U.C. 21173

Introdução à Programação

27 de fevereiro de 2020

-- INSTRUÇÕES --

- O tempo de duração da prova de exame é de 150 minutos (2h30m).
- 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 5 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 exame é constituído por 5 grupos, estando a cotação indicada em cada grupo.
- 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)

O programa seguinte pretende identificar se um dado número introduzido pelo utilizador é ou não primo. No entanto foram identificados problemas com a utilização deste programa.

Identifique e corrija os erros, de modo a que o programa tenha o funcionamento correto.

```
int main()
{
    int divisor; n;

    printf("Funcao que verifica se um numero N e' primo:\nIndique N:")
    scanf("%d",n);

    if(n<10)
        printf("Erro: o numero tem de ser maior que zero.\n");
    else {
        /* a variável divisor vai iterar até que o seu quadrado seja
           maior que o número (evitando a utilização da raiz quadrada) */
        divisor=10;
        while(divisor*divisor<=n) {
            /* o número é divisível se o resto da divisão for nula */
            if(divisor%n==0) {
                printf("\nNumero %d divisivel por %d\n",divisor,n);
                return;
            }
            /* mostrar iteração */
            printf("%d ",divisor);
            divisor++;
        }
        printf("\nNumero %d primo!\n",n);
    }
}
```

Grupo II (3 valores)

Implemente a função *MostrarDamas*, chamada no programa em baixo, que mostra num tabuleiro de lado 4 a posição das respetivas damas, representando por '#' a posição da dama, e por '.' uma casa vazia no tabuleiro. O vetor contém para cada linha a coluna em que a dama se encontra. O vetor fornecido indica que na primeira linha a dama encontra-se na coluna 1 (as colunas vão de 0 a 3), na segunda linha na coluna 3, e assim sucessivamente. O mesmo ocorre na representação do tabuleiro, na primeira linha o cardinal está na segunda posição (coluna 1), na segunda linha está na quarta posição (coluna 3), e assim sucessivamente. Faça uma função que funcione para qualquer que seja o valor do lado, desde que seja um inteiro positivo.

Programa:

```
int main()
{
    int linhasDamas[4] = { 1,3,0,2 };
    MostrarDamas(linhasDamas, 4);
}
```

Execução de Exemplo:

```
C:\...>normal1920g2
 . # . .
 . . . #
 # . . .
 . . # .
```

Grupo III (3 valores)

Implemente a função *ColocarDama*, utilizada o programa em baixo. A função recebe três argumentos, o vetor onde deve colocar as colunas de cada dama, em cada linha, compatível com a função do grupo anterior. O segundo argumento é um valor K com o número da linha que a função deverá preencher, e o terceiro argumento é o valor *N* com o lado do tabuleiro, idêntico ao segundo argumento do grupo anterior.

A função deverá colocar uma dama na linha *K*. Para escolher a coluna da dama, a mesma não pode estar em ataque direto com uma das damas colocadas nas linhas anteriores (*0* a *K-1*). Um ataque entre uma dama na linha W e a dama na linha K, existe quando ambas estão na mesma coluna, ou na mesma diagonal, portanto o valor absoluto da diferença entre as colunas em que ambas as damas se encontram, é igual K-W. Por exemplo, K=3 e W=1, se a dama W estiver na coluna 2, a dama K não pode estar nem na coluna 2, nem na coluna 0 (2-2) ou coluna 4 (2+2), já que K-W=2.

Após localizar a primeira posição válida, a função deverá procurar colocar as restantes damas, de *K+1* até *N* (a função pode ser recursiva). No caso de conseguir colocar as restantes damas, deverá retornar 1, caso contrário deve procurar na posição válida seguinte da linha K, repetir o processo. Apenas se nenhuma das posições válidas permitir colocar as restantes damas, a função deverá retornar 0.

Na execução de exemplo é possível ver que a função encontrou uma solução válida para 5 damas.

Programa:

```
int main()
{
    int linhasDamas[5];
    if(ColocarDama(linhasDamas, 0, 5))
        MostrarDamas(linhasDamas, 5);
}
```

Execução de Exemplo:

```
C:\...>normal1920g3
 # . . . .
 . . # . .
 . . . . #
 . # . . .
 . . . # .
```

Grupo IV (3 valores)

Faça agora um programa utilizando as funções dos grupos anteriores, que solicite ao utilizador um inteiro para o lado do tabuleiro, e aloque memória necessária para o valor introduzido. No caso de algum problema na alocação, o programa deve retornar um erro, caso contrário deve executar o programa do grupo anterior, libertando de seguida a memória alocada.

Execução de Exemplo:

```
C:\...>normal1920g4
Indique N:6

 . # . . . .
 . . . # . .
 . . . . . #
 # . . . . .
 . . # . . .
 . . . . # .
C:\...>normal1920g4
Indique N:10

 # . . . . . . . . .
 . . # . . . . . . .
 . . . . . # . . . .
 . . . . . . . # . .
 . . . . . . . . . #
 . . . . # . . . . .
 . . . . . . . . # .
 . # . . . . . . . .
 . . . # . . . . . .
 . . . . . # . . . .
C:\...>normal1920g4
Indique N:-10
Problema ao alocar memoria.
```

Grupo V (8 valores)

Suponha que tem de desenvolver um programa para registo de informação de uma agência imobiliária, especializada no arrendamento de casas de habitação. É necessário registar sobre imóveis a morada, número de assoalhadas, custo de aluguer, cidade e zona. Pode vir a ser necessário atualizar as cidades e as respetivas zonas.

a) Defina a estrutura de dados necessária para registar a informação referida.

b) Faça um programa que grave e leia informação da estrutura de dados para um ficheiro de texto. O formato do ficheiro é opção sua.

c) Faça o relatório que agregue as cidades, e para cada uma destas cidades, agregue as zonas, indicando a quantidade de habitações em cada cidade e zona.

d) Faça um relatório que permita listar os imóveis, após aplicar uma restrição sobre a cidade, e número de assoalhadas mínimo, ordenando tudo por custo de aluguer.

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