Depuração de programas

Existem várias ferramentas e métodos que permitem depurar programas. A dica geral é identificar o menor subconjunto possível do input que continua causar o *bug*. Apresentamos aqui algumas recomendações.

Para correr um teste particular, usem o comando seguinte.

```
./a.out < testes-enunciado/teste01.in
```

onde `a.out` é o vosso programa compilado e `testes-enunciado/teste01.in` é o input que querem testar.

Erros e Warnings (avisos)

Prestem atenção aos erros e warnings que o vosso programa apresenta aquando da compilação. As mensagens costumam ser relativamente explicitas. Quando não o forem, se googlarem o erro/warning apresentado poderão quase sempre identificar o problema em causa.

Quando o programa dá output errado

O resultado do teste acima encontra-se no ficheiro `testes-enunciado/teste01.out`.

O programa `diff` pode ser usado para comparar os dois outputs (existem outras ferramentas com interface gráfica)

```
./a.out < testes-enunciado/teste01.in > myoutput.out
diff myoutput.out testes-enunciado/teste01.out
```

apresentando as diferenças entre os dois ficheiros. Se quiserem entender em detalhe o output do `diff` podem por exemplo ver aqui.

Podem também instalar a ferramenta `icdiff` (Ubuntu: `sudo apt install icdiff`).

```
icdiff myoutput.out testes-enunciado/teste01.out | less -R
```

Quando o programa rebenta

Se um programa rebentar, a ferramenta `valgrind` ou a opção `-fsanitize` podem ser usadas para encontrar a razão.

Para além destas ferramentas, também existe a ferramenta `cppcheck`, que tenta encontrar pedaços de código que podem ser fontes de erro. É um tipo de ferramenta que pode apresentar *false positives*, o que quer dizer que pode apontar erros que de facto não o são.

Opção `fsanitize`

A opção `fsanitize` invoca uma ferramenta `AddressSanitizer` 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 proj2.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
```

Nota: O valgrind neste momento não tem suporte completo para Macs. As alternativas são utilizar uma máquina virtual com ubuntu, os computadores dos laboratórios, ou pedir ajuda a um colega/amigo.

cppcheck

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

```
$ sudo apt install cppcheck
```

Para analisar o projecto usando esta ferramenta deverá executar:

```
$ cppcheck --enable=all --language=c --std=c99 proj2.c
```

ou, para mais informações:

```
$ cppcheck --enable=all --language=c --std=c99 -v proj2.c
```