Міністерство освіти України
Криворізький державний педагогічний інститут
Кафедра інформатики та прикладної математики

О.П. Поліщук

Лабораторний практикум з інформатики

Синтаксичний аналіз та інтерпретація математичних виразів. Організація баз даних засобами мови Паскаль

Кривий Ріг

1998

Поліщук О.П. Лабораторний практикум з інформатики. Синтаксичний аналіз та інтерпретація математичних виразів. Організація баз даних засобами мови Паскаль. – Кривий Ріг, КДПІ, 1998. – 48 с.

Автор: Поліщук Олександр Павлович, кандидат технічних наук, старший науковий співробітник, доцент кафедри інформатики та прикладної математики (КДПІ)

Рецензенти:

Соловйов В.М. – доктор фізико-математич-них наук, професор (КДПІ)

Теплицький І.О. – заступник директора з наукової роботи, вчитель-методист (Центрально-Міська гімназія, м. Кривий Ріг)

Рекомендовано до друку на засіданні кафедри інформатики та прикладної математики КДПІ, протокол №2 від 17.09.98 р.

Содержание:

1. Синтаксический анализ выражений (формул), заданных символьной строкой.

Эта задача является вспомогательной в программах интерпретации формул, заданных пользователем в виде символьных строк, в частности:

- ◊ при построении графиков функций, задаваемых пользователем вводом с клавиатуры;
- ◊ при выполнении расчетов по часто и непредсказуемо изменяющимся формулам - например, при начислении налогов, в программах-калькуляторах и пр.

В общем случае в формуле могут встретиться 2 класса лексем - операции и операнды и дополнительные промежуточные классы, например "конец формулы", "строка", "выражение в скобках". К операциям мы отнесем не только очевидный набор арифметических символов (+,-,^,*,/), но и все функции, которые распознает и выполняет наш интерпретатор (например sin(выражение), cos(выражение) и пр.

Классы можем закодировать например так:
const OPERAND=1;OPERATION=2;SKOBKA=3; EOL=4;

Эти значения будем использовать для инициализации глобальной переменной token_type.

Допустимые для использования в формулах операции (подклассы класса OPERATION) тоже закодируем, чтобы ускорить операции обработки (это называется переводом во внутренний формат):

```
{Коды операций}
const
MUL=1;LDIV=2;POW=3;PLUS=4;MINUS=5;    {Арифметика}
LABS=6;  ACOS=7;ASIN=8;  ATAN=9;  LCOS=10;  COSH=11;
LEXP=12;   LOG=13;   LOG10=14;   LSIN=15;   SINH=16;
LSQRT=17; TAN=18; TANH=19;
```

Все обозначения (имена) операций и их коды сведем в таблицу (массив структур tf[]) по шаблону

```
type
```

```
 loperation=record
 on:array [0..9] of char; ov:integer;
 end;

const
 op:array [0..20] of loperation=
(
(on:'ABS';ov:LABS),              (on:'ACOS';ov:ACOS),
(on:'ASIN';ov:ASIN),             (on:'ATAN';ov:ATAN),
(on:'COS';ov:LCOS),              (on:'COSH';ov:COSH),
(on:'EXP';ov:LEXP),                 (on:'LN';ov:LOG),
(on:'LOG';ov:LOG10),              (on:'SIN';ov:LSIN),
(on:'SINH';ov:SINH),            (on:'SQRT';ov:LSQRT),
(on:'TAN';ov:TAN),               (on:'TANH';ov:TANH),
(on:'+';ov:PLUS),                  (on:'-';ov:MINUS),
(on:'*';ov:MUL),  (on:'/';ov:LDIV),  (on:'^';ov:POW),
(on:'';ov:0));
```

причем в поля op[i].on занесли имена операций, а в поля op[i].ov значения их числовых кодов.

К операндам (подклассы класса OPERAND) отнесем переменные, именованные константы, непосредственно заданные числа и закодируем:

```
const LSTRING=1; {просто неклассифицированная еще строка}
      NUMBER=2; {числовая строка}
      VARIABLE=3; {переменная}
      CONSTANT=4; {константа}
      EXPRESS =5; {выражение в скобках}
```

Значения кодов подклассов будем заносить в глобальную переменную tok.

Будем предполагать, что допустимые для использования в формулах именованные константы сведены в константный массив структур вида:

```
type cnst=record
 cn:array [0..5] of char;
 cv:real;
end;   шаблон структуры
```

```
const
 c:array [0..2] of cnst=(
 (cn:'PI';cv:3.14159265358979323846),
(cn:'E';cv:2.71828182845904523536),
 (cn:'';cv:0)
 );константный массив структур
```

При этом в поля c[i].cn записаны имена констант, а в поля c[i].cv - их числовые коды.

Предварительная обработка формульной строки может осуществляться еще при клавиатурном вводе фильтрацией недопустимых символов, а если нет уверенности, что это выполнено достаточно эффективно, то необходимо в простейшем случае удалить из нее все недопустимые символы - пробелы, символы табуляции, возврата каретки, перевода строки, всевозможные разделители типа запятой или двоеточия, буквы нелатинского алфавита, оставив только предусмотренные к распознаванию знаки.

Кроме того, необходимо перед анализом привести все буквенные символы к верхнему регистру.

Таким образом, первое, что нам понадобится - это инструмент для выделения и классификации лексем (неделимых единиц формульного языка) из текста формулы. Остальные разъяснения алгоритма синтаксического анализа формульной строки мы дадим в комментариях к отдельным строкам приводимого ниже текста программы.*)

```
uses strings,crt;

(*служебные типы*)
type
  preal=^real;
  PPChar=^PChar;(*указатель на массив ASCIIZ-строк*)

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

var
  token_type,tok:integer;
```

```
  token:array [0..9] of char; (*для хранения выде-
ленной лексемы*)

(*Коды классов лексем*)
const OPERAND=1;OPERATION=2;SKOBKA=3;EOL=4;

(*Подклассы операндов*)
const LSTRING=1;   (*просто неклассифицированная еще
строка*)
       NUMBER=2;   (*числовая строка*)
       VARIABLE=3; (*переменная*)
       CONSTANT=4; (*константа*)
       EXPRESS =5; (*выражение в скобках*)
  WasError:boolean=false;     (*индикатор ошибки при
синтаксическом разборе позволяет быстро его закон-
чить и начать новый без выхода из программы, вернув-
шись к вводу данных*)

type cnst=record
  cn:array [0..5] of char;
  cv:real;
end;  (*   шаблон структуры*)

const
  tc:array [0..2] of cnst=(
  (cn:'PI';cv:3.14159265358979323846),
  (cn:'E';cv:2.71828182845904523536),
  (cn:'';cv:0)
 );(*константный массив структур  *)

{Коды операций}
const
MUL=1;LDIV=2;POW=3;PLUS=4;MINUS=5;   {Арифметика}
LABS=6;  ACOS=7;ASIN=8;  ATAN=9;  LCOS=10;  COSH=11;
LEXP=12;LOG=13;    LOG10=14;    LSIN=15;    SINH=16;
LSQRT=17; TAN=18; TANH=19;

      (*Все  обозначения  (имена) операций  и  их
коды  сведем  в таблицу (массив структур tf[]) по
шаблону*)
type

  loperation=record
    on:array [0..9] of char;
```

```
    ov:integer;
  end;

const
  op:array [0..19] of loperation=
(
(on:'ABS';ov:LABS),              (on:'ACOS';ov:ACOS),
(on:'ASIN';ov:ASIN),             (on:'ATAN';ov:ATAN),
(on:'COS';ov:LCOS),              (on:'COSH';ov:COSH),
(on:'EXP';ov:LEXP),                 (on:'LN';ov:LOG),
(on:'LOG';ov:LOG10),              (on:'SIN';ov:LSIN),
(on:'SINH';ov:SINH),            (on:'SQRT';ov:LSQRT),
(on:'TAN';ov:TAN),               (on:'TANH';ov:TANH),
(on:'+';ov:PLUS),                  (on:'-';ov:MINUS),
(on:'*';ov:MUL),  (on:'/';ov:LDIV),  (on:'^';ov:POW),
(on:'';ov:0));

OOP:PChar='+-*/^';     (*Перечень арифметических опе-
раций*)

(*НЕКОТОРЫЕ СЛУЖЕБНЫЕ ПОДПРОГРАММЫ*)
(*Преобразование к верхнему регистру*)
function toupper(c:char):char;
var
  d:byte;
begin
  d:=byte(c);
  if(d>=byte('a'))and(d<=byte('z'))              then
dec(d,$20);
  if(d>=byte('a'))and(d<=byte('п'))              then
dec(d,$20);
  if(d>=byte('p'))and(d<=byte('я'))              then
dec(d,$50);
  if(d=byte('ё'))                   then
d:=byte('Ё');
  toupper:=char(d);
end;

(*Тест на "низость" символа, в т.ч. для русских
букв*)
function islower(key:char):boolean;
begin
  islower:=false;
  if (key='ё') or
```

```
      (           (byte(key)>=byte('a'))           and
(byte(key)<=byte('z')) ) or
      (           (byte(key)>=byte('a'))           and
(byte(key)<=byte('п')) ) or
      (           (byte(key)>=byte('p'))           and
(byte(key)<=byte('я')) ) then
    islower:=true;
end;


(*Ф-ция приведения латинских букв к верхнему регистру*)
procedure touppercase(f:PChar);
var
  j:integer;
begin
  for j:=0 to StrLen(f) do
    if islower(f[j]) then
       f[j]:=toupper(f[j]);
end;

      (*Тест на алфавитно-цифровые символы*)
function isalnum(key:char):boolean;
begin
  isalnum:=false;
  if        (        (byte(key)>=byte('0'))        and
(byte(key)<=byte('9')) ) or
      (           (byte(key)>=byte('A'))           and
(byte(key)<=byte('Z')) ) or
      (           (byte(key)>=byte('a'))           and
(byte(key)<=byte('z')) ) or
      (           (byte(key)>=byte('A'))           and
(byte(key)<=byte('п')) ) or
      (           (byte(key)>=byte('p'))           and
(byte(key)<=byte('ё')) ) then
    isalnum:=true;
end;


(*Функция очистки формульной строки от излишков и неточностей*)
procedure clearformula(f:PChar);
var
  temp:PChar;
```

```
  i:integer;
begin
  temp:=@f[0];
  while boolean(temp^) do
    if        isalnum(temp^)         or         long-
bool(StrScan(OOP,temp^))
       or(temp^='.')or(temp^='(')or(temp^=')') then
       inc(temp)(*пример адресной арифметики*)
    else
       (*Вначале удалим все недопустимые в формуле
символы*)
       move(temp[1],temp[0],StrLen(@temp[1])+1);

(*Затем несколько раз на наличие сочетаний типа '+*'
или '-*'*)
  temp:=@f[1];
  while boolean(temp^) do
  begin
    if((temp^='*')or(temp^='/')or(temp^='^'))and
       ((temp-1)^='+')or((temp-1)^='-')then
    begin
       move(temp[1],temp[0],StrLen(@temp[1])+1);
       temp:=@f[1];{перезапуск с начала строки}
    end;
    inc(temp);
  end;
end;


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

procedure look_up(s:PChar;cv:preal);
var
  i:integer;
begin
  (*Просмотрим таблицу операций*)
  i:=0;
  while wordbool(StrLen(op[i].on)) do
  begin
  (*Если нашли совпадение с допустимым именем опера-
ции
```

```
    возвращаем конкретный тип операции*)
    if not wordbool(StrComp(op[i].on,s)) then
    begin
       token_type:=OPERATION;
       tok:=op[i].ov;
       exit;
    end;
    inc(i);
  end;
  i:=0;
  (*Поиск в таблице констант*)
  while wordbool(StrLen(tc[i].cn)) do
  begin
(*Если нашли совпадение с допустимым именем кон-
станты возвращаем индекс константы в массиве
структур и ее значение записываем по заданному
адресу*)
    if not wordbool(StrComp(tc[i].cn,s)) then
    begin
token_type:=OPERAND;tok:=CONSTANT;     cv^:=tc[i].cv;
exit;
    end;
    inc(i);
  end;

  (*Если ничего не нашли*)
  token_type:=0;tok:=-1;
end;

(*Ф-ция для вывода сообщения об ошибке и возврата к
вводу *)
procedure serror(error:integer);
const
  e:array [0..6] of string=(
'Синтаксическая ошибка',
'Непарные круглые скобки',
'Где-то не выражение',
'Где-то не переменная',
'Есть нераспознанная операция',
'Не распознанное имя константы',
'Получается деление на нуль');
begin
  gotoxy(1,5);
  write(e[error],'-any key to continue');
```

```
  readkey;
  WasError:=true;
end;

(*Тест на цифровые символы*)
function isdigit(key:char):boolean;
begin
  isdigit:=false;
  if                (byte(key)>=byte('0'))        and
(byte(key)<=byte('9'))then
    isdigit:=true;
end;


(*Тест на алфавитные символы - операция пересечения
2-х множеств*)
function isalpha(key:char):boolean;
begin
  isalpha:=isalnum(key) and not isdigit(key);
end;


(*Функция выделения и классификации очередной лексе-
мы, возвращает класс лексемы. Ее параметры - указа-
тель на указатель текущего текста формулы и адрес
для значения операнда-константы - он транзитом пере-
дается в look_up(). Первый параметр мы вынуждены по-
лучать 'по ссылке', чтобы значение адреса текущей
лексемы изменялось в вызывающей программе, а не в
нашей подпрограмме*)

function get_token(cf:PPChar;cv:preal):integer;
var
  opr,temp:PChar;(*Временный указатель на лексему*)
begin
  token_type:=0;  tok:=0;
  (*Если конец формулы*)
  if(cf^^=#0)then
  begin
  token[0]:=#0;                        token_type:=EOL;
get_token:=token_type; exit;
  end;

  (*Если это открытая круглая скобка*)
```

```
  if(cf^^='(')then
  begin
    temp:=token;
    temp^:=cf^^;   (*перепишем его в token*)
    inc(cf^);         (* переход на следующую пози-
цию*)
    inc(temp);
    temp^:=#0;       (*token закроем нулем*)
    tok:=EXPRESS;
    token_type:=OPERAND;
    get_token:=token_type;
    exit;
  end;

(*Если это закрытая круглая скобка
  if(cf^^=')')then
  begin
    temp:=token;
    temp^:=cf^^;   (*перепишем его в token*)
    inc(cf^);         (* переход на следующую пози-
цию*)
    inc(temp);
    temp^:=#0;       (*token закроем нулем*)
    tok:=EXPRESS;
    token_type:=SKOBKA;
    get_token:=token_type;
    exit;
*)

(*Если это символ арифметической операции*)
  opr:=StrScan(OOP,cf^^);
  if(opr<>nil)then
  begin
    temp:=token;
    temp^:=cf^^;   (*перепишем его в token*)
    inc(cf^);         (* переход на следующую пози-
цию*)
    inc(temp);
    temp^:=#0;       (*token закроем нулем*)
    case opr^ of
      '+': tok:=PLUS;
      '-': tok:=MINUS;
      '*': tok:=MUL;
      '/': tok:=LDIV;
```

```
      '^': tok:=POW;
    end;
    (*сообщим что класс лексемы - операция*)
    token_type:=OPERATION;
    get_token:=token_type;
    exit;
  end;

(*Если встретили цифру*)
  if isdigit(cf^^)then
  begin
    (*то запишем всю числовую подстроку в token*)
    temp:=token;
    while isdigit(cf^^) or (cf^^='.')do
    begin
    temp^:=cf^^;  inc(temp);  inc(cf^);
    end;
    temp^:=#0;  tok:=NUMBER; token_type:=OPERAND;
    get_token:=token_type;
    exit;
  end;

(*Если встретили букву*)
  if isalpha(cf^^)then
  begin
(*то это переменная или операция-функция или именованная константа, пока все буквы-цифры перепишем и временно зарегистрируем просто строкой*)
    temp:=token;
    while isalnum(cf^^)do
    begin
    temp^:=cf^^; inc(temp);   inc(cf^);
    end;
    token_type:=LSTRING;
  end;
  temp^:=#0;

  (*Не отходя далеко проанализируем полученную строку*)
  (*Если это 1-буквенное имя независимой переменной*)
  if(token_type=LSTRING           )and           not wordbool(StrComp(token,'T'))then
  begin
```

```
tok:=VARIABLE;token_type:=OPERAND;get_token:=token_t
ype;exit;
  end;

(*В противном случае поищем среди унарных операций и
констант*)
  look_up(token,cv);
  if(tok<0)then  serror(4);
end;

(* Эта подпрограмма может использоваться как вспомо-
гательная, например, в интерпретаторе для вычисления
значений формулы при различных значениях независимой
переменной. В следующих разделах это будет продемон-
стрировано, а пока мы просто протестируем приведен-
ные ф-ции в программе, выводящей на экран отдельные
лексемы заданной в командной строке формулы, их
классы и подклассы.*)

{$I gettok.inc}
var
  formula:array[0..80]of  char;     (*Для  сохранения
текста формулы*)
  cv:real; t:PChar;
begin
  if(ParamCount<>1)then
  begin
  write('Неполадки с параметрами в командной стро-
ке');  exit;
  end;
    StrPCopy(formula,ParamStr(1));        (*Сохраняем
текст формулы*)
    touppercase(formula);      (*Приводим к верхнему
регистру*)
    clearformula(formula);    (*Убираем недопустимые
символы*)
  t:=PChar(@formula);
  clrscr;
  get_token(@t,@cv);
  while(token_type<>EOL)and(not WasError)do
  begin
    writeln('Лексема     ',token,',     код     класса
',token_type,', код подкласса ',tok);
    get_token(@t,@cv);
```

```
  end;
end.
```

2. Рекурсивный интерпретатор формул, заданных на стадии выполнения программы в виде символьной строки.

Задача интерпретирующей подпрограммы - вернуть вычисленное значение при заданном значении аргументов. Для упрощения и без того достаточно громоздкого алгоритма мы будем предполагать, что формула представляет собой функцию f(t) одного аргумента с фиксированным именем 't' и в нее могут входить в качестве операндов числа, именованные константы (например 'pi', 'e' и пр.), аргумент 't' и набор различных математических операций и функций.

В качестве вспомогательного средства функция - интерпретатор будет использовать подпрограммы синтаксического анализатора строковых выражений, который будет вызываться для получения каждой следующей лексемы - неделимой единицы выбранного "формульного" языка, который у нас по возможности не будет отличаться от принятого в математике. Неизбежное отличие будет конечно состоять в том, что например произведение dcos(t) придется записывать в виде d*cos(t) с явным, а не подразумеваемым обозначением операции умножения. Подпрограммы реализации синтаксического анализатора поместим в файл gettok.inc и включим его в нашу программу директивой {$Ifilename.ext}. Собственно интерпретацию и вычисление значения формулы при заданном значении аргумента t будем осуществлять по следующему алгоритму: вначале просмотрим текст формулы, предварительно "очищенный" специальной подпрограммой от пробелов и явных ошибок в расстановке операций - например от знаков умножений, делений, возведения в степень сразу после знаков других операций; в процессе просмотра заполним массив структур, полями которых являются коды операций или операндов и (для операндовой структуры) значения операндов. При получении операнда класса EXPRESS (выражение в скобках) мы просто перепишем все внутрискобочное выражение в приватный массив подпрограммы и ре-

курсивно вызовем ее для вычисления внутрискобочного выражения - еще один пример использования рекурсии для упрощения метода решения задачи.

После построения структуры формулы в виде последовательности операций и операндов просто приступим к выполнению операций в порядке их приоритетов - сначала все унарные операции, включая математические функции от скобочных выражений, затем возведения в степень, потом умножения и деления и в последнюю очередь сложения и вычитания. После выполнения каждой операции количество элементов в массиве формульных структур будем сокращать после записи результата операции в поле соответствующего операнда и перемещения влево области занимаемой структурами массива памяти.

После выполнения последней операции результат окажется в 0-м элементе массива структур - оттуда его и возвратит наша функция интерпретации выражений.

```
{$I gettok.inc} (*Выделение лексем*)

var
  formula:array[0..80]of char;      (*Для  сохранения
текста формулы*)
  t:real;             (*Для значения аргумента*)

(* Нам понадобятся некоторые вспомогательные функ-
ции, в частности функция выполнения заданных двух-
местных математических операций - ее параметры при
вызове - это код требуемой двухместной операции и
значения левого и правого операндов.*)

function oper(co:integer;lo,ro:real):real;
begin
  case(co)of
    PLUS :oper:=lo+ro;
    MINUS:oper:=lo-ro;
    MUL  :oper:=lo*ro;
    LDIV  :
         if(ro=0) then   serror(6)
         else             oper:=lo/ro;
    POW  :
       begin
         if lo=0 then    oper:=0
```

```
        else              oper:=exp(ro*ln(lo));
      end;
    else                 serror(0);
  end;
end;


(*Эта функция осуществляет унарные операции, к клас-
су которых мы отнесли помимо унарных "плюс" и "ми-
нус" все математически функции, полагая находящееся
в скобках после имени функции выражение единственным
их операндом *)

function unary(co:integer;ro:real):real;
begin
  case(co)of
    PLUS :unary:=ro;
    MINUS:unary:=-ro;
    LSIN :unary:=sin(ro);
    LCOS :unary:=cos(ro);
    TAN  :unary:=sin(ro)/cos(ro);
    LSQRT:unary:=sqrt(ro);
    else  serror(0);
  end;
end;

(* Формула состоит из операндов и операций. Типы
операндов - число, именованная константа, перемен-
ная, выражение в скобках.*)

type __formula=record     (*Для структуры формулы*)
  dtok,                        (*Код типа операнда*)
  co:word;                   (*Код операции*)
  z:real;                     (*значение операнда*)
end;

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

var
result:real; (*для промежуточных результатов вычис-
лений*)
```

```
(*для текущего фрагмента формулы*)
privat_form:array[0..80]of char;
function get_exp(f:PChar;t:real):real;
var
    tt,flag,Code,
    sc,                    (*Счетчик скобок*)
    i,j,                  (*рабочие   переменные   для
циклов*)
    co:integer;       (*для кода операции*)
    cv:real;        (*для значения именованной кон-
станты*)
  tmp:PChar;              (*Исходный адрес формулы*)
  cnt:word;          (*Количество элементов в форму-
ле*)
  frm:array[0..31]of __formula;
begin
  for i:=0 to 31 do
    FillChar(frm[i],sizeof(frm[i]),0);(*Обнулим*)
  tmp:=f; i:=0;  cnt:=0;  get_token(@tmp,@cv);
  (*Просмотрим текст формулы и заполним ее структу-
ру*)
  while(token_type<>EOL)do
  begin
    (*Если получили операнд - число, константу или
переменную*)
    if(token_type=OPERAND)then
    begin
       frm[i].dtok:=tok;
       if(tok=NUMBER)then
Val(token,frm[i].z,Code);
       if(tok=CONSTANT)then    frm[i].z:=cv;
       if(tok=VARIABLE)then    frm[i].z:=t;
       if(tok=EXPRESS)then
       begin
         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 мо-
жем использовать такой же механизм исследования*)
         j:=0;
         while wordbool(sc)and(j<78)do
         begin
            if(tmp^='(')then inc(sc);
```

```
            if(tmp^=')')then dec(sc);
            privat_form[j]:=tmp^; inc(j);inc(tmp);
          end;
          privat_form[j-1]:=#0;   (*Закроем нулем*)
if wordbool(sc)then  serror(1); (*Если не нашли пар-
ную скобку*)
          result:=get_exp(privat_form,t);
          token_type:=tt;     (*Восстановим тип лексе-
мы*)
          frm[i].dtok:=NUMBER; frm[i].z:=result;
        end;(*if EXPRESS   *)
    end;  (*if OPERAND*)

    (*Если получили операцию*)
if(token_type=OPERATION)then          frm[i].co:=tok;
inc(i);
get_token(@tmp,@cv);
end; (*while*)

cnt:=i;       (*Количество заполненных элементов*)

(*Теперь можем обрабатывать. Вначале необходимо вы-
полнить 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 - опе-
ранд*)

  flag:=1;
  while wordbool(flag)do
  begin
    flag:=0; i:=0;
    while(i<cnt)do
    begin
if((i=0)and(frm[i].co<>0)and(frm[i+1].dtok<>0))or((i
<>0)and
(frm[i-
1].co<>0)and(frm[i].co<>0)and(frm[i+1].dtok<>0))then
       begin
         frm[i+1].z:=unary(frm[i].co,frm[i+1].z);
         (*Сомкнем ряды на выполненной операции*)
         move(frm[i+1],frm[i],(cnt-i-
1)*sizeof(frm[i]));
         dec(cnt); inc(flag);
```

```
       end;
       inc(i);
    end;
  end;
  (*Теперь необходимо выполнить все возведения в
степень *)
  i:=1;
  while(i<cnt)do
  begin
if(frm[i].co=POW)and(frm[i-
1].dtok<>0)and(frm[i+1].dtok<>0)then
    begin
       frm[i-1].z:=oper(POW,frm[i-1].z,frm[i+1].z);
       (*Сомкнем ряды на выполненной операции*)
move(frm[i+2],frm[i],(cnt-i-
2)*sizeof(frm[i]));dec(i);dec(cnt,2);
    end;
    inc(i);
  end;
  (*Теперь  выполнить все деления и умножения *)
  i:=1;
  while(i<cnt)do
  begin
    if((frm[i].co=MUL)or(frm[i].co=LDIV))and(frm[i-
1].dtok<>0)
         and(frm[i+1].dtok<>0)then
    begin
       frm[i-1].z:=oper(frm[i].co,frm[i-
1].z,frm[i+1].z);
       (*Сомкнем ряды на выполненной операции*)
move(frm[i+2],frm[i],(cnt-i-
2)*sizeof(frm[i]));dec(i);dec(cnt,2);
    end;
    inc(i);
  end;
  (*Теперь  выполнить все сложения и вычитания *)
  i:=1;
  while(i<cnt)do
  begin

if((frm[i].co=PLUS)or(frm[i].co=MINUS))and(frm[i-
1].dtok<>0) and(frm[i+1].dtok<>0)then
    begin
```

```
       frm[i-1].z:=oper(frm[i].co,frm[i-
1].z,frm[i+1].z);
       (*Сомкнем ряды на выполненной операции*)
move(frm[i+2],frm[i],(cnt-i-
2)*sizeof(frm[i]));dec(i);dec(cnt,2);
    end;
    inc(i);
  end;
  (*Теперь в 0-м элементе лежит результат*)
  get_exp:=frm[0].z;
end;

{$I interpr.inc}

      (* Пусть текст формулы и значение аргумента,
при котором она должна быть вычислена, задается в
командной строке *)

var
  Code:integer;
  r:real;
begin
  if(ParamCount<>2)then
  begin
  writeln('Неполадки с параметрами в командной стро-
ке');exit;
  end;
  StrPCopy(formula,ParamStr(1));   (*Сохраняем текст
формулы*)
  Val(ParamStr(2),t,Code);
  touppercase(formula);clearformula(formula);
  r:=get_exp(formula,t);
  clrscr;
  writeln('Вычисленное значение при t=',t:5:3,' рав-
но ',r:5:3);
end.
```

3. Работа в графическом режиме с библиотекой Borland Graphics Interface (BGI) при построении графиков функций.

Предположим, что перед нами поставлена задача составить программу - графический интерпретатор функций, принимающую от пользователя уравнения плоских кривых в параметрической форме (x=fx(t); y=fy(t)) и диапазон изменения параметра t и строящая по этим уравнениям графики соответствующих функций.

Для решения этой задачи нам, очевидно, пригодятся рассмотренные в предыдущих разделах программы синтаксического анализа заданных в строковом виде формул, интерпретатор строковых выражений.

Составление любой программы лучше всего начинать с составления ее словесного плана-алгоритма, вначале крупноблочного, а затем постепенно детализируемого. Это даст возможность структурировать программу разбиением на отдельные подпрограммы (разложить сложную задачу на ряд простых), определить функциональное назначение всех подпрограмм.

Итак, начнем:

1.Сохраним содержимое экрана при входе в нашу программу, чтобы восстановить перед выходом из нее и "не наследить".

2.Оформим интерьер для ввода-вывода. Выведем приглашение пользователю для ввода уравнений кривых. Дадим ему возможность свободно редактировать текст вводимых формул, перемещаясь между предусмотренными для этого строками ввода. Результаты ввода и редактирования должны отображаться и на экране и в буфере памяти.

3.После завершения ввода вычислим значения переменных с определенным шагом и результаты вычисления сохраним в таблице для последующего построения графика. Для этого вызовем подпрограмму - интерпретатор текста формул для расчета массивов значений x и y при различных значениях параметра t.

4.Инициализируем графический режим и определим основные параметры - максимальные значения координат и пр.

5.Оформим интерьер для вывода графика функции, просчитаем масштабы по осям координат.

6.Нарисуем график и когда пользователь им налюбуется и нажмет например Esc - предложим ему продолжить работу с другой кривой или с той же самой, но с другими коэффициентами. По желанию пользователя завершим работу программы.

Пункты с 2-го по 6-й будем повторять циклически, пока пользователь не введет признак завершения программы - нам этот признак тоже предстоит придумать.

8.Выйдем из графического режима и восстановим содержимое текстового экрана, из которого мы стартовали.

Вот крупным планом и все - теперь на очереди детализация плана.

Реализация этого плана на Си приведена в разделе 6.18. Так как библиотеки BGI - графики в Паскале и Си полностью аналогичны, Паскаль - реализацию этой программы мы оставляем на самостоятельную проработку с возможностью "консультаций" в приведенном Си - варианте.

```
#include <dos.h>
#include <stdio.h>
#include <bios.h>
#include <setjmp.h> //Для функций дальних переходов
#include <math.h>
#include <ctype.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <graphics.h>
#include"keycodes.def" //Двухбайтовые коды клавиш
#include"formread.h" //Здесь редактор для ввода фор-
мул
#include"syntax.h"        //Выделение лексем
#include"interpr.h"      //Интерпретация формул
#include"tablev.h"  //Расчет таблиц для построения
графиков
#include"makegraph.h" //Построение графиков функций
по таблицам x,y

jmp_buf e_buf;//Буфер для дальнего возврата по ошиб-
кам
```

```
char sbuf[4096];   //для сохранения экрана

void main(void)
{
int key1,i;
gettext(1,1,80,25,sbuf);       //запоминаем экран в
sbuf
textattr(0x07);clrscr();

      /*Заполним буфер для дальнего возврата при
ошибке, обнаруженной синтаксическим анализатором -
именно со следующего фрагмента редактирования формул
предстоит начинать исправление ошибок */
setjmp(e_buf);

for(;;)           //В бесконечном цикле повторяем
{
//ввод текста формулы и пределов изменения аргумента
if((key1=formread())==Esc) break;

//Приведение всех строк к одному регистру и очистка
for(i=0;i<4;i++)
{touppercase(formula[i]);clearformula(formula[i]);}

/*функция расчета таблиц, использующая синтаксиче-
ский анализатор текста формулы */
tablevar();

//Переход в графику с сохранением текста
ginit();

//вывод графика функции
gshow();

fflush(stdin); //Очищаем буфер ввода
restorecrtmode();//Снова в текстовый режим
}
//Перед выходом из программы восстанавливаем экран
textattr(0x07);clrscr();
puttext(1,1,80,25,sbuf);
}
```

Приведем некоторые разъяснения.

1. Сохранение и восстановление текстового экрана.

Любая "культурная" программа должна после своего завершения оставить используемые ресурсы в том же состоянии, в котором они находились до ее начала; в нашем случае это прежде всего экран монитора, на котором мы собираемся что-то рисовать. Чтобы потом восстановить его содержимое, мы должны запомнить его при входе в программу.

Будем для упрощения предполагать, что мы стартуем из текстового режима номер 3 с 25-ю строками и 80-ю символами в строке. (Строго говоря, это не обязательно так - в наиболее общем случае программа должна определить, из какого режима она стартует и вызывать соответствующие подпрограммы сохранения и восстановления. Но пока для упрощения мы будем думать, что стартуем из наиболее распространенного режима). Так как каждый символ в видеопамяти представлен 2-мя байтами (байт кода символа и байт атрибута), то для хранения образа символьного экрана нам понадобится 2*25*80=4000 байт или 2000 2-байтовых слов.

Отведем для хранения массив

```
char video[4000];
```

и сохраним экран с помощью библиотечной функции

```
gettext(1,1,80,25,video);
```

с прототипом в conio.h. Перед выходом из программы мы сможем восстановить содержимое экрана из массива video с помощью функции

```
puttext(1,1,80,25,video);
```

Можно это же сделать прямым обращением к видеопамяти - для этого надо определить указатель на видеобуфер, адрес которого для текстового режима номер 3 равен 0xB8000000:

```
char * vptr=(char*)0xB8000000;
```

и затем просто прочитать из видеобуфера:

```
for(int i=0;i<4000;i++) video[i]=vptr[i];
```

При восстановлении в конце программы в этом случае аналогичный цикл переписи назад:

```
for(i=0;i<4000;i++) vptr[i]=video[i];
```

2. Диалог с пользователем и редактирование текстов формул.

Теперь можем заняться интерьером. Для этого нужно представить себе, как будет выглядеть удобный для пользователя экран во время работы с программой. В простейшем случае это 2 прямоугольных области, в одной из которых будет диалог с пользователем по поводу формул и диапазона изменения параметра, а во второй будет рисоваться график кривой. После отрисовки кривой во второй области активной снова должна становиться первая и желательно без стирания предыдущих формул – ведь пользователю может понадобиться изменить в формуле только какие-нибудь численные коэффициенты. Здесь не было бы особых проблем, если бы курсор в графическом режиме не был невидимым – пользователь не видит, куда будет введен очередной символ, если мы об этом не позаботимся в нашей программе. Можно было бы создать свой программно управляемый графический курсор, но это слишком
затемнило бы основную нашу задачу второстепенными деталями и осложнило бы восприятие учебного материала. Поэтому мы принимаем следующее решение: ввод пользователя осуществляем в текстовом режиме, по сигналу пользователя о завершении ввода сохраняем содержимое окна ввода в отведенном для этого буфере с помощью функции `gettext()`. После перехода в графический режим и перевода функций из `conio.h` в режим работы через прерывания `bios` (установкой переменной `directvideo=0` вместо прямого доступа к видеобуферу при `directvideo=1`) восстанавливаем окно ввода с помощью `puttext()` из буфера сохранения. При работе с видеоадаптерами устаревших моделей класса `CGA` могут возникнуть проблемы с изменением цветов; избежать этого можно предварительной обработкой байтов атрибутов в буфере сохранения, но мы этим заниматься не будем.

Мы пока займемся текстовой частью, а графикой позже. Оформим массив строк `inpstr`:

Файл `formread.h`

```
const
xbeg=2,ybeg=1,xend=78,yend=8,  //Размеры и позиция окна диалога
stroffs=12,lstr=60;      //Координаты и длина строк ввода
```

```
char *inpstr[]=                 //Массив строк диало-
га
{
"□□Введите параметрическое кривой и пределы парамет-
ра t□□",
"□ X(t)=
"□ Y(t)=
"□ tmin=
"□ tmax=
"□ Ошибка:
"□□□□□□□□F6   -рисовать□□□□□Esc-выход□□□□□□□□□□□□□□□□□
",
 ""
       };
      /*Примечание: размер рисунка рамки уменьшен
здесь по сравнению с фактическим для размещения на
книжной странице */

char x_scr,y_scr,temp; //Позиция курсора на экране и
в строках
#define x_buf x_scr-1//Для отображения позиции кур-
сора в буфере
#define y_buf y_scr-1 //связываем их через макро-
подстановку

//Буфер для сохранения окна диалога
char textbuf[2*(xend-xbeg+1)*(yend-ybeg+1)];

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

int index;        //Текущий номер строки ввода

char formula[4][80]; //Для текста формул и пределов
параметра

//Функция для ввода формул и пределов изменения па-
раметра
 formread()
 {
 int i,j,key;
 directvideo=1;    //Работаем прямо с видеопамятью
 window(1,1,80,25); //Занимаем весь экран
 //Выведем шаблон для ввода формул
```

```
 textattr(0x0A);clrscr();
 for(i=0;strlen(inpstr[i]);i++)
 {gotoxy(xbeg,ybeg+i); printf("%s",inpstr[i]);}
 //Окно внутри рамки
 window(xbeg+stroffs,ybeg+1,xbeg+lstr,yend-1);
 //Если в строках буфера ввода что-то есть - выведем
это
 for(i=0;i<4;i++)
 if(strlen(formula[i]))
 {gotoxy(1,i+1);cputs(formula[i]);}

 x_scr=1;y_scr=1;
 gotoxy(x_scr,y_scr); //Курсор в начальную позицию

 //Теперь предстоит создать 4-хстрочный текстовый
редактор
 for(;;)
 {
      fflush(stdin); //Очистим клавиатурный буфер
      key=bioskey(0);   //Получаем 2-хбайтовый код
клавиши
      switch(key)
      {
      case F6:     //Если ввод закончен
                        //Сохраняем окно с введен-
ным текстом
      gettext(xbeg,ybeg,xend,yend,textbuf);
      return 0;
      //Если пользователь закончил работу с програм-
мой
       case Esc:return Esc; //на выход!

       /*Управление курсором - действия по клавишам
- стрелкам сводятся к изменению позиции курсора и
через макроподстановку - позиции в буфере x_buf,y-
buf */

       case  Left:if(x_scr>1)
       {x_scr--;gotoxy(x_scr,y_scr);}break;

       case
Right:if(x_scr<(strlen(formula[y_buf])+1))

{x_scr++;gotoxy(x_scr,y_scr);}break;
```

```
       case Up: if(y_scr>1)
                  {y_scr--
;x_scr=1;gotoxy(x_scr,y_scr);}break;

       case Down: if(y_scr<5) {y_scr++;x_scr=1;
       gotoxy(x_scr,y_scr);}break;

       case Home: x_scr=1;gotoxy(x_scr,y_scr);break;

       case End: x_scr=strlen(formula[y_buf]);
       gotoxy(x_scr,y_scr);break;

      /*Редактирование текста
При стирании символа весь "хвост" после него смещаем
на 1 байт влево в памяти и на экране (стиранием и
выводом) */

       case Del:
       memmove(formula[y_buf]+x_buf,

formula[y_buf]+x_buf+1,strlen(formula[y_buf]+x_buf+1
)+1);

temp=x_scr;clreol();cputs(formula[y_buf]+x_buf);
       x_scr=temp;gotoxy(x_scr,y_scr); break;

       case BackSpace:
       if(x_scr>1){
       memmove(formula[y_buf]+x_buf-1,

formula[y_buf]+x_buf,strlen(formula[y_buf]+x_buf)+1)
;
       gotoxy(--x_scr,y_scr);
       temp=x_scr;clreol();
       cputs(formula[y_buf]+x_buf);
       x_scr=temp;gotoxy(x_scr,y_scr);
       }
       break;

       default:
       //Если символ допустим в тексте формулы
       if((char)key){
       if((y_scr<5) && (isalnum((char)key) ||
```

```
        strchr("*+-/^.()",(char)key)!=NULL))
        {

memmove(formula[y_buf]+x_buf+1,formula[y_buf]+
        x_buf,strlen(formula[y_buf]+x_buf));
        formula[y_buf][x_buf]=(char)key;
        temp=x_scr;clreol();
        cputs(formula[y_buf]+x_buf);
        x_scr=++temp;gotoxy(x_scr,y_scr);
        } } } }}
```

3.Наша следующая задача - заполнение таблицы значений t, x и y. При этом помним, что x вычисляется по formula[0], y - по formula[1].

Файл tablev.h

```
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 xyint[MAX_POINT][2]; //таблица для значений x,y в пикселах
```

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

```
void tablevar(void)
{
 int i;
 /*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, специфических для какой либо прикладной области - это могут быть физические, химические константы и пр. */

 char *curf=formula[2];
 tmin=get_exp(curf,0);
 curf=formula[3];
 tmax=get_exp(curf,0);
 double incr=(tmax-tmin)/MAX_POINT;
```

```
 for(index=0;index<MAX_POINT;index++)
t[index]=tmin+index*incr;
 for(i=0;i<2;i++)
 {curf=formula[i];
 for(index=0;index<MAX_POINT;index++)
    xy[index][i]=get_exp(curf,t[index]);
 }
}
```

4. Инициализация графического режима и оформление интерьера.

Наконец мы добрались до графики.

Собственно инициализация графики в простейшем случае сводится к вызову библиотечной функции `initgraph` с передачей ей адресов переменных для графического драйвера, графического режима и пути доступа к `BGI` - драйверу. Но такая программа может быть запущена только на компьютере с имеющимся на диске файлом `BGI`-драйвера (с расширением `.bgi`) и, как следствие, - переноситься на всякий случай вместе с файлом этого драйвера на дискете.

Аналогичная проблема возникает с файлами графических масштабируемых сегментированных шрифтов (с расширением `.chr`) - система `BGI` предполагает по умолчанию, что путь доступа к этим файлам такой же, как и к драйверу графики. Поэтому целесообразно включить и файл драйвера и файл используемого шрифта непосредственно в код программы. Это делается до инициализации графики следующим образом:

с помощью специальной программы-утилиты в составе пакета Си `bgiobj.exe` все необходимые программе `.bgi` и `.chr`-файлы преобразуются в объектные `.obj`-файлы, регистрируются в программе с помощью библиотечных функций

```
int registerbgidriver(void (* driver)(void))
int registerfarbgidriver(void far *driver)
int registerbgifont(void (*font)(void))
int registerfarbgifont(void far *font)
```

Эти функции информируют систему `BGI`-графики о том, что необходимые драйверы присутствуют в коде программы и не требуют загрузки с диска. Аргумент этих Функций - имя реги-

стрируемого драйвера или шрифта, образуемого присоединением к имени драйвера суффикса _driver, _driver_far или _font, _font_far, например (это имя вам сообщит утилита bgiobj.exe):

```
registerbgidriver(EGAVGA_driver);
registerbgifont(triplex_font_far);
```

После этого строка пути доступа к драйверам и графическим шрифтам при вызове может быть пустой.

Еще одно – для компоновки полученных объектных файлов совместно с вашей программой создайте файл проекта и включите в него эти объектные файлы.

Таким образом, функция инициализации BGI-графики может выглядеть так:

Файл makegraph.h

```
unsigned X,Y;                    //размеры   экрана   в
пикселах
int graphflag=1;

//инициализация графики
 void ginit(void)
 {
int gd = DETECT,gm, ec;

//Для исключения повторной регистрации драйвера и
шрифтов
if(graphflag)
{
ec = registerbgidriver(EGAVGA_driver);
if (ec < 0)
{printf("Ошибка графики: %s\n", grapherrormsg(ec));
getch();exit(1);}
ec = registerbgifont(triplex_font);
if (ec < 0)
{printf("Ошибка графики: %s\n", grapherrormsg(ec));
getch();exit(1);}

ec = registerbgifont(sansserif_font);
if (ec < 0)
```

```
{printf("Ошибка графики: %s\n", grapherrormsg(ec));
getch();exit(1);}
//Функция инициализации графического режима
initgraph(&gd, &gm, "");
ec = graphresult();
if     (ec     !=     grOk){printf("Ошибка:     %s\n",
grapherrormsg(ec));
getch(); exit(1);}
graphflag=0;
}
else setgraphmode(getgraphmode());

      /*Вывод сохраненного через gettext фрагмента
из текстового режима */
directvideo=0;     //переходим на вывод через BIOS
puttext(xbeg,ybeg,xend,yend,textbuf);
 }

      /*Функция вывода графика кривой, заданной мас-
сивом xy[RowCount][ColColnt], в графическое окно */
 void gshow()
 {
/*Для масштабов по x и y - количества пикселов на
натуральные единицы */
 double picx,picy;

 double dx,dy; //Диапазоны переменных
 int i;
 //определим максимальные значения x и y просмотром
массива
 xmax=xy[0][0];xmin=xy[0][0];
 ymax=xy[0][1];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
 X=getmaxx();Y=getmaxy();
 int scrhight=(3*Y)>>2; //Высота окна
 int scrwidth=scrhight; //Ширина окна - оно будет
квадратным
 picy=picx=scrwidth;
 if(dx<dy) picx*=(dx/dy);
 if(dx>dy) picy*=(dy/dx);

 //определяем графическое окно с отсечкой вывода на
границах
 setviewport(X-scrwidth,Y>>2,X,Y,1);
 //Установим XOR -стираемый режим вывода отрезков
 setwritemode(XOR_PUT);
 setcolor(6);                    //цвет рисования
 setfillstyle(1,7);              //заполнение
 setlinestyle(0,8,1);            //стиль линий
 rectangle(2,2,scrwidth-2,scrhight-2);     //рамка  в
окне
 bar(3,3,scrwidth-3,scrhight-3);  //заполненный  пря-
моугольник

      // координаты центра системы координат
      int xc=scrwidth>>1;
      int yc=scrhight>>1;

      //Координатные оси
moveto(5,yc);lineto(scrwidth-5,yc);
linerel(-6,-2);linerel(0,4);linerel(6,-2);//Стрелки
на осях

for(i=0;i<10;i++)      //Метки на осях
{moverel(-(int)(0.1*scrwidth),0);
if(i&1) linerel(0,-4);else linerel(0,4);
}

moveto(xc,scrhight-5);lineto(xc,5);
linerel(-2,6);linerel(4,0);linerel(-2,-6);

for(i=0;i<10;i++)      //Метки на осях
{moverel(0,(int)(0.1*scrwidth));
if(i&1) linerel(4,0);else linerel(-4,0);
}
```

```
outtextxy(scrwidth-10,yc-10,"X");
outtextxy(xc+10,10,"Y");

      //Вывод кривой
moveto((int)(xy[0][0]*picx)+xc,yc-
(int)(xy[0][1]*picy));
for(i=1;i<MAX_POINT;i++)
 {lineto((int)(xy[i][0]*picx)+xc,yc-
(int)(xy[i][1]*picy));}
 //Цикл просмотра и корректировки кривой
 int k;
 do
 {
 k=bioskey(0);
 switch(k)
 {
 case Up:
      //Стирание кривой повторным выводом
 moveto((int)(xy[0][0]*picx)+xc,yc-
(int)(xy[0][1]*picy));
 for(i=1;i<MAX_POINT;i++)
 {lineto((int)(xy[i][0]*picx)+xc,yc-
(int)(xy[i][1]*picy));}

 //Изменение масштаба и по клавишам - стрелкам вывод
 picy*=1.1;
 moveto((int)(xy[0][0]*picx)+xc,yc-
(int)(xy[0][1]*picy));
 for(i=1;i<MAX_POINT;i++)
 {lineto((int)(xy[i][0]*picx)+xc,yc-
(int)(xy[i][1]*picy));}
 break;
 case Down:
 //Стирание кривой повторным выводом
 moveto((int)(xy[0][0]*picx)+xc,yc-
(int)(xy[0][1]*picy));
 for(i=1;i<MAX_POINT;i++)
 {lineto((int)(xy[i][0]*picx)+xc,yc-
(int)(xy[i][1]*picy));}
 //Изменение масштаба и вывод
 picy/=1.1;
 moveto((int)(xy[0][0]*picx)+xc,yc-
(int)(xy[0][1]*picy));
 for(i=1;i<MAX_POINT;i++)
```

```
 {lineto((int)(xy[i][0]*picx)+xc,yc-
(int)(xy[i][1]*picy));}
 break;
 case Right:
 //Стирание кривой повторным выводом
 moveto((int)(xy[0][0]*picx)+xc,yc-
(int)(xy[0][1]*picy));
 for(i=1;i<MAX_POINT;i++)
 {lineto((int)(xy[i][0]*picx)+xc,yc-
(int)(xy[i][1]*picy));}
 //Изменение масштаба и вывод
 picx*=1.1;
 moveto((int)(xy[0][0]*picx)+xc,yc-
(int)(xy[0][1]*picy));
 for(i=1;i<MAX_POINT;i++)
 {lineto((int)(xy[i][0]*picx)+xc,yc-
(int)(xy[i][1]*picy));}
 break;
 case Left:
 //Стирание кривой повторным выводом
 moveto((int)(xy[0][0]*picx)+xc,yc-
(int)(xy[0][1]*picy));
 for(i=1;i<MAX_POINT;i++)
 {lineto((int)(xy[i][0]*picx)+xc,yc-
(int)(xy[i][1]*picy));}
 //Изменение масштаба и вывод
 picx/=1.1;
 moveto((int)(xy[0][0]*picx)+xc,yc-
(int)(xy[0][1]*picy));
 for(i=1;i<MAX_POINT;i++)
 {lineto((int)(xy[i][0]*picx)+xc,yc-
(int)(xy[i][1]*picy));}
 break;
 default:break;
 }
 }while(k!=Esc);
 //Дополнительное окно
 setviewport(0,Y>>2,X-scrwidth,Y,1);
 setcolor(6);                    //цвет рисования
 setfillstyle(1,7);              //заполнение
 setlinestyle(0,8,3);            //стиль линий
 bar(0,0,scrwidth,scrhight);
 rectangle(1,1,scrwidth-1,scrhight-1);     //рамка  в
окне
```

```
 settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
 char far* ps[]={"В этом окне можно дать",
                          "инструкцию                    по
использованию",
                          "настоящей программы"};
 for(i=0;i<3;i++)
 outtextxy(10,2*(i+1)*textheight(ps[0]),ps[i]);
 getch();
 }
```

4. Работа с массивами структур файлового хранения

Массивы или списки структур в файлах представляют собой обычно так называемые базы данных - многочисленные совокупности информационных блоков, содержащих возможно разнотипные, но связанные по смыслу и способам совместной обработки данные - это могут быть сведения о служащих некоторой фирмы, библиографический справочник, сведения об автомобилях и их владельцах в базе данных Госавтоинспекции региона, регистрационные данные о посетителях стоматологической поликлиники, операционные сведения за текущий день, с начала недели, месяца, года в коммерческом банке и многое другое.

В настоящем разделе мы рассмотрим методы работы с записями (структурами) файлового хранения на простом примере создания пополняемого толкового словаря. Реализующая его программа должна будет обеспечить предоставление пользователю следующих услуг: добавление записей в словарь, логическое и физическое удаление записей, поиск по ключевому слову, последовательный просмотр содержимого словаря и списка помеченных к удалению записей.

Каждая запись в словаре - это структура с двумя полями типа символьных массивов (строк), одно из которых содержит ключевое слово, а другое, большей длины, толкование этого слова.

Метод создания словаря будет ясен из приводимого ниже текста программы и сопровождающих его комментариев.

```
uses crt,dos;
```

```
(*Некоторые необходимые константы*)
const
  MAXWORDS=1000;(*максимальное количество слов в
словаре*)
  KEYLENGTH=20;(*длина слова*)
  VALUELENGTH=255;(*длина толкования*)
  NUMPUNKTS=8;(*количество пунктов в меню*)

(*в этой структуре хранятся основные справочные све-
дения в памяти*)
type shortrec=record
  IsDel:char;(*если ==0, запись пустая или помечена
к удалению*)
  key:string[KEYLENGTH];(*слово*)
end;
ashortrec=array [0..MAXWORDS-1] of shortrec;
pshortrec=^ashortrec;

(*эта структура для файлового хранения записей *)
type rec=record
  memrec:shortrec;(*признак занятости и слово-ключ*)
  value:string[VALUELENGTH];(*значение по данному
ключу*)
end;
frec=file of rec;

(*прототипы используемых процедур и функций*)
procedure add;forward;
procedure del;forward;
procedure find;forward;
procedure view;forward;
procedure pack;forward;
procedure undel;forward;
procedure dellist;forward;
procedure quit;forward;
procedure init;forward;
function findinmemory(key:string):boolean;forward;
function rcompare(f,s:string):boolean;forward;
function getnumber(key:string):longint;forward;
function rtolower(c:char):char;forward;

(*этот массив структур содержит список возможных
действий и реакцию на них*)
```

```
  type unknown=record
  msg:string[30];(*сообщение - пункт меню*)
  (*указатель на функцию, соответствующую данному
пункту*)
  func:procedure;
end;

const
message:array[0..NUMPUNKTS-1] of unknown=(
(*количество элементов в массиве равно количеству
пунктов меню*)
(msg:'0. Добавить запись';func:add),
(msg:'1. Удалить запись';func:del),
(msg:'2. Найти запись';func:find),
(msg:'3. Просмотр словаря';func:view),
(msg:'4. Упаковать словарь';func:pack),
(msg:'5. Отменить удаление';func:undel),
(msg:'6. Список удалённых';func:dellist),
(msg:'7. Выход';func:quit));

var
(*временная запись, используемая для промежуточных
действий*)
  temp:rec;
  dictionary:pshortrec;(*указатель на справочник в
памяти*)

(*процедура начальной инициализации*)
procedure init;
var
  fp:frec;
  i,RecordCount:longint;
begin
  (*отводим память под справочник*)
  GetMem(dictionary,MAXWORDS*sizeof(shortrec));
  if not longbool(dictionary) then(*если не удалось
отвести*)
  begin
  writeln('Не хватает памяти под справочник');
halt(1);
  end;
  (*очищаем память, выделенную под справочник*)
  Fill-
Char(dictionary^[0],MAXWORDS*sizeof(shortrec),0);
```

```
  (*открываем двоичный файл для чтения и записи*)
  {$I-}
  (*проверка на существование*)
  Assign(fp,ParamStr(1)); FileMode:=2; Reset(fp);
  if ioresult<>0 then
  begin
    Assign(fp,ParamStr(1));      FileMode:=2;      Re-
Write(fp);
    if IOResult<>0 then(*нет места, защита от записи
etc*)
    begin
       writeln('Не могу открыть файл ',ParamStr(1));
       (*освобождаем память из под справочника*)
    FreeMem(dictionary,MAXWORDS*sizeof(shortrec));
halt(0);
    end;
    close(fp);
    FileMode:=2;Reset(fp);(*переоткрываем вновь со-
зданный файл*)
  end;
  (*определяем число пар слов-значение в словаре*)
  RecordCount:=FileSize(fp);
  for i:=0 to RecordCount-1 do
  begin (*и считываем слова в справочную часть*)
    read(fp,temp);
    dictionary^[i]:=temp.memrec;
  end;
  close(fp);(*закрываем файл*)
  {$I+}
end;

(*Процедура добавления записи*)
procedure add;
var
  fp:frec;  i:longint;
begin
  {$I-}
  Assign(fp,ParamStr(1));  FileMode:=2; Reset(fp);
  FillChar(temp,sizeof(rec),0);(*очищаем рабочую за-
пись*)
  writeln('Введите слово : ');(*запрашиваем ключевое
слово*)
  readln(temp.memrec.key);
  (*если его нет в словаре*)
```

```
  if( not findinmemory(temp.memrec.key))then
  begin
(*вводим с клавиатуры значение с ограничением на
длину*)
writeln('Введите значение этого слова (Enter - конец
ввода) :');
readln(temp.value);
temp.memrec.IsDel:='A';(*признак неудалённости запи-
си*)
for i:=0 to MAXWORDS-1 do(*дрейф по справочнику в
памяти*)
if not bytebool(dictionary^[i].IsDel) then
       (*в поисках пустой (удалённой) ячейки*)
       begin (*заносим слово в память*)
         dictionary^[i]:=temp.memrec;
         Seek(fp,i);
         write(fp,temp);(*а вместе со значением - на
диск*)
         break;
       end;
    if i=MAXWORDS then
    writeln('Нет места в словаре для добавления но-
вого слова');
  end
  else(*если нашли слово в памяти*)
  writeln('Такое слово уже есть в словаре. Попробуй-
те изменить его значение');
  readkey;
  close(fp);
  {$I+}
end;

(*функция поиска слова в справочнике*)
function findinmemory(key:string):boolean;
var
  i:longint;
begin
(*слово считается найденным, если оно не помечено к
удалению и совпадает с искомым с точностью до реги-
стра*)
  for i:=0 to MAXWORDS-1 do
    if(bytebool(dictionary^[i].IsDel))and
    ( not rcompare(key,dictionary^[i].key)) then
    begin
```

```
        findinmemory:=boolean(1); exit;
    end;
  findinmemory:=false;
end;

(*сравнение двух строк без учёта регистра по
аналогии с stricmp, но с проверкой на русские бук-
вы*)

function rcompare(f,s:string):boolean;
var
  i:byte;
begin
  if(Length(f)<>Length(s))then
  begin
  rcompare:=true;  exit;
  end;
  for i:=1 to Length(f) do
    if (rtolower(f[i])<>rtolower(s[i])) then
    begin
       rcompare:=true;   exit;
    end;
  rcompare:=boolean(0);
end;

(*переводит символ в нижний регистр, возвращая ре-
зультат*)
function rtolower(c:char):char;
var
  d:byte;
begin
  d:=byte(c);
  if(d>=byte('A'))and(d<=byte('Z')) then inc(d,$20);
  if(d>=byte('A'))and(d<=byte('П')) then inc(d,$20);
  if(d>=byte('P'))and(d<=byte('Я')) then inc(d,$50);
  if(d=byte('Ё')) then d:=byte('ё');
  rtolower:=char(d);
end;

(*процедура удаления слова*)
procedure del;
var
  i:longint;  fp:frec;
begin
```

```
  {$I-}
  Assign(fp,ParamStr(1)); FileMode:=2;  Reset(fp);
  FillChar(temp,sizeof(rec),0);(*очищаем    временную
запись*)
  writeln('Введите слово : ');
  readln(temp.memrec.key);    (*вводим      удаляемое
слово*)
  if(findinmemory(temp.memrec.key))(*если     таковое
имеется*)then
  begin
    (*получаем номер записи с данным словом*)
    i:=getnumber(temp.memrec.key);
    dictionary^[i].IsDel:=#0;(*устанавливаем  признак
удаления*)
    Seek(fp,i);     (*позиционируемся  на  эту  запись
в файле*)
    read(fp,temp);    temp.memrec.IsDel:=#0;
    Seek(fp,i);     (*позиционируемся  на  эту  запись
в файле*)
    write(fp,temp);     (*записываем  с  новым  призна-
ком*)
  end
  else
    writeln('Такого слова в словаре нет');
  readkey;
  close(fp);
  {$I+}
end;

(*функция, ищущая слово в памяти и возвращающая но-
мер записи с ним*)

function getnumber(key:string):longint;
var
  i:word;
begin
  getnumber:=-1;   (*а может и не найдём...*)
  for i:=0 to MAXWORDS-1 do(*дрейфуем по словам до
тех пор,*)
    if( not rcompare(key,dictionary^[i].key))then
    begin         (*пока не найдём; тогда*)
    getnumber:=i;  (*возвращаем номер*)   exit;
    end;
end;
```

```
(*если удалённая запись ещё не затёрта, признак
удаления с неё можно снять*)

procedure undel;
var
  fp:frec;
  i:integer;
begin
  {$I-}
  Assign(fp,ParamStr(1));  FileMode:=2; Reset(fp);
  FillChar(temp,sizeof(rec),0);(*очищаем рабочую за-
пись*)
  writeln('Введите слово : ');
  (*вводим предположительно удалённое слово*)
  readln(temp.memrec.key);
  for i:=0 to MAXWORDS-1 do     (*ищем только среди
удалённых*)
    if(                                          not
rcompare(dictionary^[i].key,temp.memrec.key))
         and(   not   boolean(dictionary^[i].IsDel))
then
    begin      (*убираем признак удалённости*)
       dictionary^[i].IsDel:='R';
       Seek(fp,i);         (*позиционируемся на уда-
лённую запись*)
       read(fp,temp);
       temp.memrec.IsDel:='R';
       Seek(fp,i);       (*позиционируемся   на   эту
запись в файле*)
       write(fp,temp);  (*и восстанавливаем её*)
       writeln('Восстановлено!');
       i:=-1;            (*признак восстановления*)
       break;
    end;
  if wordbool(i+1) then(*если не удалось восстано-
вить*)
    writeln('Такого слова в словаре не было');
  readkey;
  close(fp);
  {$I+}
end;
```

```
(*поиск всех слов, содержащих заданное как подстро-
ку*)
procedure find;
var
  key:string[KEYLENGTH];
  j,i,count,RecordCount:longint;
  fp:frec;
begin
  {$I-}
  Assign(fp,ParamStr(1));  FileMode:=0;  Reset(fp);
  FillChar(temp,sizeof(rec),0);(*обнуляем     рабочую
запись*)
  writeln('Введите слово : ');
  readln(key);(*вводим подстроку для поиска*)
  count:=0;
  for i:=1 to byte(key[0]) do
    key[i]:=rtolower(key[i]);(*преобразуем к нижнему
регистру*)
  RecordCount:=FileSize(fp);  (*определяем число за-
писей*)
  for i:=0 to RecordCount-1 do
  begin
    read(fp,temp);
    for j:=1 to byte(temp.memrec.key[0]) do

temp.memrec.key[j]:=rtolower(temp.memrec.key[j]);
    if bytebool(pos(key,temp.memrec.key)) and
    bytebool(temp.memrec.IsDel) then
    begin
       writeln('Слово - ',temp.memrec.key,',
                  значение - ',temp.value);
       inc(count);
    end;
  end;
  if not longbool(count) then
    writeln('Такого слова в словаре нет');
  readkey;
  close(fp);
  {$I+}
end;

(*вывод на экран всего словаря*)
procedure view;
var
```

```
  i,count:longint;  fp:frec;
begin
  {$I-}
  Assign(fp,ParamStr(1)); FileMode:=0; Reset(fp);
  writeln('Содержимое словаря');
  FillChar(temp,sizeof(rec),0);
  count:=0;  i:=0;
  while (i<MAXWORDS) and not eof(fp) do
  begin
  if bytebool(dictionary^[i].IsDel)(*если слово не
удалено*)then
  begin
  Seek(fp,i);  read(fp,temp);(*считываем его значе-
ние*)
inc(count);
writeln(count,' Слово - ',temp.memrec.key,',
значение - ',temp.value);
readkey;(*печатаем и даём полюбоваться до нажатия
клавиши*)
   end;
    inc(i);
  end;
  writeln('Всего записей : ',count);(*вывод стати-
стики*)
  readkey;
  close(fp);
  {$I+}
end;

  (*вывести список удалённых словарных единиц*)
procedure dellist;
var
  i,count:longint;
begin
(*пара слово-значение считается удалённой, если сло-
во есть, а признак удаления установлен*)
  writeln('Список удалённых слов:');
  count:=0;
  for i:=0 to MAXWORDS-1 do
    if not bytebool(dictionary^[i].IsDel)and
        bytebool(Length(dictionary^[i].key)) then
    begin
       inc(count);
```

```
       writeln(count,'             слово             -
',dictionary^[i].key);
    end;
  writeln('Всего       удалённых       записей       :
',count);(*статистика*)
  readkey;
end;

(*Записи, помеченные к удалению, продолжают занимать
место, увеличивая размер файла, а потому их имело бы
смысл физически удалить*)

procedure pack;
var
  RecordCount,i:longint;
  fp,ff:frec;
begin
  {$I-}
  write('Упаковываю...');
  Assign(fp,ParamStr(1));  FileMode:=0;  Reset(fp);
  Assign(ff,'vocab.tmp');       FileMode:=1;       Re-
Write(ff);
  RecordCount:=FileSize(fp);
  (*переписываем только существующие записи*)
  for i:=0 to RecordCount-1 do
  begin
    read(fp,temp);
    if bytebool(temp.memrec.IsDel) then
       write(ff,temp);
  end;
  close(fp); close(ff);
  {$I+}
  Erase(fp);  Rename(ff,ParamStr(1));
  writeln(#8#8#8#8#8#8#8'ано!   ');
  readkey;
end;

(*Процедур выхода с освобождением памяти*)
procedure quit;
begin
  (*освобождаем выделенную память*)
  FreeMem(dictionary,MAXWORDS*sizeof(shortrec));
halt(0);
end;
```

```
var
  i:integer; what:byte;
begin
  (*в качества параметра передадим имя словаря*)
  if ParamCount<1 then
  begin
    writeln('Используйте:     ',ParamStr(0),'     vo-
cab.dat'); exit;
  end;
  init;(*начальная инициализация*)
  directvideo:=boolean(0);
  (*в бесконечном цикле обработки сообщений от кла-
виатуры*)
  while true do
  begin
    writeln;(*выводим пустую строку перед списком*)
    for i:=0 to -1+NUMPUNKTS do(*распечатываем все
пункты меню*)
       writeln(message[i].msg);
    write('Ваш выбор : ');(*приглашение к диалогу*)
    (*ввод с отсечением недопустимых номеров пунктов
меню*)
    repeat
       what:=byte(readkey);
    until                  (what-ord('0')>=0)and(what-
ord('0')<NUMPUNKTS);
    dec(what,ord('0'));(*преобразуем цифровой символ
в номер*)
    (*печатаем, что выбрали*)

writeln(Copy(message[what].msg,3,Length(message[what
].msg)-2));
    message[what].func;(*и  вызываем  соответствующую
функцию*)
  end;
end.
```

Литература

1. Поляков Д.Б., Круглов И.Ю. Программирование в среде Турбо Паскаль (версия 5.5.). - М.: МАИ, А/О "РосВузНаука", 1992.
2. Полищук А.П. Информатика. Персональный компьютер и его программирование (С, С++, Паскаль). – Кривой Рог, 1997.

Підп. до друку 26.09.98
Друк №3. Друк офсетний
Умовн. фарбо-відб. 2,47
Тираж 50

Формат 80х84 1/16.
Умовн. друк. арк. 2,6
Зам. №9-3978

КДПІ, 324086, Кривий Ріг-86, пр. Гагаріна, 54

Криворізька міська друкарня
324050, Кривий Ріг-50, пр. Металургів, 28.