```
1  //---------------------------------------------------------------------
2
3  #pragma hdrstop
4
5  #include "evaluador.h"
6  #include <ctype.h>
7  #include <string.h>
8  #include <math.h>
9  #include <stdlib.h>
10 #include <stdio.h>
11
12 class STACKNODE
13 {
14      public:
15      STACKNODE<S>* Next;
16      S Data;
17
18      STACKNODE(STACKNODE<S>* pNext,S pData);
19 };
20
21 template <class S>
22 class STACK
23 {
24      STACKNODE<S>* T;
25      int N;
26
27      public:
28      STACK (void);
29      void Push (S pData);
30      S Pop (void);
31      S Top (void);
32      int IsEmpty (void);
33      void Flush (void);
34 };
35
36 template <class S>
37 STACKNODE<S>::STACKNODE(STACKNODE<S>* pNext,S pData)
38 {
39      Next= pNext;
40      Data= pData;
41 }
42
43 template <class S>
44 STACK<S>::STACK (void)
45 {
46      T= 0;
47      N= 0;
48 }
49
50 template <class S>
51 void STACK<S>::Push (S pData)
52 {
53      STACKNODE<S>* Tmp= new STACKNODE<S>(T,pData);
```

```
54
55        N++;
56        T= Tmp;
57    }
58
59    template <class S>
60    S STACK<S>::Pop (void)
61    {
62        STACKNODE<S>* Tmp;
63        S Data= 0;
64
65        if (N>0)
66        {
67            Tmp = T;
68            T = T->Next;
69            Data= Tmp->Data;
70            delete Tmp;
71            N--;
72        }
73
74        return (Data);
75    }
76
77    template <class S>
78    S STACK<S>::Top (void)
79    {
80        return(T->Data);
81    }
82
83    template <class S>
84    int STACK<S>::IsEmpty (void)
85    {
86        return(N==0);
87    }
88
89    template <class S>
90    void STACK<S>::Flush (void)
91    {
92        STACKNODE<S>* Tmp;
93
94        while (N>0)
95        {
96            Tmp= T;
97            T = T->Next;
98            delete Tmp;
99            N--;
100       }
101   }
102
103   //---------------------------------------------------------
104
105   const char DecimalPoint= '.';
106   STACK<int> S;
```

```
107
108  int IsNumeral (int c)
109  {
110       return (isdigit(c)||(c==DecimalPoint));
111  }
112
113  int IsOperator (int c)
114  {
115       switch (c)
116       {
117            case '+':
118            case '-':
119            case '*':
120            case '/':
121            case '~':
122            case '^':
123                 return (1);
124
125            default:
126                 return (0);
127     }
128  }
129
130  #define COS    'A'
131  #define SEN    'B'
132  #define TAN    'C'
133  #define SENH   'D'
134  #define COSH   'E'
135  #define TANH   'F'
136  #define ACOS   'G'
137  #define ASEN   'H'
138  #define ATAN   'I'
139  #define LOG    'J'
140  #define LN     'K'
141  #define EXP    'L'
142  #define SQRT   'M'
143
144  #define NF      13
145
146
147  struct FUNCTIONS
148  {
149       char Text[5];
150       char Token;
151  } Functions[]= {"senh",SENH,
152                  "cosh",COSH,
153                  "tanh",TANH,
154                  "acos",ACOS,
155                  "asen",ASEN,
156                  "atan",ATAN,
157                  "raiz",SQRT,
158                  "cos",COS,
159                  "sen",SEN,
```

```
160                  "tan",TAN,
161                  "log",LOG,
162                  "exp",EXP,
163                  "ln",LN};
164
165
166  int IsFunction (char* pExp)
167  {
168       int n,m;
169
170       for (n=0;n<NF;n++)
171       {
172            for (m=0;(unsigned)m<strlen(Functions[n].Text);m++)
173                 if (Functions[n].Text[m]!=pExp[m])
174                      break;
175
176            if (m==strlen(Functions[n].Text))
177                 return Functions[n].Token;
178       }
179
180       return 0;
181  }
182
183  int IsFunction (int c)
184  {
185       int n;
186
187       for (n=0;n<NF;n++)
188            if (c==Functions[n].Token)
189                 return 1;
190
191       return 0;
192  }
193
194
195  int GoEP (int pOP1,int pOP2)
196  {
197       int r;
198
199       switch (pOP1)
200       {
201            case '*':
202            case '/':
203            case '%':
204            case '~':
205            case '^':
206                 r= 1;
207                 break;
208
209            case '+':
210            case '-':
211                 switch (pOP2)
212                 {
```

```
213                     case '*':
214                     case '/':
215                     case '%':
216                     case '~':
217                     case '^':
218                          r= 0;
219                          break;
220
221                     case '+':
222                     case '-':
223                          r= 1;
224                          break;
225                }
226          }
227          return(r);
228     }
229
230     void ConvertToPostFix (char *pInFix,char *pPostFix)
231     {
232          int IFc,PFc,oFlg= 1;
233
234          S.Push('(');
235          strcat(pInFix,")");
236
237          for (IFc=PFc=0;!S.IsEmpty();)
238          {
239               if (IsNumeral(pInFix[IFc]))
240               {
241                    pPostFix[PFc++]= pInFix[IFc++];
242                    oFlg= 0;
243               }
244               else
245               {
246                    if (pInFix[IFc]=='(')
247                    {
248                         S.Push('(');
249                         IFc++;
250                    }
251               else if (IsFunction(pInFix[IFc]))
252                    {
253                         pPostFix[PFc++]= ' ';
254                         S.Push(pInFix[IFc++]);
255                         oFlg= 1;
256                    }
257                    else if (IsOperator(pInFix[IFc]))
258                         {
259                              if ((oFlg)&&(pInFix[IFc]=='-'))
260                              {
261                                   pPostFix[PFc++]= '0';
262                                   pPostFix[PFc++]= ' ';
263                                   S.Push('~');
264                                   IFc++;
265                              }
```

```
266                         else
267                         {
268                              pPostFix[PFc++]= ' ';
269                              while ((IsOperator(S.Top()))&& (GoEP(S.Top(),pInFix[IFc])))
270                              {
271                                   pPostFix[PFc++]= S.Pop();
272                                   pPostFix[PFc++]= ' ';
273                              }
274                              S.Push(pInFix[IFc++]);
275                              oFlg= 1;
276                         }
277                    }
278                    else if (pInFix[IFc]==')')
279                         {
280                              while (S.Top()!='(')
281                              {
282                                   pPostFix[PFc++]= ' ';
283                                   pPostFix[PFc++]= S.Pop();
284                              }
285
286                              S.Pop();
287
288                              if (!S.IsEmpty())
289                                   if (IsFunction(S.Top()))
290                                   {
291                                        pPostFix[PFc++]= ' ';
292                                        pPostFix[PFc++]= S.Pop();
293                                   }
294
295                              IFc++;
296                         }
297                         else IFc++;
298                    }
299          }
300          pPostFix[PFc]= '\0';
301     }
302
303     STACK<double> R;
304
305     double DoOperation (double pOP1,double pOP2, int pOP)
306     {
307          double R;
308
309          switch (pOP)
310          {
311               case '+':
312                    R= pOP1+pOP2;
313                    break;
314               case '-':
315               case '~':
316                    R= pOP1-pOP2;
317                    break;
318               case '*':
```

```
319                 R= pOP1*pOP2;
320                 break;
321           case '/':
322                 R= pOP1/pOP2;
323                 break;
324           case '^':
325                 R= pow(pOP1,pOP2);
326                 break;
327     }
328     return (R);
329 }
330
331
332 double DoFunction (double pOP,int pFunc)
333 {
334     double V= 0;
335
336     switch (pFunc)
337     {
338           case COS:
339                 V= cos(pOP);
340                 break;
341
342           case SEN:
343                 V= sin(pOP);
344                 break;
345
346           case TAN:
347                 V= tan(pOP);
348                 break;
349
350           case COSH:
351                 V= cosh(pOP);
352                 break;
353
354           case SENH:
355                 V= sinh(pOP);
356                 break;
357
358           case TANH:
359                 V= tanh(pOP);
360                 break;
361
362           case ACOS:
363                 V= acos(pOP);
364                 break;
365
366           case ASEN:
367                 V= asin(pOP);
368                 break;
369
370           case ATAN:
371                 V= atan(pOP);
```

```
372                 break;
373
374           case LOG:
375                 V= log10(pOP);
376                 break;
377
378           case LN:
379                 V= log(pOP);
380                 break;
381
382           case EXP:
383                 V= exp(pOP);
384                 break;
385
386           case SQRT:
387                 V= sqrt(pOP);
388                 break;
389     }
390
391     return V;
392 }
393
394 void PreEvaluation (char* pExp,char* pNewExp,double pValue)
395 {
396     int n,i,F;
397     char x[10]={0};
398
399     sprintf(x,"%f",pValue);
400     pNewExp[0]= '\0';
401
402     for (n=i=0;(unsigned)n<strlen(pExp);)
403     {
404           if (IsNumeral(pExp[n])||IsOperator(pExp[n])||(pExp[n]=='(')||(pExp[n]==')'))
405                 pNewExp[i++]= pExp[n++];
406           else if (F=IsFunction(&(pExp[n])))
407                 {
408                       pNewExp[i++]= F;
409                       n+= 3;
410                 }
411                 else if (toupper(pExp[n])=='X')
412                       {
413                             strcat(pNewExp,x);
414                             i+= strlen(x);
415                             n++;
416                       }
417                       else if (pExp[n]==',')
418                             {
419                                   pNewExp[i++]= DecimalPoint;
420                                   n++;
421                             }
422                             else n++;
423     }
424     pNewExp[i]= '\0';
```

```
425  }
426
427  double Evaluate (char* pExp,double pValue)
428  {
429       char InFix[255]={0},PostFix[255]={0};
430       char *Tmp;
431       char *E;
432
433       PreEvaluation(pExp,InFix,pValue);
434       ConvertToPostFix(InFix,PostFix);
435
436
437       Tmp= strtok(PostFix," ");
438       while (Tmp!=NULL)
439       {
440            if (IsNumeral(Tmp[0]))
441                 R.Push(strtod(Tmp,&E));
442            else if (IsOperator(Tmp[0]))
443                      R.Push(DoOperation(R.Pop(),R.Pop(),Tmp[0]));
444                 else if (IsFunction(Tmp[0]))
445                           R.Push(DoFunction(R.Pop(),Tmp[0]));
446
447            Tmp= strtok(NULL," ");
448       }
449
450       return(R.Pop());
451  }
452
453  #pragma package(smart_init)
454
```