Вводим команду "T" снова:.

```
AX=0148 BX=0000 CX=0010 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=11B7 ES=11B7 SS=11B7 CS=11B7 IP=0106 NV UP EI PL NZ NA PE NC
11B7:0106 8BD8 MOV BX,AX
```

AX=0148 — "прибавить значение 0025h к AX". Сделали? Сделали!!

Вводим команду "T" снова:.

```
AX=0148 BX=0148 CX=0010 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=11B7 ES=11B7 SS=11B7 CS=11B7 IP=0108 NV UP EI PL NZ NA PE NC
11B7:0108 03D8 ADD BX,AX
```

AX=0148=BX — "переслать содержимое AX в BX". Сделали? Сделали!!

Вводим команду "T" снова:

```
AX=0148 BX=0290 CX=0010 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=11B7 ES=11B7 SS=11B7 CS=11B7 IP=010A NV UP EI PL NZ AC PE NC
11B7:010A 8BCB MOV CX,BX
```

"Прибавить содержимое AX к BX". Оно? А то!

Вводим команду "T" снова:

```
AX=0148 BX=0290 CX=0290 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=11B7 ES=11B7 SS=11B7 CS=11B7 IP=010C NV UP EI PL NZ AC PE NC
11B7:010C 31C0 XOR AX,AX
```

"Переслать содержимое BX в CX". Сделано!

Вводим команду "T" снова:

```
AX=0000 BX=0290 CX=0290 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=11B7 ES=11B7 SS=11B7 CS=11B7 IP=010E NV UP EI PL ZR NA PE NC
11B7:010E CD20 INT 20
```

"Очистка AX"? И точно AX=0000!

Вводим команду "T" снова... И ГРОМКО РУГАЕМСЯ!!

Потому что, по идее, сейчас наша программа должна была завершиться — у нас же там код выхода прописан, а она куда лезет? NOP'ы какие-то (если продолжать команду "T" вводить), CALL 1085 (да вы продолжайте "трассировку", продолжайте!)

Для тех, кому лень продолжать жать на букву "T", введите для разнообразия команду "G" (от английского — GO). На монитор должна вывалиться надпись "Нормальное завершение работы программы".

"Уф, — должны сказать вы — Работает!"

А то!

#9. Только непонятно вот, почему вдруг между int 20 (CD 20) и надписью "Нормальное завершение работы программы" куча всяких "левых" непонятных команд (в том случае, если вы и дальше производили трассировку, а не воспользовались "халявной" командой "G")?

А потому, дорогие наши, что вы имели счастье нарваться на ***прерывание*** (interrupt)!

Понимаете ли, завершить программу — дело непростое :). Нужно восстановить первоначальное значение регистров, восстановить переменные среды и кучу всего другого! Знаете, как это сложно?

Однако эта процедура насколько сложная, настолько и ***типичная*** для исполняемых программ. А посему, разработчики операционной системы решили избавить программистов от необходимости делать это вручную, и включили эту стандартную процедуру в ядро операционной системы. И сказали: "да будешь ты (процедура обработки прерывания) вызываться как int 20, и будешь ты обеспечивать корректную передачу управления из выполняемой программы — назад в ядро". И стало так...

Ну посудите сами, должна же операционная система ну хоть что-нибудь делать!!

Литература

1. HI-TECH. Низкоуровневое программирование. — Радиомир. Ваш компьютер, 2001, №9, С.24.

(Продолжение следует)

Программирование циклических алгоритмов

А.ШАКИРИН,
г.Минск, БАТУ, каф.ВТ.

Цель этой статьи — освоить простейшие средства отладки модулей проекта и создать приложение, которое использует циклический алгоритм.

1. Отладка модулей проекта

Отладка представляет собой процесс обнаружения, локализации и устранения ошибок в проекте. Она занимает значительную часть рабочего времени программиста, нередко большую, чем разработка проекта.

Практически любой нетривиальный проект перед началом отладки содержит хотя бы одну синтаксическую или логическую ошибку.

1.1. Отладка синтаксических ошибок

Синтаксические ошибки заключаются в нарушении формальных правил использования операторов. Эти ошибки появляются в результате недостаточного знания разработчиком языка программирования и невнимательности при наборе операторов на экране дисплея.

Поиск синтаксических ошибок в модулях проекта осуществляется компилятором. Чтобы дать программисту как можно больше информации об ошибках, допущенных в модуле, компилятор отмечает ошибки и продолжает работу до тех пор, пока не будут обработаны все операторы модуля. Следует иметь в виду, что:

- компилятор распознает *не все ошибки*;
- некоторые ошибки могут повлечь за собой то, что правильные операторы будут восприниматься компилятором как ошибочные, и наоборот — ошибочные операторы компилятор воспримет как верные;
- ошибка в одном месте модуля может повлечь за собой серию диагностических сообщений компилятора в других местах модуля;
- из-за некоторых ошибок компиляция модуля может вообще прекращаться, и проверка последующих операторов не производится.

Информация обо всех ошибках, найденных в модуле, выводится в специальное окно, которое появляется в нижней части экрана. Каждая строка этого окна содержит имя файла, номер строки, в которой обнаружена ошибка, и характер ошибки. Если дважды щелкнуть мышью на строке с описанием ошибки, курсор установится в той строке модуля, где обнаружена ошибка. Следует исправлять ошибки последовательно, сверху вниз, и после исправления каждой ошибки компилировать программу заново. С целью сокращения

времени компиляции рекомендуется осуществлять проверку наличия ошибок в режимах Syntax Check и Compile меню Project. Для получения более полной информации о характере ошибки можно обратиться к HELP нажатием клавиши F1.

Отладка синтаксиса считается завершенной, когда после очередной компиляции в режиме Build All меню Project отсутствуют диагностические сообщения об ошибках.

1.2. Отладка логических ошибок

Логические ошибки условно можно разделить на ошибки алгоритма и семантические ошибки. Причинами таких ошибок могут быть: несоответствие алгоритма поставленной задаче, неправильное понимание программистом смысла (семантики) операторов языка программирования, нарушение допустимых пределов и правил представления данных, невнимательность при технической подготовке проекта к обработке на компьютере.

Для выявления ошибок служат *тесты*. Тест — это такой набор исходных данных, который дает результат, не вызывающий сомнений. Промежуточные и конечные результаты теста используются для контроля правильности выполнения приложения.

Составление тестов — непростая задача. Тесты должны быть, с одной стороны, достаточно простыми, чтобы результат легко проверялся, с другой — достаточно сложными, чтобы комплексно проверить алгоритм.

Тесты составляются по схеме алгоритма *до программирования*, так как составление тестов помогает выявить многие ошибки в алгоритмизации.

Количество тестов и их сложность зависят от алгоритма. Комплекс тестов должен быть таким, чтобы все ветви схемы алгоритма были пройдены, по крайней мере, по одному разу.

Несовпадение результатов, выдаваемых приложением, с результатами тестов — признак наличия ошибок. Эти ошибки проявляются в том, что результат расчета оказывается неверным, либо происходит переполнение, деление на 0 и др.

Для локализации места ошибки рекомендуется поступать следующим образом. В окне Редактора Кода установите курсор в строке перед подозрительным участком и нажмите клавишу F4 (выполнить до курсора). Выполнение приложения будет остановлено на той строке модуля, в которой был установлен курсор. Текущее значение любой переменной можно увидеть, если накрыть курсором идентификатор переменной на 1...2 с. Нажимая клавишу F8 (пошаговое выполнение), можно построчно выполнять программу, контролируя содержимое переменных и правильность вычислений.

2. Пример создания приложения

Необходимо создать Windows-приложение, которое выводит таблицу значений функции

$$Y(x) = (1 - \frac{x^2}{2})\cos x - \frac{x}{2}\sin x$$

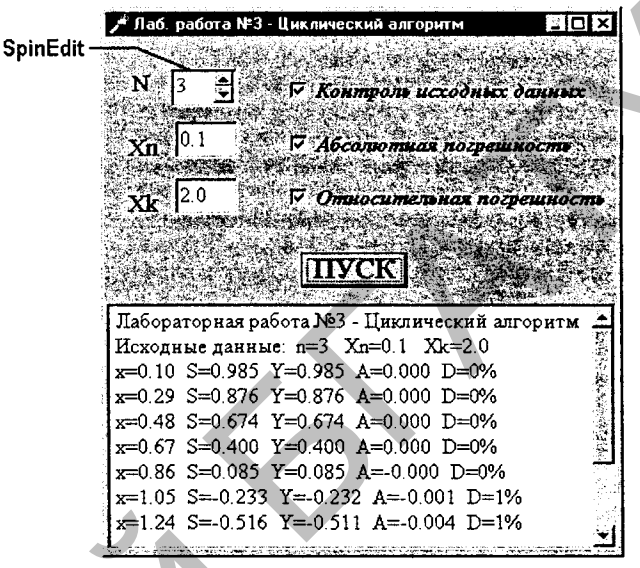
и ее разложение в ряд в виде суммы

$$S(x) = \sum_{n=0}^{n} (-1)^n \frac{2n^2+1}{(2n)!} x^{2n}$$

для значений x от x_n до x_k с шагом $h=(x_k - x_n)/10$.

В панели интерфейса следует предусмотреть возможность управления выводом исходных данных и погрешности вычислений.

Один из возможных вариантов панели интерфейса создаваемого приложения показан на рисунке.



2.1. Размещение компонентов на Форме

Вместо компонента Edit используем компонент SpinEdit, который обеспечивает отображение и редактирование целого числа с возможностью его изменения посредством двойной кнопки.

Компонент SpinEdit находится на странице Samples Палитры Компонентов. В тех случаях, когда объем выводимой информации превышает размер поля компонента Memo, целесообразно снабдить его линейками прокрутки. В свойстве ScrollBars компонента Memo1 установим значение ssVertical — появится вертикальная линейка прокрутки. Присвоим модулю имя UnCiklAlg.

2.2. Текст модуля UnCiklAlg

```
Unit UnCiklAlg;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls, Spin;
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    SpinEdit1: TSpinEdit;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    CheckBox3: TCheckBox;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
```

```
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
  SpinEdit1.text:='3'; // начальное значение N
  Edit1.text:='0.1';   // начальное значение Xn
  Edit2.text:='2.0';   // начальное значение Xk
  Memo1.Clear;
  Memo1.Lines.Add('Лабораторная работа №3 - Циклический алгоритм');
end;
procedure TForm1.Button1Click(Sender: TObject);
 var xn,xk,x,h,c,s,y,a1,del:extended;
     n,k:integer;
begin
 n:=StrToInt(SpinEdit1.Text);
 xn:=StrToFloat(Edit1.Text);
 xk:=StrToFloat(Edit2.Text);
 if CheckBox1.Checked then
   Memo1.Lines.Add('Исходные данные:  n='+IntToStr(n)+
     '   Xn='+FloatToStrF(xn,ffFixed,6,1)+
     '   Xk='+FloatToStrF(xk,ffFixed,6,1));
 h:=(xk-xn)*0.1;          // шаг h
 x:=xn;
 repeat                   // цикл по x
  c:=-x*x*0.5;
  S:=1;
  for k:=1 to n do
   begin
    s:=s+c*(2*k*k+1);
   c:=-c*x*x/((2*k+1)*(2*k+2));
   end;
  y:=(1-x*x*0.5)*cos(x)-0.5*x*sin(x);
  if CheckBox2.Checked then
    if CheckBox3.Checked then
      begin
       a1:=s-y;                  // абсолютная погрешность
       del:=abs((s-y)/y)*100;    // относительная погрешность
       Memo1.Lines.Add('x='+FloatToStrF(x,ffFixed,6,2)+
                  '   S='+ FloatToStrF(s,ffFixed,6,3)+
                  '   Y='+ FloatToStrF(y,ffFixed,6,3)+
                  '   A='+ FloatToStrF(a1,ffFixed,6,3)+
                  '   D='+ FloatToStrF(del,ffFixed,6,0)+'%');
     end
                                        else
     begin
      a1:=s-y;
  Memo1.Lines.Add('x='+FloatToStrF(x,ffFixed,6,2)+
                  '   S='+ FloatToStrF(s,ffFixed,6,3)+
                  '   Y='+ FloatToStrF(y,ffFixed,6,3)+
                  '   A='+ FloatToStrF(a1,ffFixed,6,3));
     end
                                       else
    if CheckBox3.Checked then
      begin
      del:=abs((s-y)/y)*100;
    Memo1.Lines.Add('x='+FloatToStrF(x,ffFixed,6,2)+
                  '   S='+ FloatToStrF(s,ffFixed,6,3)+
                  '   Y='+ FloatToStrF(y,ffFixed,6,3)+
                  '   D='+ FloatToStrF(del,ffFixed,6,0)+'%');
     end
                                          else
       Memo1.Lines.Add('x='+FloatToStrF(x,ffFixed,6,2)+
                  '   S='+ FloatToStrF(s,ffFixed,6,3)+
                  '   Y='+ FloatToStrF(y,ffFixed,6,3));
   x:=x+h;
  until x>xk;
end;
end.
```

3. Индивидуальные задания

Необходимо создать Windows-приложение для вычисления соответствующего выражения из таблицы, протестировать его работу и вывести на экран таблицу значений функции $Y(x)$ и ее разложения в ряд $S(x)$ для значений x от x_n до x_k с шагом $h=(x_k - x_n)/10$. Близость значений $S(x)$ и $Y(x)$ во всем диапазоне значений x указывает на правильность вычисления $S(x)$ и $Y(x)$.

№	x_n	x_k	S(x)	n	Y(x)
1	0,1	1	$x-\frac{x^3}{3!}+\ldots+(-1)^n\frac{x^{2n+1}}{(2n+1)!}$	16	$\sin x$
2	0,1	1	$1+\frac{x^2}{2!}+\ldots+\frac{x^{2n}}{(2n)!}$	10	$\frac{e^x+e^{-x}}{2}$
3	0,1	1	$1+\frac{\cos\frac{\pi}{4}}{1!}x+\ldots+\frac{\cos n\frac{\pi}{4}}{n!}x^n$	12	$e^{x\cos\frac{\pi}{4}}\cos(x\sin\frac{\pi}{4})$
4	0,1	1	$1-\frac{x^2}{2!}+\ldots+(-1)^n\frac{x^{2n}}{(2n)!}$	8	$\cos x$
5	0,1	1	$1-\frac{x^2}{1!}+\frac{x^4}{2!}-\frac{x^6}{3!}+\ldots+(-1)^n\frac{x^{2n}}{n!}$	14	e^{-x^2}
6	0,1	1	$x+\frac{x^3}{3!}+\ldots+\frac{x^{2n+1}}{(2n+1)!}$	8	$\frac{e^x-e^{-x}}{2}$
7	0,1	1	$\frac{x^3}{3}-\frac{x^5}{15}+\ldots+(-1)^{n+1}\frac{x^{2n+1}}{4n^2-1}$	12	$\frac{1+x^2}{2}arctgx-\frac{x}{2}$
8	0,1	1	$1+\frac{2x}{1!}+\ldots+\frac{(2x)^n}{n!}$	10	e^{2x}
9	0,1	1	$1+2\frac{x}{2}+\ldots+\frac{n^2+1}{n!}\left(\frac{x}{2}\right)^n$	14	$\left(\frac{x^2}{4}+\frac{x}{2}+1\right)e^{\frac{x}{2}}$
10	0,1	0,5	$x-\frac{x^3}{3}+\ldots+(-1)^n\frac{x^{2n+1}}{2n+1}$	15	$arctgx$
11	0,1	0,8	$\frac{x^2}{2}-\frac{x^4}{12}+\ldots+(-1)^{n+1}\frac{x^{2n}}{2n(2n-1)}$	10	$xarctgx-\ln\sqrt{1+x^2}$
12	0,1	1	$-\frac{(2x)^2}{2}+\frac{(2x)^4}{4}-\ldots+(-1)^n\frac{(2x)^{2n}}{(2n)!}$	8	$1(\cos^2 x-1)$

Литература

1. В.В.Феофанов. Delphi 3. Учебный курс. — М.: Нолидж, 1981.
2. Э.Возневич. Delphi. Освой самостоятельно. — М.: Восточная книжная компания, 1996.
3. Дж.Матчо, Д.Р.Фолкнер. Delphi. — М.: БИНОМ, 1995.
4. М.Канту. Delphi 2 для Windows 95/NT. — М.: ООО "Малип", 1997.