Ficheiro: fixsizedtypes.h
Este ficheiro define um conjunto de tipos e macros base. É definido um tipo que representará um byte sem sinal (uint8_t), bem como um tipo de dados que deverá representar uma expressão lógica (bool) e dois possíveis valores para o mesmo: true ou false.

Ficheiros: list.h e list.c

Podemos considerar que o coração do programa está nestes ficheiros. Eles definem uma lista genérica duplamente ligada com cabeçalho e rodapé[1], bem como um conjunto de funções para a manipular. O tipo de dados definido é:

```
typedef struct _list_t {
    void* data;
    struct _list_t* next;
    struct _list_t* prev;
} list_t;
```

Assumindo-se que em "data" se vai guardar sempre um ponteiro. Adicionalmente, também é definido um tipo compareFunc, que é um ponteiro para uma função, cujo objectivo será detalhado brevemente:

```
typedef int (*compareFunc)(void*, void*);
```

Para manipular este tipo de dados, são fornecidas as seguinte as seguintes funções:

```
list_t* ListNew(void);
```

Esta função cria uma lista e retorna um ponteiro para a mesma. Internamente, recorre à função malloc para alocar espaço para o cabeçalho e o rodapé, inicializando-os de forma apropriada (data=NULL para ambos os elementos).

```
list_t* ListAdd(list_t* list, void* g, compareFunc funct);
```

Esta função adiciona um elemento (g) de forma ordenada à lista list, recorrendo à função funct para a ordenação. Esta função receberá dois elementos que deverá comparar, retornando -1, 0 ou 1 se o primeiro elemento for, respectivamente, menor, igual ou maior do que o segundo. Podem ser utilizadas diferentes funções em diferentes alturas, consoante o critério que se queira utilizar. É retornado um ponteiro para o recém adicionado nó. Internamente, esta função recorre à função ListSearch, em seguida documentada. A memória em g não deverá ser libertada, pois a lista armazenará um ponteiro para ela.

```
list_t* ListSearch(list_t* list, void* key, compareFunc funct);
```

A função ListSearch recebe um elemento, key, e a função funct, já mencionada anteriormente, retornando o primeiro nó que seja "maior ou igual" (de acordo com os critérios de funct) a key. Se nenhum elemento nestas condições for encontrado, NULL é retornado.

```
void ListDel(list_t* l);
```

1 Definimos o nó de cabeçalho e o nó de rodapé como dois nós adicionais da lista-ligada, de conteúdo desprezável, a serem colocados no princípio e no fim, respectivamente, da lista duplamente ligada, por forma a permitir algoritmos mais genéricos e/ou mais eficientes em termos de tempo de processamento.

Esta função elimina o elemento l da lista ligada e liberta a memória associada a data, bem como a memória associada ao próprio nó.

```
void ListDelete(list_t* list);
```

A função ListDelete (não confundir com ListDel) elimina toda a lista e, por isso, deve receber um ponteiro para o cabeçalho. Internamente, o seu funcionamento é equivalente a invocar sucessicamente ListDel nos nós da lista, excepto no cabeçalho, eliminados à parte.

```
list_t* ListIterateNext(list_t* list);
list_t* ListIteratePrev(list_t* list);
```

As funções ListIterateNext e ListIteratePrev iteram sobre a lista, retornando um ponteiro para o próximo elemento ou para o elemento anterior, respectivamente.

```
bool ListIsHeader(list_t* list);
bool ListIsFooter(list_t* list);
```