**Instituto Superior de Engenharia de Lisboa**

# LEETC - LEIRT

# Programação II

**2020/21 – 2.º semestre letivo**

# Primeira Série de Exercícios

## 1. Exploração de operações *bitwise* e deslocamentos

Considere a representação de um conjunto de valores inteiros positivos, suportada por um *bit-map* em que cada bit representa a presença de um valor no conjunto. Para isso, é definido o tipo `Set` formado por uma sequência de palavras em memória, em que cada palavra armazena uma parte dos *bits*.

```
typedef unsigned long Set[BITMAP_PARTS];
```

Admita que o universo do conjunto é a gama de valores de 0 a 255. Para manipulação do conjunto usa-se um índice de *bit*, numerado de 0 até ao máximo suportado. O primeiro *bit*, que corresponde à presença do valor 0 no conjunto, tem o índice 0 e é o de menor peso da palavra localizada no índice 0 do *array*; O último bit, que corresponde à presença do valor 255 no conjunto, tem o índice 255 e é o de maior peso da palavra localizada no índice mais elevado do *array*.

Na implementação deve ter em conta a portabilidade, utilizando o operador *sizeof* e o valor de `CHAR_BIT`, como definido no *standard header file* "limits.h"

1.1. Defina o valor de `BITMAP_PARTS` de acordo com a gama de valores a representar.

1.2. Escreva as funções seguintes:

```
void setPlace( Set set, unsigned value );
```

que coloca, no conjunto representado por `set`, o elemento indicado por `value`. Se este já existir, não há alteração;

```
void setRemove( Set set, unsigned value );
```

que elimina, do conjunto representado por `set`, o elemento indicado pelo parâmetro `value`. Se este não existir, não há alteração;

```
void setUnion( Set set1, Set set2 );
```

que calcula a reunião dos conjuntos representados por `set1` e `set2`, depositando o resultado em `set1`;

```
void setIntersect( Set set1, Set set2 );
```

que calcula a interseção dos conjuntos representados por `set1` e `set2`, depositando o resultado em `set1`;

```
void setShow( Set set );
```

que apresenta, em *standard output*, a lista com os valores dos elementos presentes no conjunto representado por `set`.

1.3. Escreva um programa de teste que permita demonstrar a funcionalidade das funções anteriores. Para esse efeito deve criar, com estado inicial vazio, dois conjuntos, designados por A e B, e aceitar, através de *standard input*, os comandos seguintes:

- `p` *set val* – (*place*) coloca, no conjunto *set* (A ou B), o elemento com o valor indicado por *val*;
- `r` *set val* – (*remove*) elimina, do conjunto *set* (A ou B), o elemento com o valor indicado por *val*;
- `u` – (*union*) afeta o conjunto A com a reunião dos dois;
- `i` – (*intersection*) afeta o conjunto A com a interseção dos dois;
- `s` – (*show*) mostra, em *standard output*, os elementos presentes em cada um dos conjuntos;
- `q` – (*quit*) termina.

## 2. Manipulação de *strings*

Pretende-se a criação de uma função para processamento de *strings* que representam nomes de ficheiros, com o propósito de identificar a respetiva terminação, se pertencer a um determinado conjunto de terminações predefinidas. O formato do nome de ficheiro deve ser "*nome.terminação*". As terminações a considerar são especificadas por *strings*, sem o '.', passadas à função através de um *array* de ponteiros para as mesmas.

2.1. Escreva a função

```
int findTerm( char *str, char *terms[], int numTerms );
```

que verifica se o nome de ficheiro indicado por `str` tem uma das terminações indicadas por `terms`. O parâmetro `numTerms` representa o número de terminações identificadas pelos elementos de `terms`. A função retorna o índice, na *string* `str`, do '.' que antecede a terminação. Se não existir nenhuma das terminações previstas, retorna o número total de carateres da *string* indicada por `str`.

Por exemplo, no caso `terms` ter dois ponteiros, indicando as *strings* "c" e "h", a função `findTerm`:

- se `str` indicar "prog.c", retorna 4, indicando a posição da sequência ".c" final;
- se `str` indicar "my.c.defs.h", retorna 9, indicando a posição da sequência ".h" final;
- se `str` indicar "a.out", retorna 5, indicando a posição do terminador de *string*;

Recomenda-se a consulta da especificação das funções declaradas no *header file* "string.h" e a exploração das que forem úteis.

2.2. Escreva um programa de teste que permita demonstrar a função anterior. Deve identificar um conjunto de palavras usadas como terminação em nomes de ficheiros e definir com elas um *array* de ponteiros adequado para passar no parâmetro `terms` da função `findTerm`.

## 3. Armazenamento e ordenação de dados

Considere a leitura e armazenamento dos nomes dos ficheiros existentes numa diretoria. Propõe-se a criação do tipo `Item` para representar um nome de ficheiro ou diretoria.

```
enum ItemType { T_FILE, T_DIRECTORY };

typedef struct {
    enum ItemType type;
    char filename[MAX_FILE_NAME];
    int term;
} Item;
```

Pretende-se o desenvolvimento de um programa que leia a diretoria, indicada por argumento de linha de comando, armazenando os nomes de ficheiros ou diretorias num *array* de *struct* com o tipo `Item`. Deve descartar a informação relativa a itens que não sejam ficheiros nem diretorias, se existirem na diretoria. Após o carregamento da informação, o programa permite a apresentação dos dados, em resposta a comandos introduzidas através de *standard input*.

Considerando a criação das variáveis sem recurso a alojamento dinâmico, admita um cenário simplificado em que são definidos limites rígidos para o armazenamento de dados:

- Espaço máximo para cada nome, representado pela macro `MAX_FILE_NAME;`
- Quantidade máxima de nomes, representada pela macro `MAX_ITEMS`.

Sugere-se a criação de diretorias de ensaio com diversidade de conteúdo para o teste e demonstração do programa.

Após a leitura da diretoria e preenchimento da estrutura de dados, o programa deve esperar, em ciclo, os comandos seguintes:

- `l` (*list*) Apresenta a lista de todos os itens, mostrando, para cada um: o tipo (diretoria ou ficheiro), o nome completo e a indicação da terminação identificada, se existir.

  A lista é ordenada da seguinte forma: primeiro todos os ficheiros; segundo todas as diretorias; em cada um destes grupos, a ordem é por nome completo, alfabeticamente crescente.

- `f` (*files*) Apresenta uma lista parcial, apenas com os nomes de ficheiros.

  Esta lista é organizada em grupos de nomes com a mesma terminação. A sequência de grupos é ordenada por terminação alfabeticamente crescente. O grupo dos ficheiros sem terminação reconhecida é o primeiro a ser exibido. Em cada grupo com a mesma terminação, a ordem é por nome completo alfabeticamente crescente.

- `q` (*quit*) Termina.

3.1. Identifique as dimensões para a estrutura de dados e defina as respetivas macros.

3.2. Escreva a função

```
int loadDirList( Item list[], size_t size, char *path );
```

destinada a carregar a lista de itens, no *array* `list` com dimensão máxima `size`, a partir da diretoria indicada por `path`. Retorna a quantidade de itens preenchidos.

Para ler os dados da diretoria propõe-se a utilização das funções, de biblioteca normalizada, `opendir`, `readdir` e `closedir`. Em anexo ao enunciado é disponibilizado um exemplo de utilização destas funções.

Deve utilizar a função `findTerm` para determinar o valor do campo `term` de cada elemento do tipo `Item`.

3.3. Escreva a função

```
void sortList( Item list[], size_t num );
```

destinada a ordenar, de acordo com o critério do comando "l", os `num` itens existentes em `list`. Deve utilizar a função `qsort` da biblioteca normalizada.

3.4. Com vista à implementação do comando "f", construa a estrutura de dados necessária e escreva a função

```
void sortAuxList( Item *auxList[], size_t num );
```

destinada a ordenar, de acordo com o critério do comando "f", os `num` ponteiros do *array* `auxList`.

Tendo como objetivo a eficiência do comando "f", não prejudicando a eficiência do comando "l", deve criar um acesso específico para os itens que são ficheiros, ordenado de acordo com o comando "f", sem modificar a ordenação da lista completa. Propõe-se a criação de um *array* de ponteiros para `Item`, cujos elementos são inicialmente apontados para os itens que representam ficheiros e em seguida ordenados, usando também a função `qsort`. Na resposta ao comando "f", deve utilizar o *array* de ponteiros já ordenado.

3.5. Escreva e teste o programa de aplicação capaz de ler a diretoria, preencher os dados e responder aos comandos "l", "f" e "q" especificados.

ISEL, 2021.05.06