Mais sobre entradas e saídas de dados

1. Qual a diferença entre *declarar* e usar uma variável?

Lembramos que uma variável pode ser entendida como um "nome" associado à uma posição de *memória* do computador (e a um tamanho): a posição de seu *bit* inicial e o número de *bits* usados para representar aquele tipo de variável. Como a linguagem *C* é **compilada**, nela é necessário **declarar** cada variável e isso (geralmente) é feito no início da função. Uma vez que variável foi declarada, pode-se usá-la (recebendo valores ou em expressões).

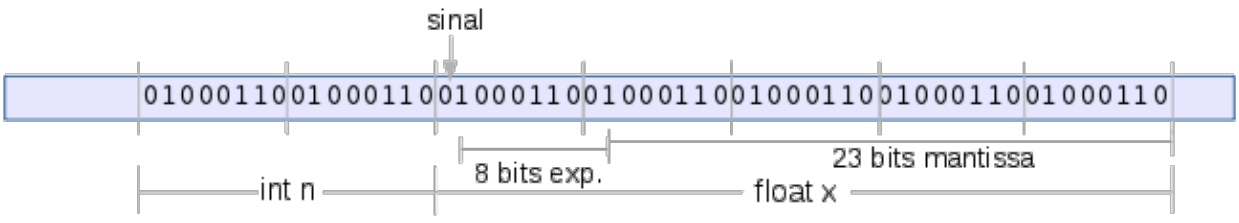
```
int main (void) {
  int n, i=3; // declara 2 variaveis para inteiro, de nomes 'n' e 'i'
(i sendo iniciada com valor 3)
  float x = 0.5, y; // declara 2 variaveis para inteiro de nome 'x' e
'y' (x sendo iniciada com valor 0.5)
  y = i/2; // 'y' recebe 1, pois 'i', entao a conta 3/2 resulta 1
  // %d e' formatador para imprimir inteiro, %f para flutuante – y=%f
e' para comprovar que 3/2 = 1
  printf("i=%d, x=%f, y=%f\n", i, x, y);
  ...
  }
```

Cód. 1. Código ilustrando a declaração de variáveis inteiras e flutuantes.

Outro conceito essencial às variáveis é o de **atribuição** de valor. Sua sintaxe em *C*, e em muitas outras linuguagens, é usar do **lado esquerdo** da atribuição um nome de variável e do **lado direito** da atribuição uma expressão aritmética válida. No código 1, `i=3` e `x=0.5` são atribuições, respectivamente, tendo como *lado esquerdo* as variáveis *i* e *x* e como *lado direito* as expressões constantes *3* e *0.5*.

Ainda usando como estrutura o código 1, outro exemplo de atribuição poderia ser `i = n/2`, que tem como *lado direito* a expressão *n/2*. Como no código 1 a variável *n* é do tipo inteiro, o mesmo para a constante *2*, então a operação é feita usando a precisão de inteiros (se desejasse *flutuante* poderia usar *n / 2.0*) e desse modo, seu resultado é o **quociente da divisão inteira da divisão**. Mas vale adiantar outra possibilidade de "forçar" o resultado a ser *flutuante*, usando o coversor de tipo "forçando" *n* a ser tratado como *flutuante*: `x = (float)n/2`. Experimente!

Saber mais



2. O que são "entradas de dados" e "saídas de dados"?

Um **algoritmo** é uma sequência finita de passos, que ao ser aplicado à um conjunto de dados (**entradas**) deve produzir sempre as mesmas **saídas**. Por exemplo, o algoritmo da divisão ao ser aplicado sobre valores fixados *a* e *b*, deve produzir sempre o mesmo valor *q* (de tal forma que *q* * *b* = *a*).

Por outro lado, um **programa**, em *C* ou em qualquer outra linguagem, nada mais é que a implementação de um **algoritmo** na referida linguagem. Desse modo, para este programa ser usado, o **usuário** (aquele que está executando) deve fornecer um conjunto de **dados de entrada**, na ordem adequada (pois *a/b* geralmente não é o mesmo que *b/a*), para que o programa possa ser executado e produzir as **saída** desejadas.

Fig. 2. Ilustração da existência de um algoritmo que aplicado sobre as entradas produz as respectivas saídas.

Podemos usar o mesmo exemplo da divisão para ilustar a necessidade dos *dados de entrada* adequados. Para que um algoritmo para a divisão de dois números reais seja adequadamente executado, devem ser fornecidos os dois valores reais, o primeiro será o **numerador** e o segundo será o **denominador**. Assim, se o *usuário* digitar apenas um valor, o programa ficará parado até que ele digite o segundo valor.

Saber mais

3. Saídas de dados simples: inteiro, real e texto em *C*

A linguagem *C* preconiza o uso de **formatador** para cada tipo de variável, tanto para as *entradas* quanto para as *saídas*. Os formatadores básicos são:

Tab. 1. Formatadores de acordo com tipo, com exemplo para entra e para saída em C

Tipo de variável	Tipo em *C*	Leitura	Impressão
Inteiro	`int a, b;`	`scanf("%d %d", &a, &b);`	`printf("%d %d\n", a, b);`
Caractere	`char a, b;`	`scanf("%c %c", &a, &b);`	`printf("%c %c\n", a, b);`

Flutuante	`float a, b;`	`scanf("%f %f", &a, &b);`	`printf("%f %f\n", a, b);`
Duplo	`double a, b;`	`scanf("%lf %lf", &a, &b);`	`printf("%lf %lf\n", a, b);`

Em *C* a conversão entre inteiros e caracteres é direta, basta trocar o formatador. Isso é possível devido à conversão inteiro-binário e binário-caractere. Para isso utiliza-se a primeira tabela de representação de caracteres que se popularizou, a tabela **ASCII**. Para saber mais consulte o texto introdutório sobre caracteres. Na tabela 1, o código à esquerda ilustra como testar a tabela *ASCII*.

Tab. 1. Exemplo de códigos para entrada e saída de inteiros e "reais" para o C 2

Exemplo de tabela ASCII	**Exemplo de tabela numérica $\sum_{i=0}^{10} 0.5^i$**

`#include <stdio.h>`
 	
`void main (void) {`
`  int i;`
`  printf("Int | ASCII\n");`
`  for (i=48; i<91; i++) {`
`  printf(" %2d |  '%2c'\n", i, i);`
`  // "Pular" os caracteres:`
`  if (i==57) // 58=':' 59=';'`
`60='<' 61='='`
`  i = 64;  // 62='>' 63='?'`
`64='@'`
`  }`
`  }`
	`#include <stdio.h>`
	`void main (void) {`
	`  float soma = 0, pot = 1;`
	`  int i;`
	`  printf("Int | Soma 0.5^0+...+0.5^i\n");`
	`  for (i=0; i<11; i++) {`
	`  soma += pot;`
	`  printf(" %2d |  %8.2f\n",`
	`i, soma);`
	`  pot *= 0.5 * pot;`
	`  }`
	`  }`

Se o *usuário* desejar inserir outra coisa (em geral uma *cadeia de caracteres* - "*string*") deve-se usar uma função especial (vide `raw_input` a seguir).

4. Entradas de dados em *C*

Em *C* o processo para *leitura* de valores via teclado é tratado em *lote*, quer dizer, durante a execução de um programa, a cada momento que o usuário *digitar* uma sequência de valores e pressionar a tecla para registro (*Enter*), todos os valores digitados serão armazenador em um *reservatório* (em Inglês *buffer*) e a cada momento que o programa encontrar um comando para *leitura* será usado a próxima parte disponível do *reservatório*.
Para verficar a existência do *reservatório*, experimente testar o código 2, de dois modos:

1. digitando todos os valores e um único *Enter* (`CR`) ao final (i.e. algo como `2 3 ABD D E 0.5 2.3 CR`)
2. digitando cada um dos 9 itens teclando *Enter* em diferentes locais (e.g. como: `2 CR 3 CR ABD CR D CR E CR 0.5 CR 2.3 CR`).

Você deverá notar que em ambos os modos todos os valores serão corretamente capturados, com qualquer variação de *Enter* (`CR`) *adequado*, os valores serão armazenados nas variáveis *i1, i1, c1, c2, c3, c4, c5, x* e *y* da mesma forma.

Tab. 2. Exemplo de código para entrada de valors, caracteres e "reais"

Exemplo de código para entrada em *C*

```c
#include <stdio.h>
void main (void) {
  float x, y;
  int i1, i2;
  char c1, c2, c3, c4, c5;
  printf("Digite 9 itens, separados por espaco em branco (2 int, 5 char, 2
float): "); // neste ordem: inteiro, inteiro, caractere,...
  scanf("%d %d %c %c %c %c %c %f %f", &i1, &i2, &c1, &c2, &c3, &c4, &c5, &x,
&y);
  // Experimente comentar a linha acima e testar com variacoes como abaixo (sera
igual)
  // scanf("%d %d", &i1, &i2);
  // scanf("%c %c %c %c %c", &c1, &c2, &c3, &c4, &c5);
  // scanf("%f %f", &x, &y);
  printf("i1=%d, i2=%d, c1=%c, c1=%c, c1=%c, c1=%c, c1=%c, x=%f, y=%f\n", i1,
i2, c1, c2, c3, c4, c5, x, y);
  }
```

Para o código da tabela 2, se compilá-lo e rodá-lo com *C* padrão, supondo que digite em uma única linha `2 3 ABD D E 0.5 2.3`, produzirá o seguinte resultado:

```
igite 9 itens, separados por espaco em branco (2 int, 5 char, 2 float): 2
3 ABD D E 0.5 2.3

1=2, i2=3, c1=A, c1=B, c1=D, c1=D, c1=E, x=0.500000, y=2.300000
```

Fig. 3. Ilustração do erro ao supor que existe comentários aninhados em Python.

Experimente copiar e colar cada um dos códigos acima em seu ambiente para programação *C* preferido. Procure alterar o código até que esteja certo de ter assimilado a sintaxe e os detalhes de cada conceito aqui explorado.

Leônidas de Oliveira Brandão
http://line.ime.usp.br

Alterações