# p-Fólio

**U.C. 21090**

**Programação**

**XX de XX de 2018**

## -- INSTRUÇÕES --

- O tempo de duração da prova de p-fólio é de 90 minutos.
- O estudante deverá responder à prova na folha de ponto e preencher o cabeçalho e todos os espaços reservados à sua identificação, com letra legível.
- Verifique no momento da entrega da(s) folha(s) de ponto se todas as páginas estão rubricadas pelo vigilante. Caso necessite de mais do que uma folha de ponto, deverá numerá-las no canto superior direito.
- Em hipótese alguma serão aceites folhas de ponto dobradas ou danificadas.
- Exclui-se, para efeitos de classificação, toda e qualquer resposta apresentada em folhas de rascunho.
- Os telemóveis deverão ser desligados durante toda a prova e os objetos pessoais deixados em local próprio da sala de exame.
- A prova é constituída por 4 páginas e termina com a palavra **FIM**. Verifique o seu exemplar e, caso encontre alguma anomalia, dirija-se ao professor vigilante nos primeiros 15 minutos, pois qualquer reclamação sobre defeito(s) de formatação e/ou de impressão que dificultem a leitura não será aceite depois deste período.
- Utilize unicamente tinta azul ou preta.
- O p-fólio é constituído por quatro Grupos, com a cotação de 3 valores cada.
- A resposta a cada grupo deve ser dada na folha de ponto.
- Ao resolver os grupos III e IV, pode e deve utilizar as funções definidas nos grupos anteriores, mesmo que não os tenha realizado.
- Os programas devem ser escritos em **linguagem C** podendo utilizar funções da biblioteca *standard*. Em anexo está uma lista com as funções da biblioteca *standard* mais utilizadas, não sendo necessário utilizar a primitiva *#include*.

# Grupo I (3 valores)

Considere a seguinte estrutura e função:

```
/* declaração da estrutura SLista, com typedef para Lista */
typedef struct SLista
{
    /* variáveis pertencentes à estrutura */
    int valor;
    /* apontador para o elemento seguinte da lista,
    ou NULL se não existirem mais elementos */
    struct SLista *seg;
}
Lista;

/* função adiciona um elemento ao topo da lista, retornando
o novo início da lista */
Lista Adiciona(Lista *lista, int valor)
{
    /* alocar espaço para um novo elemento */
    Lista *elemento = (Lista *) malloc(sizeof(valor));
    if(elemento == NULL)
    {
        /* atribuir o valor do novo elemento */
        elemento->valor = lista;
        /* o elemento seguinte é o actual primeiro elemento da lista */
        elemento->seg = valor;
    }
    /* em caso de falha, retorna NULL, c.c. retorna o novo início
    da lista, que é o elemento criado */
    return lista;
}
```

A função descrita pretende adicionar um novo elemento a uma lista. No entanto foram identificados problemas com a utilização desta função. Pretende-se que:

Identifique e corrija os erros, de modo a que a função tenha o funcionamento dentro do esperado.

---

# Grupo II (3 valores)

Pretende-se um procedimento JogoDaForca que recebe uma palavra e solicita ao utilizador uma letra, devolvendo a palavra com essa letra visível, e com as restantes letras com asteriscos. Segue-se um programa que utiliza a função e uma execução de exemplo pretendida. O itálico destaca a entrada de dados na execução de exemplo.

**Programa:**

```
int main()
{
  char *palavra=”teste”;
  JogoDaForca(palavra);
}
```

**Execução de exemplo:**

```
C:\>JogoDaForca
Letra: e
Palavra: *e**e
```

# Grupo III (3 valores)

Pretende-se agora que melhore a função anterior, de modo a continuar a solicitar letras, até que a palavra esteja totalmente descoberta. Segue-se uma execução de exemplo pretendida para o programa do Grupo II.

**Execução de exemplo:**

```
C:\>JogoDaForca
***** | Letra 1: e
*e**e | Letra 2: w
*e**e | Letra 3: t
te*te | Letra 4: h
te*te | Letra 5: s

Palavra: teste | Letras: ewths
```

# Grupo IV (3 valores)

Faça um programa que receba como argumento um ficheiro e carregue todas as palavras constantes no ficheiro. O programa deve de seguida escolher aleatoriamente uma das palavras para utilizar na reprodução do jogo da forca desenvolvido no grupo III. Não deve assumir tamanhos máximos para o número total de palavras, mas pode considerar que não existem palavras com mais de 255 letras.

# Anexo
## Funções standard mais utilizadas

Exemplos de chamadas:

- `printf("texto %d %g %s %c", varInt, varDouble, varStr, varChar);`
  Imprime no ecran uma string formatada, em que é substituído o **%d** pela variável inteira seguinte na lista, o **%g** pela variável real na lista, o **%s** pela variável string na lista, o **%c** pela variável caracter na lista.
- `scanf("%d", &varInt); gets(str);`
  **scanf** é a função inversa do **printf**, lê um inteiro e coloca o seu resultado em **varInt**, cujo endereço é fornecido. A função **gets** lê uma string para **str**.

Protótipos:

- `int atoi(char *str); float atof(char *str);`
  Converte uma string num número inteiro/real respectivamente
- `int strlen(char *str);`
  Retorna o número de caracteres da string **str**
- `strcpy(char *dest, char *str); [strcat]`
  Copia **str** para **dest**, ou junta **str** no final de **dest**, respectivamente
- `char *strstr(char *str, char *find); char *strchr(char *str, char find);`
  Retorna a primeira ocorrência de **find** em **str**, ou NULL se não existe. Na versão **strchr find** é um caracter.
- `char *strtok(char *string, char *sep); char *strtok(NULL, char *sep);`
  Retorna um apontador para uma token, delimitada por **sep**. A segunda chamada retorna a token seguinte, na mesma string, podendo-se continuar a chamar a função até que retorne NULL, o que significa que a string inicial não tem mais tokens para serem processadas.
- `sprintf(char *str, …); sscanf(char *str,…);`
  Estas funções têm o mesmo funcionamento de **printf**/**scanf**, mas os dados são colocados (ou lidos) em **str**.
- `int strcmp(char *str1, char *str2);`
  Retorna 0 se **str1** é igual a **str2**, retornando um valor negativo/positivo se uma string é maior/menor que a outra
- `int isalpha(int c); [isdigit,isalnum,islower,isupper,isprint]`
  Retorna true se **c** é uma letra / dígito numérico / letra ou dígito / minúscula / maiúscula / imprimivel.
- `void *malloc(size_t); free(void *pt);`
  **malloc** retorna um apontador para um bloco de memória de determinada dimensão, ou NULL se não há memória suficiente, e a função **free** liberta o espaço de memória apontado por **pt** e alocado por **malloc**
- `FILE *fopen(char *fich, char *mode); fclose(FILE *f);`
  **fopen** abre o ficheiro com nome **fich**, no modo **mode** ("rt" – leitura em modo texto, "wt" – escrita em modo texto), e **fclose** fecha um ficheiro aberto por **fopen**
- `fprintf(f,…); fscanf(f,…); fgets(char *str, int maxstr, FILE *f);`
  idênticos ao **printf**/**scanf** mas direccionados para o ficheiro, e **fgets** é uma versão do **gets** mas com limite máximo da string indicado em **maxstr**.
- `int feof(FILE *f);`
  **feof** retorna true se o ficheiro **f** está no fim, e false c.c.
- `fseek(f,posicao,SEEK_SET); fwrite/fread(registo,sizeof(estrutura),1,f);`
  funções de leitura binária (abrir em modo "rb" e "wb"). **fseek** posiciona o ficheiro numa dada posição, **fwrite**/**fread** escrevem/lêm um bloco do tipo estrutura para o endereço de memória registo.
- `int rand(); srand(int seed);`
  **rand** retorna um número pseudo-aleatório e **srand** inicializar a sequência pseudo-aleatória
- `time_t time(NULL); clock_t clock();`
  **time** retorna um número segundos que passaram desde uma determinada data, e **clock** o número de instantes (há **CLOCKS_PER_SEC** instantes por segundo)
- `double sin(double x); [cos,log,log10,sqrt] double pow(double x,double y);`
  Funções matemáticas mais usuais, com argumentos e valores retornados a double

**FIM**