

Instituto Superior de Engenharia de Lisboa
LEETC
Programação II
2021/22 – 2.º semestre letivo
Primeira Série de Exercícios

1. Exploração de operações *bitwise* e deslocamentos

O programa `packint.c`, disponível em anexo e reproduzido abaixo, realiza o armazenamento de sequências de valores, com o tipo `int`, em ficheiros de conteúdo binário, ocupando espaço compactado pela representação seguinte:

Cada valor é armazenando numa sequência de elementos com a dimensão de `char`, em quantidade variável, de modo a usar apenas os necessários para o valor em causa. A quantidade de elementos é identificada pelo bit de maior peso de cada um deles: No último elemento é 1; nos anteriores é 0. A ordem de armazenamento é dos bits de menor peso para os de maior peso do valor representado.

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

#define DATA_SIZE (CHAR_BIT - 1)
#define FINAL_MASK (1 << DATA_SIZE)
#define DATA_MASK (~(~0 << DATA_SIZE))
#define SIGN_SIZE 1

int main( int argc, char *argv[] ){
    if( argc != 2 ){
        fprintf( stderr, "Error on arguments;\n"
                  "Usage: %s output_filename\n", argv[0] );
        return 1;
    }
    FILE *f = fopen( argv[1], "wb" );
    if( f == NULL ){
        fprintf( stderr, "Fail opening file \"%s\"\n", argv[1] );
        return 2;
    }
    int v;
    while( scanf( "%d", &v ) == 1 ){
        while( v >> (DATA_SIZE - SIGN_SIZE) != 0 &&
              v >> (DATA_SIZE - SIGN_SIZE) != -1 ){
            putc( v & DATA_MASK, f );
            v >>= DATA_SIZE;
        }
        putc( v & DATA_MASK | FINAL_MASK, f );
    }
    fclose( f );
    return 0;
}
```

Propõe-se que ensaie o programa `packint.c` com sequências de valores, incluindo positivos e negativos com várias ordens de grandeza, e observe os ficheiros de dados gerados, com o auxílio do utilitário `hd` (*hexadecimal dump*).

Pretende-se dispor de um programa para decodificar os dados armazenados pelo programa `packint.c`, reproduzindo os valores originais. O código deve ser independente da plataforma, funcionando corretamente para qualquer valor de `sizeof(int)` e de `CHAR_BIT` (definido no *header file* `limits.h`).

1.1. Escreva a função

```
int signExtend( int value, int size );
```

que recebe um valor em código de complemento para dois nos `size` bits de menor peso de `value` e retorna o mesmo valor representado na totalidade dos bits do tipo `int`.

- 1.2. Escreva o programa `unpackint.c` que, recebendo como argumento de linha de comando um ficheiro anteriormente produzido pelo programa `packint.c`, descodifique o seu conteúdo e apresente, através de *standard output*, a sequência de valores original. Ao descodificar cada elemento, após juntar as várias partes provenientes do ficheiro, deve utilizar a função `signExtend` para reproduzir o valor com sinal na dimensão do tipo `int`.

2. Manipulação de *strings*

Pretende-se o processamento de *strings* contendo linhas de texto provenientes de um ficheiro. Cada linha é formada por uma sequência de campos, separados por ponto-e-vírgula ‘;’. Cada campo pode conter qualquer sequência de caracteres, incluindo alfabéticos, numéricos, espaços e *tab*, como no exemplo seguinte, cujos segundo e quarto campos são vazios:

```
primeiro campo;; terceiro campo \t; ; palavras do quinto campo 1234\n
```

O objetivo é separar e uniformizar os campos, obtendo acesso ao seu conteúdo e eliminando os separadores de campo ‘;’, bem como caracteres de espaçamento que possam existir no início ou fim de algum campo.

2.1. Escreva a função

```
char *cutEndingSpaces( char *str );
```

que retorna um ponteiro para a frase contida na *string* indicada por `str`, eliminando os caracteres de espaçamento (espaços, *tabs* e mudanças de linha) que eventualmente existam nas suas extremidades. Valoriza-se a eficiência, nomeadamente evitando de deslocar o conteúdo da *string*. Se esta contiver apenas separadores, deve produzir *string* vazia, retornando o endereço do terminador.

2.2. Escreva a função

```
int fields(char *line, char *ptrs[], int max_fields );
```

que faz o processamento de uma linha de texto apontada pelo parâmetro `line`, identificando os campos e separando-os com a colocação do terminador de *string*. O parâmetro `ptrs` indica um *array* de ponteiros que devem ser afetados de modo a apontar o texto dos campos, na *string* apontada por `line`, separados e sem espaços nas extremidades. O parâmetro `max_fields` é o número de elementos do *array* de ponteiros e corresponde ao número máximo de campos esperados na linha. O valor de retorno é o número de campos identificados; este pode ser superior ao número de ponteiros afetados, se a linha contiver mais campos do que o indicado por `max_fields`.

Considerando a linha de exemplo mencionada anteriormente, a função retorna o valor 5 e os ponteiros ficam a apontar para as palavras indicadas no quadro seguinte:

<code>ptrs[0]</code>	"primeiro campo"
<code>ptrs[1]</code>	" "
<code>ptrs[2]</code>	"terceiro campo"
<code>ptrs[3]</code>	" "
<code>ptrs[4]</code>	"palavras do quinto campo 1234"

Para separar os campos, deve encontrar os caracteres ‘;’ e substituí-los por terminador de *string*. Para eliminar espaços nas extremidades dos campos deve usar a função `cutEndingSpaces`.

2.3. Escreva um programa de teste que, por cada linha recebida de *standard input*, execute a função `fields` e, utilizando os ponteiros preenchidos por ela, apresente em *standard output* os campos identificados, novamente separados por ‘;’. Propõe-se que leia de *standard input* com `fgets`. Admita que cada linha tem a dimensão máxima de 255 caracteres e que os campos são 8 no máximo. Para efeitos de demonstração, deve criar ficheiros, contendo linhas com campos que permitam verificar a sua identificação e a eliminação de espaços nas extremidades, e utilizá-los com redirecionamento de *input*. Propõe-se o desenvolvimento em duas fases, especificadas nas alíneas seguintes.

- a) Na primeira fase, escreva o programa de modo a reproduzir os campos todos, pela ordem original. Por exemplo, se executável tiver o nome “a.out” e usar um ficheiro de texto de entrada “amostra” com o conteúdo seguinte:

```
abc ; ; 135; xyz
ABC; *# ; 246; XYZ
```

O comando “a.out < amostra” deve produzir o resultado seguinte:

```
abc; ; 135; xyz
ABC; *#; 246; XYZ
```

- b) Na segunda fase, adicione ao programa anterior a possibilidade de reproduzir uma seleção dos campos, que pode ter ordem diferente da original. Para isso, o programa recebe em argumento de linha de comando uma sequência de números que identifica os campos selecionados e a ordem para a sua apresentação. Nestes argumentos, o valor 1 significa a seleção do primeiro campo, 2 do segundo, etc.. No caso de haver argumentos que selecionam campos inexistentes, devem ser ignorados. Se não houver argumentos de linha de comando, deve reproduzir todos os campos como na versão da primeira fase.

Para obter o valor numérico dos argumentos de linha de comando deve usar uma função de biblioteca, como por exemplo `atoi`, `strtol` ou `sscanf`.

Considerando as condições do exemplo anterior,

o comando “a.out < amostra”, bem como o comando “a.out < amostra 1 2 3 4”, deve produzir o resultado seguinte, idêntico ao anterior:

```
abc; ; 135; xyz
ABC; *#; 246; XYZ
```

o comando “a.out < amostra 1 3” deve produzir o resultado seguinte:

```
abc; 135
ABC; 246
```

o comando “a.out < amostra 3 1 4” deve produzir o resultado seguinte:

```
135; abc; xyz
246; ABC; XYZ
```