E-fólio Global | Instruções para a realização do E-fólio

Introdução à Programação | 21173

Data de Realização

Decorre em 24 de julho de 2020

Instruções

- O tempo de duração da prova de e-fólio Global é de 90 minutos com tolerância de 90 minutos.
- 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 dos enunciados 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, com a cotação de 3 valores cada.
- 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.

Trabalho a desenvolver

Grupo I (3 valores)

O programa seguinte pretende calcular os arranjos de N, R a R, sendo N e R dados 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 comportamento expectável.

```
int main()
{
    int i; n; r; arranjos;
    printf("Calculo dos arranjos de N, R a R:\nIndique N:");
    scanf("%d",&n);
    printf("Indique R:");
    scanf("%d",&r);

          /* verificação da validade dos argumentos introduzidos */
    if(n<r && r<1);
        printf("Erro: N tem de ser maior que R e este maior que 0.\n");
    else {
        /* inicialização da variável iteradora é a expressão
           indicada no enunciado */
        i=n-r+1;
        arranjos=0;
        while(i<=n)
            arranjos*=i;
            /* parcial */
            printf("  i=%d; arranjos=%d\n",arranjos,i);
            i++;

        printf("Resultado: %d\n",n);
    }
}
```

Grupo II (3 valores)

Implemente a função *Find*, chamada no programa em baixo. Esta função recebe duas strings, e procura pela primeira ocorrência da segunda string na primeira string, retornando o número de caracteres em que esta se encontra, relativo ao início da primeira string. No caso de não ocorrer, retorna -1. No programa é dada uma string com dois títulos de notícias, e procura-se pelas sub-strings "iPad" e "Huawei", tendo-se localizado ambas as substrings como se pode confirmar na execução exemplo

Programa:

```
int main()
{
    char *str = "Huawei Mate 30 Pro com super, super, super Slow Motion.\
 Próximo iPad Pro pode incluir câmara tripla. ";
    printf("\nTexto contem 'iPad' na posicao %d e 'Huawei' na posicao %d.",
        Find(str, "iPad"), Find(str,"Huawei"));
}
```

Execução de Exemplo:

```
C:\...>recurso1920g2
Texto contem 'iPad' na posicao 64 e 'Huawei' na posicao 0.
```

Grupo III (3 valores)

Implemente outra versão da função *Find*, mas desta vez aceitando os caracteres wilcards '?' e '*' com o significado de um qualquer caracter no primeiro caso, e zero ou mais caracteres no segundo caso. Note que pode ao encontrar o caracter '*' na procura, chamar recursivamente a função para processar o resto da string de procura. O valor de retorno deverá ser o mesmo que no grupo anterior, com o início da sequência localizada. A função tem agora mais dois argumentos no final, o início e fim da procura na string. Como resultado, na segunda chamada, com argumento a 25 no início da procura, a função nunca retornaria uma posição inferior, como também nunca retornaria uma procura cujo match termine para além do último argumento, no caso dos exemplos, 1000. O programa exemplo apresenta algumas procuras, incluindo uma que falha.

Programa:

```
int main()
{
    char *str = "Huawei Mate 30 Pro com super, super, super Slow Motion.
Próximo iPad Pro pode incluir câmara tripla. ";
    printf("\nMatch 'i??d' na posicao %d", Find(str, "i??d", 0, 1000));
    printf("\nMatch 'su?er*tri?la' na posicao %d", Find(str,
"super*tripla", 25, 1000));
    printf("\nMatch 'Hua?i*30' na posicao %d", Find(str, "Hua?i*30", 0,
1000));
}
```

Execuções de Exemplo:

```
C:\...>recurso1920g3
Match 'i??d' na posicao 64
Match 'su?er*tri?la' na posicao 30
Match 'Hua?i*30' na posicao -1
```

Grupo IV (3 valores)

Faça um programa que receba dos argumentos o nome de um ficheiro e uma procura, e retorne todas as linhas em que existe uma procura válida, indicando a linha, conforme a execução exemplo. Deverá efetuar as validações exemplificadas na execução de exemplo, do número de argumentos e ficheiro inexistente.

Execuções de Exemplo:

```
C:\...>recurso1920g4
Coloque o nome de um ficheiro seguido da procura a realizar.
C:\...>recurso1920g4 recurso n?m*s?g??d
Ficheiro recurso nao abre.
C:\...>recurso1920g4 recurso1920g4.c Coloque

35:           printf("Coloque o nome de um ficheiro seguido da
procura a realizar.");

Procura localizada na posicao 16.
C:\...>recurso1920g4 recurso1920g4.c n?m*s?g??d

35:           printf("Coloque o nome de um ficheiro seguido da
procura a realizar.");

Procura localizada na posicao 26.
```

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