Programação Imperativa

Aula 20

Sumário

Ficheiros de texto

Ficheiros de texto

Um ficheiro de texto não é mais do que uma sequência de caracteres que ficam armazenados no disco do computador. Essa sequência de caracteres é terminada com um caracter especial denominado *fim-de-ficheiro*.

Um ficheiro de texto pode ser criado com um editor de texto. É precisamente isso que fazem nas aulas práticas quando escrevem os vossos programas. No entanto, nada impede que escrevam outras coisas. Com um editor de texto, podem escrever cartas, poesias, relatórios de trabalhos, e claro que também podem escrever programas em C.

Em C, podemos dar instruções ao programa para ler e escrever em ficheiros de texto, em vez de ler do teclado e escrever no ecrã. Isto tem a vantagem de não se ter de estar sempre a introduzir os dados.

Leitura e escrita em ficheiros de texto

Para um programa ler e escrever em ficheiros de texto, podemos utilizar as funções `fscanf` e `fprintf`. Estas funções são praticamente iguais ao `scanf` e `printf` que já conheçem. A única diferença é que as funções têm um argumento adicional que indica o ficheiro em que se pretende ler ou escrever. Exemplo:

```
fscanf( f, "%d", &n );
```

A variável `f` é uma variável que tem de ser associada previamente a um ficheiro, e que se declara como sendo do tipo `FILE *`.

O exemplo que se segue é um programa que lê dois números de um ficheiro chamado `"dados.txt"` e escreve o quadrado desses números num ficheiro novo chamado `"resultados.txt"`

Antes de se começar a ler ou escrever num ficheiro, deve ser chamada a função `fopen` para abrir o ficheiro. O `fopen` tem 2 argumentos, o 1º é o nome do ficheiro e o 2º indica se o ficheiro vai ser aberto para leitura ou escrita (`"r"` ou `"w"` respectivamente. Estes nomes vêm do inglês, r-read, w-write).

Depois de abrir o ficheiro deve-se testar se a operação foi bem sucedida. Para tal, compara-se o valor de `f` com a constante `NULL` que está definida em `stdio.h`. Caso haja um erro, podemos escrever no ecrã uma mensagem de erro e terminar o programa.

Se o ficheiro foi aberto com sucesso, poderemos ler e escrever. A leitura e escrita são feitas sequencialmente do inicio para o fim do ficheiro. No final, devemos fechar os ficheiros (anteriormente abertos com a função `fopen`) com a função `fclose`. Aqui vai o programa:

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

main()
{
  FILE *in, *out;
  int a, b;

  /* abre o ficheiro dados.txt para leitura e associa-o a in */
  in = fopen( "dados.txt", "r" );
  if( in == NULL )
   {
     printf("ERRO: não consigo abrir o ficheiro dados.txt\n");
     exit(1);
   }

  /* abre o ficheiro resultados.txt para escrita e associa-o a out */
  out = fopen( "resultados.txt", "w" );
  if( out == NULL )
   {
     printf("ERRO: não consigo abrir o ficheiro resultados.txt\n");
     exit(1);
   }

  /* leitura e escrita */
  fscanf( in, "%d", &a );
  fscanf( in, "%d", &b );
```

```
  fprintf( out, "Este ficheiro foi criado pelo programa em C.\n" );
  fprintf( out, "%d\n", a*a );
  fprintf( out, "%d\n", b*b );

  /* fecha os ficheiros */
  fclose( in );
  fclose( out );
}
```

O `scanf` e o `printf` (e as variantes `fscanf` e `fprintf`) fazem input/output formatado. Isto permite-lhes ler e escrever variáveis dos tipos base. Em C, também existem funções específicas que permitem ler um caracter e escrever um caracter. Essas funções chamam-se `getchar` e `putchar`.

```
char c;

c = getchar();   /* equivalente a scanf("%c", &c)     */
putchar( 'z' );  /* equivalente a printf("%c", 'z')
                    ou simplesmente printf("z")       */
```

O `getchar` lê um caracter do teclado e o `putchar` escreve um caracter para o ecrã. Existem variantes que permitem ler/escrever um caracter de/para um ficheiro. Os nomes são `fgetc` e `fputc`.

```
char c;

c = fgetc( f );   /* equivalente a fscanf(f, "%c", &c)     */
fputc( 'z' );     /* equivalente a fprintf(f, "%c", 'z')
                     ou simplesmente fprintf(f, "z")       */
```

O `getchar` e `putchar` podem ser escritos como sendo casos particulares do `fgetc` e `fputc`.

```
getchar()   é equivalente a fgetc(stdin)
putchar(c)  é equivalente a fputc(stdout,c)
```

`stdin` (standard input) é interpretado como sendo um ficheiro especial, o teclado. `stdout` (standard output) também é interpretado como sendo um ficheiro especial, o ecrã.

Leitura caracter a caracter

O programa que vimos acima não faz qualquer tipo de validação. Por exemplo, se o ficheiro de entrada tiver apenas um número, o programa irá funcionar mal. Ao tentar fazer o segundo `fscanf`, o programa já vai estar a aceder a coisas que

estão depois do *fim-de-ficheiro*. Isso é equivalente a termos um array de dimensão 10 e tentarmos aceder à posição 12!

A solução para este problema é verificar se já chegamos ao fim do ficheiro imediatamente antes de tentarmos ler qualquer coisa. Em `stdio.h` existe uma constante chamada `EOF` (end of file) que significado o *fim-de-ficheiro*.

De seguida apresenta-se um programa que converte um ficheiro de texto para maiúsculas. A ideia é abrir o ficheiro de entrada e ir lendo os caracteres um após outro até chegar ao fim do ficheiro. Cada caracter lido é convertido para maiúsculas e escrito para o ficheiro de saída. Aqui vai o código,

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

main()
{
  FILE *in, *out;
  char c;

  /* abre o ficheiro pequenas.txt para leitura e associa-o a in */
  in = fopen("pequenas.txt", "r" );
  if( in == NULL )
    {
      printf("ERRO: não consigo abrir o ficheiro pequenas.txt\n");
      exit(1);
    }

  /* abre o ficheiro grandes.txt para escrita e associa-o a out */
  out = fopen( "grandes.txt", "w" );
  if( out == NULL )
    {
      printf("ERRO: não consigo abrir o ficheiro grandes.txt\n");
      exit(1);
    }

  c = fgetc( in );
  while( c != EOF )
    {
      fputc( toupper(c), out );
      c = fgetc( in );
    }

  fclose( in );
  fclose( out );
```

```
}
```

O ciclo while faz o trabalho todo. A função `fgetc` lê um caracter de cada vez. Esse caracter pode ser qualquer coisa, incluindo o caracter espaço (' '), o caracter mudança-de-linha ('\n') e o caracter fim-de-ficheiro (EOF).

A função `toupper` está definida em `ctype.h` e converte o caracter de entrada para maiúscula.

Para tornarem o programa mais genérico, podem pedir ao utilizador para introduzir o nome dos ficheiros de entrada e saída. Experimentem fazer isso como exercício.

Outro exemplo

Vamos supor que temos um ficheiro chamado `nomes.txt` cujo conteúdo são nomes de pessoas, um nome por linha. Pretende-se fazer um programa que leia esses nomes para um array de nomes. Aqui vai uma tentativa,

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

/* retorna 1 se 'c' é uma vogal, retorna 0 caso contrário */
int eh_vogal( char c )
{
  return ( c == 'a' || c == 'A' ||
           c == 'e' || c == 'E' ||
           c == 'i' || c == 'I' ||
           c == 'o' || c == 'O' ||
           c == 'u' || c == 'U'
         );
}

/* retorna o número de vogais da string 's' */
int conta_vogais( char *s )
{
  int i, vogais;

  vogais = 0;
  for( i=0; i< strlen(s); i++ )
    if( eh_vogal(s[i]) )
      vogais++;

  return vogais;
}
```

```
main()
{
  FILE *f;
  int  i;
  int ultimo;
  char nomes[100][30];

  f = fopen("nomes.txt", "r");
  if( f == NULL )
    erro("...");

  i = 0;
  while( fgets(nomes[i], 30, f) != NULL && (i<100) )
    {
      ultimo = strlen( nomes[i] );
      nomes[i][ultimo-1] = '\0';       /* para mandar fora o '\n' */
      printf("%s %d\n", nomes[i], conta_vogais(nomes[i]) );
      i++;
    }

  fclose( f );
}
```

Novamente, o sumo do programa está no ciclo while. A função `fgets` é a variante do `gets` para ficheiros. O `fgets` tem 3 argumentos,

- um array de caracteres onde a linha vai ser guardada.
- um número que indica o número máximo de caracteres que pode ser lido (incluindo o `'\0'`).
- o ficheiro de onde pretendemos ler.

Se chegarmos ao fim do ficheiro ou acontecer outro erro qualquer, o `fgets` retorna `NULL`. Além de guardar uma linha de texto no array nome, o `fgets` também guarda o caracter de fim de linha (`'\n'`) caso exista. Ou seja, se o ficheiro `nomes.txt` for,

```
susana
toze
miguel angelo
maria
```

ao fazer o primeiro `fgets` a variável `nome[0]` vai ficar assim:

```
 0   1   2   3   4   5    6    7
's' 'u' 's' 'a' 'n' 'a' '\n' '\0'
```

Uma nota acerca do `gets` e `fgets`

O `fgets` é mais seguro que o `gets`. Aliás, de agora em diante devem deixar de utilizar o `gets` porque o `gets` não faz a validação do número máximo de caracteres do array. Se pretenderem ler um string do teclado podem fazer,

```
char s[80];

fgets( s, 80, stdin );
```

Isto permite que s guarde 79 caracteres mais o `'\0'`. Se por acaso o utilizador escrever mais do que 79, os caracteres adicionais serão ignorados. Se fizerem o mesmo com a função `gets` o programa estoira.