Aula 12- Variáveis e valores reais

Até agora todos os nossos problemas continham apenas valores inteiros e portanto variáveis do tipo **`int`**. Para resolver problemas que usam valores com parte fracionária, outros tipos de variáveis são necessários.

São chamados em C e em outras linguagens de valores e variáveis reais. Essa denominação não é precisa pois da mesma forma que no caso de inteiros, nem todos os valores podem ser representados ou mesmo armazenados dentro de uma variável real. O problema é o tamanho da palavra de memória que permite que sejam apenas armazenados valores com uma certa quantidade de dígitos. Ou seja valores com uma determinada precisão.

Melhor chamarmos de valores com parte fracionária, ou mesmo valores em ponto flutuante, aludindo ao fato que os mesmos podem ser armazenados forma (M,E), onde M é a mantissa e E é o expoente em alguma base. Supondo por exemplo a base 10, o número (M,E) representa o valor $M*10^E$. O ponto decimal flutua, dependendo do expoente considerado.

Exemplos:

1.2 = $1.2 * 10^0$, ou $0.12 * 10^1$, ou $12 * 10^{-1}$, ou ainda $0.0012 * 10^3$
-34.567 = $-0.34567*10^2 = -34567*10^{-3}$

Constantes em ponto flutuante em C

Podem ser representadas com ponto decimal ou em notação exponencial.

Exemplos:

1) `1.2`
2) `0.12e1` (o mesmo que $0.12 * 10^1$)
3) `-54.897`
4) `-467.8907e12` (o mesmo que $–467.8907 * 10^{12}$)
5) `2.45e0` (o mesmo que $2.4 * 10^0$)
6) `23.0`

Variáveis em ponto flutuante

São declaradas como **`float`** ou **`double`**

Float (de ponto flutuante) – contém valores no intervalo $–3.4 * 10^{38}$ a $3.4 * 10^{38}$
O valor positivo mais próximo de zero é $3.4 * 10^{-38}$

Double (dupla) – contém valores no intervalo $–1.7 * 10^{308}$ a $1.7 * 10^{308}$
O valor positivo mais próximo de zero é $1.7 * 10^{-308}$

Cuidado: nem todos os valores destes intervalos podem ser representados. Existem infinitos valores no intervalo e só são representados aqueles cuja precisão (quantidade de dígitos significativos) cabe dentro de uma variável do tipo correspondente.

Exemplos:

```
float x,y;
double a,b,c;
```

Podemos agora resolver outros problemas usando valores não inteiros, mas antes disso vejamos algumas outras informações que irão tornar mais interessantes os problemas resolvidos.

Abaixo um resumo de todos os tipos básicos da linguagem C:

Tipos básicos

Tipo	Tamanho (bytes)	Descrição
`char`	1	Tipo caracter. Pode ser com ou sem sinal (`unsigned`).
`short`	2	Inteiro armazenado em 16 bits. Pode ser com ou sem sinal.
`int`	4	Inteiro aramazenado em 32 bits. Pode ser com ou sem sinal.
`long`	4	Idêntico ao int.
`long long`	8	Inteiro armazenado em 64 bits. Com ou sem sinal.
`float`	4	Ponto flutuante com precisão simples.
`double`	8	Ponto flutuante em precisão dupla

Exemplos:

```
char b;
unsigned char c;
short c;
unsigned short d;
int e;
unsigned int f;
long g;
unsigned long h;
long long i;
unsigned long long j;
long int k;
unsingned long int l;
float m;
double m;
```

Funções intrínsecas

Existem no C várias funções pré-definidas. Tais funções podem ser usadas nas expressões aritméticas de um programa. Para isso basta incluir (**`#include`**) o arquivo de definições adequado.
Abaixo algumas delas. Para ver uma lista completa, consulte o help do compilador.

função e tipo	Resultado	#include
`int abs(int i)`	| i | - valor absoluto de i	`<stdlib.h>`
`int rand (void)`	um número aleatório inteiro > 0	`<stdlib.h>`
`double cos (double x)`	cos x	`<math.h>`
`double sin (double x)`	sen x	`<math.h>`
`double tan (double x)`	tan x	`<math.h>`
`double fabs (double x)`	| x | - valor absoluto x	`<math.h>`
`double exp (double x)`	e^x	`<math.h>`
`double log (double x)`	ln x	`<math.h>`
`double log10 (double x)`	log x (base 10)	`<math.h>`
`double acos (double x)`	arc cos x	`<math.h>`
`double asin (double x)`	arc sen x	`<math.h>`
`double atan (double x)`	arc tan x	`<math.h>`
`double sqrt (double x)`	raiz quadrada de x	`<math.h>`
`double cbrt (double x)`	raiz cúbica de x	`<math.h>`
`double pow (double x, double y)`	x^y	`<math.h>`

Exemplos:

```
int a,b;
double x, y, z;
:
:
:
a = abs(i);
a = rand();
z = sqrt(x) + cbrt (x);
z = sin(x) * cos(y) + tan (x+y);
if (exp(x) > 3.1416) printf("logaritmo de x = %12.5f", log10(x));
else printf("logaritmo neperiano de x = %12.5f", log(x));
z = atan(x+y) + acos(x)/asin(y)
z = pow(x, fabs(y));
```

O formato `%f` para valores float

O formato f é usado para imprimir valores reais. Pode-se especificar o número de casas com que os valores são impressos. Exemplos:

%f – imprime o valor com o comprimento necessário
%12f – imprime o valor com 12 posições. Nestas 12 posições estarão o ponto decimal e 6 casas decimais.
%12.5f – imprime o valor com 12 casas sendo 5 decimais

```
#include <stdio.h>
#include <stdio.h>
/* exemplos do formato f */
int main() {
 float x;

 /* entre com x */
 printf("digite o valor de x:");
 scanf("%f", &x);

 /* imprime x com vários formatos */
 printf("\n%f", x);
 printf("\n%12f", x);
 printf("\n%10f", x);
 printf("\n%12.5f", x);
 printf("\n%10.5f", x);
 system("pause"); return 0;
}
```

Será impresso:

```
digite o valor de x: 3.1415
3.1415
    3.141600
  3.141600
     3.14160
   3.14160
```

O formato `%lf` para valores double

Idem ao formato **%f**. Exemplo:

```
#include <stdio.h>
#include <stdio.h>
/* exemplos do formato lf */
int main() {
 double x;

 /* entre com x */
 printf("digite o valor de x:");
 scanf("%lf", &x);

 /* imprime x com vários formatos */
 printf("\n%lf", x);
 printf("\n%12lf", x);
 printf("\n%10lf", x);
 printf("\n%12.5lf", x);
 printf("\n%10.5lf", x);
 system("pause"); return 0;
}
```

Será impresso:

```
digite o valor de x:-12.34
-12.340000
  -12.340000
 -12.34000
   -12.34000
 -12.34000
```

Mistura de tipos em expressões

Podem-se misturar tipos numa expressão aritmética.

Se numa operação existe a mistura de tipos, antes da operação os operandos são convertidos para o tipo dominante.

Na atribuição o tipo resultante da expressão calculada (lado direito) é convertido para o tipo da variável que recebe o valor (lado esquerdo) antes da atribuição.

Tipos dominantes – vale a seguinte prioridade:

bool < char < int < long < float < double

Mudança de tipos

Pode-se mudar o tipo de uma expressão colocando-se o operador **(tipo)** à frente da expressão.

Exemplos:

```
int n;
double x, z;
z = x / (double) n;
n = (int) x;
```

Além disso, quando valores reais são atribuídos a variáveis inteiras, os mesmos são truncados.

Veja o programa abaixo e o que será impresso:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  int i, j, k;
  float a, b, c;
  i = 13;
  j = 5;
  k = i / j;
  printf("\n1 - valor de k = %3d", k);
  c = i / j;
  printf("\n2 - valor de c = %5.2f", c);
  k = i / (float) j;
  printf("\n3 - valor de k = %3d", k);
  c = i / (float) j;
  printf("\n4 - valor de c = %5.2f", c);

  a = 13;
  b = 5;
  k = a / b;
  printf("\n5 - valor de k = %3d", k);
  c = a / b;
  printf("\n6 - valor de c = %5.2f", c);
  c = (int) a / (int) b;
  printf("\n7 - valor de k = %5.2f", c);
  c = (int) a / b;
  printf("\n8 - valor de c = %5.2f", c);
  system("pause"); return 0;
}
```

```
1 - valor de k =   2
2 - valor de c =  2.00
3 - valor de k =   2
4 - valor de c =  2.60
5 - valor de k =   2
6 - valor de c =  2.60
7 - valor de k =  2.00
8 - valor de c =  2.60
```

P42) Dados a, b e c reais, calcular as raízes da equação do segundo grau $ax^2 + bx + c = 0$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main() {
 double a,b,c,   /* coeficientes dados */
        delta,   /* discriminante da equação */
        x1, x2;  /* raízes calculadas */
 /* entre com a,b e c */
 printf("digite o valor de a:");
 scanf("%lf", &a);

 printf("digite o valor de b:");
 scanf("%lf", &b);
```

```
 printf("digite o valor de c:");
 scanf("%lf", &c);

 /* verifica se a = 0. Neste caso não é equação do 2. Grau */
 if (a == 0.0) printf("\n*** nao e equacao do 2. grau");
 else {/* calcule o valor do discriminante */
       delta = b*b - 4*a*c;
       /* verifica se delta < 0 e neste caso não há raizes reais */
       if (delta < 0) printf ("\n*** nao ha raizes reais");
       else {/* calcule x1 e x2 */
             x1 = (-b + sqrt(delta))/(2*a);
             x2 = (-b - sqrt(delta))/(2*a);
             printf("\n***x1 = %10.4lf",x1);
             printf("\n***x2 = %10.4lf",x2);
       }
 }
 system("pause"); return 0;
}
```

P43) Calcular os valores da função senx*cosx para x = 0, 0.001, 0.002, ..., 2pi

```
#include <stdio.h>
#include <stdlib.h>
#define pi 3.1416
#define pi2 2*pi
#include <math.h>

int main() {
 double x;

 /* imprime a tabela */
 for (x = 0.0; x < pi2; x = x + 0.0001)
       printf("\nx=%10.5lf     senx*cosx=%10.5lf", x, sin(x)*cos(x));
 system("pause"); return 0;
}
```

No problema acima há 2 novidades:

Definição de constantes:

O comando **`#define`** é usado para definir um texto como sinônimo de outro texto. No exemplo acima, `3.1416` é sinônimo de `pi` e `2*pi` é sinônimo de `pi2`.

Comando `for` com passo 0.001

A variável do comando `for` é `double` e o passo de incremento é 0.001. Até agora, usávamos o comando for, incrementando ou decrementando o contador de 1 em 1 , 2 em 2, etc., mas sempre inteiros. O incremento pode ser também de valores não inteiros. Na verdade, o comando for é mais geral que isso. Será visto mais tarde.

Especialmente com relação ao **`#define`**, pode-se usá-lo para criar sinônimos dos elementos de programação e tornar mais clara a programação. Ao gosto do freguês, por exemplo:

```
#include <stdio.h>
#define abra {
#define feche }
```

```
#define enquanto while
#define se if
#define entao
#define senao else
#define inteiro int
#define leia scanf
#define imprima printf
/* programa exemplo em linguagem algorítmica */
inteiro main()
abra
 inteiro n,
         i;
 /* ler o n */
 imprima("digite o valor de n:");
 leia("%d", &n);

 /* verifique quais são e quais não são os divisores de n */
 i = 2;
 enquanto (i <= n/2)
   abra
     se (n%i == 0)
        entao imprima ("\n%d - e divisor de %d", i, n);
        senao imprima ("\n%d - nao e divisor de %d", i, n);
        i++;
   feche
 system("pause"); return 0;
feche
```

P44) Dado n inteiro e x real, calcular o valor da soma $1 + x + x^2 + x^3 + ... + x^n$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {
 double x,
        soma;
 int i,
     n;

 /* ler x e n */
 printf("\nentre com o valor de x:"); scanf("%lf", &x);
 printf("\nentre com o valor de n:"); scanf("%d", &n);

 /* calcular a soma */
 soma = 1;
 for (i = 1; i <= n; i++) soma = soma + pow(x,i);
 printf("\n*** valor da soma - %15.7lf", soma);
 system("pause"); return 0;
}
```

P45) idem sem usar `pow`

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
int main() {
 double x,       /* dado */
        termo,   /* termo da soma */
        soma;    /* resultado calculado */
 int i,  /* contador */
     n;  /* dado */

 /* ler x e n */
 printf("\nentre com o valor de x:"); scanf("%lf", &x);
 printf("\nentre com o valor de n:"); scanf("%d", &n);

 /* calcular a soma */
 soma = 1.0;
 termo = 1.0;
 for (i = 1; i <= n; i++) {
      {termo = termo * x;
       soma = soma + termo;
 }
 /* imprime o resultado */
 printf("\n*** valor da soma - %15.7lf", soma);
 system("pause"); return 0;
}
```