Entrada e saída

Esta página descreve superficialmente as funções de entrada (= *input*) e de saída (= *output*) mais importantes da linguagem C. Todas estão na biblioteca `stdio`. Para ter acesso a essa biblioteca, seu programa deve incluir a interface `stdio.h`:

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

Teclado e tela

A função **`printf`** (= *print formatted*) exibe na tela do monitor uma lista "formatada" de números, caracteres, strings, etc. O primeiro argumento da função é uma string que especifica o formato da impressão.

A função **`scanf`** (= *scan formatted*) lê do teclado uma lista de números, caracteres, strings, etc. O primeiro argumento da função é uma string que especifica o formato da lista a ser lida. Os demais argumentos são os endereços das variáveis onde os valores lidos serão armazenados. A função trata todos os brancos como se fossem espaços. Eis um exemplo:

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

int main (void) {
   int a, b;
   double media;
   scanf ("%d %d", &a, &b);
   media = (a + b)/2.0;
   printf ("A média de %d e %d é %f\n", a, b, media);
   return EXIT_SUCCESS;
}
```

Supondo que o nome do programa é `prog`, teremos o seguinte resultado na tela do monitor (o computador escreve em vermelho e o usuário em azul):

```
prompt> ./prog
222 333
A média de 222 e 333 é 277.500000
prompt>
```

Arquivos

Um *arquivo* (= *file*) é uma sequência de bytes que reside em memória "lenta", tipicamente um disco magnético. Abstratamente, um arquivo tem estrutura semelhante à memória RAM do computador. Mas, ao contrário do que acontece com a memória RAM (= *random access memory*), os bytes de um arquivo não podem ser endereçados individualmente. Assim, o acesso a um arquivo é estritamente sequencial: para chegar ao 5º byte é preciso passar pelo 1º, 2º, 3º e 4º bytes.

Para que um programa possa manipular um arquivo, é preciso associar a ele uma variável do tipo `FILE` (esse

tipo está definido na interface `stdio.h`). A operação de associação é conhecida como "abertura" do arquivo e é executada pela função `fopen` (= *file open*). O primeiro argumento da função é o nome do arquivo e o segundo argumento é `"r"` ou `"w"` para indicar se o arquivo deve ser aberto "para leitura" (= *read*) ou "para escrita" (= *write*). A função `fopen` devolve o endereço de um `FILE` (ou `NULL`, se não encontrar o arquivo especificado).

Depois de usar o arquivo, é preciso "fechá-lo" com a função `fclose` (= *file close*).

EXEMPLO: Digamos que o arquivo `dados.txt` contém uma sequência de números inteiros separados por brancos. O programa abaixo calcula a média dos números. Para ler o arquivo, o programa usa a função `fscanf` (o nome é uma abreviatura de *file scanf*):

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1

int main (void) {
   int x, n, k;
   double soma;
   FILE *entrada;
   entrada = fopen ("dados.txt", "r");
   if (entrada == NULL) {
      printf ("\nNão encontrei arquivo\n");
      exit (EXIT_FAILURE);
   }
   soma = n = 0;
   while (TRUE) {
      k = fscanf (entrada, "%d", &x);
      if (k != 1) break;
      soma += x;
      n += 1;
   }
   fclose (entrada);
   printf ("A média dos números é %f\n", soma/n);
   return EXIT_SUCCESS;
}
```

A função `fscanf`, tal como a função `scanf`, devolve o número de objetos efetivamente lidos. O programa acima usa isso para detectar o fim do arquivo. A propósito: o programa supõe que o arquivo contém pelo menos um número!

Arquivo de texto. Um *arquivo de texto* é, simplesmente, um arquivo de caracteres dividido em linhas (o fim de cada linha é sinalizado por um caractere `\n`). É uma boa prática garantir que o último caractere do arquivo seja um `\n`.

Stdin e stdout

O teclado é um "arquivo" padrão de entrada (= *standard input*). Ele está permanente "aberto" e é representado pela constante `stdin`. Portanto `fscanf (stdin,...)` equivale a `scanf (...)`.

Algo análogo acontece com as funções `printf`, `fprintf` e o "arquivo" `stdout`, que representa a tela do monitor.

As funções putc e getc

A função mais básica de entrada de dados — mais básica que `printf` — é **`putc`** (= *put character*). Cada

invocação da função grava um caractere no arquivo especificado. Se `c` é um caractere e `f` aponta um arquivo, `putc (c, f)` grava `c` no arquivo `f`. Por exemplo, `putc ('*', stdout)` exibe o caractere `*` na tela do monitor.

A função correspondente de *leitura* de caracteres é **`getc`** (= *get character*). Cada chamada da função lê um caractere do arquivo especificado (interpretando-o como um `unsigned char`). Se `f` aponta um arquivo então `getc (f)` lê o próximo caractere do arquivo. Em particular, `getc (stdin)` lê o próximo caractere do teclado.

EXEMPLO: O programa abaixo lê uma linha de caracteres do teclado, armazena essa linha em um vetor e em seguida exibe esses caracteres na tela do monitor. Vamos supor que a linha tem no máximo 100 caracteres (incluindo o `'\n'` final):

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1

int main (void) {
   char linha[100];
   int i, j;
   j = 0;
   while (TRUE) {
      linha[j] = getc (stdin);
      if (linha[j] == '\n') break;
      j = j + 1;
   }
   for (i = 0; i <= j; i += 1)
      putc (linha[i], stdout);
   return EXIT_SUCCESS;
}
```

OUTRO EXEMPLO: O programa abaixo lê o primeiro caractere do arquivo `dados.txt` e exibe esse caractere na tela do monitor:

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

int main (void) {
   char c; /* erro */
   FILE *entrada;
   entrada = fopen ("dados.txt", "r");
   if (entrada == NULL) exit (EXIT_FAILURE);
   c = getc (entrada);
   fclose (entrada);
   putc (c, stdout);
   return EXIT_SUCCESS;
}
```

O program tem um defeito, que discutiremos a seguir.

Que tipo de objeto getc devolve?

Que acontece se `getc` tenta ler o próximo caractere de um arquivo que já acabou? É preciso que `getc` devolva algum tipo de "caractere inválido". Acontece que todos os 256 caracteres são "válidos"!

Para resolver esse impasse, `getc` não devolve um caractere (sem sinal) mas um `int`, pois o conjunto de valores de `int` contém o conjunto de valores de `unsigned char` e é maior que esse último. Se o arquivo tiver acabado, `getc` devolve um `int` que não possa ser confundido com um `unsigned char`. Mais especificamente, a função faz o seguinte:

1. se houver um próximo caractere no arquivo, `getc` lê o caractere interpretando-o como um `unsigned char`, transforma-o em um `int` e devolve o resultado.
2. se o arquivo não tiver mais caracteres, `getc` devolve `-1`.

Para ser mais exato, se o arquivo não tem mais caracteres a função devolve a constante `EOF` (= *end of file*), que está definida na interface `stdio.h`. Em muitos computadores — mas não em todos! — o valor de `EOF` é `-1`.

EXEMPLO: O seguinte fragmento de código exibe o próximo caractere do arquivo *a menos que estejamos no fim do arquivo*:

```
int c;
c = getc (entrada);
if (c != EOF)
   putc (c, stdout);
else
   printf ("\nO arquivo terminou!");
```

(Se o arquivo de entrada for `stdin`, o fim do arquivo é produzido pela combinação de teclas Ctrl-D, que gera o caractere 4.)

A solução adotado por `getc` é uma boa lição de projeto de algoritmos: a função devolve um objeto que pertence a um *superconjunto* do conjunto em que estamos realmente interessados. Situações análogas acontecem em muitas outras ocasiões.

Exercícios 1

1. Suponha que o arquivo `dados.txt` contém a string `"ABCÿDEF"` e nada mais. O que o seguinte programa imprime?

   ```
   int main (void) {
      FILE *entrada;
      int c;
      entrada = fopen ("dados.txt", "r");
      while ((c = getc (entrada)) != EOF)
         printf ("%c ", c);
      fclose (entrada);
      return EXIT_SUCCESS;
   }
   ```

   Que acontece se trocarmos "`int c`" por "`char c`"? Que acontece se trocarmos "`int c`" por "`unsigned char c`"?

2. Escreva um programa completo que faça uma cópia de um arquivo. O nome do arquivo é digitado pelo usuário. [Solução]

3. Escreva um programa que remova os comentários (norma ANSI) de um programa C. O programa original está gravado em um arquivo; o programa "limpo" deve ser gravado em outro arquivo.

Exercícios 2

1. Escreva um programa que conta o número de ocorrências de cada caractere em um arquivo. O programa solicita o nome do arquivo ao usuário e imprime uma tabela que dá o número de ocorrências de cada caractere.

   Para ganhar inspiração, analise o comportamento do utilitário `wc` (o nome é uma abreviatura de *word count*).

Argumentos na linha de comando

A função `main`, como qualquer outra função, admite argumentos. Eles são conhecidos como "argumentos na linha de comando" (= *command-line arguments*). O primeiro argumento é um inteiro que dá o número de argumentos. O segundo, é um vetor de strings. Portanto, a função `main` deve ser especificada assim:

```
int main (int numargs, char *arg[]) {
   . . .
}
```

Supondo que o nome do programa é `prog`, se digitarmos a linha de comando

```
prompt> ./prog a  bb   ccc 2222
```

`numargs` terá valor `5` e

- `arg[0]` será a string `"prog"`,
- `arg[1]` será a string `"a"`,
- `arg[2]` será a string `"bb"`,
- `arg[3]` será a string `"ccc"` e
- `arg[4]` será a string `"2222"`.

Exemplo. O seguinte programa calcula a média dos números inteiros fornecidos como argumentos na linha de comando.

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

int main (int numargs, char *arg[]) {
   int soma, n;
   soma = 0;
   for (i = 1; i < numargs; ++i) {
      soma += atoi (arg[i]);
   }
   n = numargs - 1;
   printf ("média = %.2f\n", (double) soma/n);
   return EXIT_SUCCESS;
}
```

Supondo que o nome do programa é `prog`, podemos ter a seguinte interação na tela:

```
prompt> ./prog +22   33 -11      +44
média = 22.00
prompt>
```

Outro exemplo. O seguinte programa imprime uma tabela de conversão de graus Celsius para graus Fahrenheit ou vice-versa. O usuário especifica a direção da conversão, bem como o início e o fim da tabela.

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

/* Programa temperatura
// --------------------
// Digite
//             temperatura c-f 10 40
//
// para obter uma tabela de conversão de graus Celsius
// em graus Fahrenheit. A primeira coluna começará com
// 10 graus Celsius e andará em passos de 1 grau até
// 40 graus Celsius. A segunda coluna trará a
// correspondente temperatura em graus Fahrenheit.
// Troque "c-f" por "f-c" para obter a tabela de
// conversão de graus Fahrenheit em graus Celsius.
*/
int main (int numargs, char *arg[]) {
   int inf, sup;
```

```
   if (numargs != 4) {
      printf ("Número de argumentos errado.\n");
      return EXIT_FAILURE;
   }
   inf = atoi (arg[2]);
   sup = atoi (arg[3]);
   if (strcmp (arg[1], "c-f") == 0) {
      int c;
      printf ("Celsius Fahrenheit\n");
      for (c = inf; c <= sup; c += 1)
         printf ("%7d %10.2f\n", c, 9.0/5.0*c + 32);
      return EXIT_SUCCESS;
   }
   if (strcmp (arg[1], "f-c") == 0) {
      int f;
      printf ("Fahrenheit  Celsius\n");
      for (f = inf; f <= sup; f += 1)
         printf ("%10d %8.2f\n", f, 5.0*(f - 32.0)/9.0);
      return EXIT_SUCCESS;
   }
   return EXIT_FAILURE;
}
```

Perguntas e respostas

- PERGUNTA: Por que o meu arquivo aparece na tela com um "`^M`" no fim de cada linha?

  RESPOSTA: Provavelmente o arquivo foi gerado no sistema Windows e está sendo exibido no sistema Linux. Nos sistemas Linux e UNIX, o fim de uma linha é indicado pelo caractere `\n`. Já no sistemas DOS e Windows, o fim de uma linha é indicado pelo par de caracteres `\r\n`. O caractere `\r` aparece na tela como `^M`.

Last modified: Tue Dec 1 13:28:05 BRST 2015
http://www.ime.usp.br/~pf/algoritmos/
Paulo Feofiloff
DCC-IME-USP