```
/* Calculo do Numero PI
 *
 *
 * Este programa calcula o numero pi com varias (muitas) casas decimais.
 * O numero de casas decimais pode ser obtido pela formula:
 *   ncasas = log10(BASE) * CASAS
 * que, neste exemplo, resulta em 50.008 casas decimais. As oito ultimas
 * foram propositalmente acidionadas para absorver erros de
 * arrendondamento e se garantir um numero pi com precisao de 50.000
 * casas e incerteza apenas na ultima.
 *
 * A base BASE foi fixada em 10000 ao inves da base 10, que seria mais
 * natural, afim de otimizar a velocidade dos calculos. Nao se recomenda
 * alterar a BASE por dois motivos: (1) overflow nos calculos internos,
 * se aumentarmos a BASE; e (2) menor velocidade de processamento, se
 * diminuirmos a BASE.
 *
 * Autor: Roberto Marques Bekman
 * Email: kongsp@ig.com.br
 * Primeira versao: 1987
 * Versao Atual: 2002
 *
 * OBS1. Este numero pi so deve ser usado para o bem. Ou seja, nada de
 * ficar assustando as garotas com recitais infernais de 100, 500,
 * 1000 casas que parecem nao terminar nunca.
 *
 */


#include <stdio.h>
#include <time.h>

#define OVERFLOW     1
#define CASAS        12502
#define BASE         10000
#define FORMATO      "%04d"

#define MAIS         '+'
#define MENOS        '-'


/* A estrutura do numero de ponto flutuante foi fixada em um digito na
 * sase BASE antes do ponto decimal e CASAS digitos apos o ponto decimal.
 */
typedef struct
       {
       char sinal;
       int digitos[CASAS + 1];
       } numero;


/* Subrotina para somar dois numeros de ponto flutuante de estrutura
 * 'numero'.
 */
int somaxy (numero *resultado, numero *x, numero *y)
       {
       numero *p;
       int soma;
       int acumulador;
       int i;

       if ((x->sinal) == (y->sinal))
              {
              resultado->sinal = x->sinal;

              acumulador = 0;
              for (i = CASAS; i >= 0; i--)
                     {
                     soma = x->digitos[i] + y->digitos[i] + acumulador;
                     resultado->digitos[i] = soma % BASE;
                     acumulador = soma / BASE;
                     };
              }
       else
              {
```

```
		/* verifica qual numero e maior */
		for (i = 0; i <= CASAS; i++)
			if (x->digitos[i] != y->digitos[i])
				break;

		/* por elegancia faz o sinal do resultado ser +0 caso x e y
		 * sejam iguais.
		 */
		if (i > CASAS)
			{
			x->sinal = MAIS;
			y->sinal = MAIS;
			};

		/* se y for maior que x entao troca x com y */
		if (y->digitos[i] > x->digitos[i])
			{
			p = x;
			x = y;
			y = p;
			};

		/* agora que x >= y faz a subtracao */
		resultado->sinal = x->sinal;

		/* faz o truque de emprestar BASE a priori para facilitar o algoritmo.
		 * este truque e possivel porque nunca havera a necessidade de tomar
		 * mais que BASE emprestado.
		 */
		acumulador = 0;
		for (i = CASAS; i >= 0; i--)
			{
			soma = x->digitos[i] - y->digitos[i] + acumulador + BASE;
			resultado->digitos[i] = soma % BASE;
			acumulador = soma / BASE - 1;
			};
		};

	if (acumulador != 0)
		return (OVERFLOW);

	return (0);
	}


/* Subrotina para multiplicar dois numeros de ponto flutuante com
 * estrutura 'numero'. A multiplicacao e realizada na forma exata
 * (2 * CASAS) e depois desprezadas as casas menos significativas,
 * retornando um numero de ponto flutuante de estrutura 'numero'.
 * A variavel interna mult foi propositalmente colocada fora da
 * subrotina pois ela e bastante grande. Fora da subrotina ela
 * aliviando o stack durante a execucao.
 */
int mult[2 * CASAS + 2];
int multxy (numero *resultado, numero *x, numero *y)
	{
	long soma;
	long acumulador;
	int yi;
	int i, j, k;

	/* inicializa a variavel com o valor completo de x vezes y */
	for (i = 0; i <= 2 * CASAS + 1; i++)
		mult[i] = 0;

	for (i = CASAS; i >= 0; i--)
		{
		acumulador = 0;
		for (j = CASAS; j >= 0; j--)
			{
			k = i + j + 1;

			soma = (long)mult[k] + (long)(x->digitos[i]) * (long)(y->digitos[j]) + acumulador;
			mult[k] = soma % BASE;
			acumulador = soma / BASE;
```

```
                        };

                k--;
                while ((k >= 0) && (acumulador != 0))
                        {
                        soma = mult[k] + acumulador;
                        mult[k] = soma % BASE;
                        acumulador = soma / BASE;
                        k--;
                        };

                /* e impossivel dar overflow neste ponto mas o codigo esta ai para
                 * desencargo de conciencia
                 */
                if (acumulador > 0)
                        return (OVERFLOW);
                };

        if (x->sinal == y->sinal)
                resultado->sinal = MAIS;
        else
                resultado->sinal = MENOS;

        for (i = 0; i <= CASAS; i++)
                resultado->digitos[i] = mult[i + 1];

        /* agora sim faz o teste de overflow pois o algoritmo pressupoe apenas
         * 1 casa do lado esquerdo da virgula
         */
        if (mult[0] != 0)
                return (OVERFLOW);

        return (0);
        }


/* Subrotina para dividir um numero de ponto flutuante (com estrutura
 * 'numero') por numero do tipo 'long'.
 */
int divxn (numero *resultado, numero *x, long n)
        {
        long dividendo;
        char sinal;
        int i;

        /* testa se o valor de n nao vai ocasionar overflow */
        if ((n == 0) || (n > 200000))
                return (OVERFLOW);

        if (n > 0)
                sinal = MAIS;
        else
                sinal = MENOS;

        if (x->sinal == sinal)
                resultado->sinal = MAIS;
        else
                resultado->sinal = MENOS;

        dividendo = 0;
        for (i = 0; i <= CASAS; i++)
                {
                dividendo = BASE * dividendo + x->digitos[i];

                resultado->digitos[i] = dividendo / n;
                dividendo = dividendo % n;
                };

        return (0);
        }


/* Subrotina para testar se o numero de ponto flutuante (com estrutura
 * 'numero' e igual a zero.
 */
```

```
int ezero (numero *x)
   {
   int i;

   for (i = 0; i <= CASAS; i++)
      if (x->digitos[i] != 0)
        return (1);

   x->sinal = MAIS;
   return (0);
   }


/* Subtorina para imprimir na tela um numero de ponto flutuante de
 * estrutura 'numero'.
 */
int printx (numero *x)
       {
       int i;

       putchar (x->sinal);
       for (i = 0; i <= CASAS; i++)
              {
              printf (FORMATO, x->digitos[i]);
              if (i == 0)
                     putchar ('.');
              };
       putchar ('\n');

       return (0);
       }


/* Subrotina para carregar do arquivo ('pi.dat') um numero de ponto
 * flutuante de estrutura 'numero'.
 */
int loadx (numero *x)
       {
       FILE *src;
       int valor;
       int i, j;
       int ch;

       x->sinal = '+';
       x->digitos[0] = 3;
       for (i = 1; i <= CASAS; i++)
              x->digitos[i] = 0;

       src = fopen ("pi.dat", "rt");
       if (src == NULL)
              return (0);

       ch = fgetc (src);
       if ((ch != '+') && (ch != '-'))
              {
              fclose (src);
              return (0);
              };

       x->sinal = ch;
       for (i = 0; i <= CASAS; i++)
              {
              valor = 0;

              for (j = BASE; j > 1; j = j / 10)
                     {
                     ch = fgetc (src);
                     if ((ch < '0') || (ch > '9'))
                            break;
                     valor = valor * 10 + (ch - '0');
                     };

              x->digitos[i] = valor;
              };
```

```
   fclose (src);
   return (0);
   }


/* Subrotina para salvar para arquivo ('pi.dat') um numero de ponto
 * flutuante de estrutura 'numero'.
 */
int savex (numero *x)
       {
       FILE *dst;
       int i;

       dst = fopen ("pi.dat", "wt");
       if (dst == NULL)
              return (0);

       fputc (x->sinal, dst);
       for (i = 0; i <= CASAS; i++)
              fprintf (dst, FORMATO, x->digitos[i]);

       fclose (dst);
       return (0);
       }

/* Subrotina principal
 *
 * O numero pi e calculado atraves de algoritmo iterativo.
 * O algoritmo implementado e uma versao ligeiramente melhorada
 * do metodo de Newton-Raphson para se encontar a raiz de uma
 * funcao, no caso, encontar o zero de sen(x) na vizinhanca
 * de pi (sen(pi) = 0). Para se obter o valor de sen(x) foi
 * utilizada a expansao em serie de sen(x)
 * sen(x) = x - x^3/3! + x^5/5! - x^7/7! + ...
 * O tempo de processamento e altamente sensivel ao numero de
 * casas que se pretende calcular, requerendo da ordem do
 * numero de casas elevado ao cubo.
 */
int main ()
       {
       numero x, x2, fator, pi, tmp;
       long n;
       int i, j;
       float segundos;

       loadx (&pi);
       i = 0;
       n = 0;
       do
              {
              printf ("Pi(%d): ", ++i);
              printx (&pi);

              segundos = (float)clock() / (float)CLOCKS_PER_SEC;
              printf ("Iteracao %d: %d Casas decimais em %04.2f segundos (nmax=%ld)\n",
              i - 1, (log(BASE) / log(10)) * CASAS, segundos, n);

              x.sinal = pi.sinal;
              for (j = 0; j <= CASAS; j++)
                     x.digitos[j] = pi.digitos[j];

              multxy (&x2, &x, &x);

              fator.sinal = x.sinal;
              for (j = 0; j <= CASAS; j++)
                     fator.digitos[j] = x.digitos[j];

              n = 1;
              while (ezero (&fator) != 0)
                     {
                     multxy (&fator, &fator, &x2);

                     n++;
                     divxn (&fator, &fator, n);
                     n++;
```

```
                divxn (&fator, &fator, n);

                if (fator.sinal == MAIS)
                        fator.sinal = MENOS;
                else
                        fator.sinal = MAIS;

                somaxy (&x, &x, &fator);
                };

        somaxy (&pi, &pi, &x);
        savex (&pi);
        }
while ((ezero (&x) != 0) && (i < 10));

printf ("\n");
printf ("Pi(%d): ", i);
printx (&pi);

segundos = (float)clock() / (float)CLOCKS_PER_SEC;
printf ("Iteracao %d: %d Casas decimais em %04.2f segundos\n",
        i, (log(BASE) / log(10)) * CASAS, segundos);

return (0);
}
```