FOLHA DE PROBLEMAS Nº 1

Comandos da *shell* e Aspectos do gerais dos programas em C no Unix/Linux

---

RECOMENDAÇÕES:

A) Crie pastas para guardar os ficheiros dos programas (código-fonte, executável, etc.) que vier a desenvolver em cada uma das aulas práticas desta disciplina, bem como os ficheiros de dados utilizados por esses programas. Sugestão: dê a essas pastas os nomes: `prob01`, `prob02`, etc. .

B) Em cada aula, guarde os programas com um nome indicativo do número do problema e da alínea respectiva, por exemplo, `p2a.c`, para o programa correspondente ao "problema 2, alínea a)". Se um programa for constituído por vários ficheiros (`.c` e `.h`), ou se o programa necessitar de usar/criar ficheiros de dados, crie uma pasta para conter todos os ficheiros, dando à pasta um nome adequado (ex: `p6`, no caso do problema 6, desta folha de problemas).

C) Em cada programa:

- **Use nomes sugestivos para as variáveis, funções, etc.; isso facilitará a interpretação do código sempre que solicitar ajuda ao docente e também, por si, no futuro.**
- **Faça indentação do código mas não exageradamente (1 ou 2 caracteres é suficiente); isso também facilitará muito a leitura e interpretação do mesmo. Note que, em geral, os editores de texto permitem configurar o espaço de indentação.**
- **Use comentários para descrever a funcionalidade do programa, das funções que desenvolver e dos seus principais blocos de código. Sempre que necessário descreva o significado dos argumentos do programa e das variáveis utilizadas.**

D) Use o manual *online* para esclarecer dúvidas sobre a utilização das funções da biblioteca de C e das chamadas ao sistema, para determinar quais as directivas `#include` que deve incluir nos programas e para interpretar o valor de retorno das chamadas quando elas não são bem sucedidas.

E) Use sempre a opção de compilação `-Wall` e certifique-se, antes de executar qualquer programa, que a compilação não deu origem a nenhum *warning*.

---

1. – Comandos da *shell.*

a) Experimente alguns comandos da *shell* (interpretador de comandos) do Unix/Linux, nomeadamente: `cd`, `ls`, `mkdir`, `rmdir`, `cp`, `mv`, `cat`, `more`, `tail`, `pwd`, `ps`, `find`, `grep`, `chmod`, `clear` e `echo`. Consulte o manual *online* e os documentos de apoio indicados na página Web da disciplina ou outros disponíveis na Web, para esclarecer dúvidas sobre a utilização destes comandos.

b) Experimente os mecanismos de redireccionamento de entradas/saídas de um comando, de execução de comandos em sequência, em *pipeline* e em *background*.

2. – Geração de um executável em C.

a) Compile e execute o seguinte programa:

```
#include <stdio.h>
int main(void)
{
  printf("Hello !\n");
  return 0;
}
```

b) Substitua o nome da função `main` por outro nome e tente compilar novamente. Interprete o resultado. Volte a repor o nome da função `main`.

3. – Código de retorno de um programa.

a) Execute o programa do problema 2 e verifique que o código de retorno do programa é zero, como esperado, recorrendo ao comando da *bash* `echo $?` que mostra o código de retorno do último comando executado.

b) Altere o programa por forma a que o código de retorno tome outro valor e confirme, na *shell*, o valor retornado.

c) Execute o comando `ls` e volte a testar o valor de retorno.

4. – Acesso aos argumentos da linha de comando. Argumentos de diferentes tipos.

a) Altere o programa do problema 2 por forma a aceitar um nome de uma pessoa como argumento da linha de comandos, escrevendo a saudação "Hello *nome* !". Procure descobrir como é possível passar como parâmetro um nome que seja constituído por mais do que uma palavra (ex: Rui Silva).

b) Modifique o programa da alínea anterior por forma a aceitar como parâmetros um nome de uma pessoa e um número e que repita a saudação tantas vezes quantas as indicadas pelo número.

5. – Acesso às variáveis de ambiente. Criação de variáveis de ambiente.

a) Escreva um programa que apresente no ecrã o valor de todas as variáveis de ambiente.

b) Modifique o programa do problema 2 por forma a que ao executar, sem qualquer parâmetro da linha de comando, apresente uma mensagem personalizada, consoante o utilizador que o executar, do tipo "Hello *user* !" em que *user* representa o valor da variável de ambiente USER. Sugestão: use a função da biblioteca de C `strncmp` para determinar qual o valor da variável de ambiente USER.

c) Modifique o programa do problema anterior por forma a usar a função da biblioteca de C `getenv()`.

d) Crie, com comandos da *shell*, uma variável de ambiente chamada USER_NAME, atribuindo-lhe um valor igual ao seu nome completo. Modifique a saudação do programa da alínea anterior por forma a usar o valor de USER_NAME em vez do valor de USER.

6. – Análise de situações de erro.

a) Compile o seguinte programa e corrija os erros de compilação que forem assinalados.

```
// PROGRAMA p6a.c
#include <stdio.h>
#define BUF_LENGTH 256
int main(void)
{
  FILE *src, *dst;
  char buf[buf_length];
  if ( ( src = fopen( "infile.txt", "r" ) ) == NULL )
  {
    exit(1)
  }
  if ( ( dst = fopen( "outfile.txt", "w" ) ) == NULL )
  {
    exit(2)
  }
```

```
    while( ( fgets( buf, MAX, src ) ) != NULL )
    {
      fputs( buf, dst );
    }
    fclose( src );
    fclose( dst );
    exit(0);  // zero é geralmente indicativo de "sucesso"
}
```

b) Mesmo depois de compilado com sucesso, o programa não executará correctamente. Procure identificar o motivo, antes de executar o programa, mas não efectue ainda nenhuma alteração.

c) Execute o programa. Verifique, na *shell*, qual o seu código de terminação.

d) Altere o programa por forma a usar os mecanismos de detecção e descrição de erros, inserindo chamadas às funções `perror()` ou `strerror()` antes das chamadas `exit()`. Consulte, no manual, a descrição daquelas funções de descrição de erros e interprete os resultados obtidos.

e) Substitua as chamadas a `perror()` ou `strerror()` por uma instrução que mostre no ecrã o valor da variável global `errno`.

f) Usando um editor de texto crie um ficheiro com um conteúdo à sua escolha e dê-lhe o nome `infile.txt`. Volte a executar o programa e interprete os resultados. Verifique se o conteúdo do ficheiro `outfile.txt` é o esperado.

g) Altere o programa por forma a poder ser utilizado com quaisquer outros dois ficheiros cujos nomes deverão ser passados como parâmetros da linha de comando. Verifique o que acontece quando executa o programa sem argumentos.

h) Altere o programa por forma a apresentar uma mensagem de utilização "usage: *nome_do_executável* file1 file2" caso não sejam passados os nomes de dois ficheiros como argumentos da linha de comando, terminando imediatamente após apresentar aquela mensagem.

7. – Instalação e execução de *atexit handlers.*

a) Escreva um programa que teste a possibilidade de utilização de *atexit handlers* recorrendo à função **atexit**. O programa deverá registar dois desses *handlers*, cada um dos quais deverá, simplesmente, imprimir uma mensagem do tipo "Executing exit handler x", em que x é um número diferente para cada *handler*; o programa principal deverá escrever a mensagem "Main done!", imediatamente antes de terminar.

b) Tire conclusões sobre a relação entre a ordem de instalação e de execução dos *handlers*. Será possível instalar um *handler* mais do que uma vez ? O que acontece se fizer uma chamada `abort()` na função `main`, antes de escrever "Main done!" no ecrã ? E se algum dos *handlers* terminar com `exit()` ?

8. – Medição de tempos de execução de um programa

a) Escreva um programa que tenha como parâmetros da linha de comando dois números, n1 e n2, e que gere repetidamente números aleatórios no intervalo [0..n1[terminando quando for gerado o número n2 (ex: o comando `p8a 1000 13`, deve gerar números no intervalo [0..999[terminando quando for gerado o número 13). O programa deve apresentar no ecrã os números gerados, sendo cada número precedido do número da iteração. Sugestão: use `srand()` para garantir que, em cada execução, são geradas sequências de números aleatórios diferentes.

b) Modifique o programa da alínea anterior por forma a apresentar no ecrã os seguintes tempos de execução: o tempo real, e os tempos de *CPU* nos modos *user* e *system*.

9. – Linguagem de *scripting* da *shell.*

a) Escreva um *script* (dê-lhe, por exemplo, o nome `cx`, de *compile and execute*) que admita como parâmetro o nome de um ficheiro contendo um programa em C (o nome não deve ter a extensão ".c" que deverá ser acrescentada pelo *script*) e de seguida compile e execute o programa. Se a compilação for bem sucedida, o executável deve ser guardado num ficheiro com o nome do parâmetro (ex: `cx p2a`, deverá compilar o ficheiro `p2a.c`). O *script* deve executar a seguinte sequência de operações:

- remover ficheiros (executável e ficheiro com a extensão ".o") eventualmente existentes como resultado de compilações anteriores do mesmo programa; caso estes ficheiros não existam não deve ser mostrada nenhuma mensagem no ecrã (ver opções do comando "rm" da *shell*)
- compilar o programa (não esquecer de acrescentar a extensão ".c" ao parâmetro do *script*)
- se a compilação for bem sucedida, executar o programa resultante da compilação, se não mostrar uma mensagem de erro "COMPILATION ERROR"

Notas:
1) Comece por procurar informação sobre a forma de construir e executar um *script* na *bash (Bourne-Again Shell*).
2) Adiante será introduzida uma ferramenta mais adequada para fazer a compilação de programas: o comando `make`.

b) Escreva um *script* para criar uma pasta para cada aula prática, com os nomes prob01, prob02, etc., como sugerido nas "Recomendações" iniciais. Sugestão: pesquise informação sobre como executar um ciclo na linguagem de *scripting* da *bash*.

---

10. – Decomposição de uma *string* em elementos constituintes.

a) Escreva um programa em C que leia uma *string* representando uma linha de comando simples, de Unix, por exemplo,

- `ps`
- `ls -la`
- `find / -name gcc`
- `cat *.txt > all.txt`

e decomponha essa linha de comando nos seus *tokens* (símbolos) mostrando-os no ecrã. Os *tokens* devem estar separados por um ou mais espaços.
Nota: Use a função `fgets()` para ler a linha de comando e a função `strtok()` para decompor a linha.

b) Altere o programa da alínea anterior por forma a guardar os *tokens* num *array* de *strings* (um *token* em cada elemento do *array*), só apresentado o resultado no ecrã depois de concluída a decomposição.
Nota: Pode admitir que é conhecido o número máximo de *tokens*.

c) Altere o programa da alínea anterior por forma a permitir a separação de uma linha de comando composta por vários sub-comandos (comandos simples) a serem executados em sequência (separados por '`;`') ou em *pipeline* (separados por '`|`') como, por exemplo,

- `cd ~; ls -la` (os comandos `cd ~` e `ls -la` são executados em sequência)
- `cat *.c| grep "Hello"| more` (os comandos simples `cat *.txt`, `grep "Hello"` e `more` são executados em *pipeline*)

Nota: Pode admitir que é conhecido o número máximo de sub-comandos .