должно быть только **одно** решение
- **большое число шагов** для достижения высокой точности
- только для функций **одной** переменной

Метод дихотомии (в программе)

```
//-----------------------------------------
// Solve находит точку пересечения на [a,b]
// Вход:  a, b - границы интервала,  a < b
//        eps  - точность решения
// Выход: x - решение уравнения f1(x)=f2(x)
//-----------------------------------------
float Solve ( float a, float b, float eps )
{
  float c, fa, fc;
  while ( b - a > eps )
    {
    c = (a + b) / 2;
    fa = f1(a) - f2(a);
    fc = f1(c) - f2(c);
    if ( fa*fc < 0 ) b = c;
    else             a = c;
    }
  return (a + b) / 2;
}
```

$$f(a) = f_1(a) - f_2(a)$$
$$f(c) = f_1(c) - f_2(c)$$

Метод дихотомии (в программе)

```
float xc1, xc2;
...
float Solve ( float a, float b, float eps )
{
  ...
}
...
void Cross ()
{
  char s[20];
  xc1 = Solve ( 1, 2, 0.0001 );
  sprintf(s, "x=%.2f", xc1);
  outtextxy ( 150, 100, s );
  sprintf(s, "y=%.2f", f1(xc1));
  outtextxy ( 150, 120, s );
  ...
}
```

глобальные переменные: абсциссы точек пересечения

найти решение на интервале [1,2] с точностью 0,0001

вывод на экран через символьную строку

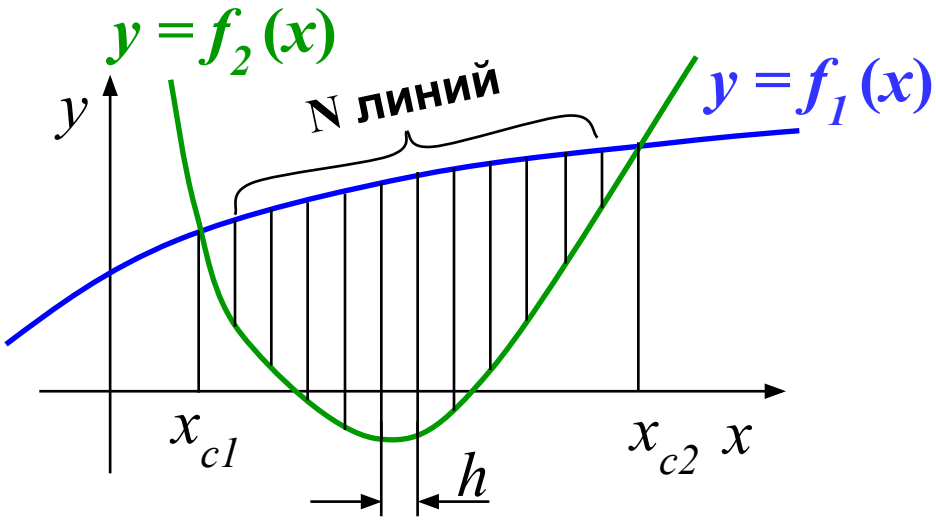
… и значение y!

то же самое для остальных точек

Структурное программирование на языке Си

Тема 5. Штриховка

Штриховка (две функции)



$$h = \frac{x_{c2} - x_{c1}}{N + 1}$$

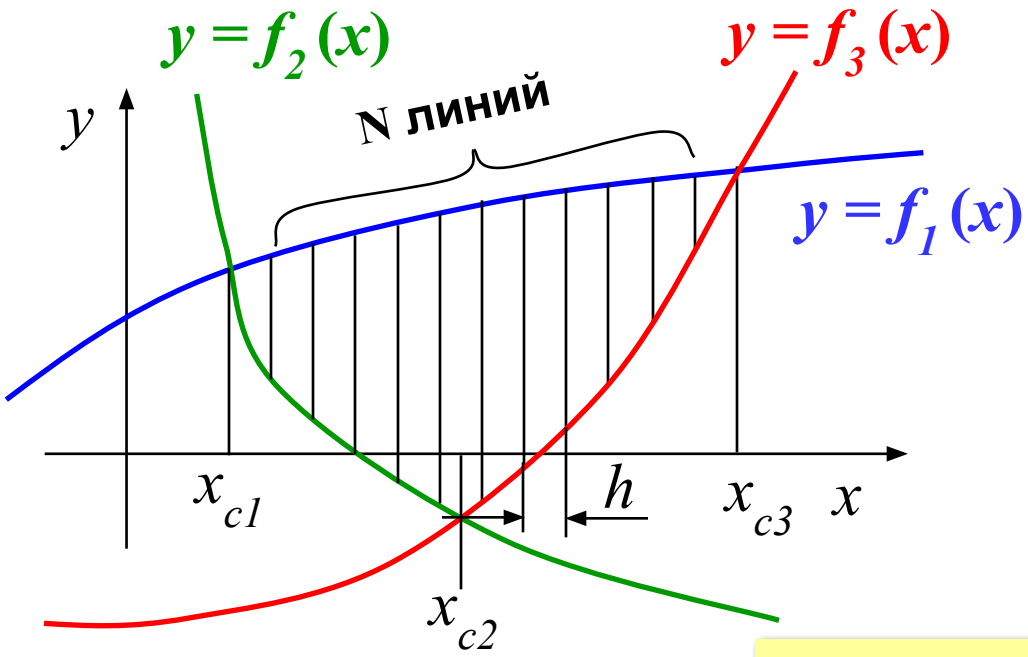
```
void Hatch()
{
  const int N = 10;
  float x, h = (xc2 - xc1) / (N + 1);
  int xe, yUp, yDown;
  for (x = xc1+h; x < xc2; x += h )
    {
    xe = ScreenX ( x );
    yUp = ScreenY ( f1(x) );
    yDown = ScreenY ( f2(x) );
    line ( xe, yUp, xe, yDown );
    }
}
```

шаг по x

экранная координата x

экранные координаты границ области по оси y

Штриховка (составная нижняя граница)



$$h = \frac{x_{c3} - x_{c1}}{N+1}$$

```
void Hatch()
{
...
h = ( xc3 - xc1) / (N + 1);
...
yDown = ScreenY ( Down(x) );
...
}
```

```
//----------------------------
// Down нижняя граница области
//----------------------------
float Down ( float x )
{
if ( x < xc2 )
      return f2(x);
else return f3(x);
}
```

Штриховка (общий случай)

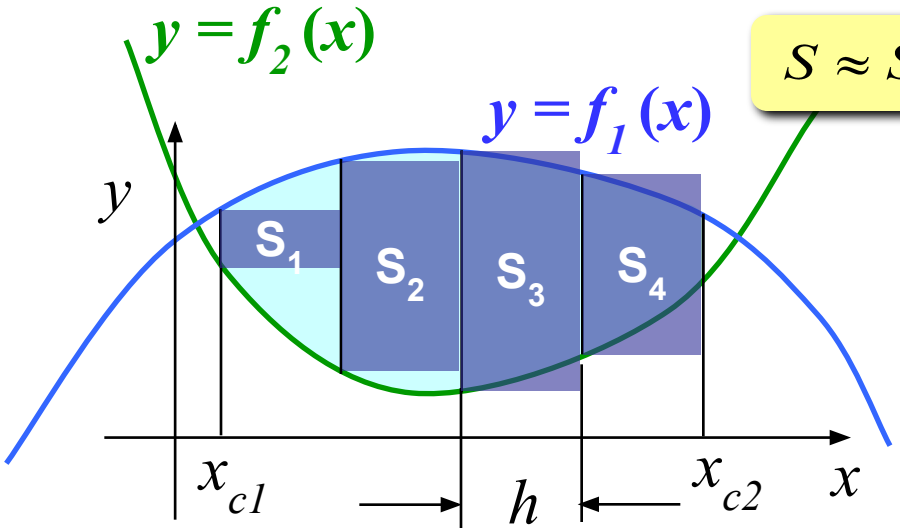
```
float Up ( float x )   { ... }
float Down ( float x ) { ... }
...
void Hatch()
{
  const N = 10;
  float x, h = ( xc3 - xc1 ) / (N + 1);
  int xe, yUp, yDown;
  for ( x = xc1+h; x < xc3; x += h )
    {
    xe = ScreenX ( x );
    yUp = ScreenY ( Up(x) );
    yDown = ScreenY ( Down(x) );
    line ( xe, yUp, xe, yDown );
    }
}
```

у всех по-разному…

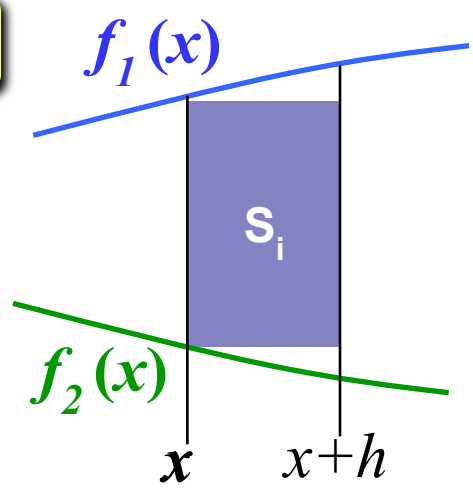
Структурное программирование на языке Си

Тема 6. Вычисление площади

Метод (левых) прямоугольников



$$S \approx S_1 + S_2 + S_3 + S_4$$



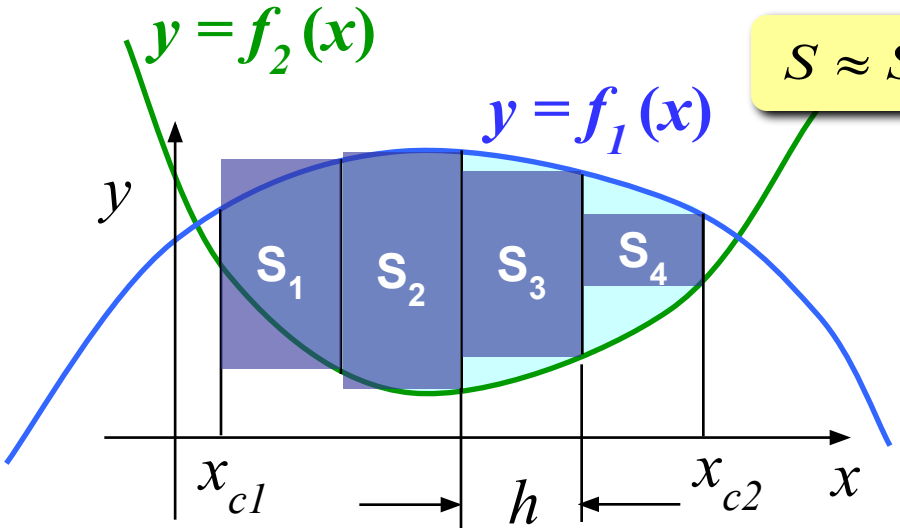
$$S_i = (f_1(x) - f_2(x)) \cdot h$$

```
void Area()
{
  float x, S = 0, h=0.001;
  char out[20];
  for ( x = xc1; x < xc2; x += h )
    S += f1(x) - f2(x);
  S *= h;
  sprintf ( out, "S=%7.3f", S );
  outtextxy ( 300, 300, out );
}
```

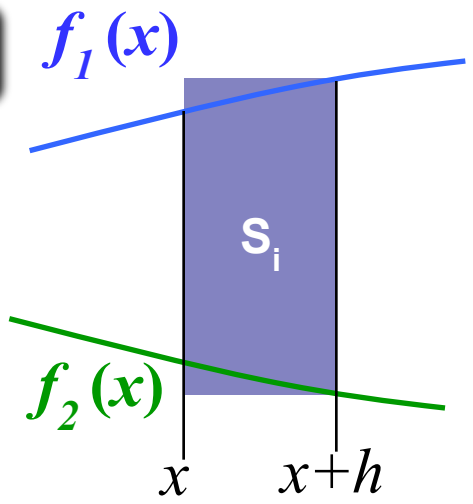
Почему не x <= xc2?

Как улучшить решение?

Метод (правых) прямоугольников



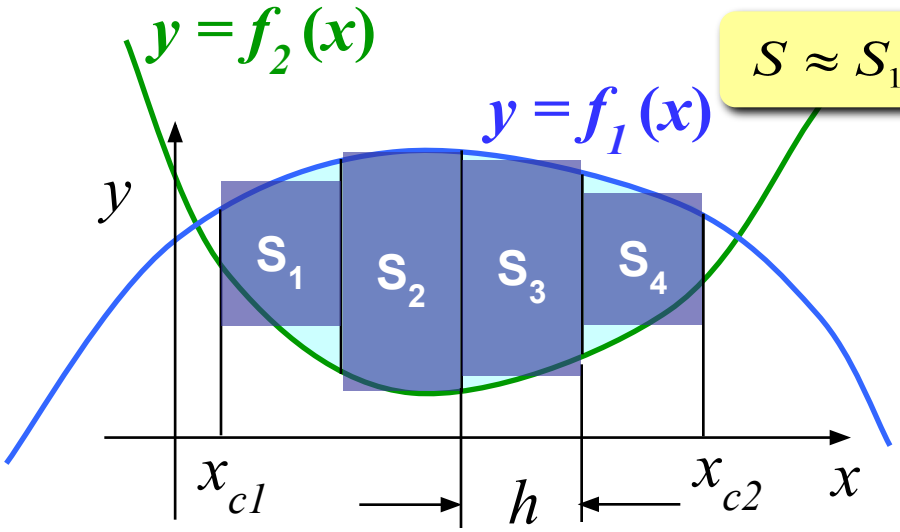
$$S \approx S_1 + S_2 + S_3 + S_4$$



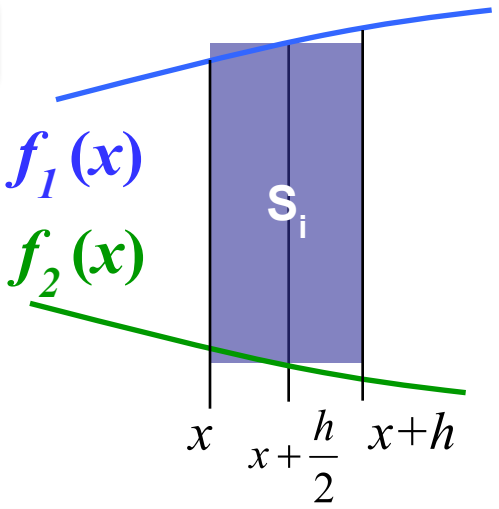

```
void Area()
{
  float x, S = 0, h=0.001;
  char out[20];
  for ( x = xc1; x < xc2; x += h )
    S += f1(x+h) - f2(x+h);
  S *= h;
  sprintf ( out, "S=%7.3f", S );
  outtextxy ( 300, 300, out );
}
```

$$S_i = (f_1(x+h) - f_2(x+h)) \cdot h$$

Метод (средних) прямоугольников



$$S \approx S_1 + S_2 + S_3 + S_4$$




```
void Area()
{
  float x, S = 0, h=0.001;
  char out[20];
  for ( x = xc1; x < xc2; x += h )
    S += f1(x+h/2) - f2(x+h/2);
  S *= h;
  sprintf ( out, "S=%7.3f", S );
  outtextxy ( 300, 300, out );
}
```

$$S_i = \left[f_1(x + \frac{h}{2}) - f_2(x + \frac{h}{2}) \right] \cdot h$$

Какой метод точнее?

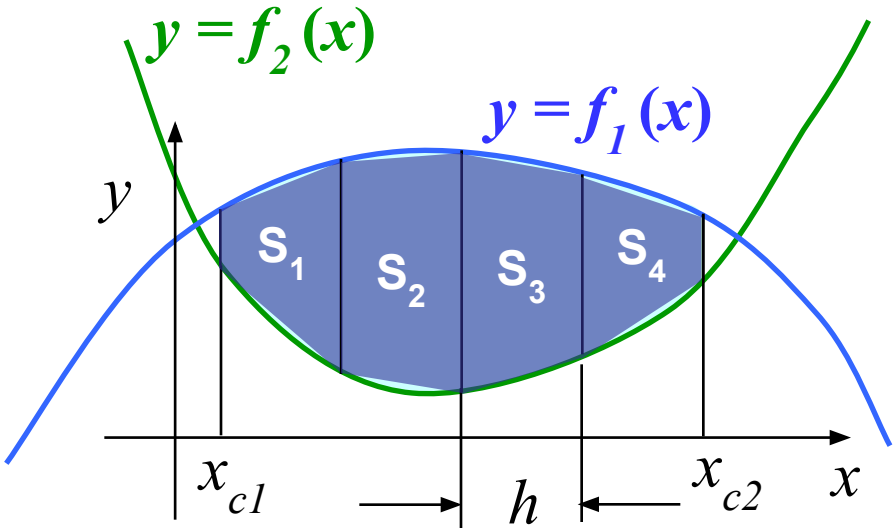
левые (правые):

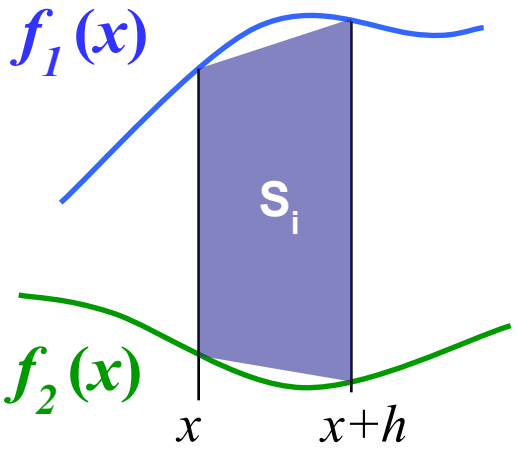
$$\varepsilon = O(h)$$

средние

$$\varepsilon = O(h^2)$$

Метод трапеций





$$S_i = \frac{f_1(x) - f_2(x) + f_1(x+h) - f_2(x+h)}{2} \cdot h$$

```
for ( x = xc1; x < xc2; x += h )
S
```

```
S = ( f1(xc1) - f2(xc1)
    + f1(xc2) - f2(xc2) )/2.;
for ( x = xc1+h; x < xc2; x += h )
    S += f1(x) - f2(x);
S *= h;
```

Как улучшить?

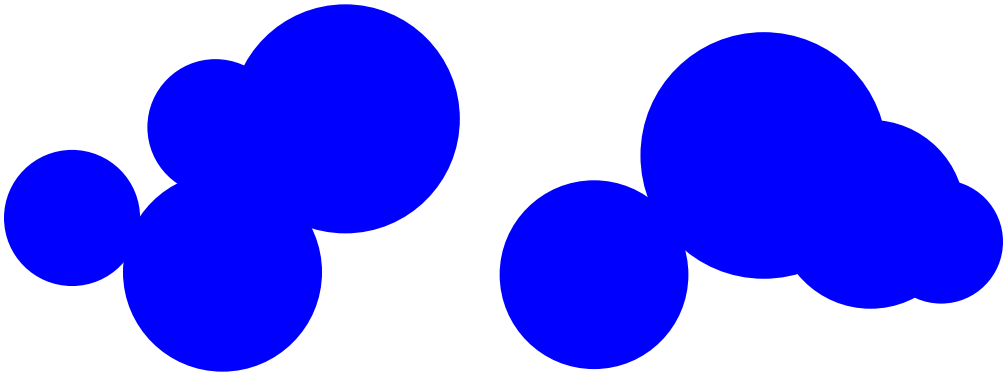
Ошибка $\varepsilon = O(h^2)$

Метод Монте-Карло

Применение: вычисление площадей сложных фигур (трудно применить другие методы).

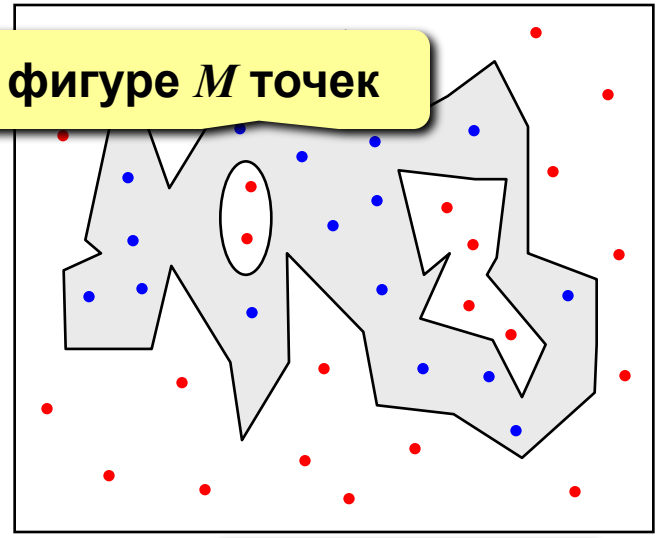
Требования: необходимо уметь достаточно просто определять, попала ли точка (x, y) внутрь фигуры.

Пример: заданы 100 кругов (координаты центра, радиусы), которые могу пересекаться. Найти площадь области, перекрытой кругами.



Метод Монте-Карло

1. Вписываем сложную фигуру **в другую фигуру**, для которой легко вычислить площадь (прямоугольник, круг, …).
2. **Равномерно** N точек со случайными координатами внутри прямоугольника.
3. Подсчитываем количество точек, **попавших на фигуру**: M.
4. Вычисляем **площадь**: $\frac{S}{S_0} \approx \frac{M}{N} \Rightarrow S \approx S_0 \cdot \frac{M}{N}$



1. **Метод приближенный.**
2. **Распределение должно быть равномерным.**
3. **Чем больше точек, тем точнее.**
4. **Точность ограничена датчиком случайных чисел.**

Случайное число в заданном интервале

`rand()`	целое `[0, RAND_MAX]`
`rand()/RAND_MAX`	всегда 0!!!
`1.*rand()/RAND_MAX`	`[0,1]`
`1.*rand()/RAND_MAX + a`	`[a,a+1]`
`(b-a)*rand()/RAND_MAX + a`	`[a,b]`

```
//---------------------------------------
// randF - случайное вещественное число
//         в заданном интервале
//---------------------------------------
float randF ( float a, float b)
{
return (b-a)*rand() / RAND_MAX + a;
}
```

Проверка точки (внутри или нет?)

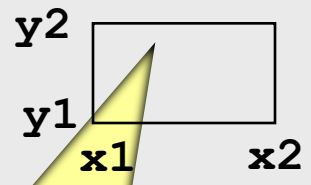
```
//----------------------------------------
// Inside – определяет, находится ли точка
//          внутри фигуры
// Вход:  x, y – координаты точки
// Выход: 1, если точка внутри фигуры,
//        0, если точка вне фигуры
//----------------------------------------
int Inside ( float x, float y )
{
  if ( Down(x) <= y  &&  y <= Up(x) )
       return 1;
  else return 0;
}
```

```
int Inside ( float x, float y )
{
  return (Down(x) <= y  &&  y <= Up(x));
}
```

Метод Монте-Карло (реализация)

```
//-------------------------------------------------
// Area2 - вычисление площади методом Монте-Карло
//-------------------------------------------------
void Area2 ()
{
  int i, N = 200000, M = 0;
  float x1 = xc1, x2 = xc2, y1 = 1, y2 = 4;
  float x, y, S;
  char out[20];
  for (i=1; i<=N; i++)
    {
    x = randF ( x1, x2 );
    y = randF ( y1, y2 );
    if ( Inside(x,y) ) M++;
    }
  S =
    (x2-x1)*(y2-y1)*M/N;
  sprintf(out, "S=%.3f", S);
  outtextxy(300, 320, out);
}
```

границы прямоугольника (у каждого свои!)



если на фигуре, увеличить счетчик

вычисление площади

Структурное программирование на языке Си

Тема 7. Оформление отчета

Титульный лист

Министерство образования и науки РФ
Государственное образовательное учреждение СОШ № 163

Отчет о выполнении проекта

"Графики функций"

Вариант 6

Выполнил: ученик 9Б класса
Пупкин Василий

Проверил:
Поляков К. Ю.

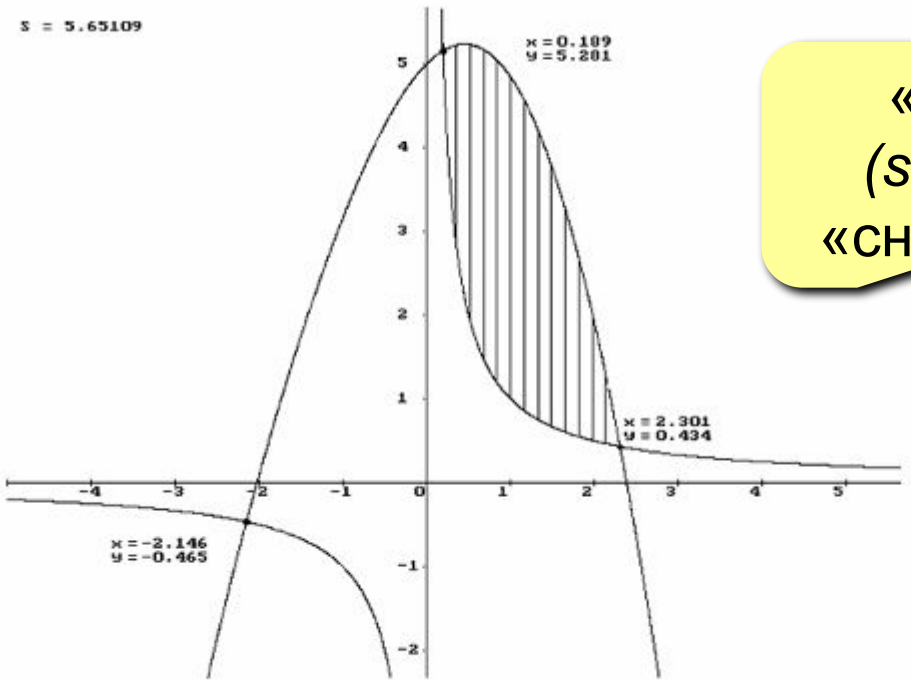
Санкт-Петербург
2007

Графики функций

I. Графики функций

$$y = \frac{1}{x}$$

$$y = -x^2 + \sin(x) + 5$$



через Редактор формул (**Вставка – Объект – Microsoft Equation**)

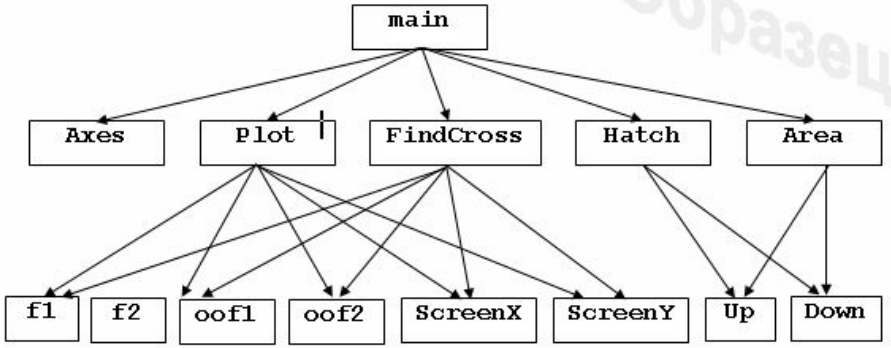
«скриншот» *(screenshot)* – «снимок» экрана

Как получить копию экрана?

1. Поменять цвета так, чтобы все линии и текст были белые.
2. Запустить программу (она должна все нарисовать).
3. Нажать клавишу **PrtScr** (*Print Screen* – «снимок» экрана) на клавиатуре или комбинацию **Alt+PrtScr** («снимок» активного окна).
4. В графическом редакторе (*Paint*): **Правка – Вставить**.
5. Перевести в черно-белую палитру (**Рисунок – Атрибуты – Палитра – Черно-белая**).
6. Инверсия (черный ↔ белый), **Рисунок – Обратить цвета**.
7. Выделить нужную часть рисунка. 
8. Вставить в отчет через буфер обмена (**Ctrl+C, Ctrl+V**).

Структура программы

II. Структура программы



main	основная программа
Axes	построение и разметка осей координат
Plot	построение графиков
FindCross	определение точек пересечения
Hatch	штриховка
Area	вычисление площади
f1	функция $y = \frac{1}{x}$
f2	функция $y = -x^2 + \sin x + 5$
oof1	область определения функции $y = \frac{1}{x}$
oof2	область определения функции $y = -x^2 + \sin x + 5$
ScreenX	пересчет x-координаты точки из математической системы в экранную
ScreenY	пересчет y-координаты точки из математической системы в экранную
Up	верхняя граница заштрихованной области
Down	нижняя граница заштрихованной области

Текст программы

III. Текст программы


```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>

const a = 300, b = 340, k = 50;
float xmin, xmax, ymin, ymax;
float xc[20];
int   xcN = 0;

typedef float fFuncF ( float );
typedef int   iFuncF ( float );
fFuncF* func[2];
iFuncF* ooFunc[2];

//-----------------------------------------------
// F1, F2 - исходные функции
// Вход:  x в математических координатах
// Выход: y в математических координатах
//-----------------------------------------------
float f1(float x) {  return 1. / x; }
float f2(float x) {  return - x*x + sin(x) + 5; }

//-----------------------------------------------
// OOF1, OOF2 - область допустимых значений аргумента
// Вход:  x в математических координатах
// Выход: 1, если x допустим
//        0, если недопустим
//-----------------------------------------------
int   oof1(float x) {  return fabs(x) > 0.0001; }
int   oof2(float x) {  return 1; }

//-----------------------------------------------
// UP - верхняя граница области
// Вход:  x в математических координатах
// Выход: y в математических координатах
//-----------------------------------------------
float Up(float x)
{
    return f2(x);
}

//-----------------------------------------------
// DOWN - нижняя граница области
// Вход:  x в математических координатах
// Выход: y в математических координатах
//-----------------------------------------------
float Down(float x)
{
    return f1(x);
}

//-----------------------------------------------
// SCREENX, SCREENY - ïåðåñ÷åò èç математических
// координат в экранные
// Вход:  x или y в математических координатах
// Выход: x или y в экранных координатах
//-----------------------------------------------
int ScreenX (float x) { return  round(a+k*x); }
int ScreenY (float y) { return  round(b-k*y); }

//-----------------------------------------------
// AXES - ïîñòðîåíèå îñåé êîîðäèíàò
//-----------------------------------------------
void Axes()
{
  int d, sx, sy;
  char s[20];
```

шрифт **Courier New**, (моноширинный) размер 10 пт

Конец фильма