Лицей при СПбГУТ

А.В. Красов

Программирование на языке Си.

Часть вторая.

Использование графики и Программная обработка данных

Санкт – Петербург

1998 г.

А.В. Красов

Программирование на языке Си. Вторая часть, Использование графики и программная обработка данных. СПб.: Лицей СПбГУТ 1998, -41 с.

Предлагаемое Вашему вниманию пособие посвящено обучению программированию на языке Си для IBM PC и совместимых с ними компьютеров.

Данный курс читается автором пособия с 1989 года, для подшефных школьников и студентов СПбГУТ а также СПбГЭТУ.

Отличительным характерным моментом методики рассмотрения материала является ориентация на наглядное графическое представления материала. Тем самым акцентируется внимание в первую очередь на алгоритм, суть задачи, и лишь затем реализация. Ориентация на выбранный стиль разработки программ, сразу приучает к определенным подходам к программированию и оформления текста своих программ, это принесет несомненные преимущества при разработки сложных приложений.

Большое количество примеров и наглядных задач делает курс интересным и занимательным. Приведенный план занятий позволяет оценивать темпы изучения материала и вносит элементы соревнования.

Рассмотренные в этой части темы: вывод графиков, сжатие и кодирование информации, численные методы, позволяют более плавно адаптироваться к задачам решаемым на специализированных кафедрах.

Лучшие работы студентов и школьников в рамках данной учебной программы а также информацию о новых методических разработках можно найти на

http://www.lank.ru

Пособие предназначено для учащихся лицея, базовых школ университета, студентов младших курсов СПбГУТ, и всех тех кто интересуется программированием.

Замечания, предложения о контактах можно направлять по адресу

KRASOV@mail.wplus.net

Пособие предназначено для учащихся лицея, базовых школ, студентов младших курсов СПбГУТ, и всех тех кто интересуется программированием.

Изложены приемы использования графических функций для решения прикладных задач. Рассматриваются основные алгоритмы программной обработки данных и даются элементы вычислительных методов.

Изложение материалов иллюстрируется графическими блок-схемами.

Рецензент:

Часть 2
Графика Си модуль graphics.h

Монитор ПК может работать в двух режимах текстовый и графический. В этих режимах по разному представляется видео память. Переход из режима в режим очищает экран.

Все выше перечисленные функции ввода вывода работали с текстовым режимом. В графики они не доступны. В графическом режиме необходимо пользоваться функциями из графической библиотеки.

Типы видео мониторов и их режимы

Существует много типов мониторов, на каждом из которых доступны кроме своего режима, и все более низкие режимы. Под режимом понимается разрешающая способность количество цветов.

Кроме того мониторы делятся по аппаратной реализации: ЦИФРОВЫЕ и АНАЛОГОВЫЕ.

Тип монитора	Режим	Разрешающая способность	Кол-во цветов
CGA	CGAC0-2	320 X 200	4
	CGAHI	640 X 200	2
HERCMONO		720 X 347	2
EGA	EGALO	640 X 200	16
VGA	VGAHI	640 X 480	16

Остальные стандартные типы являются мало употребительными или повторяют более слабые режимы.

Режимы SVGA не являются стандартными, хотя драйвера для них иногда и встречаются, но не входят в комплект Borland Си. Положение ухудшает и большое разнообразие особенностей SVGA карт выпускаемыми разными фирмами, не придерживающихся одинакового формата.

Инициализация графики

Функции:
```
initgraph(int *GrDr,int *GrMod,char *Path) ;
i=graphresult() ;
closegraph() ;
```
Функция initgraph(...) инициализирует графический режим. В параметрах ей передается:

GrDr - Тип графического монитора, или DETECT - Определить максимально возможный. Тип установленного оборудования возвращается в этих же переменных (поэтому они и передаются указателем).

GrMod - Режим.

Path - Путь до файлов *.bgi - драйверов графических режимов. Если указанно " " - то в текущем каталоге.

Значения и имена можно взять из предыдущей таблицы.

Функция graphresult() - возвращает код ошибки инициализации графики (недопустимый графический режим, не найден файл *.bgi и т.п.). Если все в порядке функция возвращает значение grOk. Проверку правильного выполнения функции initgraph(...) обязательно надо производить, так как, если графический режим не установлен, выполнение любой графической команды приведет к аварийному останову программы.

Функция closegraph() обеспечивает корректное возвращение в текстовый режим. Ее выполнение, как говорилось раньше, очищает экран. При необходимости вставляйте задержку до нажатия клавиши: getch().

Внимание !

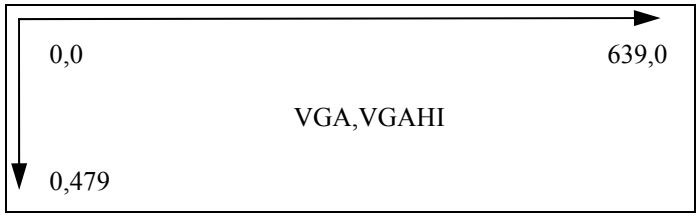
Работа с графикой возможна только в моделях памяти >= medium, это устанавливается в опциях компилятора.

Для работы с библиотекой графики ее необходимо подключить. В опциях Си: Option\Linker\Libriry\Graphics: X

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
int main(void)
{
        int GrDr,GrMod,rez ;
        GrDr=DETECT ;
        initgraph(&GrDr,&GrMod," ") ;
        rez=graphresult() ;
        if(rez != grOk)
        {
                printf("\n Ошибка инициализации графики") ; return(0) ;
        } /* Кон. if */
        line(0,0,100,100) ;
        getch() ;
        closegraph() ;
        return(1) ;
} /* Кон. main() */
```

Система координат

Система координат в графическом режиме начинается с точки 0,0. Размеры экрана зависят от установленного графического режима.



Основные графические функции

Ниже используются обозначения:

x,y,x1,y1,x2,y2	– координаты;
*St	– строка символов;
rx,ry,r	– радиусы;
h	– толщина изображения;
Alfa1,Alfa2	– углы в градусах;
DX,DY	– размеры символа;
Color	– цвета.

Функции рисования

line(int x1,int y1,int x2,int y2)	Линия
rectangle(int x1,int y1,int x2,int y2)	Прямоугольник
bar(int x1,int y1,int x2,int y2)	Закрашенный прямоугольник
bar3d(int x1,int y1,int x2,int y2,int h)	Закр. прям. с оттененением
ellipse(int x,int y,int rx,int ry)	Эллипс
fillellipse(int x,int y,int rx,int ry)	Закрашенный эллипс
arc(int x,int y,int Alfa1,int Alf2,int r)	Дуга (круг)
outtextxy(int x,int y,char *St)	Вывод строки текста
putpixel(int x,int y,char Color)	Поставить точку
Color=getpixel(int x,int y)	Получить цвет точки
floodfill(int x,int y,char Color)	Залить до границы указанного цвета
cleardevice()	Очистить экран
clearviewport()	Очистить порт вывода
setviewport(int x1,int int y1, int x2,int y2,char flg)	Установить порт вывода flg-вывод за пред. окна

Функции изменения параметров рисования

setcolor(char Color) Установить цвет рисования
setbkcolor(char Color) Установить цвет фона
Внимание! эта команда меняет все цвета экрана
setfillstyle(<Шаблон>,cahr Color) Установить цвет и стиль закраски фигур
setlinestyle(<Шаблон>,int Bit,char h) Установить стиль и толщину линий.

Шаблоны линий

Шаблон	Ном	Изображение
SOLID_LINE	0	Сплошная линия
DOTTED_LINE	1	Линия из точек
CENTER_LINE	2	Centered line
DASHED_LINE	3	Dashed line

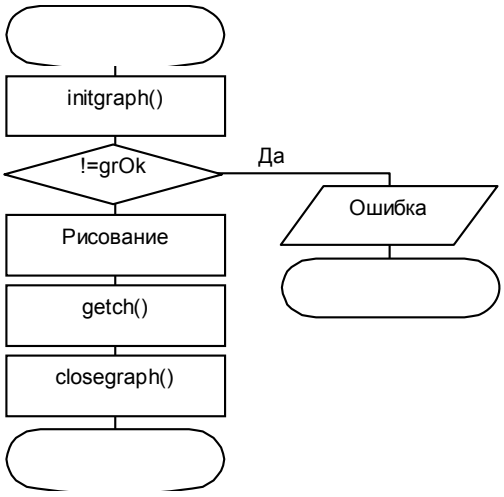
Шаблоны закраски

Шаблон	Ном.	Изображение
EMPTY_FILL	0	Цветом фона
SOLID_FILL	1	Выбранным цветом
|LINE_FILL	2	---
LTSLASH_FILL	3	///
SLASH_FILL	4	///
BKSLASH_FILL	5	\\\
LTBKSLASH_FILL	6	\\\
HATCH_FILL	7	Light hatch
XHATCH_FILL	8	Heavy crosshatch
INTERLEAVE_FILL	9	Interleaving line
WIDE_DOT_FILL	10	Точками
CLOSE_DOT_FILL	11	Частыми точками

Функции получения информации

MaxX=getmaxx() Получить макс. значение по оси X
MaxY=getmaxy() Получить макс. значение по оси Y
DX=textwitdh("C") Получить размер символа (строки) по оси X
DY=textheight("T") Получить размер символа (строки) по оси Y

Блок схема графической программы



Рисование графиков

Постановка задачи

Заданна функция float F(float x), необходимо на выбранном пользователем интервале построить ее график. Программа должна автоматически определять масштаб, содержать разметку, линейку для сканирования точек графика. Значения сканируемых точек должны писаться под графиком.

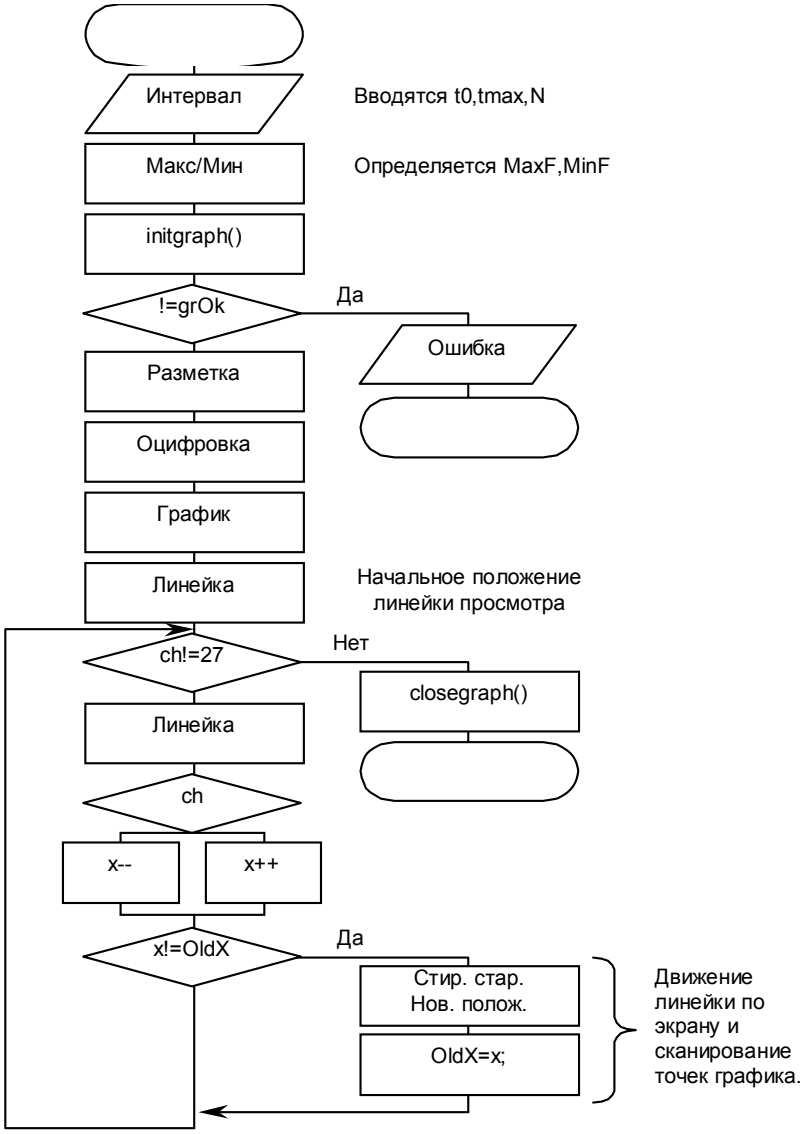
Этапы реализации

Программа реализуется в 4 этапа: Рисование разметки; Нахождение минимума и максимума функции; Оцифровка; Вывод графика; и Сканирование значений точек.

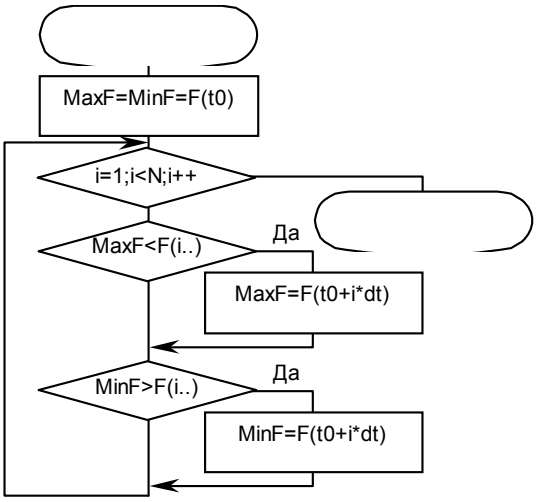
Рисование разметки

Для рисования разметки рекомендуется воспользоваться стилем линий DOTTED_LINE. После завершения рисования, не забудьте вернуться к стилю SOLID_LINE. Рисовать удобнее функцией line(...) выводимой в цикле.

Программа вывода графиков



Нахождение минимума и максимума функции



Необходимо построить график на интервале
t0..tmax. Кол-во точек N. Шаг между точками
dt=(tmax-t0)/N. float t0,tmax,dt; int N;
В начале считаем что макс.=мин.= значение в
первой точке.

Просматриваем все остальные точки графика.

Каждая точка сравнивается с макс. значение, если больше то макс. берется эта точка.

t0+i*dt - значение по горизонтали i точки.
F(t0+i*dt) - значение функции в i точки.

Оцифровка

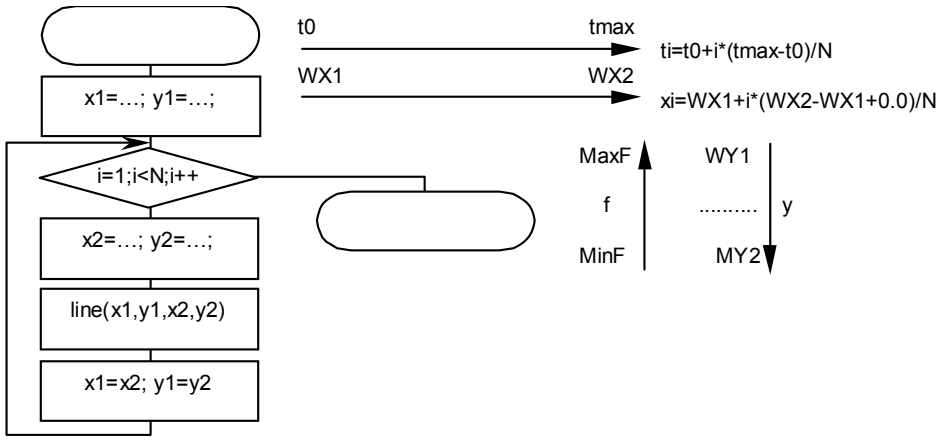
Для вывода цифр около линий разметки необходимо использовать функцию outtextxy(...), но предварительно число необходимо записать в строку, т.к. outtextxy(...) работает только со строками. Для этого удобнее всего пользоваться функцией sprintf(<Строка>,<"Текст, метки форматов",[<Переменные>]>). Эта функция аналогична printf(...), но вывод происходит не на экран, а в строку.
Пример:

```
char st[80] ;
. . . . .
sprintf(st,"x=%f , F(x)=%f",x,F(x)) ;
 outtextxy(100,100,st) ;
```

При выводе на экран, необходимо помнить что координаты переданные в функцию outtextxy(...), относятся к левому верхнему углу строки. Чтобы строка была напротив (под) линей разметки, ее необходимо сдвинуть. Полезно учесть размер символа (функции textwidth(...) и textheight(..).

Вывод графика

Для вывода графика рекомендуется ввести целые переменные: x1,y1,x2,y2 - координаты точек, WX1,WY1,WX2,WY2 - координаты окна вывода. Алгоритм рисования графика следующий.



Сложность заключается только в пересчете вертикальных координат. Обычно они направленны снизу вверх, а у машины наоборот.

Сканирование значений точек

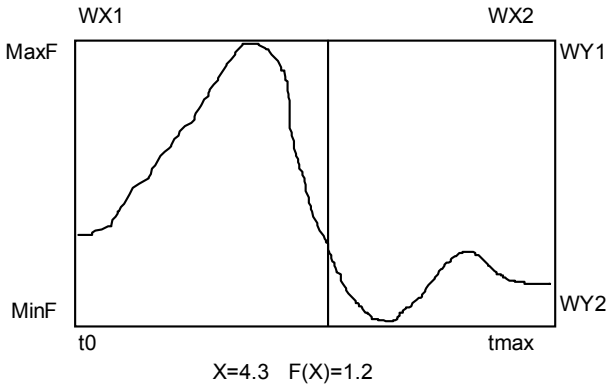
Под сканированием точек в задании понимается вертикальная линия перемещающаяся по графику, и выводимые под графиком значения аргумента и функции. Как организовать движение по клавишам и вывод числовых значений в графики уже рассматривалось выше.

Новое: Изображение линии, с последующим восстановлением изображеного под ней. Это рекомендуется реализовать используя режим инверсии экрана. В Си существует несколько способов рисования линий и прямоугольников. Первый режим COPY_PUT, указанные объекты рисуются цветом установленным в setcolor(...). Другие режимы позволяют рисовать комбинацией текущего цвета и цвета точек находящихся под рисунком. Удобнее всего использовать режим XOR_PUT. Повторно проведя линию того же цвета, возвращаемся к исходным цветам XOR(XOR(X))==X.

Установку режима осуществляет функция setwritemode(<Режим>). Установленный режим действует до установки нового. По умолчанию используется режим COPY_PUT.

Для того чтобы не забыть отключить режим инверсии, удобнее вынести все это в отдельную функцию, которая будет и стирать, и рисовать линю курсора.

```
void PutLine(int x,int y1,int y2)
{
        setwritemode(XOR_PUT) ;
        setcolor(RED) ; /* Инверсия др. цветом даст др. результ. */
                        /* Линия будет рисоваться не цветом RED */
        line(x,y1,x,y2) ; /* Линия вертикальная x1=x2=x */
        setwritemode(COPY_PUT) ;  /*  Иначе  и все остальное будет с инверсией цветов. */
} /* Кон. PutLine() */
```



Работа со спрайтами

Спрайт - фрагмент видео памяти (прямоугольное изображение). Библиотека Си содержит функции getimage(...) и putimage(...) позволяющие запоминать фрагменты экрана в памяти и выводить их в указанное место.

Функция getimage(int x1,int y1,int x2,int y2,void far *Buf)

x1,y1,x2,y2 - Размеры запоминаемой области,

Buf - Адрес выделенного буфера для запоминания изображения.

Функция putimage(int x,int y,void far *Buf,<Режим>)

x,y - Куда выводить изображение, левый верхний угол.

Buf - Адрес буфера где хранится изображение.

<Режим> - Режим вывода. COPY_PUT

В месте с этими функциями используются функции работы с памятью: выделения malloc(), farmalloc() (calloc(),farcalloc()), освобождения free(),farfree()

Обычно удобнее использовать память за пределами рабочего сегмента данных, по этому чаще используются farmalloc() и farfree(). Выделенную в программе память обязательно надо освобождать в конце работы программы. Иначе с каждым запуском программы у компьютера будет оставаться все меньше ресурсов, и в конце концов их не хватит для нормальной работы. Если попытка выделения памяти закончилась неудачно, то функции возвращают значение NULL. Если Вы хотите чтобы Ваша программа работала без сбоев, обязательно проверяйте успешность операции выделения памяти. За один вызов невозможно выделить более 64 кило байт.
Функции работы с памятью описаны в модуле alloc.h

Функция farmalloc(<Размер>)
Функция farfree(<Адрес буфера>)

Для правильного выделения памяти необходимо знать сколько ее нужно для выбранного фрагмента экрана. Это делает функция imagesize(...).
Функция imagesize(int x1,int y1,int x2,int y2) x1,y1,x2,y2 - Координаты выбранной области. Функция возвращает размер в байтах.

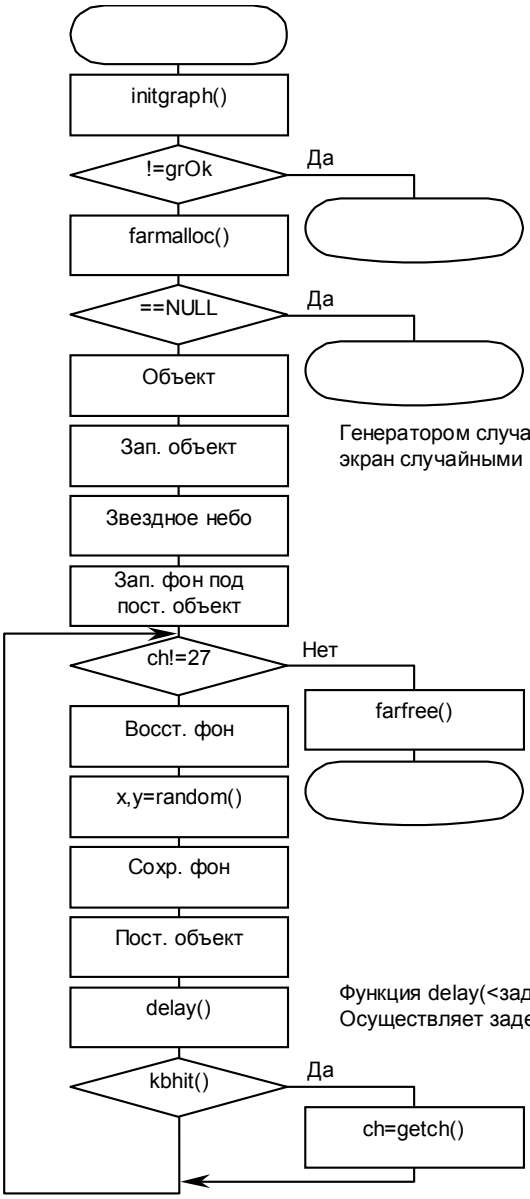
Внимание!!! Размеры запоминаемой картинки должны совпадать с размером выделенного буфера!

Пример вывода картинки

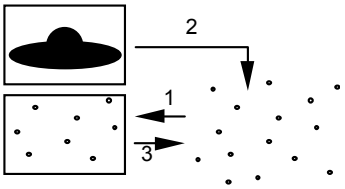
```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
int main(void)
{
        int GrDr, GrMod, Rez;
        long size ;     /* Переменная для хранения размера  запоминаемой
                           картинки  */
        void far *Buf;  /* Указатель на начало буфера изображения                  */
        GrDr=DETECT ;
        initgraph(&GrDr, &GrMod, " ");
        Rez = graphresult();
        if(Rez != grOk)
        {
                printf("\n Ошибка инициализации памяти"); return(0) ;
        } /* Кон. if */
        circle(20,20,5) ;               /* Рисование объекта который будет запомнен  */
```

```
        size=imagesize(10,10,30,30) ;  /* Опред. требуемого объема памяти      */
        Buf=farmalloc(size) ;              /* Выделение требуемой памяти           */
        if(Buf==NULL)
        {       /* Проверка, что выделение прошло успешно */
                printf("\n Ошибка выделения памяти.") ;
                getch() ;
                closegraph() ;
                return(0) ;
        } /* Кон. if  */
        getimage(10,10,30,30,Buf) ; /* Запоминание фрагмента экрана */
        getch() ;
        putimage(100,100,Buf,COPY_PUT) ;  /* Вывод в другое место */
        getch() ;
        farfree(Buf) ;  /* Освобождение памяти */
        closegraph();
        return(1);
} /* Кон. main() */
```

Программа “летающая тарелка”



Генератором случайных чисел заполнить экран случайными цветными точками.



Функция delay(<задержка [мс]>)
Осуществляет задержку на указанное время

Компьютерная анимация

Для организации анимации используются несколько фаз движения объекта. Каждый рисунок целесообразно пометить в свою функцию, в качестве параметров в нее передается точка с которой надо рисовать и направление рисунка по оси X. Все функции удобнее объединить в одну функцию рисования:

```
void Ris(int *faza,int x,int y,int napr)
{
        switch (*faza)
        {
                case 1:Ris1(x,y,napr) ; break ;
                case 2:Ris2(x,y,napr) ; break ;
                . . .
        } /* Кон. switch */
        (*faza)++
        if(*faza>MAX_FAZA) *faza=0 ;
} /* Кон. Ris() */
```

Каждая функция начинает с очистки места для рисунка. Все координаты в функциях рисования отсчитываются от точки x,y и умножаются на направление ; 1 (направо) и -1 (налево). Этим же параметром при необходимости можно менять и масштаб изображения.

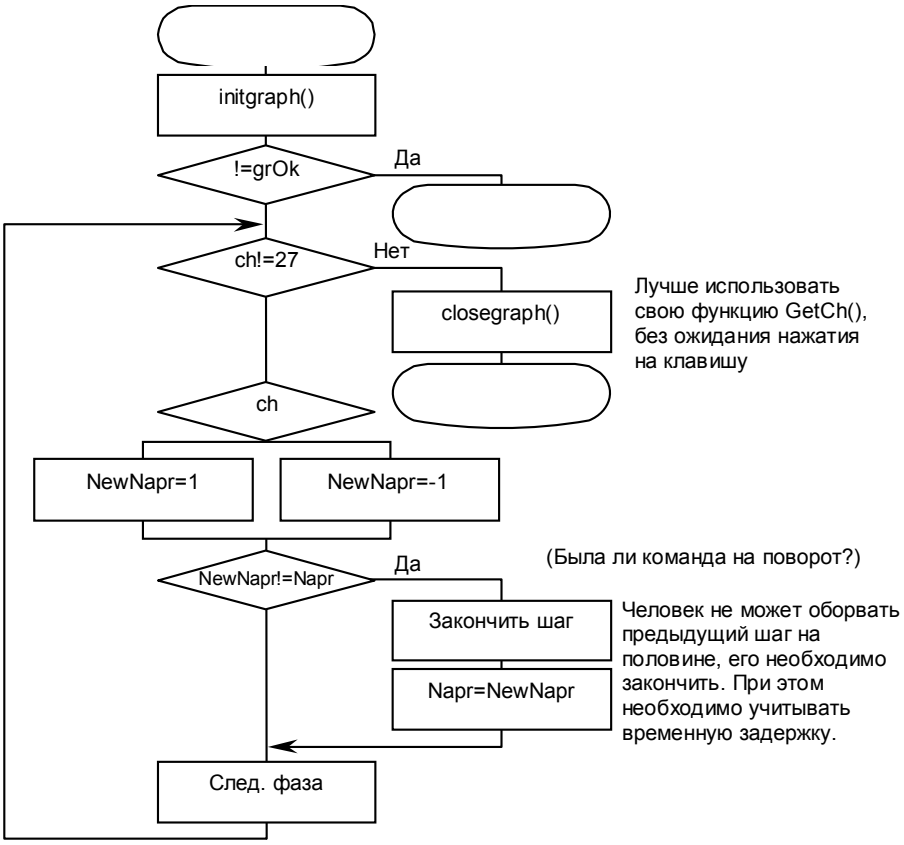
При организации изображения движения старайтесь делать минимум команд рисования. Например, в часах необходимо перемещать только стрелки и маятник, у движущегося автомобиля проще крутить колеса и двигать телеграфные столбы в доль дороги, чем каждый раз перерисовывать кузов машины.

Программа движения объекта (человек)

Даны N фаз движения объекта (человек или животное). Необходимо организовать его движение (по клавишам), повороты. Возможно движение с разной скоростью.

Изображение движущегося объекта - контурное.

Программа движения человечка



Программа движения винтовой лестницы

При написании программы рекомендуется использовать двух-страничный режим монитора. Он есть не на всех VGA дисплеях, поэтому если не пройдет следующий пример, используйте режим EGA.
В приведенной программе используются новые графические функции:
setactivepage(<Номер граф. стр.>) - На какую страницу выводить.
setvisualpage(<Номер граф. стр.>) - Какую страницу показывать. Нумерация графических страниц начинается с 0.

Демонстрационная программа количества видео страниц

```
#include<graphics.h>
int main(void)
{
        int GrDr,GrMod,Rez ;
        char ch=0 ;
        GrDr=EGA; /* В режиме VGA вторя страница используется не полностью
*/
        GrMod=EGAHI
        initgraph(&GrDr,&GrMod,"") ;
        Rez=grpahresult() ;
        if(Rez!=grOk)
        {
                printf("\n Ошибка инициализации графики.") ; return(0) ;
        } /* Кон. if */
        outtextxy(100,100,"Это первая видео страница. Нажмите \'2\' или \'Esc\' ") ;
        setactivepage(1) ;
        outtextxy(100,150,"Это втрая видео страница. Нажмите \'1\' или \'Esc\' ") ;
        while(ch!=27)
        {
                ch=getch() ;
                switch(ch)
                {
                        case '1':setvisualpage(0) ; break ;
                        case '2':setvisualpage(1) ; break ;
                } /* Кон. switch(ch) */
        } /* Кон. while */
 /*  Кон. main()  */
```

Количество видео страниц определяется объемом памяти на видео карте, и от разрешающей способности, количества цветов (сколько ОЗУ необходимо для представления изображения на экране).

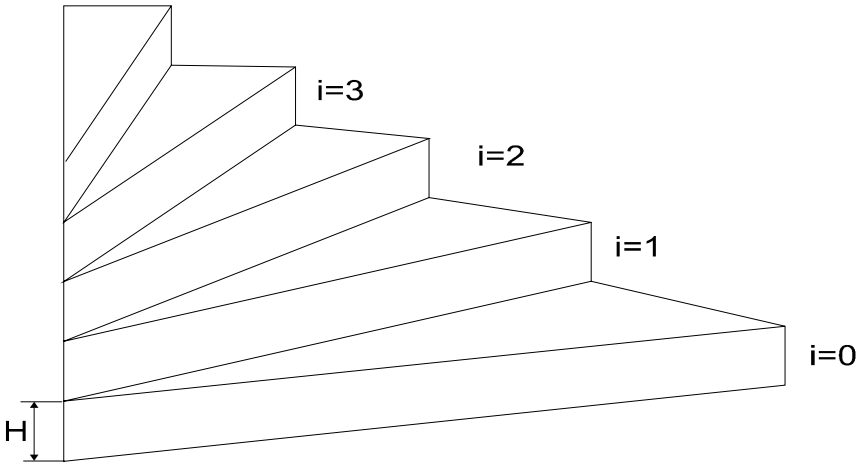
Драйвер	Режим	Кол-во граф. стр.	Разреш.	Кол-во цветов
VGA	VGAHI	1 - 2 ?	640X480	16
EHA	EGAHI	2 (2.5) 4	640X350	16
EGA	EHALO		640X200	16

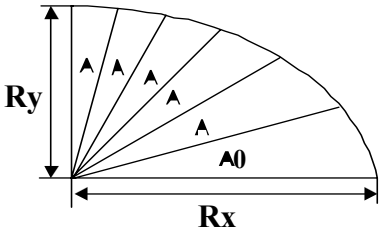
Отсутствие двух страниц в режимах VGA с высокой разрешающей способностью отчасти объясняется более высокой скоростью их работы. Двух страничный режим позволяет скрыть от пользователя этапы создания картинки,

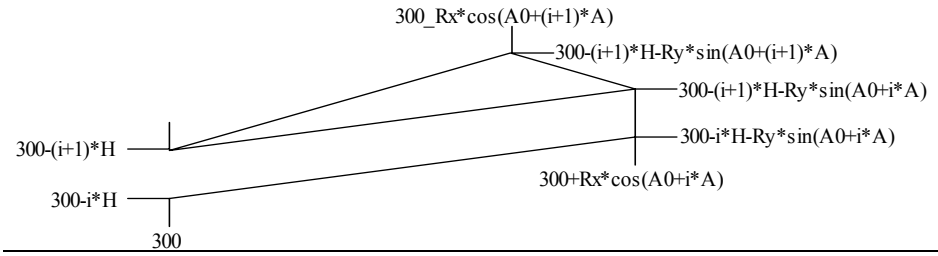
раздражая его мельканием элементов. Изображение формируется на невидимой странице, и показывается сразу, целиком, как в мультфильме.

Винтовая лестница

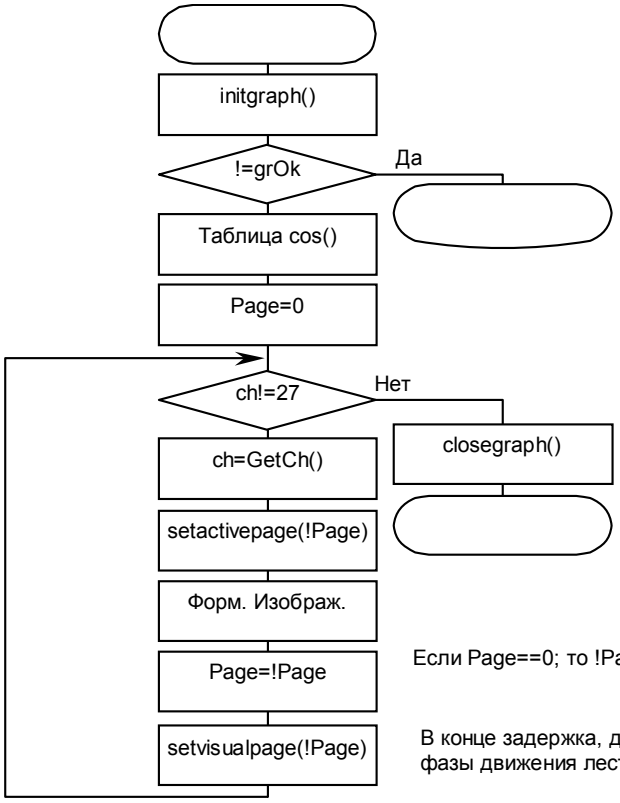
Программа должна имитировать вид возникающий при подъеме по винтовой лестнице. Необходимо изобразить четверть круга ступенек. Координаты вычисляются через функции sin(...) и cos(...). Скорость можно значительно поднять если не вычислять их каждый раз заново, а запомнить в массиве. Причем достаточно помнить таблицу только одной функции, на интервале [0..pi/2]. Другую всегда можно получить sin(x)=cos(pi-x).







Программа Винтовая лестница



Так как углы ступенек имеют повторяющиеся значения можно заранее сделать их таблицу, а не вычислять в цикле.

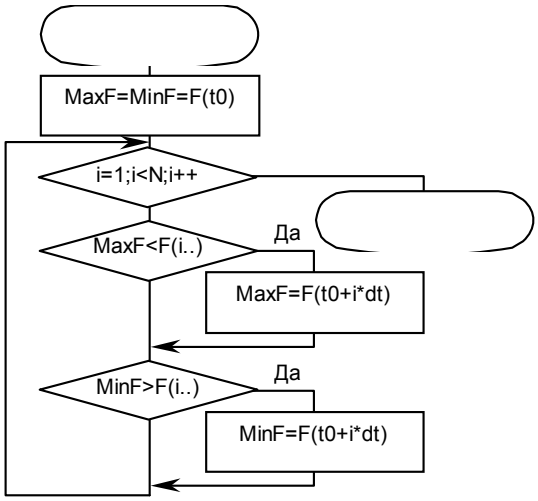
Лучше использовать свою функцию GetCh(), без ожидания нажатия на клавишу

Если Page==0; то !Page==1, !1=0.

В конце задержка, для демонстрации фазы движения лестницы

Часть 3
Основные алгоритмы
Поиск и сортировка
Нахождение минимума и максимума функции

Данная задача уже рассматривалась ранее, при выводе графиков. Необходимо найти максимальное и минимальное значение в массиве элементов. Имя исходного массива X, общее количество элементов N.




```
#include <stdio.h>
void GetMinMax(float *X,int N,float *MaxX,float *MinX)
{
   /* Функция пишется самостоятельно. */
} /* Кон. GetMinMax() */
void main(void)
{
        float X[100],Max,Min ;
        int i ;
        printf("\n Нахождение мин. и макс. значения в массиве.") ;
        for(i=0;i<100;i++) X[i]=10*sin(i/12.0)*cos(i/8) ;
        GetMinMax(X,100,&Max,&Min) ;
        printf("\n Макс. значение = %f",Max) ;
        printf("\n Мин. значение  = %f",Min) ;
} /* Кон. main() */
```

Для организации своей библиотеки может быть удобней сделать две функции float GetMax(...), float GetMin(...).

Поиск

Методы поиска делятся на линейные и бинарные. Пример алгоритма линейного поиска приведен выше (просматриваются все значения).

Алгоритмы бинарного поиска применяются для поиска в отсортированном массиве. Массив делится пополам и проверяется в какой половине находится нужное значение. Деление повторяется до нахождения искомого элемента.

Сортировка

Сортировка - процесс перегруппировки элементов в заданном порядке. Цель сортировки - облегчить последующий поиск элементов. Алгоритмы сортировки делятся на два класса:

- Сортировка массивов (внутренняя).
- Сортировка файлов (внешняя).

Рассматривается только первый класс методов. Они делятся на несколько групп. Прямые методы более простые и очевидные. Быстрые методы позволяют повысить скорость за счет значительного усложнения алгоритмов.

Во всех алгоритмах используются обозначения: a[] - сортируемый массив целых чисел, N - длинна массива.

Сортировка методом прямого включения

Сортируемый массив делится на две части: 1- a[0]..a[i], вторая конец массива. Алгоритм сортировки следующий:

```
for(i=0;i<N;i++)
{
x=a[i] ;
<Включение x  на соответствующее место среди a[0]..a[i]>
}
```

Пример:

0:	34	76	23	98	35	12
1:	34	76	23	98	35	12
2:	23	34	76	98	35	12
3:	23	34	76	98	35	12
4:	23	34	35	76	89	12
5:	12	23	24	35	76	98

Способы повышения скорости: Поиск места для включения искать не линейным перебором, а бинарным поиском.

Сортировка с помощью простого выбора

Алгоритм метода:

```
for(i=0;i<N;i++)
{
        <Нахождение мин. элемента a[k] из a[i]...a[N-1]>
        <Поменять местами a[i] и a[k]>
}
```

Пример:

```
0:        34  76  23  98  35  12
1:        12  76  23  98  35  34
2:        12  23  76  98  35  34
3:        12  23  34  98  35  76
4:        12  23  34  35  98  76
5:        12  23  34  35  76  98
```

Сортировка с помощью прямого обмена
Метод "пузырька"

Алгоритм сортировки основан на обмене соседних элементов массива. Сортировка продолжается до тех пор пока за весь проход массива не будет сделано ни одной перестановки.

```
<Флаг перестановки>=1
while(<Флаг перестановки>)
{
        <Флаг перестановки>=0
        for(i=1;i<N;i++)
        <Сравнение a[i-1] и a[i]> Если требуется перестановка:
        {
                <Перестановка местами a[i-1] и a[i]>
                <Флаг перестановки>=1
        }
}
```

Пример:

```
0:  34 76 23 98 35 40
1:  34 23 76 35 40 98
2:  23 34 35 40 76 98
```

Однако если минимальный элемент массива будет последним скорость сортировки заметно увеличится. Исправить это можно меняя направления просмотра, так называемая шейкерная сортировка.

Быстрые методы сортировок

Сортировка Шейлла, алгоритм основан на комбинациях сортировок: частей массива и целиком.

Сортировка с помощью дерева, алгоритм основан на создании двоичного дерева переставляемых значений и записи из него в отсортированный массив.

Сравнение методов

Сравнение методов удобнее производить по количеству перестановок и количеству сравнений.

Сжатие информации и кодирование

Сжатие и кодирование информации требуется прежде всего для организации долговременного хранения и передачи данных. Обычно пакеты программ, игрушки и т.п. передают в виде архивов, запакованных специальными программами упаковщиками. Наиболее распространенными из них являются ARJ, ZIP, RAR. Любой упаковщик работает на трех основных принципах.

Если взять любой файл на диске, например этот текст, в нем есть часто встречающиеся символы, редко встречающиеся и вовсе отсутствующие. Например пробел в этом тексте встречается 2756 раз, точка 1139 раз, буква i 620 раз, но из 255 возможных символов алфавита 162 символа в этом тексте не встречаются ни разу. Это более половины алфавита ЭВМ. Следовательно, для этого файла, два символа можно закодировать одним.

Кроме этого существуют повторяющиеся последовательности, например строка из пробелов. Тогда тоже достаточно в архиве сохранить один символ и указать количество его повторений.

Для маленьких файлов существует третий способ сократить занимаемое ими место. Файловая система MS DOS не может выделить для записи на диск блоками по 4 кБ. Даже если указывается, что файл занимает 1 байт, на самом деле он занимает 4 кБ, или для большего файла кратный 4 кБ размер. По этому даже простое слияние нескольких файлов в один, уже сокращает занимаемое ими место.

При передачи данных по каналам связи, например модем, встает обратная задача. Необходимо добавить к каждому передаваемому байту проверочную информацию, которая должна позволить принимаемой стороне распознать и если возможно исправить ошибку возникшую в канале связи. Этот процесс называется кодированием.

Может быть применено и шифрование. Тогда с информацией производят преобразования, например замену всех символов на другие значения из специальной таблицы. Понять такой текст без этой таблицы (ключа) невозможно.

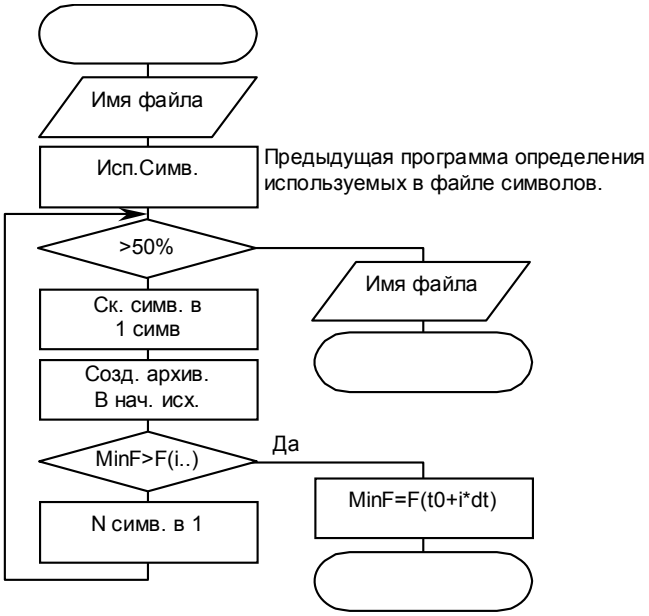
Программа анализа файла

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
        char name1[20],ch ;         /* Имя обрабатываемого файла       */
        FILE *File1 ;               /* Указатель на файл                */
        unsigned int  nc[256],i,n ; /* nc - счетчик вхождений символов */
        printf("\n Введите имя исходного файла : ") ;
        scanf("%s",name1) ;
        File1=fopen(name1, "rt") ;
        if(File1==NULL)
        {
                printf("\n Открыть указанный файл невозможно."); return(1) ;
        } /* Кон. if(ошибка открытия файла)  */
        for(i=0;i<256;i++) nc[i]=0 ; /* Обнуление счетчика */
        while(!feof(File1))
        {
                fscanf(File1,"%1c",&ch) ;
                nc[ch]++ ;
        } /* Кон. while() просмотр файла  */
        fclose(File1);
        n=0 ;
        for(i=0;i<256;i++)  if(nc[i]==0)  n++  ;  /* Подсчет числа неиспользуемых
символов */
        printf("\n В файле содержатся:") ;
        for(i=1;i<256;i++)
        {
                printf("\n Символ: %c (%3d)  содержимся %5d раз.",i,i,nc[i]) ;
                if((i%20)==0) getch() ; /* Задержка после вывода страницы */
        } /* Кон. for()  */
        printf("\n В файле ни разу не использовались  %d символов.",n) ;
        return(1) ;
} /*  Кон. main()   */
```

Упаковщик файлов

Необходимо реализовать простейший упаковщик реализующий следующий алгоритм сжатия. Составляется таблица перекодировки символов, символы не используемые в файле в ней отсутствуют. Если количество не используемых символов велико, то можно записать в один символ несколько. Для этого рекомендуется использовать операцию сдвига.

В предлагаемой программе предлагается записывать два символа в один, это возможно если число используемых символов в тексте не превышает 32, только заглавные или строчные буквы. Иначе программа должно вывеси сообщение о невозможности сжать файл.



Позиционирование по файлу

Позиционирование по файлу осуществляет функция: fseek(*<Файл>, <Смещение>, <Откуда>) <Откуда> - одно из
SEEK_SET начало файла
SEEK_CUR - текущая позиция.
SEEK_END - конец файла.
*<Файл> - Указатель на файл полученный из open().

Двоичный сдвиг

<Переменная или число> >> <Количество разрядов>
<Переменная или число> << <Количество разрядов>

Операция выполняет двоичный сдвиг числа на указанное число разрядов, так как числа представляются в двоичной системе счисления, сдвиг на 1 разряд эквивалентен делению или умножению на 2.
Операция допустима только над типами int, long, char.

Улучшение программы упаковки

Можно организовать упаковку не одного файла, а например всего каталога. Для этого удобно пользоваться функциями findfirst() и findnext().

Другим вариантом улучшения программы может быть устранение повторяющихся символов.

Поиск файлов

Поиск файлов осуществляют функции findfirst() и findnext(). Они описаны в модуле dir.h.
done = findfirst(<Шаблон поиска>,&ffblk,<Атрибуты>)
done = findnext(&ffblk)
Информация о найденном файле помещается в структуру ffblk.

Функции возвращают значение int done. Если поиск окончился успешно, то done=1, если указанных файлов больше найти не удалось, то done=0. Пример:

```
#include <stdio.h>
#include <dir.h>
int main(void)
{
        struct ffblk ffblk ; /* Информация о файлах */
        int done ; /* Результат поиска  */
        printf("\n Содержимое каталога:") ;
        done = findfirst("*.*",&ffblk,0) ;
        while (!done)
        {
        printf("\n %s", ffblk.ff_name) ;
        done = findnext(&ffblk) ;
        } /* Кон. while   */
        return(0) ;
} /* Кон. main()   */
```

Часть 4
Основные математические алгоритмы
Решение системы линейных уравнений

В вычислительной практике очень часто приходится сталкиваться с системами линейных уравнений

a11x1 + a12x2 + . . . +a1nxn = b1
a21x1 + a22x2 + . . . +a2nxn = b2
. . . . . .
an1x1 + an2x2 + . . . +annxn = bn

В матричной форме записи эта система принимает вид:

A*x=B,

где -

$$A = \begin{bmatrix} a11 & a12 & \dots & a1n \\ a21 & a22 & \dots & a2n \\ \dots & \dots & \dots & \dots \\ an1 & an2 & \dots & ann \end{bmatrix}, \quad x = \begin{bmatrix} x1 \\ x2 \\ \dots \\ xn \end{bmatrix}, \quad b = \begin{bmatrix} b1 \\ b2 \\ \dots \\ bn \end{bmatrix}.$$

Наиболее часто для решения линейных систем применяется метод Гауса. Он основан на приведении матрицы коэффициентов A к треугольному виду.

В методе Гауса используется следующие фундаментальное свойства линейных систем:

- Перестановка строк (в массивах A, x и b) не приводит к изменению результата.
- Прибавляя (вычитая) к какой либо строке элементы другой строки или их линейную комбинацию (умноженные на коэффициент) не меняет решение системы.

Алгоритм решения следующий:

Цикл по всем столбцам for(i=0;i<N;i++)
{
    Найти максимальный по модулю элемент в i столбце A[k][i].
    Поменять строки местами.
    Разделить все элементы строки на A[i][i].
    Цикл по оставшимся строкам for(j=i+1;j<N;j++)
    {
        Из j-й строки вычесть i-ю строку умноженную на A[j][i].
    }
}

На выходе так называемая треугольная матрица (см. пример).

Обратный цикл вычисления X.

Пример:

3*X1 + 9*X2 + 16*X3 = -2
2*X1 + 8*X2 + 16*X3 = 2
4*X1 + 8*X2 - 4*X3 = -12

Или в матричной форме:

$$\begin{bmatrix} 3 & 9 & 16 \\ 2 & 8 & 16 \\ 4 & 8 & -4 \end{bmatrix} * \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \\ -12 \end{bmatrix}$$

Этапы решения:

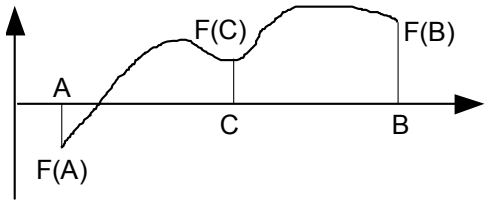
1 i=0; Макс. элемент в ім столбце 2й, меняем местами ію и 2ю строки	3 9 16 2 8 16 4 8 -4	-2 2 -12
2 Делим ію строку на A[i][i], и вычитаем из всех следующих, j строк, умнож. ее на A[j][i]	4 8 4 2 8 16 3 9 16	-12 2 -2
3 i=1;Макс. по модулю элем. в ім столбце, с ій строки 1й, менять строки не требуется.	1 2 -1 0 4 18 0 3 19	-3 8 7
4 Делим ію строку на A[i][i], и вычитаем из всех следующих, j строк, умнож. ее на A[j][i]	1 2 -1 0 4 18 0 3 19	-3 8 7
5 Делим ію строку на A[i][i]. Получаем треугольную матрицу (нижняя половина == 0)	1 2 -1 0 1 4.5 0 0 (5.5	-3 2 1)

6 Обратный цикл X3 = B3 = 0.18
7 X2 = B2 - A23 * X3 = 1.18
8 X1 = B1 - A13 * X3 - A12 * X2 = -5.18

Решение нелинейных уравнений

Большинство уравнений описывающих реальные явления содержат нелинейности. Под нелинейностями понимаются элементы, характеристики которых (y=F(x), где y - выход, x - вход) не являются линейными. На практике, как правило пользуются упрощенными моделями, например, пружинный маятник, нелинейность коэффициента упругости пружины считают постоянным, а это истинно только на небольшом интервале растяжений пружины. Но для нахождения точного решения этого не достаточно. Учет нелинейностей приводит к образованию сложных математических выражений, для которых не возможно записать решение в явной форме. В этом случае применяются приближенные, численные, методы решения.

Метод деления пополам



Метод основан на последовательном приближении к решению. Используются обозначения: FA=f(A), FB=f(B), FC=f(C).

1 -Выбирается отрезок на котором решение существует (FA>0)&&(FB<0) или (FA<0)&&(FB>0). Иначе решения нет.

2 - Отрезок [A B] делится пополам, получаем точку C.

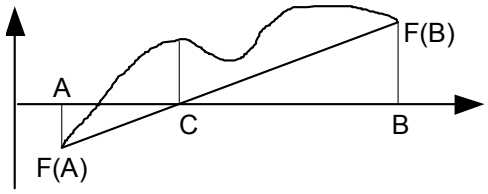
3 - Проверяем является ли точка C решением с выбранной точностью (fabs(FC)<Точность). Если да, то решение найдено.

4 - Если знак FB совпадает с FA, как на рис., то принимаем за новое значение точки B точку C. Соответственно FB=FC. Возвращаемся к пп 2.

5 - Иначе перемещаем правую границу, те точка A. A=C, FA=FC. Переход к пп 2.

За каждый проход этого алгоритма отрезок, на котором находится решение, сжимается в два раза, постепенно локализуя решение.

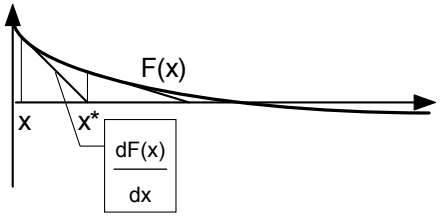
Метод секущих



Метод очень похож на предыдущий, иначе определяется только точка C. Можно предположить, что решение ближе к тому краю отрезка, где значение функции меньше. Проводим прямую линию [FA FB], и из двух подобных треугольников находим точку C. (|FA|/|FB|= (C-A)/(B-C). В остальном алгоритм аналогичен предыдущему.

Метод Ньютона

На практике часто встречаются уравнения, для которых нельзя указать интервал (A-B) на котором находится решение. В током случае обычно используют метод Ньютона.



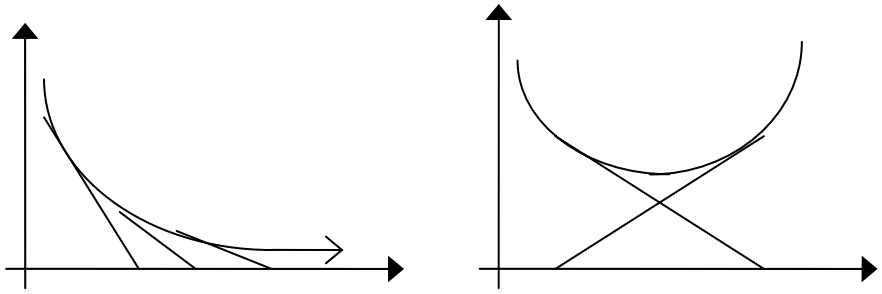
Из точки F проводится касательная к графику F(x), точка пересечения ее с осью x дает новое приближение решения X*. Это повторяется пока :

|F(x)| > <Точности решения>.

Угол наклона касательной можно найти через центральные разности:

$$\frac{dF(x)}{dx} = \frac{F(x+h) - F(x-h)}{2 \cdot h}$$

При использовании данного метода необходимо учитывать возможность зацикливания и отсутствия решения. Эти случаи представленны ниже. Для того чтобы избежать эти ситуации рекомендуется ввести переменную int n_iter и увеличивать ее на каждой итерации n_iter++. Если она превышает устновленный придел, то возможно решения нет.



Решение систем нелинейных уравнений

На практике задача отыскания решения системы уравнений встречается чаще, чем решение уравнения с одним неизвестным. Рассмотрим для примера систему:

$$\begin{cases} x1' + 10*(x1 - x2) = 0 \\ x2' + x1 * x3 - 28*x1 + x2 = 0 \\ x3' - x1 * x2 + 2.6*x3 = 0 \end{cases}$$ или в векторной форме F(X)=0

Прямое решение такой системы невозможно. Единственный путь итерационные методы.

1.Выбирается некоторое приближение к решению (начальные значения)

2.Массиву предыдущих значений X присваиваем текущие значения X.

3.Вычисляется функция Якоби:

$$F'(x) = \begin{bmatrix} \frac{dF1(x)}{dx1} & \frac{dF1(x)}{dx2} & \frac{dF1(x)}{dx3} \\ \frac{dF2(x)}{dx1} & \frac{dF2(x)}{dx2} & \frac{dF2(x)}{dx3} \\ \frac{dF3(x)}{dx1} & \frac{dF3(x)}{dx2} & \frac{dF3(x)}{dx3} \end{bmatrix}$$

$$x1\big|_{t=0} = -8$$
$$x2\big|_{t=0} = 8$$
$$x3\big|_{t=0} = 27$$

Эта функция является линеаризацией (упрощением) исходной системы уравнений в окрестностях точек x1,x2,x3

4. Считаем

F(X(n+1))=F(X(n)) + F'(X(n)) * (X(n+1)-X(n)) = 0

Или

F'(X(n)) * D X(n+1) = - F(X(n))

Решая эту систему относительно D X(n+1), получаем:

X(n+1) = X(n) + D X(n+1)

5. Полученные точки x1,x2,x3 подставляем в исходную систему уравнений. И если S | Fi(X) | <= Заданной точности, то решение найдено, иначе возвращаемся к пп. 3.

6. Если количество итераций затраченных на нахождение решения больше установленного предела, то решение не найдено, прерываем расчет.

7. Выводим или сохраняем рассчитанные точки, увеличиваем счетчик времени. Если необходимо продолжать расчет, то возвращаемся к пп. 2.

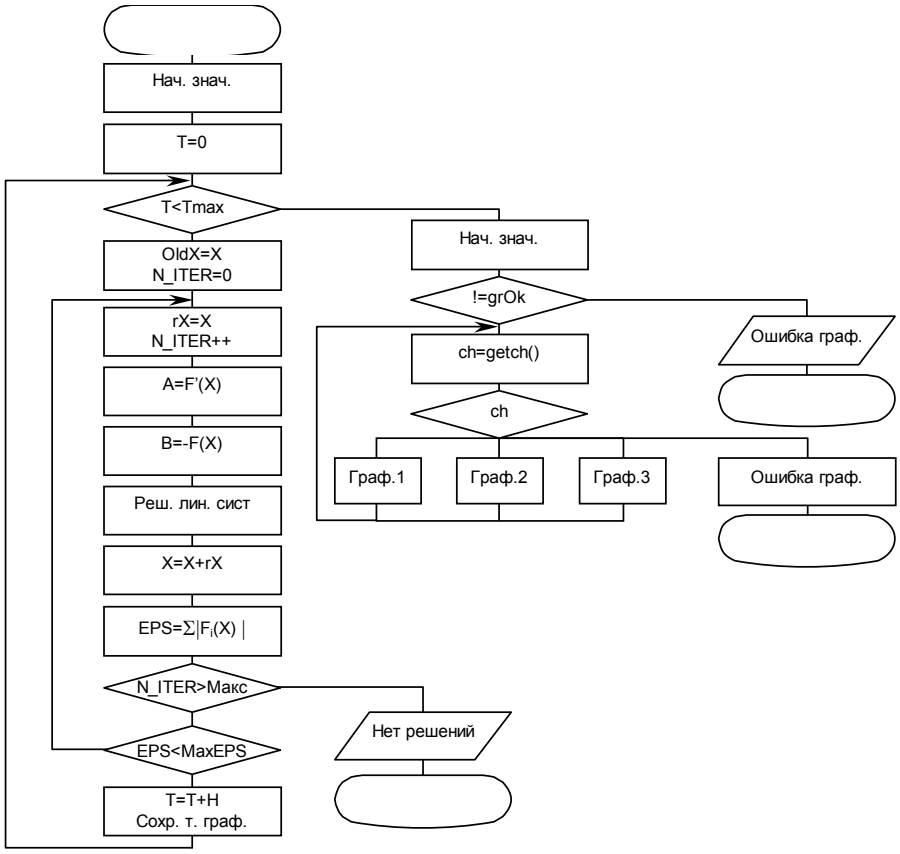
Рекомендуется использовать:

```
float X[4]              ; Значения независимых переменных                *
float OldX[4]           ; Значения переменных на предыдущем шаге         *
float rX[4]             ; Рабочий массив
float H                 ; Шаг интегрирования                             *
float dt                ; Шаг дифференцирования                          *
float EPS               ; Ошибка вычислений
float MaxEPS=0.05       ; Максимально допустимая ошибка вычислений
int N_ITER              ; Счетчик количества итераций
int MaxITER=10          ; Максимально допустимое количество итераций
float A[4][4]           ; Значения Якобиана
float B[4]              ; Значения функции
float T                 ; Текущее время
float MaxT              ; Общее время расчета
float G1[500],
      G2[500],G3[500]   ; Точки для графиков функции

float dX(int n)
{
        return((X[n]-OldX[n])/H) ;
} /* Кон. dX() - Функция для вычисления производной n-ой  перем.  */
float F(int n)
{
        float r ;
        switch(n)
        {
                case  0:r=dX(0)+10*(X[0]-X[1]) ; break ;
                case  1:r=dX(1)+X[0]*X[2]-28*X[0]+X[1] ; break ;
                case 2:r=dX(2)+X[0]*X[1]+2.6*X[2] ; break ;
                default:printf("\n Недопустимое значение функции %d",n) ;
        } /* Кон. switch() */
} /* Кон. F() - Функция описания модели  */
```

* - Рекомендуется описать как глобальные переменные.

Блок схема программы расчета НСУ с выводом результатов в форме графиков



Варианты курсовых работ
Требования к курсовой работе

Курсовая работа выполняется по индивидуальным (на бригаду) заданиям. Тема задания согласованная с преподавателем. Срок выполнения один семестр. Кроме оговоренных вариантов, режим работы программы графический.

Отчет по курсовой работе должен содержать постановку задачи, блок схему алгоритма с пояснениями, описание используемых данных и распечатку исходного текста программы.

Прикладные программы

1. Оконная библиотека.
2. Редактор строки (режимы Inset, Delete, клавиши управления курсором, маска допустимых значений).
3. Программа графического меню. Горизонтальное, вертикальное, выпадающие подменю, выбор по функциональным клавишам.
4. Вывод на экран PCX картинки.
5. Калькулятор.
6. Интерпретатор простейшего языка программирования.
7. Простая издательская система.
8. Графический редактор.
9. Архиватор.
10. Простой вариант рограмы Norton Commander

Программы для учебного процесса

1, Демонстрация видов сортировок.
2. Демонстрация численного решения нелинейных уравнений.
3. Лабораторная работа "Исследование случайных процессов".
4. Лабораторная работа "Прохождение сигнала через нелинейный элемент".
5. Программа колоквиума (опроса) с регистрацией и ведение журнала. Текст вопросов и ответы зашифрованы.
6. Программы демонстрации криптографических программ: шифры Сциталь, Цезаря, Виженера, трафаретная, «плящушие человечки», и др.

Учебные программы

1. Имитация колонии жизни.
2. Имитация машины Поста.

Игровые программы

1, Игра шарики.
2. Лабиринт.
3. Морской бой.
4. Popcorn или подобное.
5. Игра в стиле prince.
6. Пинбол.
7. Сражение в космосе (starcon или mach3).
8. Война роботов.
9. Тактические игры.
10. Игра на бирже.
11. Тесты памяти.
12. Музыкальный редактор.
13. Домино.
14. Тетрис

Литература

Использование ПК.

1. В.Э. Фигурнов. IBM PC для пользователей. (любое издание).

Язык программирования

1. С.О. Бочков, Д.М. Субботин.
Язык программирования Си для персонального компьютера.
М. СП: "Диалог" "Радио и связь", 1990 г.
2. М.А. Аксёнкин, О.Н. Целобёнок.
Язык С. Минск: "Ууіверсітэцкае", 1995 г.
3. Справочник по функциям Borland C++ 3.1/4.0
Киев: "Диалектика" 1994 г.
4. Прокофьев Б.П. и др. Графические средства Turbo C и Turbo C++.
Москва: "Финансы и статистика", 1992 г.
5. Страуструп Б. Язык программирования Си++.
Москва: "Радио и связь", 1991 г.

Теория программирования

1. Ален И. Голуб. C & C++ правила программирования.
М.: Бином, 1996г.

Вычислительная математика

1. Амосов А.А. Дубинский Ю.А. Копченова Н.В.
Вычислительные методы для инженеров. М.: Высшая школа, 1994 г.

Основные алгоритмы

1. Вирт Н. Алгоритмы + структура данных = программа.
М.: Мир, 1985 г.
2. Вирт Н. Алгоритмы и структуры данных. М.: Мир, 1989 г.

Оглавление