Exame | Instruções para a realização de exame

INTRODUÇÃO À PROGRAMAÇÃO | 21173

Data de realização

15 de julho de 2021

Duração da prova

120m + 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.
- 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.

- 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/UAb21173Recurso2021
  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 limpar uma string, substituindo qualquer caracter não imprimível por espaços, bem como outros caracteres brancos por espaços. 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.

```
/* substituir caracteres não imprimíveis por espaços, bem como
   outros caracteres brancos por espaços */
void LimparTexto(char str)
{
        int i;
        for(i=1; str[i]!=0; i++)
                if(isprint(str[i]) ||
        strchr("\t\n\r", str[i])==NULL)
                        str[i]=' ';
}

int main(int argc, char argv)
{
        char str[0];
        strcpy(str, argv[1]);
        LimparTexto(argv[1]);
        printf("\nEntrada: %s\nTexto Limpo: %s\n",
                argv[1], str);
}
```

Grupo II (3 valores)

Pretende-se validar os dados de entrada, de modo a saber se o utilizador introduziu uma carta válida. Cada carta tem um de 4 naipes, representados pelos caracteres "POCE" e um dos 10 números representados pelos caracteres "23567DVRA".

Implemente a função ***VerificaCarta*** de modo a que o programa funcione corretamente.

Programa:

```
int main()
{
    char str[BUFFER];
    scanf("%s", str);
    if (VerificaCarta(str))
        printf("Carta valida.");
    else
        printf("Carta invalida.");
}
```

Considere os seguintes casos de teste:

Entrada	Saída
2P	Carta valida.
P2	Carta invalida.
4O	Carta valida.
RE	Carta valida.
AA	Carta invalida.
8P	Carta invalida.
VC	Carta valida.
DO	Carta valida.
DODO	Carta invalida.
3	Carta invalida.

Grupo III (3 valores)

Considere que o utilizador introduz um conjunto de cartas válidas todas seguidas. As cartas são recebidas por qualquer ordem. No entanto, pretende-se que se mostrem as cartas agrupadas por naipe, devendo o naipe aparecer uma só vez, mantendo a ordem original dos números. Os naipes devem ser mostrados pela ordem POCE, de acordo com os casos de teste. Os números podem ser um de 234567DVRA.

Implemente o procedimento ***MostrarCartas*** para que o programa tenha o efeito pretendido.

Programa:

```
int main()
{
    char str[BUFFER];
    scanf("%s", str);
    MostrarCartas(str);
}
```

Considere o seguinte conjunto de casos de teste:

Entrada	**Saída**
DP2C	[D]P []O [2]C []E
4P3ODEVO2E	[4]P [3V]O []C [D2]E
8C6P2P9P6ODC	[629]P [6]O [8D]C []E
RPAP7EAC8PVC7O6EDP	[RA8D]P [7]O [AV]C [76]E
6EVCRP7EDP8C3O6P6O2EDCDE8P4PVOAPAC9P2C7O2P	[RD684A92]P [36V7]O [V8DA2]C [672D]E

Grupo IV (3 valores)

Faça um programa que receba um conjunto de cartas tal como no grupo anterior, e distribua as cartas por quatro jogadores, de forma alternada pelos jogadores. Ou seja, a primeira carta deve ir para o primeiro jogador, a segunda carta para o segundo jogador e assim sucessivamente, sendo a quinta carta dada ao primeiro jogador. No final, deve mostrar as cartas de cada jogador, agrupadas por naipe como no grupo III.

Considere os seguintes casos de teste:

Entrada	**Saída**
4P3ODEVO	Jogador 1: [4]P []O []C []E Jogador 2: []P [3]O []C []E Jogador 3: []P []O []C [D]E Jogador 4: []P [V]O []C []E
8C6P2P9P6ODCDP2C	Jogador 1: []P [6]O [8]C []E Jogador 2: [6]P []O [D]C []E Jogador 3: [2D]P []O []C []E Jogador 4: [9]P []O [2]C []E
RPAP7EAC8PVC7O6EDPDEVO2E	Jogador 1: [R8D]P []O []C []E Jogador 2: [A]P []O [V]C [D]E Jogador 3: []P [7V]O []C [7]E Jogador 4: []P []O [A]C [62]E
6EVCRP7EDP8C3O6P6O2EDCDE8P4PVOAPAC9P2C7O2P	Jogador 1: [D82]P [6]O [A]C [6]E Jogador 2: [49]P []O [V8]C [2]E Jogador 3: [R]P [3V]O [D2]C []E Jogador 4: [6A]P [7]O []C [7D]E
7E4P6ODO7C7OAO2O4EVODP3OVEAE6EDE4C6P5E6CRO 5P3PREVC4O3CVP2PAPDC3E2E5CRP7P2CAC	Jogador 1: [2]P [R]O [74V2]C [74V2]E Jogador 2: [465A]P [7V4]O [5A]C [A]E Jogador 3: [D3R]P [6A]O [3D]C [65]E Jogador 4: [V7]P [D23]O [6]C [DR3]E

Grupo V (8 valores)

Suponha que tem de desenvolver um programa mapeamento das cadeias de transmissão do vírus Covid-19. Para cada pessoa que tenha teste positivo, é necessário registar sobre os doentes anteriores, com quais esteve em contacto. No caso de não ter estado em contacto com nenhum doente anterior, uma nova cadeia de transmissão tem de ser criada, no caso de ter estado em contacto com um doente já registado, o mesmo deve ser associado à cadeia e ao doente concreto em que esteve em contacto. Caso tenha estado em contacto com mais que um doente de cadeias distintas, ambas as cadeias de transmissão devem ser fundidas.

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 uma função que adicione um novo doente, e outra que adicione um contacto com um doente existente, de acordo com o enunciado (deve juntar cadeias de transmissão se necessário).

d) Faça um relatório que indique o número de cadeias com 1 doente, número de cadeias com 2 doentes, e assim sucessivamente, até à cadeia com mais doentes.

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