INTRODUÇÃO À PROGRAMAÇÃO | 21173

Data e hora de realização

17 de fevereiro de 2022, às 15h de Portugal Continental

Duração da prova

90m + 60m

Instruções

- O estudante deverá responder à prova na folha de resolução.
- A cotação é indicada junto de cada pergunta.
- A prova é individual, mas pode ser realizada com consulta. Todos os elementos consultados devem ser referenciados na prova.
- A interpretação do enunciado das perguntas também faz parte da sua resolução, pelo que, se existir alguma ambiguidade, deve indicar claramente como foi resolvida.
- A prova é constituída por 4 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.
- Caso já tenha utilizado o VPL, pode realizar os grupos II, III e IV no VPL através do link: https://elearning.uab.pt/mod/quiz/view.php?id=728838 Independentemente de utilizar ou não o VPL, tem de colocar o código e resultados na folha de resolução.

Enunciado

Grupo I (3 valores)

A função seguinte pretende converter um número em árabe para numeração romana. No entanto foram identificados problemas com a utilização desta função (comentários corretos).

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

```
/* ArabeParaRomana: recebe um número em árabe, e retorna a sua string em
numeração romana  */
void ArabeParaRomana(int arabe, char romana)
{
    /* valores e respetivas designações em numeração romana, por ordem */
    static int valor[] =
    {
        1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1, 0
    };
    static char *texto[] =
    {
        "M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I", ""
    };
    int i;
    /* inicializar a numeração romana com a string vazia */
    strcpy(romana, "vazia");
    /* processar todos os valores, do maior para o menor */
    for(i = 0; valor[i] >= 0; i++)
        /* adicionar a string se o número for maior ou igual ao seu valor */
        while(arabe <= valor[i])
        {
            /* concatenar a string */
            strcpy(romana, texto[i]);
            /* subtrair o valor colocado na numeração romana, no número árabe */
            valor[i] -= arabe;
        }
}
```

Grupo II (3 valores)

Complete o programa em baixo, faltando definir a função *MostraMundo*, utilizada na função main.

```
#define MAX_SIZE 100
unsigned int randaux()
{
    static long seed = 1;
    return(((seed = seed * 214013L + 2531011L) >> 16) & 0x7fff);
}
void GerarMundo(int mundo[MAX_SIZE][MAX_SIZE], int L, int C)
{
    int i, j;
    for (i = 0; i<L; i++)
        for (j = 0; j<C; j++)
            mundo[i][j] = randaux() % 5 - 2;
}


void main() {
    int i, j, L, C;
    int mundo[MAX_SIZE][MAX_SIZE];
    scanf("%d %d", &L, &C);
    GerarMundo(mundo, L, C);
    MostraMundo(mundo, L, C);
}
```

Notar que o valor de *L* e *C* é introduzido pelo utilizador, após o qual um array *mundo* é preenchido pela função *GerarMundo*. A função *MostraMundo* é chamada posteriormente, devendo ter em atenção a saída prevista nos casos de teste em baixo.

Casos de execução (disponíveis no VPL 4 casos de teste):

Entrada	Saída
10 5	Linha 1: -1 0 2 -2 2 Linha 2: 2 1 1 0 2 Linha 3: -2 -2 -1 0 -1 Linha 4: -1 -2 0 0 -1 Linha 5: -1 2 0 1 0 Linha 6: 0 -1 -1 1 -2 Linha 7: 0 -1 -1 1 2 Linha 8: 0 0 2 -2 2 Linha 9: 1 -1 0 1 1 Linha 10: 2 -1 -1 1 1
5 10	Linha 1: -1 0 2 -2 2 2 1 1 0 2 Linha 2: -2 -2 -1 0 -1 -1 -2 0 0 -1 Linha 3: -1 2 0 1 0 0 -1 -1 1 -2 Linha 4: 0 -1 -1 1 2 0 0 2 -2 2 Linha 5: 1 -1 0 1 1 2 -1 -1 1 1
10 10	Linha 1: -1 0 2 -2 2 2 1 1 0 2 Linha 2: -2 -2 -1 0 -1 -1 -2 0 0 -1 Linha 3: -1 2 0 1 0 0 -1 -1 1 -2 Linha 4: 0 -1 -1 1 2 0 0 2 -2 2 Linha 5: 1 -1 0 1 1 2 -1 -1 1 1 Linha 6: 0 2 0 0 0 2 1 -1 2 1 Linha 7: -1 -2 -2 0 1 -1 -2 0 2 1 Linha 8: -1 -2 -2 2 -2 -2 -1 -1 1 1 Linha 9: 2 1 2 2 -1 -2 -1 -1 -1 1 Linha 10: 2 2 -1 1 0 1 1 0 2 -1

Contextualização: nos grupos II a IV, pretende-se fazer um simulador de uma variante do clássico Jogo da Vida.

O Jogo da Vida desenvolve-se num mundo bidimensional de L por C casas. Em cada casa pode ou não existir vida, e no caso de existir vida esta pode ter mais ou menos saúde, e pode ou não ter Covid. O valor associado a cada casa, varia de -2 a 2, no caso de ser nulo, a casa não tem vida, caso seja positivo tem vida **semCovid**, caso seja negativo tem vida **comCovid**. Quanto mais afastado do 0 maior é a saúde.

O mundo parte de uma posição inicial, gerada neste grupo II, e de acordo com um conjunto de regras do grupo III, cada casa irá ter o seu valor alterado na geração seguinte, sendo solicitado no grupo IV que se simule um determinado número de gerações.

Grupo III (3 valores)

Neste grupo apresentam-se as regras para determinar o valor de uma casa, na geração seguinte, que dependem do estado das casas vizinhas. As regras devem ser aplicadas por ordem:

1. Se a casa não tem vida (= 0):
    1. E existe vida **semCovid** (> 0) em exatamente 2 casas na vizinhança, a casa ganha vida ficando com o valor 1
    2. Caso contrário a casa permanece com o valor 0
2. Se a casa tem vida **semCovid** (> 0):
    1. E existe 1 ou menos casas na vizinhança com vida **semCovid** (> 0), trata-se de uma vida de solidão e o seu valor é reduzido em uma unidade.
    2. E existem 5 ou mais casas com vida **semCovid** (> 0) na vizinhança, trata-se de uma vida saturada e contrai Covid, ficando o valor simétrico do atual
    3. E existe pelo menos 3 casas com vida **comCovid** (< 0) na vizinhança, a vida é contaminada ficando o valor simétrico do atual
    4. Caso contrário, trata-se de uma vida saudável e aumenta a saúde, o seu valor é incrementado (até ao máximo de 2)
3. Se a casa tem vida **comCovid** (< 0):
    1. Se existirem no máximo 2 casas com vida **comCovid** (< 0) na vizinhança, a vida é curada e o valor fica simétrico do atual
    2. Caso contrário, trata-se de uma vida em doença e a saúde diminui, o seu valor absoluto é reduzido em uma unidade (de -2 passa a -1, e de -1 passa a 0).

Para teste destas regras, neste grupo recebe-se não apenas os valores de L e C, como também as posições de uma casa, linha, coluna, com valores entre 0 e L-1 e entre 0 e C-1 respetivamente. O programa deve retornar o valor da casa na geração seguinte. O programa em baixo implementa já a leitura dos dados e impressão do resultado, faltando a função *Saude*.

```
void main() {
    int i, j, L, C, linha, coluna;
    int mundo[MAX_SIZE][MAX_SIZE];
    scanf("%d %d %d %d", &L, &C, &linha, &coluna);
    GerarMundo(mundo, L, C);
    printf("%d", Saude(mundo, L, C, linha, coluna));
}
```

Nota: pode utilizar uma função auxiliar, para calculo do número de casas vizinhas que estão vivas **comCovid**, e estão vivas **semCovid**. Estas duas variáveis são suficientes para implementar todas as regras especificadas.

Nota2: caso não saiba como calcular as variáveis **comCovid** e **semCovid**, assuma a existência de uma função que as calcula, para poder realizar o grupo para 50% da cotação.

As regras vão ser exemplificadas no primeiro caso de teste do grupo II, em que se assinalou as casas testadas:

```
-1  0  2 -2  2
 2  1  1  0  2
-2 -2 -1  0 -1
-1 -2  0  0 -1
-1  2  0  1  0
 0 -1 -1  1 -2
 0 -1 -1  1  2
 0  0  2 -2  2
 1 -1  0  1  1
 2 -1 -1  1  1
```

Caso 10 5 3 2, aplica-se a regra 1.1, sendo o resultado 1. Notar que as casas vizinhas neste caso são os seguintes:

```
-2 -1  0
-2     0
 2  0  1
```

Há exatamente dois vizinhos com valor positivo, pelo que aplica-se a regra.

Caso 10 5 3 3, aplica-se a regra 1.2, sendo o resultado 0.

Caso 10 5 9 0, aplica-se a regra 2.1, sendo o resultado 1.

Caso 10 5 8 3, aplica-se a regra 2.2, sendo o resultado -1.

Caso 10 5 6 3, aplica-se a regra 2.3, sendo o resultado -1.

Caso 10 5 8 4, aplica-se a regra 2.4, sendo o resultado 2.

Caso 10 5 5 4, aplica-se a regra 3.1, sendo o resultado 2.

Caso 10 5 6 1, aplica-se a regra 3.2, sendo o resultado 0.

Grupo IV (3 valores)

Neste grupo pretende-se uma simulação de N gerações, aplicadas após uma primeira geração do mundo de acordo com o grupo II. Cada mundo é construído a partir do mundo anterior, por aplicação da função do grupo III a todas as casas, tendo sido desenvolvida a função *NovoMundo* para esse efeito, fornecida em baixo.

```
void NovoMundo(
    int novo[MAX_SIZE][MAX_SIZE],
    int antigo[MAX_SIZE][MAX_SIZE],
    int L, int C)
{
    int i, j;
    for (i = 0; i<L; i++)
        for (j = 0; j<C; j++)
            novo[i][j] = Saude(antigo, L, C, i, j);
}
```

Implemente a função main que receba o valor de L, C e N, do utilizador e mostre o mundo final após N gerações.

No caso de não ter realizado as alíneas anteriores, pode à mesma realizar esta alínea, assumindo que as funções anteriores estão disponíveis.

Casos de teste (no VPL estão disponíveis mais casos de teste):

Entrada	Saída
10 5 0	Linha 1: -1 0 2 -2 2 Linha 2: 2 1 1 0 2 Linha 3: -2 -2 -1 0 -1 Linha 4: -1 -2 0 0 -1 Linha 5: -1 2 0 1 0 Linha 6: 0 -1 -1 1 -2 Linha 7: 0 -1 -1 1 2 Linha 8: 0 0 2 -2 2 Linha 9: 1 -1 0 1 1 Linha 10: 2 -1 -1 1 1
10 5 1	Linha 1: 1 0 2 2 1 Linha 2: 1 -1 -1 0 1 Linha 3: -1 -1 1 1 1 Linha 4: 0 -1 1 0 1 Linha 5: 0 1 0 0 1 Linha 6: 0 0 0 -1 2 Linha 7: 0 0 0 -1 2 Linha 8: 0 1 -2 2 2 Linha 9: 0 1 0 -1 2 Linha 10: 1 1 1 2 2

Entrada	Saída
10 5 2	Linha 1: 0 0 1 2 2 Linha 2: 0 0 1 0 2 Linha 3: 0 0 -1 -1 2 Linha 4: 0 1 2 0 2 Linha 5: 0 0 1 0 2 Linha 6: 0 0 0 1 2 Linha 7: 0 0 1 1 2 Linha 8: 1 0 2 -2 2 Linha 9: 0 2 0 1 2 Linha 10: 2 2 2 2 2
10 5 20	Linha 1: 1 2 1 2 2 Linha 2: 2 2 2 0 2 Linha 3: 2 0 2 -1 2 Linha 4: 2 -2 2 0 2 Linha 5: 2 0 2 0 2 Linha 6: 2 -2 2 0 2 Linha 7: 2 0 2 0 2 Linha 8: 2 2 1 -1 1 Linha 9: 2 2 0 1 2 Linha 10: 2 2 2 2 1

No primeiro caso de teste, N=0, o mundo é apenas gerado, podendo-se ver que é igual ao primeiro caso de teste do grupo II. O segundo caso de teste com N=1, existiu uma só geração, pelo que permite confirmar os casos do grupo III.

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