Криворожский государственный педагогический институт
Кафедра информатики и прикладной математики

Курсовая работа

на тему

Составление примеров программ в процессе обучения программированию в среде Windows

Выполнил
студент группы МИ-93-2
Семериков С.А.

Проверил:

Доц., к.т.н. Полищук А.П.

Кривой Рог 1997

Содержание

Введение

При обучении программированию в среде Windows важную роль играет подбор примеров, на которых обучаемый усваивает принципы событийного программирования в графической оболочке.

В связи с этим была поставлена задача перевести под Windows интегрированные примеры из книги А.П.Полищука “Информатика. Персональный компьютер и его программирование”, которые не только иллюстрируют работу с графическим интерфейсом, но и решают конкретные прикладные задачи.

Первая программа – это синтаксический анализатор и интерпретатор заданных параметрически формул с построением графика. В процессе перевода не было реализовано динамическое изменение графика – вместо этого он автоматически масштабируется. Кроме того, несколько упрощены алгоритмы фильтрованого ввода путём отслеживания изменений в строках редактирования.

Вторая программа – это переделанный вариант многофункциональной программы обработки векторов с тремя списковыми окнами: для условия задачи, для выбора файла данных и для вывода результатов выполнения задачи.

Третья программа – летающий мячик – с небольшими модификациями позаимствована из книги Чарльза Петцольда. Основным отличием её от оригинала является дополнительная обработка сообщений мыши и наличие специальной процедуры обработки сообщений от таймера.

Программа, рисующая отражающийся от границ окна шар

```
#define STRICT//для строгой проверки типов
#include <windows.h>
#include <stdlib.h>

//2 макроса - для определения минимума и максимума
#define min(a, b)  (((a) < (b)) ? (a) : (b))
#define max(a, b)  (((a) > (b)) ? (a) : (b))

//прототип функции окна
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
//прототип функции для обработки сообщения WM_TIMER
void CALLBACK TimerProc(HWND, UINT, UINT, DWORD);

HBITMAP hBitmap ;/*идентификатор изображения мяча*/
/*служебные данные*/
int     cxClient, cyClient, xCenter, yCenter, cxTotal, cyTotal,
        cxRadius, cyRadius, cxMove, cyMove, xPixel, yPixel ;

//основная программа, принимающая
/*четыре параметра - идентификатор программы, идентификатор предыдущего
экземпляра программы, параметры командной строки и режим отображения
окна (нормальное, развёрнутое на весь экран, свёрнутое в пиктограмму и
т.д.)*/
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR szCmdLine, int iCmdShow)
{
  static char szAppName[] = "Bounce" ;/*имя класса окна*/
  HWND        hwnd ;                  /*идентификатор окна*/
  MSG         msg ;                   /*структура сообщения*/
  WNDCLASS    wndclass ;              /*структура класса окна*/
  ATOM        IsReg;                  /*для регистрации окна*/

  /*стиль окна определяем как перерисовываемое при изменении
    как горизонтального, так и вертикального его размеров*/
  wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc   = WndProc ; /*указатель на функцию окна*/
  /*дополнительных данных как для класса, так и для окна у нас не будет*/
  wndclass.cbClsExtra    = 0 ;
  wndclass.cbWndExtra    = 0 ;
  wndclass.hInstance     = hInstance ;/*идентификатор приложения*/
  /*загружаем стандартную иконку*/
  wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
  /*загружаем стандартный курсор в виде стрелочки*/
  wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
  /*цвет окна, 0 - окно прозрачно*/
  wndclass.hbrBackground = (HBRUSH)(1+COLOR_WINDOW);
  wndclass.lpszMenuName  = NULL ;/*меню у нас не будет*/
  wndclass.lpszClassName = szAppName ;/*имя класса окна*/

  IsReg=RegisterClass(&wndclass) ;/*пытаемся зарегистрировать класс*/
  if(!IsReg)/*если не удалось зарегистрироваться*/
  {
    /*вызываем стандартную функцию вывода диалогового окна с параметрами:
      идентификатор окна-родителя (0-если нет такового), сообщение,
      заголовок диалога, характеристики диалога (здесь - иконка + кнопка
OK)*/
    MessageBox (0, "Ошибка регистрации класса окна",
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return FALSE ;/*выходим из программы*/
  }
```

```
  /*пытаемся создать окно и получить его идентификатор*/
  hwnd = CreateWindow (szAppName,/*имя класса окна*/
                       "Арканоид",/*заголовок окна*/
                       WS_OVERLAPPEDWINDOW,/*стиль окна - перекрывающееся*/
                       /*координаты x,y верхнего левого угла окна и его
                         размеры позволим определить ОС самостоятельно*/
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       NULL,/*идентификатор окна-предка, 0-если нет таково-
го*/
                       NULL,/*идентификатор меню, 0-если его нет*/
                       hInstance,/*идентификатор приложения*/
                       NULL) ;/*данные для создания окна*/
  if(!hwnd)/*если не удалось родить окно*/
  {
    /*вызываем стандартную функцию вывода диалогового окна с параметрами:
      идентификатор окна-родителя (0-если нет такового), сообщение,
      заголовок диалога, характеристики диалога (здесь - иконка + кнопка
OK)*/
    MessageBox (0, "Ошибка создания окна",
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return FALSE ;/*выходим из программы*/
  }
  /*пытаемся создать таймер для заданного окна с идентификатором 1,
    вызывающемся 1 раз в 50 миллисекунд, сообщения от которого обрабатываются
    специальной таймерной процедурой*/
  if (!SetTimer (hwnd, 1, 50, TimerProc))/*если не удалось создать таймер*/
  {
    /*диагностика невозможности создания таймера*/
    MessageBox (hwnd, "В системе слишком много таймеров",
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return FALSE ;/*выход из программы*/
  }
  /*после создания окна необходимо отобразить вначале его рамку и т.д.,*/
  ShowWindow (hwnd, iCmdShow) ;
  /*а лишь затем инициализировать прорисовку окна*/
  UpdateWindow (hwnd) ;
  /*запускаем цикл обработки сообщений*/
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;/*трансляция клавиатурных сообщений -
                                  можно в данном случае и не делать*/
    DispatchMessage (&msg) ; /*диспетчеризация сообщений по функциям*/
  }
  return msg.wParam ;        /*чем плох такой код возврата*/
}


/*эта функция обратного вызова занимается только обработкой сообщений
  от таймера. Её параметрами являются
  - идентификатор окна, владеющего таймером
  - идентификатор сообщения (всегда WM_TIMER)
  - идентификатор таймера и
  - системное время*/
//эта прагма призывает компилятор не беспокоить нас сообщениями о неиспользу-
емых параметрах
#pragma argsused
void CALLBACK TimerProc(HWND hwnd,UINT msg,UINT idTimer,DWORD dwTime)
{
  HDC hdc,hdcMem;/*для контекстов устройств*/

  if (!hBitmap)/*если нечего отображать*/
    return;/*выход из функции*/

  /*получаем контекст устройства для клиентской области данного окна*/
  hdc = GetDC (hwnd) ;
```

```
  /*создаём в памяти контекст устройства, совместимый с данным контекстом*/
  hdcMem = CreateCompatibleDC (hdc) ;
  /*выбираем в контекст памяти битовое изображение*/
  SelectObject (hdcMem, hBitmap) ;
  /*копируем из битовую маску из контекста памяти в контекст устройства*/
  BitBlt (hdc, xCenter - cxTotal / 2,
          yCenter - cyTotal / 2, cxTotal, cyTotal,
          hdcMem, 0, 0, SRCCOPY) ;
  /*освобождаем оконный контекст*/
  ReleaseDC (hwnd, hdc) ;
  DeleteDC (hdcMem) ;/*уничтожаем контекст памяти*/

  xCenter += cxMove ;/*смещаем центр*/
  yCenter += cyMove ;

  /*если достигли границы окна по координате - меняем направление перемеще-
ния*/
  if ((xCenter + cxRadius >= cxClient) ||
     (xCenter - cxRadius <= 0))
    cxMove = -cxMove ;

  if ((yCenter + cyRadius >= cyClient) ||
     (yCenter - cyRadius <= 0))
    cyMove = -cyMove ;
}


/*функция окна, как и функция таймера, принимает 4 параметра:
  1. идентификатор окна
  2. сообщение окну
  3. слово
  4. двойное слово - дополнительные данные для сообщения*/
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  HBRUSH         hBrush ;/*ID кисти, контекстов экрана и памяти*/
  HDC            hdc, hdcMem ;
  int            iScale ;

  switch (iMsg)/*обработка сообщений*/
  {
    case WM_CREATE :/*при создании окна*/
    {
      hdc = GetDC (hwnd) ;/*получаем контекст окна*/
      /*возвращает разрещающую способность по координатам*/
      xPixel = GetDeviceCaps (hdc, ASPECTX) ;
      yPixel = GetDeviceCaps (hdc, ASPECTY) ;
      ReleaseDC (hwnd, hdc) ;/*освобождаем контекст устройства для окна*/
      return 0 ;/*сообщение обработано*/
    }
    case WM_SIZE :/*при изменении размеров окна*/
    {
      /*ширина и высота клиентской области окна находятся
        соответственно в младшем и старшем словах двойного слова*/
      xCenter = (cxClient = LOWORD (lParam)) / 2 ;
      yCenter = (cyClient = HIWORD (lParam)) / 2 ;

      iScale = min (cxClient * xPixel, cyClient * yPixel) / 16 ;

      cxRadius = iScale / xPixel ;
      cyRadius = iScale / yPixel ;

      cxMove = max (1, cxRadius / 2) ;
      cyMove = max (1, cyRadius / 2) ;

      cxTotal = 2 * (cxRadius + cxMove) ;
      cyTotal = 2 * (cyRadius + cyMove) ;
```

```
      if (hBitmap)/*если битовая маска есть - уничтожаем её*/
        DeleteObject (hBitmap) ;

      hdc = GetDC (hwnd) ;/*получаем контекст окна*/
      hdcMem = CreateCompatibleDC (hdc) ;/*создаём контекст памяти, совмести-
мый
                                          с контекстом окна*/
      /*создаём битовую маску, совместимую с данным окном*/
      hBitmap = CreateCompatibleBitmap (hdc, cxTotal, cyTotal) ;
      ReleaseDC (hwnd, hdc) ;/*освобождаем контекст окна*/

      SelectObject (hdcMem, hBitmap) ;/*выбираем битовую маску в контекст па-
мяти*/
      Rectangle (hdcMem, -1, -1, cxTotal + 1, cyTotal + 1) ;/*прямоугольник*/
      /*создаём кисть с шаблоном "сетка под углом 45 градусов"*/
      hBrush = CreateHatchBrush (HS_DIAGCROSS, 0L) ;
      SelectObject (hdcMem, hBrush) ;/*выбираем кисть в контекст памяти*/
      /*устанавливаем цвет фона*/
      SetBkColor (hdcMem, RGB (random(256), random(256), random(256))) ;
      /*рисуем в памяти эллипс*/
      Ellipse (hdcMem, cxMove, cyMove, cxTotal - cxMove, cyTotal - cyMove) ;
      DeleteDC (hdcMem) ;/*уничтожаем контекст памяти*/
      DeleteObject (hBrush) ;/*уничтожаем кисть*/
      return 0 ;/*сообщение обработано*/
    }
    case WM_LBUTTONDOWN :/*при нажатии на мышь*/
    case WM_RBUTTONDOWN :
    case WM_MBUTTONDOWN :
    {
      /*координаты мышиного курсора в окне находятся
        соответственно в младшем и старшем словах двойного слова*/
      xCenter = LOWORD (lParam);
      yCenter = HIWORD (lParam);
      return 0 ;/*сообщение обработано*/
    }
    case WM_DESTROY :/*при закрытии окна*/
    {
      if (hBitmap)/*если есть битовое изображение*/
        DeleteObject (hBitmap) ;/*уничтожаем его*/

      KillTimer (hwnd, 1) ;/*останавливаем таймер*/
      PostQuitMessage (0) ;/*помещаем в очередь сообщений окна
                             посмертное сообщение, прерывающее
                             цикл обработки сообщений*/
      return 0 ;/*сообщение обработано*/
    }
  }
  /*для всех необработанных сообщений вызываем функцию обработки сообщений по
умолчанию*/
  return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
```

Программа для работы со списками файлов и задач на одномерные массивы

```
#define STRICT//для строгой проверки типов
#include <windowsx.h>/*файл макросов для облегчения работы*/
#include <stdlib.h>
#include <string.h>
#include <dir.h>
#include <stdio.h>
#include <alloc.h>


//прототип функции окна
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

HINSTANCE hInst;/*копия идентификатора программы*/

HWND hCommonWindow;/*окно списка, в которое пользовательским программам
                     позволено производить вывод*/

/*этот набор констант предназначен для идентификации дочерних окон*/
const
  TaskList=1,/*список задач*/
  TaskStr=2,/*статическая строка с выбранным условием*/
  FileList=3,/*список файлов*/
  FileStr=4,/*строка с выбранным файлом*/
  FileButton=5,/*кнопка для выбора файла и т.д.*/
  StartButton=6,/*кнопка, по нажатию которой программа выполняется*/
  StEdit=7,/*надпись перед строкой ввода*/
  Edit=8,/*строка ввода*/
  CommonWindow=9;/*общий список*/


/*функции-пустышки, которые надо будет отпрограммировать*/
void func1(char*, unsigned ){};
void func2(char*, unsigned ){};
void func3(char*, unsigned ){};
void func4(char*, unsigned ){};
void func5(char*, unsigned ){};
void func6(char*, unsigned ){};
void func7(char*, unsigned ){};
void func8(char*, unsigned ){};
void func9(char*, unsigned ){};
void func10(char*,unsigned ){};
void func11(char*,unsigned ){};
void func12(char*,unsigned ){};
void func13(char*,unsigned ){};


//Среднее арифметическое элементов массива
/*каждая из функций принимает 2 параметра - имя файла с данными и размер мас-
сива*/
void func0(char* fname,unsigned n)
{
  char buf[80],all[80];/*просто две строки*/
  double sum;

  ListBox_ResetContent(hCommonWindow);/*очистка окна общего списка*/
  //Откроем файл для чтения
  FILE* f=fopen(fname,"r");
  if(f==NULL)/*а смогли ли мы открыть файл?*/
  {
     sprintf(buf,"Неудача при открытии файла %s",fname);/*печать в строку*/
```

```
     /*и занесение строки в список для диагностики*/
     ListBox_AddString(hCommonWindow,buf);
     return;
  }
  //Выделим память для массива
  double* a=(double*)farmalloc(n*sizeof(double));
  if(a==NULL)/*если не удалось выделить*/
  {
    sprintf(buf,"Неудача выделения памяти");/*печать в строку*/
    ListBox_AddString(hCommonWindow,buf);/*добавляем эту строку в список*/
    fclose(f);/*закрываем файл*/
    return;/*выходим*/
  }
  //Читаем слова из файла с преобразование к double в массив a
  for(int i=0;i<n;i++)
    fscanf(f,"%lf",a+i);/*no comments*/
  //Закрываем ненужный больше файл
  fclose(f);
  /*добавляем в список две строки*/
  ListBox_AddString(hCommonWindow,"Исходный массив");
  ListBox_AddString(hCommonWindow,"");
  /*в каждую строку списка заносим по три числа*/
  *all=0;
  for(i=0;i<n;i++)
  {
    sprintf(buf," %-5.2lf ",a[i]);
    strcat(all,buf);
    if(!((i+1)%3))
    {
      ListBox_AddString(hCommonWindow,all);
      *all=0;
    }
  }
  /*выводим остаток чисел*/
  ListBox_AddString(hCommonWindow,all);
  for(i=sum=0;i<n;sum+=a[i++]);/*суммируем*/
  sum/=n;/*вычисляем среднее арифметическое*/
  farfree(a);    //Освобождаем арендованную память
  //Выводим результат обработки
  ListBox_AddString(hCommonWindow,"");
  ListBox_AddString(hCommonWindow,"");
  sprintf(buf,"Среднее арифметическое = %lf",sum);
  ListBox_AddString(hCommonWindow,buf);
}

     /*Функция сравнения элементов типа double - указатель на нее
надо дать аргументом библиотечной функции быстрой сортировки
qsort().*/
int cmpf(const void* a, const void* b)
{
  if(*(double*)a==*(double*)b)
    return 0;
  if(*(double*)a<*(double*)b)
    return -1;
  else
    return 1;
}


//Отсортировать массив быстрой сортировкой
void func14(char* fname,unsigned n)
{
  char buf[80],all[80];

  /*чистим список*/
  ListBox_ResetContent(hCommonWindow);
```

```
  //Откроем файл для чтения
  FILE* f=fopen(fname,"r");
  if(f==NULL)/*если не удалось открыть файл*/
  {
     /*печатаем диагностику в строку*/
     sprintf(buf,"Неудача при открытии файла %s",fname);
     /*добавляем строку в список*/
     ListBox_AddString(hCommonWindow,buf);
     return;/*и выходим*/
  }
  //Выделим память для массива
  double* a=(double*)farmalloc(n*sizeof(double));
  if(a==NULL)/*если не удалось выделить память*/
  {
    sprintf(buf,"Неудача выделения памяти");/*так и скажем*/
    ListBox_AddString(hCommonWindow,buf);/*добавим строку в список*/
    fclose(f);/*закрываем файл*/
    return;/*выходим*/
  }
  //Читаем слова из файла с преобразование к double в массив a
  for(int i=0;i<n;i++)
    fscanf(f,"%lf",a+i);
  //Закрываем ненужный больше файл
  fclose(f);
  /*выводим массив в список по три числа в строке*/
  ListBox_AddString(hCommonWindow,"Исходный массив");
  ListBox_AddString(hCommonWindow,"");
  *all=0;
  for(i=0;i<n;i++)
  {
    sprintf(buf," %-5.2lf ",a[i]);
    strcat(all,buf);
    if(!((i+1)%3))
    {
      ListBox_AddString(hCommonWindow,all);
      *all=0;
    }
  }
  ListBox_AddString(hCommonWindow,all);
  //Сортируем с помощью библиотечной ф-ции qsort()
  qsort((void *)a, n, sizeof(double), cmpf );
  //Выводим результат обработки
  ListBox_AddString(hCommonWindow,"");
  ListBox_AddString(hCommonWindow,"");
  ListBox_AddString(hCommonWindow,"Отсортированный массив");
  ListBox_AddString(hCommonWindow,"");
  *all=0;
  for(i=0;i<n;i++)
  {
    sprintf(buf," %-5.2lf ",a[i]);
    strcat(all,buf);
    if(!((i+1)%3))
    {
      ListBox_AddString(hCommonWindow,all);
      *all=0;
    }
  }
  ListBox_AddString(hCommonWindow,all);
  farfree(a);    //Освобождаем арендованную память
}
```

/* Для каждой задачи у нас есть ее изложение в виде строки текста - позже мы собираемся использовать его при выводе списка предоставляемых программой услуг по обработке одномерных массивов. К этим строкам надо "привязать" функции решения

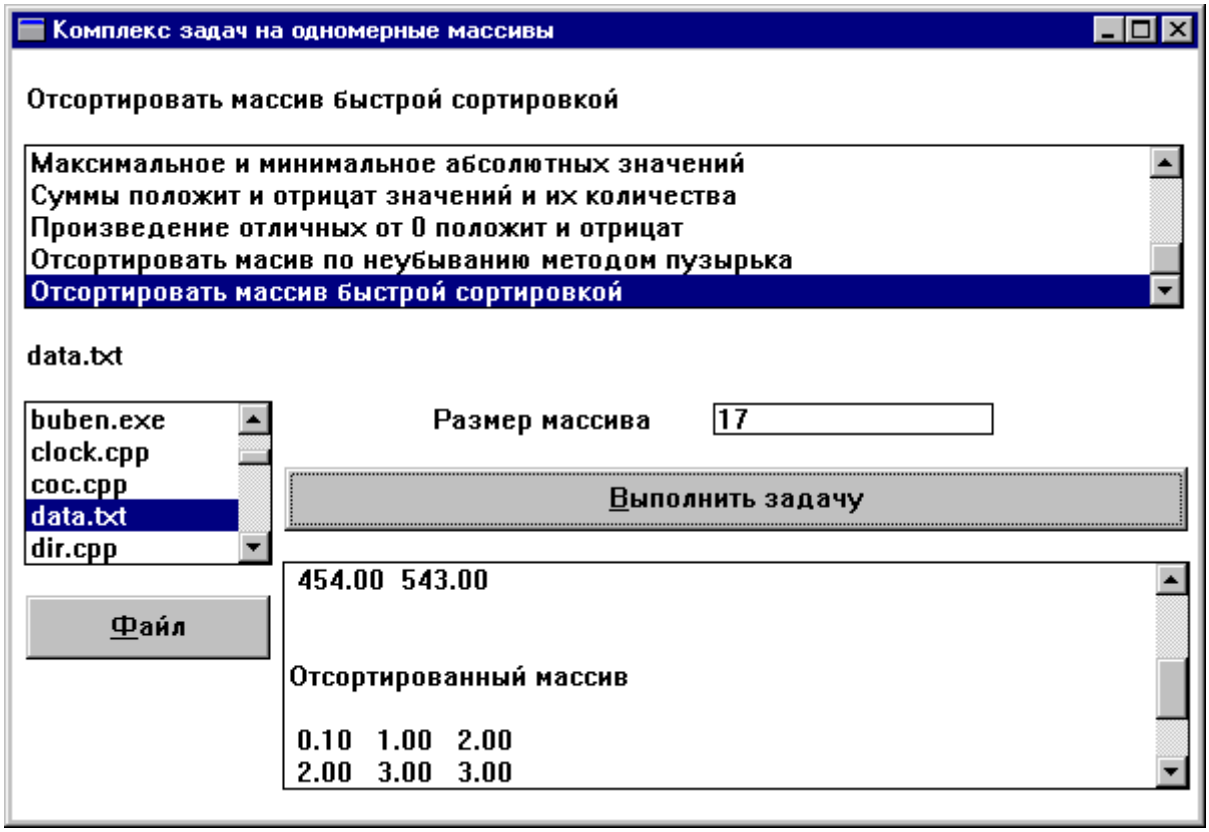
```
поставленных задач - и эту привязку мы осуществляем  организацией
массива структур с 2-мя полями - комментариями и указателями на
функции.*/

struct list_task
{
  char * str;   //Поле - указатель на строку названия задачи
  void (*func)(char*,unsigned);//Поле указатель на функцию решения
}task[]=
     /*Каждому полю мы сразу присваиваем значение, а для удобства
последующего вывода конец списка обозначаем нуль - указателями на
строку и на функцию*/
{
"Среднее арифметическое элементов массива",func0,
"Центрировать массив относительно среднего",func1,
"Среднее квадратов элементов центрированного массива",func2,
"Среднее поэлементного произведения\
2-х центрированных массивов",func3,
"Скалярное произведение 2-х векторов",func4,
"Модуль массива как вектора и направляющие косинусы",func5,
"Сумма и разность 2-х векторов",func6,
"Проекция 1-го вектора на направление 2-го",func7,
"Угол между 2-мя векторами в градусах",func8,
"Максимальное, минимальное значение и их индексы",func9,
"Максимальное и минимальное абсолютных значений",func10,
"Суммы положит и отрицат значений и их количества",func11,
"Произведение отличных от 0 положит и отрицат",func12,
"Отсортировать масив по неубыванию методом пузырька",func13,
"Отсортировать массив быстрой сортировкой",func14,
NULL,NULL
};


/*основная программа, принимающая четыре параметра -
идентификатор программы, идентификатор предыдущего
экземпляра программы, параметры командной строки и
режим отображения окна (нормальное, развёрнутое на \
весь экран, свёрнутое в пиктограмму и т.д.)*/
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR szCmdLine, int iCmdShow)
{
  static char szAppName[] = "FileList";/*имя класса окна*/
  HWND        hwnd ;                   /*идентификатор окна*/
  MSG         msg ;                    /*структура сообщения*/
  WNDCLASS    wndclass ;               /*структура класса окна*/
  ATOM        IsReg;                   /*для регистрации окна*/

  /*стиль окна определяем как перерисовываемое при изменении
    как горизонтального, так и вертикального его размеров*/
  wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc   = WndProc ; /*указатель на функцию окна*/
  /*дополнительных данных как для класса, так и для окна у нас не будет*/
  wndclass.cbClsExtra    = 0 ;
  wndclass.cbWndExtra    = 0 ;
  wndclass.hInstance     = hInstance ;/*идентификатор приложения*/
  /*загружаем стандартную иконку*/
  wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
  /*загружаем стандартный курсор в виде стрелочки*/
  wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
  /*цвет окна, 0 - окно прозрачно*/
  wndclass.hbrBackground = (HBRUSH)(1+COLOR_WINDOW);
  wndclass.lpszMenuName  = NULL ;/*меню у нас не будет*/
  wndclass.lpszClassName = szAppName ;/*имя класса окна*/

  IsReg=RegisterClass(&wndclass) ;/*пытаемся зарегистрировать класс*/
```

```
  if(!IsReg)/*если не удалось зарегистрироваться*/
  {
    /*вызываем стандартную функцию вывода диалогового окна с параметрами:
      идентификатор окна-родителя (0-если нет такового), сообщение,
      заголовок диалога, характеристики диалога (здесь - иконка + кнопка
OK)*/
    MessageBox (0, "Ошибка регистрации класса окна",
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return FALSE ;/*выходим из программы*/
  }
  /*пытаемся создать окно и получить его идентификатор*/
  hwnd = CreateWindow (szAppName,/*имя класса окна*/
                       "Комплекс задач на одномерные массивы",/*заголовок ок-
на*/
                       WS_OVERLAPPEDWINDOW,/*стиль окна - перекрывающееся*/
                       /*координаты x,y верхнего левого угла окна и его
                         размеры позволим определить ОС самостоятельно*/
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       NULL,/*идентификатор окна-предка, 0-если нет таково-
го*/
                       NULL,/*идентификатор меню, 0-если его нет*/
                       hInstance,/*идентификатор приложения*/
                       NULL) ;/*данные для создания окна*/
  if(!hwnd)/*если не удалось родить окно*/
  {
    /*вызываем стандартную функцию вывода диалогового окна с параметрами:
      идентификатор окна-родителя (0-если нет такового), сообщение,
      заголовок диалога, характеристики диалога (здесь - иконка + кнопка
OK)*/
    MessageBox (0, "Ошибка создания окна",
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return FALSE ;/*выходим из программы*/
  }
  /*после создания окна необходимо отобразить вначале его рамку и т.д.,*/
  ShowWindow (hwnd, iCmdShow) ;
  /*а лишь затем инициализировать прорисовку окна*/
  UpdateWindow (hwnd) ;
  hInst=hInstance;/*копируем в глобальную переменную идентификатор програм-
мы*/
  /*запускаем цикл обработки сообщений*/
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;/*трансляция клавиатурных сообщений -
                                    можно в данном случае и не делать*/
    DispatchMessage (&msg) ; /*диспетчеризация сообщений по функциям*/
  }
  return msg.wParam ;        /*чем плох такой код возврата?*/
}



/*функция окна принимает 4 параметра:
  1. идентификатор окна
  2. сообщение окну
  3. слово
  4. двойное слово - дополнительные данные для сообщения*/
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  /*набор идентификаторов дочерних окон*/
  static HWND hwndList1, hwndText1,hwndList2, hwndText2,
              hFileButton, hStartButton, hwndEdit, hwndStEdit;
  HDC         hdc ;/*контекст окна*/
  int         i ;
  TEXTMETRIC  tm ; /*структура для свойств текста */
  char szBuffer[128];/*промежуточный буфер*/
```

```
  switch (iMsg)/*обработка сообщений*/
  {
    case WM_CREATE :/*при создании окна*/
    {
      hdc = GetDC (hwnd) ;/*получаем контекст окна*/
      GetTextMetrics (hdc, &tm) ;/*получаем метрики текста для данного кон-
текста*/
      ReleaseDC (hwnd, hdc) ;/*освобождаем контекст окна*/

      /*создаём окно списка задач*/
      hwndList1 = CreateWindow
        ("listbox",/*имя класса окна*/
        NULL,/*заголовка не будет*/
        /*стиль списка - дочернее окно, видимый, стандартный,
          но не сортируемый*/ WS_CHILD | WS_VISIBLE |
(LBS_STANDARD&~LBS_SORT) ,
        tm.tmAveCharWidth, tm.tmHeight * 3,
        tm.tmAveCharWidth * 80 + GetSystemMetrics (SM_CXVSCROLL),
        tm.tmHeight * 5,/*размеры списка, привязанные к размерам шрифта*/
        hwnd,/*папа*/ (HMENU) TaskList,/*идентификатор этого окна списка*/
        hInst,/*идентификатор программы*/NULL) ;
      /*текстовая строка с условием задачи*/
      hwndText1 = CreateWindow(
        "static",/*имя класса окна*/
        NULL,/*пока без текста в строке*/
        /*строка - видимое дочернее окно с выравниванием текста по левому
краю*/
        WS_CHILD | WS_VISIBLE | SS_LEFT ,
        tm.tmAveCharWidth,  tm.tmHeight, tm.tmAveCharWidth * 200,
tm.tmHeight,
        hwnd,/*папа*/ (HMENU) TaskStr,/*идентификатор ребёнка*/
        hInst,/*идентификатор программы*/ NULL) ;
      /*список, но сортируемый (для файлов)*/
      hwndList2 = CreateWindow ("listbox", NULL, WS_CHILD | WS_VISIBLE |
LBS_STANDARD ,
        tm.tmAveCharWidth, tm.tmHeight * 11,
        tm.tmAveCharWidth * 15 + GetSystemMetrics (SM_CXVSCROLL),
        tm.tmHeight * 5, hwnd, (HMENU)FileList, hInst, NULL) ;
      /*строка с имкнем файла*/
      hwndText2 = CreateWindow ("static", NULL, WS_CHILD | WS_VISIBLE |
SS_LEFT,
        tm.tmAveCharWidth, tm.tmHeight*9, tm.tmAveCharWidth * 200,
tm.tmHeight,
        hwnd, (HMENU) FileStr,hInst, NULL) ;
      /*кнопка для выбора файла*/
      hFileButton = CreateWindow("button", "&Файл", WS_CHILD | WS_VISIBLE |
BS_PUSHBUTTON,
         tm.tmAveCharWidth, tm.tmHeight * 17, tm.tmAveCharWidth * 15 +
         GetSystemMetrics (SM_CXVSCROLL), tm.tmHeight * 2,
         hwnd, (HMENU) FileButton, hInst, NULL);
      /*кнопка для выполнения задачи*/
      hStartButton = CreateWindow("button", "&Выполнить задачу",
         WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, tm.tmAveCharWidth * 17 +
         GetSystemMetrics (SM_CXVSCROLL), tm.tmHeight * 13, tm.tmAveCharWidth
* 62 +
         GetSystemMetrics (SM_CXVSCROLL), tm.tmHeight * 2,
         hwnd, (HMENU) StartButton, hInst, NULL);
      /*общий список - не сортируемы*/
      hCommonWindow = CreateWindow ("listbox", NULL,  WS_CHILD | WS_VISIBLE |
        (LBS_STANDARD&~LBS_SORT), tm.tmAveCharWidth * 17 + GetSystemMetrics
(SM_CXVSCROLL),
        tm.tmHeight * 16, tm.tmAveCharWidth * 62 + GetSystemMetrics
(SM_CXVSCROLL),
        tm.tmHeight * 7, hwnd, (HMENU) CommonWindow, hInst, NULL) ;
      /*строка с надписью*/
```

```
      hwndStEdit = CreateWindow ("static", "Размер массива", WS_CHILD |
WS_VISIBLE | SS_LEFT,
        tm.tmAveCharWidth*30, tm.tmHeight*11, tm.tmAveCharWidth * 20,
tm.tmHeight,
        hwnd, (HMENU) StEdit, hInst, NULL) ;
      /*строка ввода для размера массива*/
      hwndEdit = CreateWindow ("edit", NULL,
        WS_CHILD | WS_VISIBLE | SS_LEFT | WS_BORDER | WS_TABSTOP,
        tm.tmAveCharWidth*50, tm.tmHeight*11, tm.tmAveCharWidth * 20,
tm.tmHeight,
        hwnd, (HMENU) Edit, hInst, NULL) ;
      /*добавляем в список задач их условия из массива*/
      for (i = 0 ; task[i].str ; i++)
        ListBox_AddString(hwndList1, task[i].str) ;
      /*заполняем список файлов именами дисков, файлов и каталогов*/
      SendMessage(hwndList2, LB_DIR, DDL_READWRITE | DDL_READONLY |
DDL_HIDDEN |
        DDL_SYSTEM | DDL_DIRECTORY | DDL_DRIVES | DDL_ARCHIVE,
(LPARAM)(LPSTR)"*.*");
      return 0 ;/*сообщение обработано*/
    }
    /*Здесь будем обрабатывать сообщения от дочерних окон*/
    case WM_COMMAND :
    {
      if(wParam==TaskList && HIWORD (lParam) == LBN_SELCHANGE)/*выбран эле-
мент в списке задач*/
      {
        i = ListBox_GetCurSel(hwndList1) ;/*получаем номер выбранного элемен-
та,*/
        ListBox_GetText(hwndList1, i, szBuffer) ;/*его текст,*/
        Static_SetText(hwndText1, szBuffer) ;/*который заносим в строку*/
      }
      // Сообщение от кнопки запуска задачи
      if(wParam == StartButton)
      {
        /*получаем размер массива как текстовую строку из окна редактирова-
ния*/
        Edit_GetText(hwndEdit,szBuffer,7);
        int n=atoi(szBuffer);/*преобразуем строку в число*/
        if(n<=0)/*проверяем число на корректность*/
        {
           /*если что-то не так - ругаемся*/
           MessageBox (hwnd, "Некорректный размер массива",
                "Что-то не в порядке", MB_ICONEXCLAMATION | MB_OK) ;
           return 0;/*и выходим*/
        }
        /*если хотя бы одна из строк с условием задачи или именем файла пу-
стая*/

if(!Static_GetText(hwndText1,szBuffer,2)||!Static_GetText(hwndText2,szBuffer,
2))
           MessageBox (hwnd, "Файл или задача не выбраны",/*ругаемся*/
                "Что-то не в порядке", MB_ICONEXCLAMATION | MB_OK) ;
        else
        {
          i=ListBox_GetCurSel(hwndList1) ;//номер задачи
          Static_GetText(hwndText2,szBuffer,15);          //имя файла
          task[i].func(szBuffer,n);/*вызываем функцию для данной задачи*/
        }
        return 0;
      }
      if(wParam == FileButton)/*кнопка "Файл"*/
      {
        // Определяем номер выделенной строки
        i = ListBox_GetCurSel(hwndList2);
        if(i==LB_ERR)/*если строка не выделена*/
```

```
          return 0;
        // Получаем выделенную строку
        ListBox_GetText(hwndList2, i, szBuffer);
        // Если выбрали имя каталога или диска, пытаемся изменить сначала те-
кущий
        // каталог, а затем текущий диск
        if(szBuffer[0] == '[')
        {
          // Выделяем в выбранной строке имя каталога
          szBuffer[lstrlen(szBuffer) - 1] = '\0';
          // Пытаемся изменить каталог
          if(chdir(szBuffer+1) != 0)
          {
            // Если не удалось, значит выбрали имя диска. В этом случае
            // выделяем букву имени диска и добавляем строку ":\\:
            szBuffer[3] = '\0';
            lstrcat(szBuffer, ":\\");
            // Изменяем текущий диск
            if(chdir(szBuffer+2) == 0)
            {
              // Преобразуем букву диска в  номер диска
              AnsiLowerBuff(szBuffer+2, 1);
              // Устанавливаем новый диск
              setdisk(szBuffer[2] - 'a');
            }
          }
          // Сбрасываем содержимое списка файлов
          ListBox_ResetContent(hwndList2);
          // Заполняем список информацией о файлах и каталогах в текущем
          // каталоге, а также вносим туда имена дисков
          SendMessage(hwndList2, LB_DIR, DDL_READWRITE | DDL_READONLY  |
DDL_HIDDEN |
            DDL_SYSTEM    | DDL_DIRECTORY | DDL_DRIVES | DDL_ARCHIVE,
(LPARAM)(LPSTR)"*.*");
        }
        // Если выбрали файл, выводим его имя в соответствующую строку
        else
          Static_SetText(hwndText2, szBuffer) ;
      }
      /*сообщение от списка файлов*/
      if(wParam == FileList)
        // Двойной щелчок по имени файла, каталога или диска
        if(HIWORD(lParam) == LBN_DBLCLK)
          // Имитируем приход сообщения от кнопки
          SendMessage(hwnd, WM_COMMAND, FileButton, 0L);
      return 0 ;/*сообщение обработано*/
    }
    case WM_DESTROY :/*при закрытии окна*/
    {
      PostQuitMessage (0) ;      /*помещаем в очередь сообщений окна
                                       посмертное сообщение, прерывающее
                                   цикл обработки сообщений*/
      return 0 ;/*сообщение обработано*/
    }
  }
  /*для всех необработанных сообщений вызываем функцию обработки сообщений по
умолчанию*/
  return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
```


Комплекс задач на одномерные массивы
Отсортировать массив быстрой сортировкой
Максимальное и минимальное абсолютных значений
Суммы положит и отрицат значений и их количества
Произведение отличных от 0 положит и отрицат
Отсортировать масив по неубыванию методом пузырька
Отсортировать массив быстрой сортировкой
data.txt
buben.exe
clock.cpp
coc.cpp
data.txt
dir.cpp
Файл
Размер массива
17
Выполнить задачу
454.00 543.00
Отсортированный массив
0.10 1.00 2.00
2.00 3.00 3.00

Программа, интерпретирующая заданную параметрически формулу и строющая по ней график

```
#define STRICT//для строгой проверки типов
#include <windowsx.h>/*файл макросов для облегчения работы*/
#include <stdlib.h>
#include <string.h>
#include <dir.h>
#include <stdio.h>
#include <alloc.h>
#include <ctype.h>
#include <math.h>

     /*Для хранения кода класса лексемы выделим  глобальную
переменную token_type, а для кода ее конкретного значения - tok*/

const MAX_POINT=50; //Количество вычисляемых точек для графика
double t[MAX_POINT];     //массив значений параметра
//таблица для значений x,y в натуральных единицах
double xy[MAX_POINT][2];

int token_type,tok;
char token[10]; //для хранения выделенной лексемы

//Коды классов лексем
const OPERAND=1,OPERATION=2,SKOBKA=3,EOL=4;

//Подклассы операндов
const STRING=1,  //просто неклассифицированная еще строка
      NUMBER=2,  //числовая строка
      VARIABLE=3,//переменная
      CONSTANT=4,//константа
      EXPRESS =5;//выражение в скобках

//шаблон структуры констант
struct cnst
{
  char cn[6];
  double cv;
}
//константный массив структур
tc[2]=
{
  {"PI",M_PI},
  {"E",M_E}
};

//Коды операций
const
  MUL=1,DIV=2,POW=3,PLUS=4,MINUS=5,       //Арифметика
  ABS=6, ACOS=7,ASIN=8, ATAN=9, COS=10, COSH=11, EXP=12, LOG=13,
  LOG10=14, SIN=15, SINH=16, SQRT=17, TAN=18, TANH=19;

//Все обозначения (имена) операций и их коды сведем в таблицу
//(массив структур tf[]) по шаблону
struct
{
  char on[10];
  int ov;
}
op[]=
{
  {"ABS",ABS},{"ACOS",ACOS},{"ASIN",ASIN},{"ATAN",ATAN},
  {"COS",COS},{"COSH",COSH},{"EXP",EXP},{"LN",LOG},{"LOG",LOG10},
```

```
  {"SIN",SIN},{"SINH",SINH},{"SQRT",SQRT},{"TAN",TAN},
  {"TANH",TANH},{"+",PLUS},{"-",MINUS},{"*",MUL},{"/",DIV},
  {"^",POW},{"",0}
};

char *OP="+-*/^";    //Перечень арифметических операций

//для пределов переменных
float tmin,tmax,xmin,xmax,ymin,ymax;

char formula[2][80]; //Для текста формул и пределов параметра

//НЕКОТОРЫЕ СЛУЖЕБНЫЕ ПОДПРОГРАММЫ
//Ф-ция приведения латинских букв к верхнему регистру
void  touppercase(char *f)
{
  for(char *temp=f;*temp;temp++)
    if(islower(*temp))
      *temp=toupper(*temp);
}


     /*Ф-ция поиска по таблицам для  определения  типа  строковых
лексем параметры ф-ции - искомая  строка  и  адрес,  по  которому
положить значение обнаруженной именованной константы*/

void look_up(char *s,double* cv)
{
  int i;
  //Просмотрим таблицу операций
  for(i=0;strlen(op[i].on);i++)
    //Если нашли совпадение с допустимым именем операции
    //возвращаем конкретный тип операции
    if(!strcmp(op[i].on,s))
    {
      token_type=OPERATION;
      tok=op[i].ov;
      return;
    }

  //Поиск в таблице констант
  for(i=0;strlen(tc[i].cn);i++)
  /*Если нашли совпадение с допустимым именем константы
    возвращаем  индекс константы в массиве структур и ее значение
    записываем по заданному адресу*/
    if(!strcmp(tc[i].cn,s))
    {
      token_type=OPERAND;
      tok=CONSTANT;
      *cv=tc[i].cv;
      return;
    }
  //Если ничего не нашли
  token_type=0;
  tok=-1;
}


//Функция очистки формульной строки от излишков и неточностей
void clearformula(char *f)
{
  for(char *temp=f;*temp;)
    if(isalnum(*temp) || strchr(OP,*temp)
       || *temp=='.' || *temp=='(' || *temp==')')
      temp++;
      //Вначале удалим все недопустимые в формуле символы
```

```
     else
       memmove(temp,temp+1,strlen(temp+1)+1);
  //Затем несколько раз на наличие сочетаний типа "+*" или "-*"
  for(temp=f+1;*temp!='\0';temp++)
    if((*temp=='*' || *temp=='/' || *temp=='^') &&
      (*(temp-1)=='+' || *(temp-1)=='-'))
    {
      memmove(temp,temp+1,strlen(temp+1)+1);
      temp=f;
    }
}


//Ф-ция для вывода сообщения об ошибке и  возврата  к  вводу
//пользователя

void serror(int error)
{
  static char *e[]=
  {
    "Синтаксическая ошибка",
    "Непарные круглые скобки",
    "Где-то не выражение",
    "Где-то не переменная",
    "Есть нераспознанная операция",
    "Не распознанное имя константы",
    "Получается деление на нуль"
  };
  MessageBox(0,e[error],"",MB_OK|MB_ICONASTERISK);
  exit(0);
}
```

/*Функция выделения и классификации очередной лексемы возвращает класс лексемы. Ее параметры - указатель на указатель текущего текста формулы и адрес для значения операнда-константы - он транзитом передается в look_up(). Первый параметр мы вынуждены получать "по ссылкее", чтобы значение адреса текущей лексемы изменялось в вызывающей программе, а не в нащей подпрограмме*/

```
int get_token(char **cf, double *cv)
{
  char *temp;    //Временный указатель на лексему
  token_type=0;tok=0;
  char*opr;

  //Если конец формулы
  if(**cf=='\0')
  {
    *token=0;
    return(token_type=EOL);
  }

  //Если это открытая круглая скобка
  if(**cf=='(')
  {
    temp=token;
    *temp=**cf;          //перепишем лексему в token
    (*cf)+=1;           // переход на следующую позицию
    temp++;
    *temp=0;    //token закроем нулем
    tok=EXPRESS;
    return(token_type=OPERAND);
  }

  /*Если это закрытая круглая скобка
```

```
  if(**cf==')')
  {//(*cf)+=1;
    temp=token;
    *temp=**cf;      //перепишем его в token
    (*cf)+=1;        // переход на следующую позицию
    temp++;
    *temp=0; //token закроем нулем
    tok=EXPRESS;
    return(token_type=SKOBKA);
  }
*/

  //Если это символ арифметической операции
  if((opr=strchr(OP,**cf))!=NULL)
  {
    temp=token;
    *temp=**cf;      //перепишем его в token
    (*cf)+=1;        // переход на следующую позицию
    temp++;
    *temp=0; //token закроем нулем
    switch(*opr)
    {
      case '+':
        tok=PLUS;
        break;
      case '-':
        tok=MINUS;
        break;
      case '*':
        tok=MUL;
        break;
      case '/':
        tok=DIV;
        break;
      case '^':
        tok=POW;
        break;
    }
    //сообщим что класс лексемы - операция
    return (token_type=OPERATION);
  }

  //Если встретили цифру
  if(isdigit(**cf))
  {
    //то запишем всю числовую подстроку в token
    temp=token;
    while(isdigit(**cf) || **cf=='.')
    {
      *(temp++)=**cf;
      (*cf)+=1;
    }
    *temp = '\0';
    tok=NUMBER;
    return(token_type = OPERAND);
  }

  //Если встретили букву
  if(isalpha(**cf))
  {
     /*то это переменная или операция - функция  или  именованная
       константа;  пока  все  буквы  -  цифры  перепишем  и  временно
       зарегистрируем просто строкой*/
    temp=token;
    while(isalnum(**cf))
    {
```

```
      *temp++=**cf;
      (*cf)+=1;
    }
    token_type=STRING;
  }
  *temp = '\0';

  //Не отходя далеко проанализируем полученную строку

  //Если это 1-буквенное имя независимой переменной
  if(token_type==STRING && !strcmp(token,"T"))
  {
    tok=VARIABLE;
    return(token_type=OPERAND);
  }

  //В противном случае поищем среди унарных операций и констант
  look_up(token,cv);
  if(tok<0)
    serror(4);
}
```

/* Нам понадобятся некоторые вспомогательные функции, в частности функция выполнения заданных двухместных математических операций - ее параметры при вызове - это код требуемой двухместной операции и значения левого и правого операндов.*/

```
double oper(int co,double lo, double ro)
{
  switch(co)
  {
    case PLUS:
      return(lo+ro);
    case MINUS:
      return(lo-ro);
    case MUL:
      return(lo*ro);
    case DIV:
    {
      if(ro==0)
        serror(6);
      else
        return(lo/ro);
    }
    case POW:
      return(pow(lo,ro));
    default:
      serror(0);
  }
}
```

/*Эта функция осуществляет унарные операции, к классу которых мы отнесли помимо унарных "плюс" и "минус" все математические функции, полагая находящееся в скобках после имени функции выражение единственным их операндом. Для сокращения дефицитного места в книге мы привели реализации всего нескольких математических функций, а вы смжете расширить эти возможности по аналогии */

```
double unary(int co,double ro)
{
  switch(co)
  {
    case PLUS:
```

```
      return(ro);
    case MINUS:
      return(-ro);
    case SIN:
      return sin(ro);
    case COS:
      return cos(ro);
    case TAN:
      return tan(ro);
    case SQRT:
      return sqrt(ro);
    default:
      serror(0);
  }
}
```

/*Это уже основная подпрограмма вычисления значений по тексту формулы, она возвращает вычисленное значение, а ее параметры - указатель на текст формулы и значение независимой переменной*/

```
double get_exp(unsigned char* f, double t)
{
  int sc,                //Счетчик скобок
      i,j,               //рабочие переменные для циклов
      co;                //для кода операции
  double cv;             //для значения именованной константы
  double result;         //для промежуточных результатов вычислений
  char privat_form[80];  //для текущего фрагмента формулы
```

/* Формула состоит из операндов и операций. Типы операндов - число, именованная константа, переменная, выражение в скобках.*/

```
  struct
  {               //Для сруктуры формулы
    unsigned dtok,         //Код типа операнда
    co;                    //Код операции
    double z;              //значение операнда
  }frm[32];

  for(i=0;i<32;i++)
    memset(&frm[i],0,sizeof(frm[i]));//Обнулим
  unsigned cnt=0;     //Количество элементов в формуле
  char* tmp=f; //Исходный адрес формулы

  //Просмотрим текст формулы и заполним ее структуру
  for(i=0,get_token(&tmp,&cv);token_type!=EOL;i++,get_token(&tmp,&cv))
  {
    //Если получили операнд - число, константу или переменную
    if(token_type==OPERAND)
    {
      frm[i].dtok=tok;
      if(tok==NUMBER)
        frm[i].z=atof(token);
      if(tok==CONSTANT)
        frm[i].z=cv;
      if(tok==VARIABLE)
        frm[i].z=t;
      if(tok==EXPRESS)     //Если операнд - выражение в скобках
      {
        int tt=token_type;//Сохраним тип лексемы
        sc=1;
```

/*Перепишем все после нее до парной ей закрытой в массив privat_form создавая таким образом другую, частную формулу, как фрагмент общей - для нее мы можем использовать такой же механизм

```
      исследования*/
      for(j=0;sc && j<78;j++,tmp++)
      {
        if(*tmp=='(')
          sc++;
        if(*tmp==')')
          sc--;
        privat_form[j]=*tmp;
      }
      privat_form[j-1]=0;//Закроем нулем
      if(sc)
        serror(1); //Если не нашли парную скобку
      result=get_exp(privat_form,t);
      token_type=tt;//Восстановим тип лексемы
      frm[i].dtok=NUMBER;
      frm[i].z=result;
    }//if EXPRESS
  }//if OPERAND
  //Если получили операцию
  if(token_type==OPERATION)
    frm[i].co=tok;
} //for
cnt=i;       //Количество заполненных элементов

          /*Теперь можем обрабатывать. Вначале необходимо выполнить
      все унарные операции - у них самый высокий приоритет. Признаком
      унарной операции является либо если она первая в структуре
      формулы, либо предыдущий в формуле элемент - операция, а
      последующий - операнд*/

int flag=1;
for(;flag;)
{
  flag=0;
  for(i=0;i<cnt;i++)
    if((!i && frm[i].co && frm[i+1].dtok) ||
      (i && frm[i-1].co && frm[i].co && frm[i+1].dtok))
    {
      frm[i+1].z=unary(frm[i].co,frm[i+1].z);
      //Сомкнем ряды на выполненной операции
      memmove(&frm[i],&frm[i+1],(cnt-i-1)*sizeof(frm[i]));
      cnt--;
      flag++;
    }
}
/*Теперь необходимо выполнить все возведения в степень */
for(i=1;i<cnt;i++)
  if(frm[i].co==POW && frm[i-1].dtok && frm[i+1].dtok)
  {
    frm[i-1].z=oper(POW,frm[i-1].z,frm[i+1].z);
    //Сомкнем ряды на выполненной операции
    memmove(&frm[i],&frm[i+2],(cnt-i-2)*sizeof(frm[i]));
    i--;
    cnt-=2;
  }
/*Теперь  выполним все деления и умножения */
for(i=1;i<cnt;i++)
  if((frm[i].co==MUL || frm[i].co==DIV) && frm[i-1].dtok && frm[i+1].dtok)
  {
    frm[i-1].z=oper(frm[i].co,frm[i-1].z,frm[i+1].z);
    //Сомкнем ряды на выполненной операции
    memmove(&frm[i],&frm[i+2],(cnt-i-2)*sizeof(frm[i]));
    i--;
    cnt-=2;
  }
  /*Теперь  выполним все сложения и вычитания */
```

```
  for(i=1;i<cnt;i++)
    if((frm[i].co==PLUS || frm[i].co==MINUS) && frm[i-1].dtok &&
frm[i+1].dtok)
    {
      frm[i-1].z=oper(frm[i].co,frm[i-1].z,frm[i+1].z);
        //Сомкнем ряды на выполненной операции
      memmove(&frm[i],&frm[i+2],(cnt-i-2)*sizeof(frm[i]));
      i--;
      cnt-=2;
    }
  //Теперь в 0-м элементе лежит результат
  return frm[0].z;
}


     /*Функция расчета элементов таблиц - использует
интерпретатор для вычисления значений x и y при различных
значениях параметра t.*/

void tablevar(void)
{
  int i,index;
    /*Вначале с помощью интерпретатора обрабатываются строки с
значениями пределов параметра t - так предоставляется возможность
использовать для задания пределов именованные константы типа PI,
E и любых других, специфичкеских для какой либо прикладной
области - это могут быть физические, химические константы и пр. */

  double incr=(tmax-tmin)/MAX_POINT;
  for(index=0;index<MAX_POINT;index++)
    t[index]=tmin+index*incr;
  for(i=0;i<2;i++)
    for(index=0;index<MAX_POINT;index++)
      xy[index][i]=get_exp(formula[i],t[index]);
}


/*Ф-ция вывода графика кривой,заданной массивом
  xy[RowCount][ColColnt], в графическое окно*/
void gshow(HWND hwnd)
{
  /*Для масштабов по x и y - количества пикселов на нату-
    ральные единицы*/
  double picx,picy;
  double dx,dy;    //Диапазоны переменных
  int i;
  HDC hdc;/*контекст окна */
  RECT r; /*структура для хранения размеров прямоугольной области*/
  PAINTSTRUCT ps;/*контекст+параметры рисования*/

  hdc = BeginPaint(hwnd,&ps) ;/*получаем контекст окна*/
  //определим макс знач x и y просмотром массива
  xmax=xmin=xy[0][0];
  ymax=ymin=xy[0][1];
  for(i=1;i<MAX_POINT;i++)
  {
    if(xy[i][0]>xmax)
      xmax=xy[i][0];
    if(xy[i][0]<xmin)
      xmin=xy[i][0];
    if(xy[i][1]>ymax)
      ymax=xy[i][1];
    if(xy[i][1]<ymin)
      ymin=xy[i][1];
  }
  dx=fabs(xmax-xmin);
```

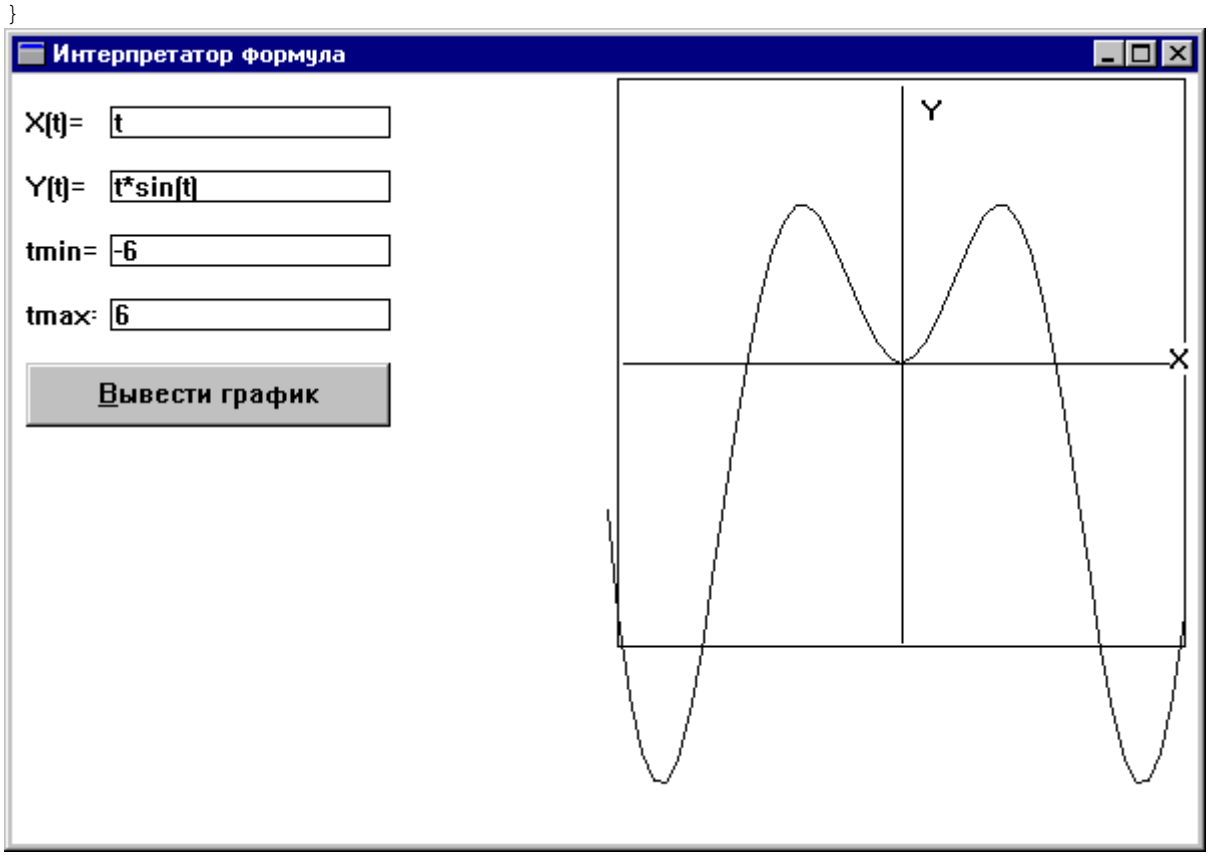
```
  dy=fabs(ymax-ymin);
  //Нормируем данные относительно диапазонов
  for(i=0;i<MAX_POINT;i++)
  {
     xy[i][0]/=dx;
     xy[i][1]/=dy;
  }

  //Определимся с размерами окна и масштабами
  GetClientRect(hwnd,&r);
  int scrheight=(3*r.bottom)>>2; //Высота окна
  int scrwidth=scrheight;  //Ширина окна - оно будет квадратным
  picy=picx=scrwidth;
  if(dx<dy)
    picx*=(dx/dy);
  if(dx>dy)
    picy*=(dy/dx);

  Rectangle(hdc,2+300,2,scrwidth-2+300,scrheight-2); //рамка в окне
  int xc=scrwidth>>1;
  int yc=scrheight>>1;
  //Координатные оси
  MoveTo(hdc,5+300,yc);
  LineTo(hdc,scrwidth-5+300,yc);
  MoveTo(hdc,xc+300,scrheight-5);
  LineTo(hdc,xc+300,5);
  TextOut(hdc,scrwidth-10+300,yc-10,"X",1);
  TextOut(hdc,xc+10+300,10,"Y",1);

  //Вывод кривой
  MoveTo(hdc,xy[0][0]*picx+xc+300,yc-xy[0][1]*picy);
  for(i=1;i<MAX_POINT;i++)
    LineTo(hdc,xy[i][0]*picx+xc+300,yc-xy[i][1]*picy);
  EndPaint(hwnd, &ps);/*возвращаем контекст*/
}


//прототип функции окна
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

HINSTANCE hInst;/*копия идентификатора программы*/

/*этот набор констант предназначен для идентификации дочерних окон*/
const
  XtSt=1,
  XtEdit=2,
  YtSt=3,
  YtEdit=4,
  tminSt=5,
  tminEdit=6,
  tmaxSt=7,
  tmaxEdit=8,
  DrawButton=9;


/*основная программа, принимающая четыре параметра -
идентификатор программы, идентификатор предыдущего
экземпляра программы, параметры командной строки и
режим отображения окна (нормальное, развёрнутое на \
весь экран, свёрнутое в пиктограмму и т.д.)*/
#pragma argsused
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR szCmdLine, int iCmdShow)
{
  static char szAppName[] = "ColorDraw";/*имя класса окна*/
  HWND        hwnd ;                  /*идентификатор окна*/
```

```
  MSG         msg ;                   /*структура сообщения*/
  WNDCLASS    wndclass ;              /*структура класса окна*/
  ATOM        IsReg;                  /*для регистрации окна*/

  /*стиль окна определяем как перерисовываемое при изменении
    как горизонтального, так и вертикального его размеров*/
  wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
  wndclass.lpfnWndProc   = WndProc ; /*указатель на функцию окна*/
  /*дополнительных данных как для класса, так и для окна у нас не будет*/
  wndclass.cbClsExtra    = 0 ;
  wndclass.cbWndExtra    = 0 ;
  wndclass.hInstance     = hInstance ;/*идентификатор приложения*/
  /*загружаем стандартную иконку*/
  wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
  /*загружаем стандартный курсор в виде стрелочки*/
  wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
  /*цвет окна, 0 - окно прозрачно*/
  wndclass.hbrBackground = (HBRUSH)(1+COLOR_WINDOW);
  wndclass.lpszMenuName  = NULL ;/*меню у нас не будет*/
  wndclass.lpszClassName = szAppName ;/*имя класса окна*/

  IsReg=RegisterClass(&wndclass) ;/*пытаемся зарегистрировать класс*/
  if(!IsReg)/*если не удалось зарегистрироваться*/
  {
    /*вызываем стандартную функцию вывода диалогового окна с параметрами:
      идентификатор окна-родителя (0-если нет такового), сообщение,
      заголовок диалога, характеристики диалога (здесь - иконка + кнопка
OK)*/
    MessageBox (0, "Ошибка регистрации класса окна",
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return FALSE ;/*выходим из программы*/
  }
  /*пытаемся создать окно и получить его идентификатор*/
  hwnd = CreateWindow (szAppName,/*имя класса окна*/
                       "Интерпретатор формула",/*заголовок окна*/
                       WS_OVERLAPPEDWINDOW,/*стиль окна - перекрывающееся*/
                       /*координаты x,y верхнего левого угла окна и его
                         размеры позволим определить ОС самостоятельно*/
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       NULL,/*идентификатор окна-предка, 0-если нет таково-
го*/
                       NULL,/*идентификатор меню, 0-если его нет*/
                       hInstance,/*идентификатор приложения*/
                       NULL) ;/*данные для создания окна*/
  if(!hwnd)/*если не удалось родить окно*/
  {
    /*вызываем стандартную функцию вывода диалогового окна с параметрами:
      идентификатор окна-родителя (0-если нет такового), сообщение,
      заголовок диалога, характеристики диалога (здесь - иконка + кнопка
OK)*/
    MessageBox (0, "Ошибка создания окна",
                szAppName, MB_ICONEXCLAMATION | MB_OK) ;
    return FALSE ;/*выходим из программы*/
  }
  /*после создания окна необходимо отобразить вначале его рамку и т.д.,*/
  ShowWindow (hwnd, iCmdShow) ;
  /*а лишь затем инициализировать прорисовку окна*/
  UpdateWindow (hwnd) ;
  hInst=hInstance;/*копируем в глобальную переменную идентификатор програм-
мы*/
  /*запускаем цикл обработки сообщений*/
  while (GetMessage (&msg, NULL, 0, 0))
  {
    TranslateMessage (&msg) ;/*трансляция клавиатурных сообщений -
                                  можно в данном случае и не делать*/
```

```
    DispatchMessage (&msg) ; /*диспетчеризация сообщений по функциям*/
  }
  return msg.wParam ;        /*чем плох такой код возврата?*/
}



/*функция окна принимает 4 параметра:
  1. идентификатор окна
  2. сообщение окну
  3. слово
  4. двойное слово - дополнительные данные для сообщения*/
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
  /*набор идентификаторов дочерних окон*/
  static HWND
    hXtSt,  hXtEdit, hYtSt, hYtEdit, htminSt,
    htminEdit, htmaxSt, htmaxEdit, hDrawButton;
  HDC         hdc ;/*контекст окна*/
  int         i,len ;
  TEXTMETRIC  tm ; /*структура для свойств текста */
  char szBuffer[128];/*промежуточный буфер*/

  switch (iMsg)/*обработка сообщений*/
  {
    case WM_CREATE :/*при создании окна*/
    {
      /*по умолчанию графиком будет такая линия*/
      for(xy[0][0]=xy[0][1]=i=1;i<MAX_POINT;i++)
        xy[i][0]=xy[i][1]=0;
      hdc = GetDC (hwnd) ;/*получаем контекст окна*/
      GetTextMetrics (hdc, &tm) ;/*получаем метрики текста для данного кон-
текста*/
      ReleaseDC (hwnd, hdc) ;/*освобождаем контекст окна*/

      /*строка с надписью*/
      hXtSt = CreateWindow ("static", "X(t)=", WS_CHILD | WS_VISIBLE |
SS_LEFT,
        tm.tmAveCharWidth, tm.tmHeight, tm.tmAveCharWidth * 5, tm.tmHeight,
        hwnd, (HMENU) XtSt, hInst, NULL) ;
      /*окно редактирования*/
      hXtEdit = CreateWindow ("edit", NULL,
        WS_CHILD | WS_VISIBLE | ES_LEFT | WS_BORDER | ES_AUTOHSCROLL ,
        tm.tmAveCharWidth*7, tm.tmHeight, tm.tmAveCharWidth * 20,
tm.tmHeight,
        hwnd, (HMENU) XtEdit, hInst, NULL) ;
      hYtSt = CreateWindow ("static", "Y(t)=", WS_CHILD | WS_VISIBLE |
SS_LEFT,
        tm.tmAveCharWidth, tm.tmHeight*3, tm.tmAveCharWidth * 5, tm.tmHeight,
        hwnd, (HMENU) YtSt, hInst, NULL) ;
      hYtEdit = CreateWindow ("edit", NULL,
        WS_CHILD | WS_VISIBLE | ES_LEFT | WS_BORDER | ES_AUTOHSCROLL ,
        tm.tmAveCharWidth*7, tm.tmHeight*3, tm.tmAveCharWidth * 20,
tm.tmHeight,
        hwnd, (HMENU) YtEdit, hInst, NULL) ;
      htminSt = CreateWindow ("static", "tmin=", WS_CHILD | WS_VISIBLE |
SS_LEFT,
        tm.tmAveCharWidth, tm.tmHeight*5, tm.tmAveCharWidth * 5, tm.tmHeight,
        hwnd, (HMENU) tminSt, hInst, NULL) ;
      htminEdit = CreateWindow ("edit", NULL,
        WS_CHILD | WS_VISIBLE | ES_LEFT | WS_BORDER | ES_AUTOHSCROLL ,
        tm.tmAveCharWidth*7, tm.tmHeight*5, tm.tmAveCharWidth * 20,
tm.tmHeight,
        hwnd, (HMENU) tminEdit, hInst, NULL) ;
      htmaxSt = CreateWindow ("static", "tmax=", WS_CHILD | WS_VISIBLE |
SS_LEFT,
```

```
        tm.tmAveCharWidth, tm.tmHeight*7, tm.tmAveCharWidth * 5, tm.tmHeight,
        hwnd, (HMENU) tmaxSt, hInst, NULL) ;
      htmaxEdit = CreateWindow ("edit", NULL,
        WS_CHILD | WS_VISIBLE | ES_LEFT | WS_BORDER | ES_AUTOHSCROLL,
        tm.tmAveCharWidth*7, tm.tmHeight*7, tm.tmAveCharWidth * 20,
tm.tmHeight,
        hwnd, (HMENU) tmaxEdit, hInst, NULL) ;
      /*кнопка, по которой обновляется график*/
      hDrawButton = CreateWindow("button", "&Вывести график",
         WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
         tm.tmAveCharWidth, tm.tmHeight * 9, tm.tmAveCharWidth * 26 ,
tm.tmHeight * 2,
         hwnd, (HMENU) DrawButton, hInst, NULL);
      return 0 ;/*сообщение обработано*/
    }
    /*Здесь будем обрабатывать сообщения от дочерних окон*/
    case WM_COMMAND :
    {
      // Сообщение от кнопки запуска интерпретатора
      if(wParam == DrawButton)
      {
        /*получаем формулы как текстовые строки из окон редактирования*/
              Edit_GetText(hXtEdit,formula[0],79);
        Edit_GetText(hYtEdit,formula[1],79);
        /*получаем границы*/
        Edit_GetText(htminEdit,szBuffer,79);
        tmin=atof(szBuffer);/*преобразуем строку в число*/
        Edit_GetText(htmaxEdit,szBuffer,79);
        tmax=atof(szBuffer);/*преобразуем строку в число*/
          if(tmax<=tmin)/*проверяем границы на корректность*/
        {
           /*если что-то не так - ругаемся*/
           MessageBox (hwnd, "Некорректные минимум или максимум",
                    "Что-то не в порядке", MB_ICONEXCLAMATION | MB_OK) ;
             return 0;/*и выходим*/
        }
        /*если хотя бы одна из строк с формулой пустая*/
        if(!lstrlen(formula[0])||!lstrlen(formula[1]))
        {
           MessageBox (hwnd, "X(t) или Y(t) не введены",/*ругаемся*/
                "Что-то не в порядке", MB_ICONEXCLAMATION | MB_OK) ;
           return 0;
        }
        //Приведение всех строк к одному регистру и очистка
        for(i=0;i<2;i++)
        {
          touppercase(formula[i]);
          clearformula(formula[i]);
        }
        /*функция расчета таблиц, использующая синтаксический анализатор
          текста формулы*/
        tablevar();
        InvalidateRect(hwnd,0,TRUE);/*перерисовываем график*/
        return 0;
      }
    }
    /*это сообщение приходит при изменении содержимого окна редактирования*/
    case EN_UPDATE:
    {
      /*получаем текст из строки ввода*/
      len=Edit_GetText((HWND)LOWORD(lParam),szBuffer,80);
      if(len)/*если таковой имеется*/
        /*проходимся по всем символам,*/
        for(i=0;i<lstrlen(szBuffer);i++)
          /*отбрасывая некорректные*/
           if(!isalnum(szBuffer[i])&&!strchr("*+-/^.()",szBuffer[i]))
```

```
            {
              memmove(szBuffer+i,szBuffer+i+1,len);
              /*устанавливаем корректный текст в то же окно*/
              Edit_SetText((HWND)LOWORD(lParam),szBuffer);
              /*и сердито бибикаем*/
                  MessageBeep(-1);
            }
        return 0;/*обработано*/
      }
      /*сообщение о том, что пора бы окно перерисовать*/
      case WM_PAINT:
      {
        gshow(hwnd);/*перерисовываем график*/
        return 0;/*обработано*/
      }
      case WM_DESTROY :/*при закрытии окна*/
      {
        PostQuitMessage (0) ;      /*помещаем в очередь сообщений окна
                                        посмертное сообщение, прерывающее
                                    цикл обработки сообщений*/
        return 0 ;/*сообщение обработано*/
      }
    }
    /*для всех необработанных сообщений вызываем функцию обработки сообщений по
умолчанию*/
    return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
}
```



Литература

1. Чарльз Петцольд. Программирование в среде Windows.- К.:BHV? 1997/
2. Фролов А.В., Фролов Г.В. Библиотека системного программиста, т. 11-14. – М.: Диалог-МИФИ, 1994.
3. Хонекамп Д., Вилькен П. Программирование под Windows.- М.:ECOM, 1996.