CUTIL: BIBLIOTECA DE APOIO AO DESENVOLVIMENTO DE SOFTWARE EM LINGUAGEM C

Versão 3.0

Julho de 1996

Álvaro F. M. Azevedo
http://www.fe.up.pt/~alvaro

Faculdade de Engenharia da Universidade do Porto

1 - Introdução

A biblioteca CUTIL destina-se a facilitar o desenvolvimento de aplicações em linguagem ANSI C, por intermédio da ocultação de algumas características associadas ao seu baixo nível, como por exemplo a validação da entrada de dados e a atribuição de memória. Quando um programador muda de uma linguagem de mais alto nível, como o Pascal, Fortran ou Basic, para a linguagem C, nota imediatamente as limitações das funções de entrada de dados (e.g., scanf, fscanf, gets, fgets), principalmente no que respeita à validação daquilo que é lido e ao controle das excepções que surgem quando os dados provenientes do teclado ou de um ficheiro são inválidos.

A facilidade na gestão de memória, recorrendo por exemplo à função malloc, é uma vantagem que motiva a transição de linguagens mais rígidas para o C. Com esta função torna-se muito simples lidar com vectores cuja dimensão se ajusta a cada situação e que pode mesmo ser modificada durante a execução do programa (e.g., função realloc). Contudo, a impossibilidade de estender esta versatilidade às matrizes de um modo claro e evidente, obriga o programador a familiarizar-se com os *arrays* de apontadores e a escrever várias linhas de código sempre que necessita de atribuir memória para uma nova matriz. Esta questão motivou o desenvolvimento das funções descritas na Secção 2.4. Uma vez que a programação de métodos numéricos é prejudicada pelo facto de os vectores e matrizes terem os seus índices pertencentes ao intervalo [0, n-1], optou-se por desfasar os apontadores que resultam da atribuição de memória de modo que todos os índices se situem no intervalo [1, n]. Assim, da utilização das funções descritas na Secção 2.4 resulta que, por exemplo, ao atribuir memória para um vector v com n elementos, o primeiro elemento é o v[1] e o último elemento é o v[n].

Na Secção 2 é referido o modo de utilizar cada uma das funções, sugerindo-se uma consulta às listagens da Secção 4 para esclarecimento de qualquer dúvida que subsista.

2 - Utilização das Funções

As diversas funções que constituem a biblioteca CUTIL foram classificadas nos seguintes grupos:

- *input*;
- *output*;
- *strings*;
- atribuição de memória e inicialização;
- tempo decorrido;
- funções privadas.

No caso das funções de *input*, distinguem-se as que fazem a leitura do teclado das que procedem à leitura de informação em ficheiros do tipo texto. Uma vez que um programa que faça alguma leitura a partir do teclado pode ser executado com o *input* redireccionado para um ficheiro de dados, as funções que processam a leitura do teclado foram preparadas para serem utilizadas nestas circunstâncias (e.g., prog < data_file.txt).

2.1 - Input

2.1.1 - *Input* do Teclado

Função

void askcha(char *messg, char choi1, char choi2, char choid, char *cansw);

Argumentos

messg	Texto com a pergunta.
choi1	Primeira resposta possível.
choi2	Segunda resposta possível.
choid	Resposta por defeito. Tem de coincidir com choi1 ou com choi2.
cansw	Resposta. A resposta introduzida pelo utilizador tem de coincidir com choi1 ou com choi2.

Tarefa

Solicitar a introdução de um caracter pelo teclado. Apenas existem duas respostas válidas. Uma resposta em maiúsculas é passada para minúsculas. Se choi1 ou choi2 forem caracteres maiúsculos a função começa por efectuar a sua conversão para caracteres minúsculos. Se o utilizador se limitar a pressionar a tecla Return é considerada a resposta por defeito.

Retorno

Nenhum.

Exemplo

```
char cansw;

askcha("Are you sure", 'y', 'n', 'y', &cansw);

askcha("Metodo directo ou iterativo", 'd', 'i', 'i', &cansw);
```

Função

void askint(char *messg, int iansd, int *iansw);

Argumentos

messg	Texto com a pergunta.
iansd	Resposta por defeito.
iansw	Resposta. A resposta introduzida pelo utilizador tem de ser um número inteiro.

Tarefa

Solicitar a introdução de um número inteiro pelo teclado. Se o utilizador se limitar a pressionar a tecla Return é considerada a resposta por defeito.

Retorno

Nenhum.

Exemplo

```
int n_obj

askint("Numero de objectos", 10, &n_obj);
```

Função

void askrea(char *messg, double ransd, double *ransw);

Argumentos

messg	Texto com a pergunta.
ransd	Resposta por defeito.
ransw	Resposta. A resposta introduzida pelo utilizador tem de ser um número real.

Tarefa

Solicitar a introdução de um número real pelo teclado. Se o utilizador se limitar a pressionar a tecla Return é considerada a resposta por defeito.

Retorno

Nenhum.

Exemplo

```
double toler;

askrea("Tolerancia", 1.0e-6, &toler);
```

Função

void askstr(char *messg, char *choid, char *sansw);

Argumentos

messg	Texto com a pergunta.
choid	Resposta por defeito. Se este argumento for um *string* de comprimento nulo a resposta por defeito não é mostrada.
sansw	Resposta introduzida pelo utilizador. O *string* sansw tem de estar dimensionado para 80 ou mais caracteres. Se o utilizador introduzir mais do que 80 caracteres os excedentes são descartados.

Tarefa

Solicitar a introdução de um *string* pelo teclado. Se o utilizador se limitar a pressionar a tecla Return é considerada a resposta por defeito.

Retorno

Nenhum.

Exemplo

```
char sansw[100];

askstr("Morada do local de trabalho", "Nao especificada", sansw);

askstr("What's your name", "", sansw);
```

Função

```
void rfword(char *fword);
```

Argumentos

fword	Primeira palavra que foi introduzida pelo teclado. O *string* fword tem de estar dimensionado para 80 ou mais caracteres. Se o utilizador introduzir mais do que 80 caracteres os excedentes são descartados.

Tarefa

Solicitar a introdução de uma palavra pelo teclado É lida uma linha de texto e é extraída a sua primeira palavra. Espaços em branco à cabeça e no fim da palavra são retirados. Esta função destina-se essencialmente a ser utilizada por outras funções da biblioteca CUTIL.

Nota: o caracter *TAB* não é considerado um espaço.

Retorno

Nenhum.

Exemplo

```
char fword[100];

rfword(fword);
```

Função

```
void rjname(char *jname);
```

Argumentos

jname	*Job name*. Primeira palavra que foi introduzida pelo teclado. Se esta palavra contiver caracteres que não podem figurar num *pathname* é de novo solicitada a introdução de um *job name* válido. O *string* jname tem de estar dimensionado para 80 ou mais caracteres. Se o utilizador introduzir mais do que 80 caracteres os excedentes são descartados.

Tarefa

Solicitar a introdução de uma palavra pelo teclado É lida uma linha de texto e é extraída a sua primeira palavra. Espaços em branco à cabeça e no fim da palavra são retirados. Esta função destina-se à leitura de um *job name*. Um *job name* é um *string* sem espaços ao qual pode ser posteriormente acrescentada uma extensão. Em PC's com MS-Windows o *job name* pode conter uma letra (volume) seguida do caracter ':' e de um conjunto de directórios separados pelo caracter '\' (ver abaixo os exemplos). Só são aceites no *job name* os seguintes caracteres: [0-9], [A-Z], [a-z], '$', '.', '/', ':', '\' e '_'.

Nota: o caracter *TAB* não é considerado um espaço.

Retorno

Nenhum.

Exemplo

```
char jname[100]; /* Job name (e.g., viga5) */

char fname[100]; /* File name (e.g., viga5.dat) */

rjname(jname);        /*  Exemplos de respostas válidas:
                              viga5
                              ..\Vig_3
                              Edif_Norte\Cave\viga7
                              D:\users\joao\viga2   */

strcpy(fname, jname); /* Copia o jname para fname */

strcat(fname, ".dat"); /* Acrescenta ".dat" a fname */
```

Função

void rtline(char *tline, int mposi);

Argumentos

tline	Linha de texto.
mposi	Número máximo de caracteres que se pretende colocar no *string* tline. O valor de mposi não pode exceder o número de caracteres que o *string* tline é capaz de armazenar.

Tarefa

Solicitar a introdução de um *string* pelo teclado. Os caracteres que excederem o número máximo de caracteres (mposi) são descartados. O caracter '*newline*' não é colocado no *string*. Esta função destina-se essencialmente a ser utilizada por outras funções da biblioteca CUTIL.

Retorno

Nenhum.

Exemplo

```
char tline[100];

rtline(tline, 90);
```

2.1.2 - *Input* de Ficheiro

Função

char get_chr(FILE *f_ptr, int *iline_a, int *llast_a);

Argumentos

f_ptr	*File pointer*: apontador para o ficheiro que está a ser lido.
iline_a	Apontador para um inteiro que conta as linhas que estão a ser lidas. Deve ser inicializado com a unidade (ver exemplo abaixo). Quando se utiliza a família de funções get_... para ler um ficheiro de texto, não se pode utilizar outras funções (e.g., fscanf, fgets), porque isso implicaria a não actualização do contador de linhas.
llast_a	Apontador para um inteiro que indica o número da linha em que foi lido o último caracter. Este argumento destina-se apenas a ser utilizado em posteriores validações (ver exemplo abaixo).

Tarefa

Ler num ficheiro de texto um caracter situado entre delimitadores. Os caracteres que são considerados delimitadores são o espaço, o *TAB* e os caracteres que marcam o fim de uma linha. Os delimitadores são ignorados até aparecer um caracter que não seja um delimitador. Os comentários são ignorados (considera-se como comentário o caracter '#' e todos os caracteres situados à sua direita e até ao fim da linha). As linhas em branco ou contendo apenas comentários são ignoradas.

Retorno

O caracter que foi lido no ficheiro.

Exemplo

```
FILE *f_ptr; /* File pointer. */

int iline = 1; /* Line count (initialize with 1). */

int llast; /* Line where the input data is located. */

char c;

f_ptr = fopen("file.txt", "r");

if (f_ptr == NULL) wermsg("File not found.", 0);

c = get_chr(f_ptr, &iline, &llast);

if (c < 'A' || c > 'L') wermsg("Character outside range [A, L].", llast);
```

Função

double get_dbl(FILE *f_ptr, int *iline_a, int *llast_a);

Argumentos

f_ptr	*File pointer*: apontador para o ficheiro que está a ser lido.
iline_a	Apontador para um inteiro que conta as linhas que estão a ser lidas. Deve ser inicializado com a unidade (ver exemplo abaixo). Quando se utiliza a família de funções get_... para ler um ficheiro de texto, não se pode utilizar outras funções (e.g., fscanf, fgets), porque isso implicaria a não actualização do contador de linhas.
llast_a	Apontador para um inteiro que indica o número da linha em que foi lido o último double. Este argumento destina-se apenas a ser utilizado em posteriores validações (ver exemplo abaixo).

Tarefa

Ler num ficheiro de texto um número real (double) situado entre delimitadores. Os caracteres que são considerados delimitadores são o espaço, o *TAB* e os caracteres que marcam o fim de uma linha. Os delimitadores são ignorados até aparecer um caracter que não seja um delimitador. Os comentários são ignorados (considera-se como comentário o caracter '#' e todos os caracteres situados à sua direita e até ao fim da linha). As linhas em branco ou contendo apenas comentários são ignoradas.

Retorno

O número real (double) que foi lido no ficheiro.

Exemplo

```
FILE *f_ptr; /* File pointer. */

int iline = 1; /* Line count (initialize with 1). */

int llast; /* Line where the input data is located. */

double distance;

f_ptr = fopen("file.txt", "r");

if (f_ptr == NULL) wermsg("File not found.", 0);

distance = get_dbl(f_ptr, &iline, &llast);

if (distance < 0.0) wermsg("Distance cannot be negative.", llast);
```

Função

int get_int(FILE *f_ptr, int *iline_a, int *llast_a);

Argumentos

f_ptr	*File pointer*: apontador para o ficheiro que está a ser lido.
iline_a	Apontador para um inteiro que conta as linhas que estão a ser lidas. Deve ser inicializado com a unidade (ver exemplo abaixo). Quando se utiliza a família de funções get_... para ler um ficheiro de texto, não se pode utilizar outras funções (e.g., fscanf, fgets), porque isso implicaria a não actualização do contador de linhas.
llast_a	Apontador para um inteiro que indica o número da linha em que foi lido o último int. Este argumento destina-se apenas a ser utilizado em posteriores validações (ver exemplo abaixo).

Tarefa

Ler num ficheiro de texto um número inteiro (int) situado entre delimitadores. Os caracteres que são considerados delimitadores são o espaço, o *TAB* e os caracteres que marcam o fim de uma linha. Os delimitadores são ignorados até aparecer um caracter que não seja um delimitador. Os comentários são ignorados (considera-se como comentário o caracter '#' e todos os caracteres situados à sua direita e até ao fim da linha). As linhas em branco ou contendo apenas comentários são ignoradas.

Retorno

O número inteiro (int) que foi lido no ficheiro.

Exemplo

```
FILE *f_ptr; /* File pointer. */

int iline = 1; /* Line count (initialize with 1). */

int llast; /* Line where the input data is located. */

int n;

f_ptr = fopen("file.txt", "r");

if (f_ptr == NULL) wermsg("File not found.", 0);

n = get_int(f_ptr, &iline, &llast);

if (n < 0) wermsg("Integer number cannot be negative.", llast);
```

Função

char *get_str(char *tline, int maxst, FILE *f_ptr, int *iline_a, int *llast_a, int isdel);

Argumentos

tline	Linha de texto onde vai ser colocado o *string* lido no ficheiro.
maxst	Número máximo de caracteres que se pretende colocar no *string* tline. O valor de maxst não pode exceder o número de caracteres que o *string* tline é capaz de armazenar.
f_ptr	*File pointer*: apontador para o ficheiro que está a ser lido.
iline_a	Apontador para um inteiro que conta as linhas que estão a ser lidas. Deve ser inicializado com a unidade (ver exemplo abaixo). Quando se utiliza a família de funções get_... para ler um ficheiro de texto, não se pode utilizar outras funções (e.g., fscanf, fgets), porque isso implicaria a não actualização do contador de linhas.
llast_a	Apontador para um inteiro que indica o número da linha em que foi lido o último *string*. Este argumento destina-se apenas a ser utilizado em posteriores validações (ver exemplo abaixo).
isdel	Código ASCII do caracter que define o fim do *string*. Se o valor de isdel for 10 o *string* é lido até ao fim da linha. Se, por exemplo, o valor de isdel for o código do caracter ';' o *string* é lido até ser encontrado este caracter. Em quaisquer circunstâncias os comentários iniciados com o caracter '#' são ignorados.

Tarefa

Ler um *string* num ficheiro de texto. Os delimitadores são ignorados até aparecer um caracter que não seja um delimitador. Os caracteres que são considerados delimitadores são o espaço, o *TAB* e os caracteres que marcam o fim de uma linha. Os comentários são ignorados (considera-se como comentário o caracter '#' e todos os caracteres situados à sua direita e até ao fim da linha). O *string* é então lido até aparecer o caracter que define o seu fim (argumento isdel). Este caracter é lido e descartado.

Retorno

Um apontador para o *string* tline que foi lido no ficheiro. Em geral, este retorno pode ser ignorado.

Exemplo

```
FILE *f_ptr; /* File pointer. */
int iline = 1; /* Line count (initialize with 1). */
int llast; /* Line where the input data is located. */
char tline[100];
f_ptr = fopen("file.txt", "r");
if (f_ptr == NULL) wermsg("File not found.", 0);
get_str(tline, 90, f_ptr, &iline, &llast, 10);
if (strlen(tline) > 60) wermsg("String length cannot be greater than 60.", llast);
```

Função

char *get_wrd(char *tline, int maxst, FILE *f_ptr, int *iline_a, int *llast_a);

Argumentos

tline	Linha de texto onde vai ser colocado o *string* lido no ficheiro.
maxst	Número máximo de caracteres que se pretende colocar no *string* tline. O valor de maxst não pode exceder o número de caracteres que o *string* tline é capaz de armazenar.
f_ptr	*File pointer*: apontador para o ficheiro que está a ser lido.
iline_a	Apontador para um inteiro que conta as linhas que estão a ser lidas. Deve ser inicializado com a unidade (ver exemplo abaixo). Quando se utiliza a família de funções get_... para ler um ficheiro de texto, não se pode utilizar outras funções (e.g., fscanf, fgets), porque isso implicaria a não actualização do contador de linhas.
llast_a	Apontador para um inteiro que indica o número da linha em que foi lido o último *string*. Este argumento destina-se apenas a ser utilizado em posteriores validações (ver exemplo abaixo).

Tarefa

Ler um *string* num ficheiro de texto. Os delimitadores são ignorados até aparecer um caracter que não seja um delimitador. Os caracteres que são considerados delimitadores são o espaço, o *TAB* e os caracteres que marcam o fim de uma linha. Os comentários são ignorados (considera-se como comentário o caracter '#' e todos os caracteres situados à sua direita e até ao fim da linha). O *string* é então lido até aparecer um delimitador. A função get_wrd destina-se à leitura de uma palavra (*word*) que não contenha nenhum delimitador (espaço ou *TAB*).

Retorno

Um apontador para o *string* tline que foi lido no ficheiro. Em geral, este retorno pode ser ignorado.

Exemplo

```
FILE *f_ptr; /* File pointer. */
int iline = 1; /* Line count (initialize with 1). */
int llast; /* Line where the input data is located. */
char tline[100];
f_ptr = fopen("file.txt", "r");
if (f_ptr == NULL) wermsg("File not found.", 0);
get_wrd(tline, 90, f_ptr, &iline, &llast);
if (strlen(tline) != 3) wermsg("String must be 3 characters long.", llast);
```

Função

int getwrd(char *tline, char **ipbeg);

Notas:

- esta função encontra-se obsoleta;
- não deve ser utilizada em novos programas;
- encontra-se presente apenas para assegurar que programas antigos que a utilizam possam ser recompilados e linkados com a versão mais recente da biblioteca CUTIL.

2.2 - Output

Função

void wermsg(char *ermsg, int iline);

Argumentos

ermsg	Linha de texto contendo a mensagem de erro.
iline	Número da linha em que ocorreu o erro. Este parâmetro destina-se a ser utilizado quando se pretende informar o utilizador que ocorreu um erro na leitura de um ficheiro de texto. O argumento iline deve receber o argumento llast presente na família de funções get_... (ver exemplo abaixo). Quando o erro não estiver relacionado com uma leitura num ficheiro de texto, deve ser atribuído a iline o valor zero.

Tarefa

Produzir uma mensagem de erro e parar a execução do programa.

Retorno

Nenhum.

Exemplo

```
FILE *f_ptr; /* File pointer. */

int iline = 1; /* Line count (initialize with 1). */

int llast; /* Line where the input data is located. */

double distance;

f_ptr = fopen("file.txt", "r");

if (f_ptr == NULL) wermsg("File not found.", 0);

distance = get_dbl(f_ptr, &iline, &llast);

if (distance < 0.0) wermsg("Distance cannot be negative.", llast);
```

Função

void wwamsg(char *wamsg, int iline);

Argumentos

wamsg	Linha de texto contendo a mensagem de aviso (*warning*).
iline	Número da linha em que ocorreu a situação que provocou o envio da mensagem de aviso (*warning*). Este parâmetro destina-se a ser utilizado quando se pretende informar o utilizador que ocorreu algo potencialmente perigoso na leitura de um ficheiro de texto. O argumento iline deve receber o argumento llast presente na família de funções get_... (ver exemplo abaixo). Quando a situação não estiver relacionada com uma leitura num ficheiro de texto, deve ser atribuído a iline o valor zero.

Tarefa

Produzir uma mensagem de aviso (*warning*). Este tipo de mensagens devem ser produzidas sempre que for detectada uma situação potencialmente perigosa, mas que não é suficientemente grave para justificar uma paragem da execução do programa.

Retorno

Nenhum.

Exemplo

```
FILE *f_ptr; /* File pointer. */

int iline = 1; /* Line count (initialize with 1). */

int llast; /* Line where the input data is located. */

double tolerance;

f_ptr = fopen("file.txt", "r");

if (f_ptr == NULL) wermsg("File not found.", 0);

tolerance = get_dbl(f_ptr, &iline, &llast);

if (tolerance < 0.0) wermsg("Tolerance cannot be negative.", llast);

if (tolerance < 1.0e-15) wwamsg("Tolerance is too small.", llast);
```

2.3 - Strings

Função

void elicom(char *tline);

Argumentos

tline	Linha de texto.

Tarefa

Eliminar um eventual comentário num *string*. Um comentário é constituído pelo caracter '#' e por todos os caracteres situados na mesma linha à sua direita. A função elicom procura a primeira ocorrência do caracter '#' e substitui-o pelo caracter nulo (fim do *string*). Se o utilizador da biblioteca CUTIL utilizar a família de funções get_... para ler ficheiros, em geral não precisará de chamar a função elicom.

Retorno

Nenhum.

Exemplo

```
char tline[100];

strcpy(tline, "  abc def  #  ghi jkl"); /* tline == "  abc def  #  ghi jkl" */

elicom(tline); /* tline == "  abc def  " */

strlsh(tline); /* tline == "abc def" */
```

Função

```
void elimnl(char *tline);
```

Argumentos

tline	Linha de texto terminada com um caracter de fim de linha.

Tarefa

Eliminar o caracter de fim de linha presente no *string* tline. O caracter de fim de linha é o caracter designado *Line Feed* (LF - código ASCII = 10 - caracter '\n'). A função elimnl substitui o caracter *Line Feed* pelo caracter nulo (fim do *string*). Se o último caracter do *string* não for o *Line Feed*, ocorre um erro fatal. Se o penúltimo caracter do *string* for o caracter *Carriage Return* (CR - código ASCII = 13 - caracter '\r'), este também é substituído pelo caracter nulo. Se o utilizador da biblioteca CUTIL utilizar a família de funções get_... para ler ficheiros, em geral não precisará de chamar a função elimnl.

Nota: no UNIX as linhas de um ficheiro de texto são terminadas pelo caracter LF, enquanto que no MS-DOS e no MS-Windows 95/98/NT/2000 são terminadas pelo par de caracteres CR/LF. A função elimnl está preparada para funcionar correctamente em todos estes sistemas operativos.

Retorno

Nenhum.

Exemplo

```
char tline[100];

strcpy(tline, "  abc def  \n"); /* tline == "  abc def  \n" */

elimnl(tline); /* tline == "  abc def  " */

strlsh(tline); /* tline == "abc def" */
```

Função

```
int length(char *tline);
```

Argumentos

tline	Linha de texto.

Tarefa

Calcular o comprimento de um *string*. Os espaços situados à direita são ignorados.

Nota: o caracter *TAB* não é considerado um espaço.

Retorno

O comprimento do *string*.

Exemplo

```
char tline[100];

strcpy(tline, "abc   ");

printf("length = %d\n", length(tline) ); /* length = 3 */
```

Função

```
int nlefts(char *tline);
```

Argumentos

tline	Linha de texto.

Tarefa

Calcular o número de espaços à esquerda dos caracteres visíveis.

Nota: o caracter *TAB* não é considerado um espaço.

Retorno

O número de espaços à esquerda dos caracteres visíveis.

Exemplo

```
char tline[100];

strcpy(tline, "   abcdef");

printf("N. of leading spaces = %d\n", nlefts(tline) ); /* N. of leading spaces = 3 */
```

Função

```
void strlsh(char *tline);
```

Argumentos

tline	Linha de texto.

Tarefa

Eliminar os espaços à esquerda do primeiro caracter visível e os espaços à direita do último caracter visível.

Nota: o caracter *TAB* é considerado um caracter visível e não é eliminado.

Retorno

Nenhum.

Exemplo

```
char tline[100];

strcpy(tline, "   abc def   "); /* tline == "   abc def   " */

strlsh(tline); /* tline == "abc def" */
```

Função

void valint(char *stnum, char *valok, int *inder);

Argumentos

stnum	Linha de texto contendo um *string* que se pretende converter num número inteiro.
valok	TRUE (1) se o *string* for susceptível de ser convertido num inteiro recorrendo à função atoi declarada em stdlib.h. FALSE (0) se o *string* não for susceptível de ser convertido num inteiro.
inder	Se o *string* não for susceptível de ser convertido num inteiro, a variável inder indica a posição do primeiro caracter que invalida a conversão.

Tarefa

Verificar se um *string* é susceptível de ser convertido num inteiro recorrendo à função atoi declarada em stdlib.h. Se o argumento valok indicar que o *string* não pode ser convertido num inteiro e se apesar dessa indicação o utilizador chamar a função atoi, o *string* é convertido num inteiro nulo (ver a documentação da função atoi). Se o utilizador da biblioteca CUTIL utilizar a função get_int para ler inteiros num ficheiro e a função askint para ler inteiros do teclado, em geral não precisará de chamar a função valint.

Nota: o caracter *TAB* é considerado um caracter visível que invalida a conversão.

Retorno

Nenhum.

Exemplo

```
char stnum[100];

char valok; /* TRUE when the string contains a valid integer number */

int inder; /* Position of the offending character [1..n] */

strcpy(stnum, "   -123   ");

valint(stnum, &valok, &inder); /* valok == TRUE (1); inder == 0 */

strcpy(stnum, "12x3   ");

valint(stnum, &valok, &inder); /* valok == FALSE(0); inder == 3 */
```

Função

void valrea(char *stnum, char *valok, int *inder);

Argumentos

stnum	Linha de texto contendo um *string* que se pretende converter num número real.
valok	TRUE (1) se o *string* for susceptível de ser convertido num real recorrendo à função atof declarada em stdlib.h. FALSE (0) se o *string* não for susceptível de ser convertido num real.
inder	Se o *string* não for susceptível de ser convertido num real, a variável inder indica a posição do primeiro caracter que invalida a conversão.

Tarefa

Verificar se um *string* é susceptível de ser convertido num número real recorrendo à função atof declarada em stdlib.h. Se o argumento valok indicar que o *string* não pode ser convertido num real e se apesar dessa indicação o utilizador chamar a função atof, o *string* é convertido num número real nulo (ver a documentação da função atof). Se o utilizador da biblioteca CUTIL utilizar a função get_dbl para ler números reais num ficheiro e a função askrea para ler números reais do teclado, em geral não precisará de chamar a função valrea.

Nota: o caracter *TAB* é considerado um caracter visível que invalida a conversão.

Retorno

Nenhum.

Exemplo

```
char stnum[100];

char valok; /* TRUE when the string contains a valid real number */

int inder; /* Position of the offending character [1..n] */

strcpy(stnum, "   -123.456e-07   ");

valrea(stnum, &valok, &inder); /* valok == TRUE (1); inder == 0 */

strcpy(stnum, "12x3   ");

valrea(stnum, &valok, &inder); /* valok == FALSE(0); inder == 3 */
```

2.4 - Atribuição de Memória e Inicialização

Quando se utiliza a biblioteca CUTIL, a atribuição de memória (*memory allocation*) deve ser efectuada com as funções dim..., a respectiva libertação com as funções fre... e a inicialização (com valores nulos) com as funções zer..., encontrando-se no seguinte quadro as 56 funções disponíveis para realizar estas tarefas:

	dim...	fre...	zer...
char - 1 índice	dimec1	freec1	zeroc1
char - 2 índices	dimec2	freec2	zeroc2
char - 3 índices	dimec3	freec3	zeroc3
double - 1 índice	dimed1	freed1	zerod1
double - 2 índices	dimed2	freed2	zerod2
double - 3 índices	dimed3	freed3	zerod3
float - 1 índice	dimef1	freef1	zerof1
float - 2 índices	dimef2	freef2	zerof2
float - 3 índices	dimef3	freef3	zerof3
int - 1 índice	dimei1	freei1	zeroi1
int - 2 índices	dimei2	freei2	zeroi2
int - 3 índices	dimei3	freei3	zeroi3
long - 1 índice	dimel1	freel1	zerol1
long - 2 índices	dimel2	freel2	zerol2
long - 3 índices	dimel3	freel3	zerol3
unsigned char - 1 índice	dimeu1	freeu1	zerou1
unsigned char - 2 índices	dimeu2	freeu2	zerou2
unsigned char - 3 índices	dimeu3	freeu3	zerou3
double - 2 índices (matriz triangular superior)	dimtd2	(*)	zertd2

(*) - A libertação de memória deve ser efectuada com a função freed2

Apenas são descritas com pormenor as funções dimec1, freec1, zeroc1, dimec2, freec2, zeroc2, dimec3, freec3, zeroc3, dimtd2 e zertd2. A descrição das restantes funções seria semelhante, uma vez que apenas diferem no tipo de dados utilizado.

Função

```
void dimec1(char **a, int npos1);
```

Argumentos

a	Endereço da variável do tipo char * que vai ser inicializada pela função dimec1.
npos1	Número de elementos do vector de caracteres.

Tarefa

Reservar memória para um vector de caracteres (char) com npos1 elementos. O primeiro elemento do vector possui o índice 1 e o último elemento possui o índice npos1. Se não existir memória suficiente a execução do programa termina com uma mensagem de erro.

Retorno

Nenhum.

Exemplo

```
char *a;

int npos1 = 5; /* number of elements */

dimec1(&a, npos1);

zeroc1(a, npos1);

a [1] = 'A';

a [5] = 'Z';

freec1(a);
```

Função

void freec1(char *a);

Argumentos

a	Valor da variável do tipo char * que foi inicializada com a função dimec1.

Tarefa

Libertar a memória reservada com a função dimec1.

Retorno

Nenhum.

Exemplo

```
char *a;
int npos1 = 5; /* number of elements */
dimec1(&a, npos1);
zeroc1(a, npos1);
a [1] = 'A';
a [5] = 'Z';
freec1(a);
```

Função

void zeroc1(char *a, int npos1);

Argumentos

a	Variável do tipo char * que foi inicializada com a função dimec1.
npos1	Número de elementos do vector de caracteres.

Tarefa

Inicializar com valor nulo todos os elementos de um vector de caracteres.

Retorno

Nenhum.

Exemplo

```
char *a;
int npos1 = 5; /* number of elements */
dimec1(&a, npos1);
zeroc1(a, npos1);
a [1] = 'A';
a [5] = 'Z';
freec1(a);
```

Função

void dimec2(char ***a, int npos1, int npos2);

Argumentos

a	Endereço da variável do tipo char ** que vai ser inicializada pela função dimec2.
npos1	Número de linhas da matriz de caracteres.
npos2	Número de colunas da matriz de caracteres.

Tarefa

Reservar memória para uma matriz de caracteres (char) com npos1 linhas e npos2 colunas. O índice correspondente às linhas varia de 1 a npos1. O índice correspondente às colunas varia de 1 a npos2. Se não existir memória suficiente a execução do programa termina com uma mensagem de erro.

Retorno

Nenhum.

Exemplo

```
char **a;

int npos1 = 3; /* number of rows */

int npos2 = 5; /* number of columns */

dimec2(&a, npos1, npos2);

zeroc2(a, npos1, npos2);

a [1] [1] = 'A';

a [3] [5] = 'Z';

freec2(a);
```

Função

void freec2(char **a);

Argumentos

a	Valor da variável do tipo char ** que foi inicializada com a função dimec2.

Tarefa

Libertar a memória reservada com a função dimec2.

Retorno

Nenhum.

Exemplo

```
char **a;
int npos1 = 3; /* number of rows */
int npos2 = 5; /* number of columns */
dimec2(&a, npos1, npos2);
zeroc2(a, npos1, npos2);
a [1] [1] = 'A';
a [3] [5] = 'Z';
freec2(a);
```

Função

void zeroc2(char **a, int npos1, int npos2);

Argumentos

a	Variável do tipo char ** que foi inicializada com a função dimec2.
npos1	Número de linhas da matriz de caracteres.
npos2	Número de colunas da matriz de caracteres.

Tarefa

Inicializar com valor nulo todos os elementos de uma matriz de caracteres.

Retorno

Nenhum.

Exemplo

```
char **a;
int npos1 = 3; /* number of rows */
int npos2 = 5; /* number of columns */
dimec2(&a, npos1, npos2);
zeroc2(a, npos1, npos2);
a [1] [1] = 'A';
a [3] [5] = 'Z';
freec2(a);
```

Função

void dimec3(char ****a, int npos1, int npos2, int npos3);

Argumentos

a	Endereço da variável do tipo char *** que vai ser inicializada pela função dimec3.
npos1	Número de linhas do "paralelepípedo" de caracteres.
npos2	Número de colunas do "paralelepípedo" de caracteres.
npos3	Número de "camadas" do "paralelepípedo" de caracteres.

Tarefa

Reservar memória para um "paralelepípedo" de caracteres (char) com npos1 linhas, npos2 colunas e npos3 "camadas". O índice correspondente às linhas varia de 1 a npos1. O índice correspondente às colunas varia de 1 a npos2. O índice correspondente às "camadas" varia de 1 a npos3. Se não existir memória suficiente a execução do programa termina com uma mensagem de erro.

Retorno

Nenhum.

Exemplo

```
char ***a;

int npos1 = 3; /* number of rows */

int npos2 = 5; /* number of columns */

int npos3 = 2; /* number of "layers" */

dimec3(&a, npos1, npos2, npos3);

zeroc3(a, npos1, npos2, npos3);

a [1] [1] [1] = 'A';

a [3] [5] [2] = 'Z';

freec3(a);
```

Função

void freec3(char ***a);

Argumentos

a	Valor da variável do tipo char *** que foi inicializada com a função dimec3.

Tarefa

Libertar a memória reservada com a função dimec3.

Retorno

Nenhum.

Exemplo

```
char ***a;
int npos1 = 3; /* number of rows */
int npos2 = 5; /* number of columns */
int npos3 = 2; /* number of "layers" */
dimec3(&a, npos1, npos2, npos3);
zeroc3(a, npos1, npos2, npos3);
a [1] [1] [1] = 'A';
a [3] [5] [2] = 'Z';
freec3(a);
```

Função

void zeroc3(char ***a, int npos1, int npos2, int npos3);

Argumentos

a	Variável do tipo char *** que foi inicializada com a função dimec3.
npos1	Número de linhas do "paralelepípedo" de caracteres.
npos2	Número de colunas do "paralelepípedo" de caracteres.
npos3	Número de "camadas" do "paralelepípedo" de caracteres.

Tarefa

Inicializar com valor nulo todos os elementos de um "paralelepípedo" de caracteres.

Retorno

Nenhum.

Exemplo

```
char ***a;
int npos1 = 3; /* number of rows */
int npos2 = 5; /* number of columns */
int npos3 = 2; /* number of "layers" */
dimec3(&a, npos1, npos2, npos3);
zeroc3(a, npos1, npos2, npos3);
a [1] [1] [1] = 'A';
a [3] [5] [2] = 'Z';
freec3(a);
```

Função

void dimtd2(double ***a, int npos1);

Argumentos

a	Endereço da variável do tipo double ** que vai ser inicializada pela função dimtd2.
npos1	Número de linhas da matriz de números reais (double). Uma vez que a matriz é triangular superior, o número de colunas é sempre igual ao número de linhas.

Tarefa

Reservar memória para uma matriz triangular superior, cujos elementos são números reais (double). O número de linhas é npos1, que coincide sempre com o número de colunas. O índice correspondente às linhas varia de 1 a npos1. Na linha ipos1 o índice correspondente às colunas varia de ipos1 a npos2. Se não existir memória suficiente a execução do programa termina com uma mensagem de erro.

Retorno

Nenhum.

Exemplo

```
double **a;

int npos1 = 3; /* number of rows/columns */

dimtd2(&a, npos1);

zertd2(a, npos1);

a [1] [1] = 3.14;

a [2] [3] = 4.34e2;

a [3] [2] = 4.0; /* ERROR: i > j */

a [3] [3] = -0.34e-4;

freed2(a); /* freed2 can be used in conjunction with dimtd2 */
```

Função

void zertd2(double **a, int npos1);

Argumentos

a	Variável do tipo double ** que foi inicializada com a função dimtd2.
npos1	Número de linhas da matriz de números reais (double). Uma vez que a matriz é triangular superior, o número de colunas é sempre igual ao número de linhas.

Tarefa

Inicializar com valor nulo todos os elementos de uma matriz triangular superior, cujos elementos são números reais (double).

Retorno

Nenhum.

Exemplo

```
double **a;

int npos1 = 3; /* number of rows/columns */

dimtd2(&a, npos1);

zertd2(a, npos1);

a [1] [1] = 3.14;

a [2] [3] = 4.34e2;

a [3] [2] = 4.0; /* ERROR: i > j */

a [3] [3] = -0.34e-4;

freed2(a); /* freed2 can be used in conjunction with dimtd2 */
```

2.5 - Tempo Decorrido

Esta Secção apenas possui uma função, designada elacpu, que se baseia nas funções time e clock (ANSI C).

Função

void elacpu(double *relat, double *rcput);

Argumentos

relat	Endereço da variável que vai receber o *elapsed time* (em segundos). O *elapsed time* consiste no número de segundos que decorreram desde 1 de Janeiro de 1970 às 0H00.
rcput	Endereço da variável que vai receber o *CPU time* (em segundos). O *CPU time* consiste no número de segundos que decorreram desde o início do processo. No sistema operativo UNIX, o *CPU time* é o tempo correspondente apenas ao processo que chamou a função clock. No MS-Windows, o *CPU time* é o tempo total que decorreu desde o início do processo.

Tarefa

Calcular o *elapsed time* (em segundos) e o *CPU time* (também em segundos).

Retorno

Nenhum.

Exemplo

```
double rcpu1; /* CPU time (1) */
double rcpu2; /* CPU time (2) */
double rcput; /* CPU time */
double rela1; /* Elapsed time (1) */
double rela2; /* Elapsed time (2) */
double relat; /* Elapsed time */
/* ...  Code ... */
/* Control the elapsed time and CPU time (1) */
elacpu(&rela1, &rcpu1);
/* ...  Code ... */
/* Control the elapsed time and CPU time (2) */
elacpu(&rela2, &rcpu2);
/* Calculate the elapsed time and CPU time between (1) and (2) */
relat = rela2 - rela1;
rcput = rcpu2 - rcpu1;
/* ...  Code ... */
/* Write the elapsed time and CPU time */
printf("\n");
printf("Elapsed time between (1) and (2) = %10.3lf seconds\n", relat);
printf("CPU time      between (1) and (2) = %10.3lf seconds\n", rcput);
```

2.6 - Funções Privadas

Na biblioteca CUTIL estão presentes algumas funções que apenas se destinam a ser utilizadas por outras funções da mesma biblioteca. Essas funções são designadas privadas e não devem ser invocadas pelos utilizadores da biblioteca CUTIL. Apresentam-se em seguida apenas os seus nomes:

chkstr

delmtr

fixstr

goodchar

nextchar

3 - Instalação e Utilização

As funções que chamarem funções da biblioteca CUTIL deverão ter à cabeça as seguintes linhas:

```
#include <stdio.h>

#include <aa/cutil.h>
```

Para que o ficheiro cutil.h seja encontrado pelo compilador deve-se proceder do seguinte modo:

- Criar algures, por exemplo em C:\, um directório chamado CUTIL
- Dentro de C:\CUTIL criar um directório chamado aa
- Copiar para C:\CUTIL\aa o ficheiro cutil.h
- Colocar o directório C:\CUTIL na lista dos directórios em que o compilador procura ficheiros com a extensão .h (*include files*). Por exemplo no Microsoft Developer Studio deve-se seguir: Tools/Options/Directories/Include Files

Em lugar de se incluírem no projecto os ficheiros da biblioteca CUTIL que contêm as funções que foram utilizadas, aconselha-se a *linkagem* com um dos ficheiros libcutil.lib. Deve-se atender à versão do compilador e ao facto de se tratar de uma versão Debug ou Release.

4 - Listagens (ordem alfabética)

Nesta Secção apenas se incluem, por ordem alfabética, as listagens das 85 funções que constituem a biblioteca CUTIL, precedidas do correspondente *include file* (cutil.h).