INTRODUÇÃO À PROGRAMAÇÃO | 21173

Data e hora de realização

24 de fevereiro de 2021, às 10h 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 HackerRank durante o semestre, pode realizar os grupos II, III e IV no HackerRank, através do link: https://hr.gs/UAb21173Normal2021
  Independentemente de utilizar ou não o HackerRank, deve colocar o código e resultados na folha de resolução.

Enunciado

Grupo I (3 valores)

O programa seguinte pretende identificar os divisores, para um dado número introduzido pelo utilizador. 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.

```
void Divisores(int N)
{
    int i;
    for(i=2; i<N/2; i++)
        if(N%i != 0)
            printf("%d "; i);
}

// dado um número inteiro N, colocar os divisores,
// testando a divisão pelos números de 2 a N/2

int main(int argc; char **argv)
{
    if(argv!=2)
        return;
    printf("Divisores: ", N);
    Divisores(atof(argv[1]));
}
```

Grupo II (3 valores)

Pretende-se validar a entrada de dados, de modo a saber se o utilizador introduziu dois dígitos correspondendo a uma peça de dominó válida. As peças de dominó têm dois lados, com número de pintas entre 0 e 6. Pretende-se validar entradas com dois dígitos que estejam entre 0 e 6.

Programa:

```
int main()
{
    char str[BUFFER];

    scanf("%s", str);
    if (PecaDomino(str))
        printf("Peca valida.");
    else
        printf("Peca invalida.");
}
```

Implemente apenas a função PecaDomino utilizada no código. Tenha em atenção aos casos de teste seguintes:

Entrada	Saída
25	Peca valida.
60	Peca valida.
70	Peca invalida.
4	Peca invalida.
11	Peca valida.
HD	Peca invalida.
z	Peca invalida.
56	Peca valida.
dfgdfsg	Peca invalida.
05	Peca valida.

Grupo III (3 valores)

Dada uma sequência de peças bem emparelhadas (pode ser uma só peça), e uma peça a adicionar, junte a peça nas restantes de forma a que fique bem emparelhada, ou retornar impossibilidade.

Programa:

```
int main()
{
    char peca[BUFFER], sequencia[BUFFER];

    scanf("%s", sequencia);
    scanf("%s", peca);

    if (Juntar(sequencia, peca))
        printf("%s", sequencia);
    else
        printf("Peca %s nao pode ser adicionada a %s.",
            peca, sequencia);
}
```

Implemente o procedimento Juntar de acordo com o descrito. Tenha em atenção o seguinte conjunto de casos de teste:

Entrada	Saída
166335 21	21166335
4442233 53	444223335
24 55	Peca 55 nao pode ser adicionada a 24.
56 61	5661
544660022224 52	25544660022224

No primeiro caso a peça 21 pode-se juntar à esquerda porque termina com o valor 1, igual ao início da sequência. No segundo caso, a peça 53 pode ser junta à direita. A sequência termina com 3 e tem de se inverter a peça para que esta possa ser emparelhada.

Grupo IV (3 valores)

Dado uma sequência de peças não emparelhadas na entrada, faça um programa que coloque a primeira peça na mesa, e emparelhe todas as restantes peças por ordem, parando na primeira peça que não dá para emparelhar. No final deverá mostrar as peças na mesa, a última peça não emparelhada, e as restantes peças na sequência.

Considere os seguintes casos de teste:

Entrada	Saída
53113163	Mesa: 53 Peca: 11 Sequencia: 3163
313440226100	Mesa: 044331 Peca: 22 Sequencia: 6100
36613511	Mesa: 53366111 Peca: Sequencia:
010511251300	Mesa: 2550011113 Peca: 00 Sequencia:
25	Mesa: 25 Peca: Sequencia:

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