U.C. 21173

Introdução à Programação

29 de julho de 2019

-- 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 o seguinte programa:

```
int main()
{
    int n, i, soma;

    printf("Calculo da soma dos primeiros N numeros.\nIndique N:")
    /* ler um número inteiro */
    scanf("%d", &n);
    /* na variável soma, será acumulado o resultado */
    soma = 0;
    /* a variável i vai iterar de 1 a N */
    i = 1;
    while(i < n);
    {
        /* a variável soma vai acumulando o resultado de 1 até i */
        soma = i + i;
        /* mostrar o resultado parcial */
        printf("\n  adicionar %d, parcial %d"; i; soma);
        /* incrementar a variável i */
        i = i + i;
    }
    /* mostrar resultado final */
    printf("\nTotal: %s\n", soma);
}
```

O programa pretende calcular a soma dos primeiros N números inteiros. No entanto foram identificados problemas com a utilização deste programa. Pretende-se que:

Identifique e corrija os erros, de modo a que o programa tenha o comportamento expectável.

Grupo II (3 valores)

Implemente a função *Baralhar*, chamada no programa em baixo, que troca o valor do vetor *v* com *N* elementos, de forma aleatória. Nas execuções de exemplo, pode-se ver que o vetor inicializado de 1 a N de forma sequencial, é retornado de forma baralhada.

Programa:

```
int main() {
    int N, i, *v;
    srand(1);

    printf("N: ");
    scanf("%d", &N);

    if (N > 0) {
        v = (int*) malloc(sizeof(int)*N);
        if (v != NULL) {
            for (i = 0; i < N; i++)
                v[i] = i+1;
            Baralhar(v, N);
            printf("Vetor: ");
            for (i = 0; i < N; i++)
                printf("%d ", v[i]);
        }
    }
}
```

Execuções de Exemplo:

```
C:\...>recurso1819g2
N: 4
Vetor: 2 4 3 1
C:\...>recurso1819g2
N: 10
Vetor: 2 10 9 3 1 5 4 8 7 6
```

Grupo III (3 valores)

Implemente as funções *MostraQuadrado* e *VerificaMagico*, utilizadas no programa em baixo. Este programa foi alterado do grupo anterior, em que a variável *N* é agora a largura de um quadrado. A função *Baralhar* recebe agora um vetor de *N*N* elementos. As funções *MostraQuadrado* e *VerificaMagico*, recebem o mesmo vetor *v*, mas que deverá ser interpretado como um quadrado. A função *MostraQuadrado* deve apresentar os números no vetor em quadrado, de acordo com as execuções exemplo. A função *VerificaMagico* deve retornar um valor não nulo no caso de todas as linhas e colunas tiverem a mesma soma (não precisa de verificar as diagonais). Pode-se ver nas execuções de exemplo que nenhum dos quadrados gerados aleatoriamente são na verdade mágicos.

Programa:

```
int main() {
    int N, i, *v;
    srand(1);

    printf("N: ");
    scanf("%d", &N);

    if (N > 0) {
        v = (int*) malloc(sizeof(int)*N*N);
        if (v != NULL) {
            for (i = 0; i < N*N; i++)
                v[i] = i+1;
            Baralhar(v, N*N);
            MostraQuadrado(v, N);
            if (VerificaMagico(v, N))
                printf("\nQuadrado magico!");
        }
    }
}
```

Execuções de Exemplo:

```
C:\...>recurso1819g3
N: 3

Quadrado:
   6   5   9
   8   3   1
   7   4   2
C:\...>recurso1819g3
N: 4

Quadrado:
  10   4   9   1
   2  11  15   8
   6  16   5  12
  14  13   3   7
```

Grupo IV (3 valores)

Faça um programa que tente encontrar quadrados mágicos, e mostre um quadrado mágico assim que o encontre. O programa deve tentar gerar 10000 quadrados aleatórios, verificando cada um se é mágico ou não. Nas execuções de exemplo pode-se ver que este método foi suficiente para encontrar um quadrado mágico de 3x3 (não verificadas as diagonais), mas não conseguiu encontrar um quadrado mágico de 4x4.

Execuções de Exemplo:

```
C:\...>recurso1819g4
N: 3

Quadrado:
   6   1   8
   2   9   4
   7   5   3
Quadrado magico!
C:\...>recurso1819g4
N: 4
```

Grupo V (8 valores)

Suponha que tem de desenvolver um programa para uma empresa de transportes de mercadorias. Pretende-se mais especificamente, registar informação sobre as encomendas transportadas em camiões, entre dois mercados. Sobre cada encomenda deve ser registado o volume e o peso. Para cada camião deve ser registada a capacidade de transporte (volume/peso), as encomendas associadas e respetiva data. Cada camião pode efetuar uma viagem por dia.

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 com o registo das viagens de um dado camião, uma linha por viagem/dia, indicando para cada viagem o número de encomendas transportadas, bem como o total de volume e peso;

d) Faça um relatório para um dado mês, com o total de viagens realizadas, total do número de encomendas, total de volume e total de peso. O relatório deverá estar desagregado por dias, com um dia por linha, apresentando no final os totais.

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