Guidelines

Compilação

- O programa deve ser compilado usando o comando `gcc -Wall -Wextra -Werror -ansi -pedantic`.
- O compilador não deve apresentar erros ou avisos ("warnings").
- Recomenda-se usar a versão 7, ou superior, do compilador (consulte `gcc --version`).

Formatação

- A formatação do código deve ser **consistente**. Deve utilizar o tabulador e não espaços brancos. Os editores permitem definir quantos espaços devem ser usados para representar o tabulador. Recomenda-se a utilização de um editor de texto com indentação automática.
- Os carateres acentuados devem ser representados em UTF-8.
- As linhas de código não deverão exceder um limite de 80 caracteres.
- Abrir cada chaveta na mesma linha do cabeçalho da função. Fechar cada chaveta numa linha nova, também alinhada com o cabeçalho. O mesmo aplica-se às chavetas associadas aos comandos `for`, `if`, `switch`, e `while`.

Comentários

- O código deve que ser adequadamente comentado. Comentários em excesso devem ser evitados. Por exemplo, não é necessário comentar sobre uma variável que representa um contador de um ciclo.
- Cada ficheiro deve ter um comentário, no início, com uma descrição sucinta e o(s) nome(s) do(s) autor(es).
- Cada função deve ter um comentário, antes do cabeçalho, com uma descrição sucinta.
- Cada variável global e/ou estática deve ter um comentário com uma descrição sucinta. O comentário pode ser colocado na mesma linha da declaração da variável, ou na linha antes.

Organização do código

- Não deve haver repetição de código (código repetido deverá ser colocado numa função).
- O programa deverá estar dividido em diversos ficheiros, implementando cada um deles um módulo coerente.
- Deve poupar memória, evitando repetições de valores.
- Evite funções demasiado longas (e pouco legíveis).
- Recomenda-se usar a ferramenta lizard para calcular a complexidade ciclométrica e o número de linhas de código. A ferramenta não deve emitir avisos ao executar

```
lizard -L 50 -T nloc=25 -C 12 -m
```

Constantes

- Constantes (números, strings) nunca devem aparecer directamente no código. Deverá ser usada a directiva `#define` no início dos ficheiros ou num ficheiro `.h` separado.
- Os comandos e mensagens de erro devem ser agrupadas para facilitar a tradução para outras línguas.

Variáveis globais

- Neste projeto não deve utilizar variáveis globais (*nenhuma*).

Exemplo

```
/*
 * File:  ex.c
 * Author:  mikolas
 * Description: A program exemplifying the use of comments and formatting in C.
*/
#include<stdio.h>

/* The maximum number of values stored. */
#define SZ 100

/* Adds a given value to the vector v at cnt.
 * Returns the number of added values.
*/
int add(int v[], int cnt, int val) {
    if (cnt == SZ)
        return 0;
    vals[cnt] = val;
    return 1;
}

/* Reads n values from stdin and inserts them into the vector vals. */
int main() {
    int vals[SZ]; /* Array of values. */
    int count; /* The number of values stored. */
    int v;

    while (scanf("%d", &v) == 1) {
        count += add(vals, count, v);
    }
    return 0;
}
```

Opção `fsanitize`

A opção `fsanitize` é uma ferramenta útil para analisar o projecto, em particular erros de memória.

Para analisar um teste que está a falhar, deverá efetuar os seguintes passos:

1. Compilar com as flags `-g -fsanitize=address`, e.g. `gcc -g -fsanitize=address -Wall -Wextra -Werror -ansi -pedantic proj2.c`.
2. executar o projecto compilado usando o teste que está a falhar como standard input, e.g.

```
$ ./a.out < testes_publicos/teste27.in
```

Podem também ignorar o output do projecto por forma a ver só os erros da seguinte forma:

```
$ ./a.out < testes_publicos/teste27.in > /dev/null
```

Valgrind

É aconselhável analisar o projecto usando também a ferramenta Valgrind. Para instalar no Ubuntu:

```
$ sudo apt install valgrind
```

Para analisar com o valgrind um teste que está a falhar, deverá efectuar os seguintes passos:

1. Compilar com a flag `-g`, e.g. `gcc -g -Wall -Wextra -Werror -ansi -pedantic *.c`.
2. executar o projecto compilado com o Valgrind usando o teste que está a falhar como standard input, e.g.

   ```
   $ valgrind ./a.out < testes_publicos/teste27.in
   ```

Podem também ignorar o output do projecto por forma a ver só os erros da seguinte forma:

```
$ valgrind ./a.out < testes_publicos/teste27.in > /dev/null
```

Deverá estar particularmente atento/a a mensagens como:

```
Invalid read e Invalid write
```

que indicam está a tentar ler ou escrever fora da área de memória reservada por si. O *valgrind* também detecta a utilização de variáveis não inicializadas dentro expressões condicionais. Nesse caso receberá a mensagem:

```
Conditional jump or move depends on uninitialised value(s).
```

Por fim, o *valgrind* oferece no final a preciosa informação sobre a quantidade de memória alocada e libertada no heap, indicando quando essas duas quantidades não são iguais. Nesse caso o programa tem *memory leaks*. Quando tudo corre bem, o *valgrind* deverá paresentar o seguinte output:

```
HEAP SUMMARY:
in use at exit: 0 bytes in 0 blocks
total heap usage: 1,024,038 allocs, 1,024,038 frees, 28,682,096 bytes alloc
All heap blocks were freed -- no leaks are possible
```

Nota: O valgrind neste momento não tem suporte para Macs. As alternativas são utilizar uma máquina virtual com ubuntu, os computadores dos laboratórios, o nós sigma (`ssh ist1xxxxx@sigma.tecnico.ulisboa.pt` com password do fénix) ou pedir ajuda a um colega/amigo.