Programação II (CCINF1002) 2024/2025

Exame Modelo Duração: 2h

Nº Mecanográfico: | | | | | | | | | |

Nome Completo: _____

Grupo 1 (50%) – Fundamentos de C

1.1. Qual será o resultado da seguinte expressão? `7 % 3` [ ]

1.2. Qual será o resultado da seguinte expressão? `5 + 6 * 3 / 4` [ ]

1.3. Classifica cada um dos seguintes 4 nomes: **sim** se for um identificado válido para uma variável e **não** caso contrário.

`break123` [ ] `123break` [ ] `break` [ ] `_123break` [ ]

1.4. Completa o espaço em branco de modo a que o código seguinte leia um inteiro e coloque-o na variável `x`:

```
int x;
scanf( [          ] );
```

1.5. Completa o espaço em branco de modo a que o código seguinte escreva exatamente `1.50`:

```
float f = 1.5;
printf( [          ] );
```

1.6. O que escreve o seguinte pedaço de código? [ ]

```
int i = 2;
int j = 4;
int k = ++i * j++;
printf("%d %d %d\n", i, j, k);
```

1.7. O que escreve o seguinte pedaço de código? [ ]

```
int a = 1;
int b = 0;
int c = 1;
if ( (a || b) && c ) printf("true");
else printf("false");
```

1.8. Quantas vezes irá o código seguinte escrever `"Hello"`? ☐

```
int n = 1;
while (n < 8) {
   printf("Hello");
   n *= 2;
}
```

1.9. Qual será o valor da variável `count` depois de executado o seguinte pedaço de código? ☐

```
int n = 5, count=0;
for (int i=0; i<n; i++)
   for (int j=n; j>=0; j--)
      count++;
```

1.10. Qual será o valor da variável `count` depois de executado o seguinte pedaço de código? ☐

```
int n=10, count = 0;
for (int i=0; i<20; i++) {
   if (i>3 && i<7) continue;
   if (i>10) break;
   count++;
}
```

1.11. Assume que a função `add` é definida da maneira seguinte:

```
void add(int x, int k) {
   x += k;
}
```

O que escreve o seguinte pedaço de código? ☐

```
int x = 1, k = 2;
add(x, k);
printf("%d", x);
```

1.12. Qual é o valor devolvido por uma chamada a `foo(5)`? ☐

```
int foo(int x) {
   if (x==0) return 0;
   else return x + foo(x-1);
}
```

1.13. Qual é o valor devolvido por uma chamada a `secret(30, 12)`? ☐

```
int secret(int a, int b) {
   int tmp;
   while (b != 0) {
      tmp = a % b;
      a = b;
      b = tmp;
   }
   return a;
}
```

1.14. O que escreve o seguinte pedaço de código? ☐

```
int a[] = {2,4,6,8};
printf("%d", a[2]);
```

1.15. Qual será o conteúdo do array `a[]` depois de executado o seguinte código? ☐ ☐ ☐ ☐ ☐

```
int a[] = {5,4,3,2,1};
for (int i=1; i<5; i++) a[i] = a[i-1] + 1;
```

1.16. Completa os espaços em brancos de modo a que o código seguinte escreva `4`:

```
int m[3][3] = { {1,2,3}, {4,5,6}, {7,8,9} };
printf("%d", m[   ][   ];
```

1.17. O que escreve o seguinte pedaço de código? ☐

```
char c1 = 'a';
char c2 = c1 + 3;
printf("%c", c2);
```

1.18. O que escreve o seguinte pedaço de código? ☐

```
char str[] = "hello123";
int i = 0;
while (!isdigit(str[i])) i++;
printf("%d", i);
```

1.19. Completa o espaço em branco de modo a que o código seguinte devolva o último caracter da string `str`: *(podes usar as funções de `string.h`)*

```
char lastChar(char str[]) {
    return                          ;
}
```

1.20. Completa o espaço em branco de modo a que o código seguinte devolve um valor *verdadeiro* se as strings *a* e *b* são iguais (isto é, têm o mesmo conteúdo) ou um valor *falso* caso contrário: *(podes usar as funções de `string.h`)*

```
int equal(char a[], char b[]) {
    return                          ;
}
```

Grupo 2 (50%) – Criando Código

2.1. (11%) Escreve um **programa em C completo** (com *includes* e função *main*) que lê dois inteiros *a* e *b* e escreve todos os números pares entre *a* e *b* (inclusive) separados por um espaço e terminados com uma mudança de linha.

Exemplo: Se o utilizador der como input `2` e `11`, o teu código deve escrever: `2 4 6 8 10`

2.2. (11%) Escreve uma **funcão** `int count_digits(int n, int d)` que devolve o número de vezes que o dígito d aparece no inteiro n. É garantido que d será sempre um inteiro entre 0 e 9.

Exemplo 1: Chamar `count_digits(122321, 2)` deve devolver 3 (o dígito 2 aparece três vezes)

Exemplo 2: Chamar `count_digits(4242, 5)` deve devolver 0 (o dígito 5 nunca aparece)

2.3. (11%) Escreve uma **funcão** `void swap(int a[], int n)` que recebe um array a com n inteiros e troca todos os pares em posições consecutiva, mudando o próprio array: o 1º elemento deve trocar com o 2º, o 3º com o 4º, etc. Se n for ímpar, o último elemento deve permanecer na mesma posição.

Exemplo: Chamar com $a = \{2, 4, 6, 8, 10\}$ e $n = 5$ deve modificar o conteúdo do array para $\{4, 2, 8, 6, 10\}$.

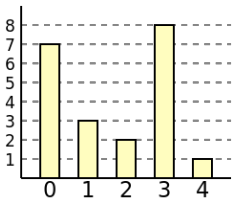
2.4. (11%) Escreve uma **função** `int palindrome(char str[])` que recebe uma string *str* de letras e devolve 1 se for um palíndromo e 0 caso contrário. Uma string é um palíndromo se for lida da mesma forma da frente para trás ou de trás para a frente. Uma letra minúscula deve ser considerada a mesma que a sua versão maiúscula. *(receberás pontos parciais se o teu código não conseguir lidar com maiúsculas/minúsculas mas funcionar tirando isso)*

Exemplo 1: Chamar `palindrome("anna")` deve devolver 1

Exemplo 2: Chamar `palindrome("Radar")` deve devolver 1

Exemplo 3: Chamar `palindrome("WoWo")` deve devolver 0

2.5. (6%) Considera um array h com n inteiros distintos representando a altura de torres. Por exemplo, o array $h = \{7, 3, 2, 8, 1\}$ representa a figura da direita. Chamemos $score(i)$ à distância da torre mais próxima que tem altura maior que $h[i]$, ou seja:



$$score(i) = min\{|i - j| : \text{para todo o } j \text{ tal que } h[j] > h[i]\}$$

O valor de $score(i)$ para a torre mais alta é 0. Para a figura da direita temos que:

- $score(0) = 3$ (a torre mais próxima que é mais alta é a torre da posição 3 e $|0 - 3| = 3$)
- $score(1) = 1$ (a torre mais próxima que é mais alta é a torre da posição 0 e $|1 - 0| = 1$)
- $score(2) = 1$ (as torres mais próximas que são mais altas estão nas posições 1 e 3)
- $score(3) = 0$ (esta é a torre mais alta)
- $score(4) = 1$ (a torre mais próxima que é mais alta é a torre da posição 3 e $|4 - 3| = 1$)

Escreve uma **funcão** `int total_score(int h[], int n)` que calcula o score total do array, isto é, a soma de todos os scores. Formalmente, deve devolver $\sum_{i=0}^{n-1} score(i)$.

Para obter pontuação total, a tua solução tem de ser eficiente no que toca ao tempo de computação.
(capaz de resolver um caso com $n = 200\,000$ em menos do que 1 segundo num portátil normal)

Exemplo: Chamar com $h = \{7, 3, 2, 8, 1\}$ e $n = 5$ deve devolver 6
(este é o exemplo explicado no enunciado da pergunta)

(resposta à pergunta 2.5 – função `total_score`)

Espero que tenhas gostado desta unidade curricular
e que tenhas aprendido muitas coisas novas!