INTRODUÇÃO À PROGRAMAÇÃO | 21173

Data e hora de realização

26 de fevereiro de 2024

Duração da prova

90m + 60m

Competências:

- Identificação dos conceitos da programação imperativa, de forma a utilizá-los em outras linguagens de programação;
- Produção de pequenos programas numa linguagem imperativa;
- Desenvolver algoritmos e estruturas de dados para pequenos problemas.

Normas a respeitar:

- Deve redigir o seu E-fólio Global na WISEflow. A prova não será de consulta pelo que tem de instalar o bloqueio de ecrã e utilizar o reconhecimento facial caso aceda à prova remotamente.
- Deve identificar claramente, e em bold, o número de cada questão que está a responder. As respostas devem ser ordenadas de ordem crescente. Sendo a identificação automática não deve colocar uma folha de rosto na resposta à prova pois esta será gerada automaticamente na WISEflow.
- Se fizer a prova remotamente deve ter um comportamento em tudo semelhante à realização da prova em contexto presencial num centro de exame.
- O(a) estudante em avaliação remota deve, durante a prova online realizada através da WISEflow, seguir as seguintes instruções:
    - Não se pode levantar durante a prova, incluindo ir à casa de banho;

- Deve procurar um lugar calmo, onde possa estar sozinho, com as costas viradas para uma parede;
- Deve ter realizado uma prova teste para familiarizar-se com o sistema;
- Deve garantir que tem acesso à Internet (através de Wi-fi ou de rede fixa);
- Deve testar o seu computador previamente (por exemplo, a webcam e o bloqueio de navegador quando estes são necessários);
- Deve desligar o telemóvel, ou outro qualquer dispositivo informático, com o qual possa aceder à Internet;
- Deve retirar todas as folhas, livros ou fotocópias de cima da mesa onde realizará a prova;
- Durante a prova, não pode conversar com pessoas independentemente do teor da conversa.

- Assim que estiver pronto(a) para submeter a prova deve selecionar a opção ir para entrega que está sinalizado a verde no canto superior direito da página.
- A cotação é indicada junto de cada pergunta.
- 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.

Votos de bom trabalho!

Enunciado

Grupo I (3 valores)

Pergunta 1 [0,5 valores]

heap

50% stack; malloc; alocação dinâmica de memória

Identifique o conceito de programação que é o local onde se pode alocar memória, podendo libertar quando se quiser.

Pergunta 2 [0,5 valores]

estrutura de dados

50% struct, declaração de variáveis

Identifique o conceito de programação que é essencial para assegurar que é possível o carregamento dos dados disponíveis, e existe suporte para as operações necessárias

Pergunta 3 [1 valor]

d g f e c a b h

Considere os seguintes elementos:

ftell

1. função para obter a posição num ficheiro

nome da variável numa expressão

2. necessário para utilizar uma variável

for

3. ?ciclo?(<inicialização>;<condição>;<atualização>)
   <instrução>;

continue

4. segue para a próxima iteração do ciclo

float

5. tipo para número real

expressão lógica e instrução

6. necessário para criar um ciclo

fclose

7. função para fechar um ficheiro

não acedem à estrutura de dados

8. num Tipo Abstrato de Dados as funções externas

Associe cada um dos seguintes conceitos / entidades / respostas:

a) expressão lógica e instrução
b) fclose
c) float
d) ftell
e) continue
f) for

g) nome da variável numa expressão

h) não acedem à estrutura de dados

Pergunta 4 [1 valor]

3 1 4 2

b d a c

Faça corresponder o erro de qualidade apropriado a cada bloco de código.

Erros de qualidade:

a) Não libertar memória após alocar

bloco de código 3

b) Variáveis com nomes quase iguais

bloco de código 1

c) Má utilização do switch ou cadeia if-else

bloco de código 4

d) Abrir um ficheiro e não chegar a fechar

bloco de código 2

Bloco de código 1:

```
int main()
{
  int segunda, terca, quarta, quinta, sexta, sabado, domingo;
  int total;

  /* introdução de valores */
  printf("Segunda: ");
  scanf("%d",&segunda);
  printf("Terca: ");
  scanf("%d",&terca);
  printf("Quarta: ");
  scanf("%d",&quarta);
  printf("Quinta: ");
  scanf("%d",&quinta);
  printf("Sexta: ");
  scanf("%d",&sexta);
  printf("Sabado: ");
  scanf("%d",&sabado);
  printf("Domingo: ");
  scanf("%d",&domingo);

  /* calculos */
  total=segunda+terca+quarta+quinta+sexta+sabado+domingo;
  printf("Soma: %d\n",total);
  printf("Media: %f\n",total/7.0);
}
```

Bloco de código 2:

```
int main()
{
  int ano, mes, dias;
  FILE *f, *f2;
  f=fopen("in.txt","rt");
  f2=fopen("out.txt","wt");
  if(f==NULL || f2==NULL)
    return 0;
  fscanf(f,"%d", &ano);
  fscanf(f,"%d", &mes);

  if(mes==2)
  {
    /* teste de ano bissexto */
    if(ano%400==0 || ano%4==0 && ano%100!=0)
      fprintf(f2,"29");
    else
      fprintf(f2,"28");
  } else if(mes==1 || mes==3 || mes==5 || mes==7 ||
    mes==8 || mes==10 || mes==12)
  {
    fprintf(f2,"31");
  } else
  {
    fprintf(f2,"30");
  }
}
```

Bloco de código 3:

```
char *Concatenar(char *str, char *str2)
{
  char *pt;
  /* duplicar str2 se str é nulo */
  if(str==NULL)
  {
    pt=(void *)malloc(strlen(str2)+1);
    if(pt!=NULL)
      strcpy(pt,str2);
  } else {
    pt=(void *)malloc(strlen(str)+strlen(str2)+1);
    if(pt!=NULL)
    {
      strcpy(pt,str);
      strcat(pt,str2);
    }
  }
  return pt;
}

int main()
{
  char *texto=NULL, str[MAXSTR];
  do {
    gets(str);
    texto=Concatenar(texto,str);
  } while(strlen(str)>0);
  printf("%s",texto);
}
```

Bloco de código 4:

```
int DiasDoMes(int ano, int mes)
{
  switch(mes) {
  case 2:
    if(Bissexto(ano))
      return 29
    else
      return 28;
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12:
    return 31;
  default:
    return 30;
  }
}
```

Grupo II (3 valores)

Considere neste grupo e nos grupos III e IV, o programa para jogar a uma versão do jogo 2048. Num tabuleiro de 4x4 inicialmente vazio, começa por aparecer o número 1 numa casa aleatória vazia. O jogador pode depois ter um de 4 movimentos: cima, esquerda, direita, baixo. Esta operação desloca todos os números para a direção escolhida. No entanto, quando dois números ficam juntos sendo iguais e na mesma direção, somam-se ficando na mesma casa. O jogo acaba quando não há mais movimentos, sendo o valor máximo no tabuleiro o resultado do jogo.

Uma execução de exemplo, com alguns movimentos:

```
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
|    |    |    |   1|
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
Jogada (C/B/E/D): C

+----+----+----+----+
|    |    |    |   1|
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
|    |   1|    |    |
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
Jogada (C/B/E/D): D

+----+----+----+----+
|    |    |    |   1|
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
|    |    |    |   1|
+----+----+----+----+
|    |   1|    |    |
+----+----+----+----+
Jogada (C/B/E/D): C

+----+----+----+----+
|    |   1|    |   2|
+----+----+----+----+
|    |   1|    |    |
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
Jogada (C/B/E/D): B
```

```
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
|   1|    |    |    |
+----+----+----+----+
|    |   2|    |   2|
+----+----+----+----+
Jogada (C/B/E/D): E

+----+----+----+----+
|    |    |    |    |
+----+----+----+----+
|   1|    |    |    |
+----+----+----+----+
|   1|    |    |    |
+----+----+----+----+
|   4|    |    |    |
+----+----+----+----+
Jogada (C/B/E/D):



… sempre a jogada E …

+----+----+----+----+
|   8|   4|   2|   1|
+----+----+----+----+
|   8|   4|   2|   1|
+----+----+----+----+
|   8|   4|   2|   1|
+----+----+----+----+
|   8|   4|   2|   1|
+----+----+----+----+
Jogada (C/B/E/D): e

Resultado: 8.
```

Podemos ver na primeira jogada C, que o número inicial, vai para cima. É também gerado outro número. Na segunda jogada D, o segundo número vai para a direita, aparecendo um terceiro. Tendo dois números iguais na mesma coluna e seguidos, jogamos agora C para que os números se juntem em cima. É o que acontece, ficando o número 2. O jogo prossegue, e após umas jogadas, é sempre utilizada a jogada E. Como resultado os números não se somaram de forma conveniente, e o jogo termina sem espaço para colocar um novo número. O resultado é 8, o maior número no tabuleiro.

O programa a baixo implementa o jogo, não existindo alguns dos procedimentos necessários.

```
int main()
{
    int tabuleiro[4][4];
    char jogada[256];

    srand(0);
    InicializaTab(tabuleiro);

    while (LivresTab(tabuleiro) > 0) {
        InsereNumeroTab(tabuleiro);
        MostraTab(tabuleiro);
        printf("\nJogada (C/B/E/D): ");
        scanf("%s", &jogada);
        JogarTab(tabuleiro, jogada[0]);
    }
    printf("\nResultado: %d.", MaiorNumeroTab(tabuleiro));
}
```

Implemente neste grupo o procedimento *InicializaTab()*, que recebe o tabuleiro e coloca todo o seu conteúdo com o valor 0.

Grupo III (3 valores)

Neste grupo esteve-se a trabalhar no procedimento *JogarTab()*, conforme a implementação em baixo:

```
void JogarTab(int tabuleiro[4][4], char jogada)
{
    int i, j;

    if (jogada == 'C' || jogada == 'c') {
        // percorrer tabuleiro de cima para baixo
        for (i = 0; i < 4; i++)
            for (j = 0; j < 4; j++)
                if (tabuleiro[i][j] > 0)
                    MoveNumero(tabuleiro, i, j, -1, 0);
    }
    else if (jogada == 'B' || jogada == 'b') {
        // percorrer tabuleiro de baixo para cima
        for (i = 3; i >= 0; i--)
            for (j = 0; j < 4; j++)
                if (tabuleiro[i][j] > 0)
                    MoveNumero(tabuleiro, i, j, 1, 0);
    }
    else if (jogada == 'E' || jogada == 'e') {
        // percorrer tabuleiro da esquerda para a direita
        for (j = 0; j < 4; j++)
            for (i = 0; i < 4; i++)
                if (tabuleiro[i][j] > 0)
                    MoveNumero(tabuleiro, i, j, 0, -1);
    }
    else if (jogada == 'D' || jogada == 'd') {
        // percorrer tabuleiro da direita para a esquerda
        for (j = 3; j >= 0; j--)
            for (i = 0; i < 4; i++)
                if (tabuleiro[i][j] > 0)
                    MoveNumero(tabuleiro, i, j, 0, 1);
    }
}
```

No entanto falta ainda o procedimento *MoveNumero()*. Este procedimento deve deslocar o número no tabuleiro em (i,j), na direção indicada nos outros dois argumentos, o primeiro para o i (linha) e o segundo para o j (coluna). Sempre que a posição vizinha de (i,j), na direção fornecida, estiver vazia, o número deve deslocar-se para essa posição. Não deve parar de executar esta operação, excepto quando chega à borda, ou se encontrar um número maior que 0. Nesse caso, se o número for igual, junta-se a esse número. Caso contrário deve parar deixando o número nessa posição. Exemplos destes movimentos podem-se ver na execução de exemplo.

Grupo IV (3 valores)

Pretendemos agora terminar o programa, sendo que faltam ainda 4 funções: *LivresTab()*, *InsereNumeroTab()*, *MostraTab()*, *MaiorNumeroTab()*. **Escolha 3 destas funções** para implementar neste grupo. Cada função vale 1 valor. A função *LivresTab()*, retorna o número de casas livres, que é igual ao número de posições no tabuleiro com o valor 0. O procedimento *InsereNumeroTab()*, gera uma posição aleatória, e se estiver vazia, coloca a posição a 1, caso contrário continua a gerar posições aleatórias até que encontre uma vazia. O procedimento *MostraTab()*, mostra o tabuleiro no ecrã, de acordo com a execução exemplo. A função *MaiorNumeroTab()*, retorna o maior número existente no tabuleiro.

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