U.C. 21090

Programação

XX de XXX de 2018

-- INSTRUÇÕES --

- O tempo de duração da prova de exame é de 150 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 exame é constituído por 5 grupos, estando a cotação indicada em cada grupo.
- 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 função:

```
float mdc(int x, int y)
{
    /* caso y=0, retornar x */
    if(x = 0);
        return y;
    /* c.c. retornar mdc(y,mod(x,y)), exactamente o comando seguinte
    após substituir o mod pelo comando do resto da divisão % */
    return mdc(y, y / x)
}
```

A função descrita pretende retornar o mínimo divisor comum entre x e y. No entanto foram identificados problemas com a utilização desta função. Pretende-se que:

Identifique e corrija os erros, de modo a que a função retorne o valor expectável.

Grupo II (3 valores)

Pretende-se um procedimento GeraVetor, para inicializar um vetor de dimensão N, com números inteiros aleatórios entre A e B, não repetidos, não sendo necessário confirmar que A<B e que B-A+1>=N. O programa em baixo exemplifica a utilização do procedimento para N=5, A=24 e B=34.

Programa:

```
int main()
{
  int sequencia[5];
  int i;
  srand(1);
  GeraVetor(sequencia,5,24,34);
  for(i=0;i<5;i++)
    printf("%d ", sequencia[i]);
}
```

Grupo III (3 valores)

Defina uma estrutura de dados para uma carta BINGO e faça um programa que permita gerar uma carta. A carta deve conter números de 1 a 75 não repetidos, numa matriz de 5x5, sem a casa central. Os números da primeira coluna devem ser gerados de entre os números de 1 a 15, os da segunda coluna de entre os números de 16 a 30, na terceira coluna de 31 a 45 (apenas 4 números já que a casa central não tem número), na quarta de 46 a 60, e na quinta de 61 a 75. Faça também um procedimento para mostrar uma dada carta BINGO no ecrã.

Grupo IV (3 valores)

Faça uma função que receba uma carta BINGO e um vetor com um determinado número de números saídos, e verifique se existe uma linha, coluna ou diagonal completamente preenchida na carta fornecida.

Grupo V (8 valores)

Suponha que tem de desenvolver um programa para uma esquadra de polícia. Pretende-se um sistema de informação de assaltos, que resultam em queixa na esquadra da polícia. A cada delito deve ser associada uma data, o montante e as pessoas envolvidas. O tipo de envolvimento de uma pessoa num delito está classificado, no entanto este pode vir a ser atualizado (vítima, agressor, testemunha). Não deve assumir nenhum valor máximo para o número de queixas, pessoas, e tipos de delitos.

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 um relatório mensal para um dado mês, que contenha informação diária sobre o número de assaltos, bem como o número de vítimas, número de agressores, testemunhas, e restantes tipos de envolvimentos;

d) Faça um relatório que permita identificar os 10 agressores que tenham o maior valor no somatório dos montantes dos seus assaltos.

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