CES-10 Introdução à Computação

Prof. Carlos Henrique Q. Forster, forster@ita.br

Lab04 – Aritmética Intervalar

Material para auxiliar a construção do programa.

```
struct interval
{
  double min, max;
};

typedef struct interval interval;

interval ivmake(double min, double max);
/*retorna um intervalo com minimo e maximo dados */
```

Retorna o intervalo $[\min, \max]$

```
interval ivadd(interval x, interval y);
/*retorna um intervalo que eh a soma de dois intervalos */
```

Retorna $[x_0 + x_1, y_0 + y_1]$ para a soma de $[x_0, x_1]$ com $[y_0, y_1]$.

```
interval ivneg(interval x);
/*retorna o oposto de um intervalo */
```

Retorna $[-x_1, -x_0]$ para a entrada $[x_0, x_1]$

```
interval ivscale(double s, interval x);
/*multiplica um intervalo por escalar */
```

Se s for positivo, retorna $[sx_0, sx_1]$, se for negativo retorna $[sx_1, sx_0]$.

```
interval ivsub(interval x, interval y);
/*subtrai dois intervalos */
```

Retorna $[x_0, x_1] + [-y_1, -y_0]$.

```
interval ivmul(interval x, interval y);
/*multiplica dois intervalos */
```

Retorna $\bar{z} = \bar{x} \cdot \bar{y}$ como $z_{\min} = \min\{k_0, k_1, k_2, k_3\}$, $z_{\max} = \max\{k_0, k_1, k_2, k_3\}$,
onde $k_0 = x_0 y_0$, $k_1 = x_1 y_0$, $k_2 = x_0 y_1$, $k_0 = x_1 y_1$.

```
interval ivinv(interval x);
/*retorna o reciproco ou inverso de um intervalo */
```

Testar se o zero está dentro do intervalo (usando ivin). Se estiver, abortar o programa após enviar mensagem de erro. Caso contrário, retornar $\left[\frac{1}{x_1}, \frac{1}{x_0}\right]$.

```
interval ivdiv(interval x, interval y);
/*divisao de dois intervalos */
```

Retornar multiplicação pelo recíproco.

```
interval ivexp(interval x);
/*exponencial de intervalo */
```

Retornar $[\exp x_0, \exp x_1]$.

```
interval ivlog(interval x);
/*logaritmo de intervalo */
```

Testar se o intervalo contém valores negativos, quando se deve terminar o programa com erro. Senão, retornar $[\ln x_0, \ln x_1]$. A função log da linguagem C é o logaritmo neperiano.

```
interval ivlog10(interval x);
/*logaritmo base 10 do intervalo */

interval ivpow(interval x, interval y);
/*potenciacao de intervalos */
```

Retornar $\exp(y \cdot \ln x)$, onde a exponencial, o logaritmo e a multiplicação são as operações de intervalo já definidas.

```
interval ivabs(interval x);
/*valor absoluto de intervalo */
```

Se o intervalo contiver o zero, o valor mínimo do intervalo resposta é zero e o máximo é o maior dos valores absolutos $|x_0|, |x_1|$.

```
interval ivsin(interval x);
/*seno de intervalo */

interval ivcos(interval x);
/*cosseno de intervalo */

int ivin(double a, interval x);
/*retorna 1 se a esta no intervalo x, senao 0*/
```

Retorna o resultado da expressão booleana que avalia se $a \geq x_0 \wedge a \leq x_1$.

```
interval ivpoly(double p[], int n, interval x);
/*retorna a avaliacao do polinomio de coeficientes
  dados no vetor p em funcao do intervalo x
  n=grau p[0]+p[1]*x+p[2]*x^2 ... +p[n]*x^n */
```

A avaliação de polinômios pode ser feita pela regra de Horner: $a_0 + x(a_1 + x(a_2 + xa_3))$.

$y \leftarrow a_n$

$\text{for}\, i = n \cdots 1$

$\{y \leftarrow y \cdot x + a_{i-1}$

$\text{return}\, y$

onde as operações de adição e multiplicação são as operações de intervalos já definidas.

```
void ivprint(interval x);
/*imprime um intervalo na forma [xmin,xmax] */


typedef void ivcallback(interval iv);
typedef interval iveval(interval iv);

void ivsolve(iveval f, interval x, ivcallback resp, double prec);
```

O algoritmo para encontrar zeros de uma função real $f(x) = 0$ é recursivo e funciona da seguinte forma: Partindo de um intervalo $\bar{x} = [x_0, x_1]$, se $0 \notin \bar{f}(\bar{x})$ então não há solução nesse intervalo. Caso contrário, se o tamanho do intervalo $x_1 - x_0$ for menor que a precisão desejada, imprima o intervalo. Caso contrário, aplique o mesmo algoritmo recursivamente para os intervalos $\bar{x}_L = \left[x_0, \frac{x_1 + x_0}{2} \right]$ e $\bar{x}_R = \left[\frac{x_1 + x_0}{2}, x_1 \right]$.

$y \leftarrow f(x);$
$\text{Se}(0 \notin y), \text{return}$
$\text{Senão se}(prec < x_1 - x_0), resp(x)$
Senão

$$\begin{cases} ivsolve\left(f, \left[x_0, \frac{x_1 + x_0}{2} \right], resp, prec \right) \\ ivsolve\left(f, \left[\frac{x_1 + x_0}{2}, x_1 \right], resp, prec \right) \end{cases}$$

A avaliação da pertinência é feita com a função ivin, a construção dos intervalos entre colchetes pode ser feita com a função ivmake

O algoritmo não garante a existência de zeros nos intervalos da resposta, mas garante que se os zeros existirem, estarão naqueles intervalos.

```
#endif
```

Não esquecer de testar todas as funções definidas através da função main.