Universidade da Beira Interior - Dep. Informática

Exame E. Normal | Programação – LEI | 2025/2026

Possível resolução

1.

(a) (A >= -50) && (A < 50)

(b) (B % 2 == 1) && (B < 10 || B >= 300)

(c) B = A / 1000 % 10 + A % 100 / 10; ou B = A % 10000 / 10 + A / 10 % 10;

(d) P = 5 * (N % 2) – 1;

(e)

- ordem de cálculo: **1, 3, 2, 5, 4**
- valor: **V (verdadeiro/true)**

(**9 <= 9** || 16 < 8) && 3 > 2	aplicar operador 1
(V || **16 < 8**) && 3 > 2	aplicar operador 3
(**V || F**) && 3 > 2	aplicar operador 2
V && **3 > 2**	aplicar operador 5
V && V	aplicar operador 4
V	

(f)

- ordem de cálculo: **5, 4, 3, 1, 2**
- valor: **14.0**

35 / 16 + (3 * (1.0 + **9 % 6**))	aplicar operador 5
35 / 16 + (3 * (**1.0 + 3**))	aplicar operador 4
35 / 16 + (**3 * 4.0**)	aplicar operador 3
35 / 16 + 12.0	aplicar operador 1
2 + 12.0	aplicar operador 2
14.0	

2.

```
#include <stdio.h>
int main()
{
   int  N, k, R1, R2;
   float  X, A;
   do{
      printf("Insira um inteiros > 1 ");
      scanf("%d", &N);
   }while(N <= 1);
   printf("Insira um real entre 50 e 100 ");
   scanf("%f", &X);
   while(X < 50 || X > 100)
   {
      printf("Insira um real entre 50 e 100 ");
      scanf("%f", &X);
   }
   R1 = 0;
   R2 = 0;
   k = 1;
   do{
      printf("Insira um numero real ");
      scanf("%f", &A);
      if(A >= 0 && A <= X)
         R1 = R1 + 1;
      else
         R2 = R2 + 1;
      k = k + 1;
   }while(k <= N);
   printf("R1 = %d, R2 = %d, X = %f\n", R1, R2, X);
}
```

3.

```
int  soma (int M, int N)
{
   int  S, min, max, k;
   S = 0;
   if(M > N)
   {
      for(k = N; k <= M; k++)
         S = S + k;
   }
   else
   {
      for(k = M; k <= N; k++)
         S = S + k;
   }
   return S;
}
```

4.

```
void  numerosPositivos (float X[], int N, int *numPos, float *media)
{
   float soma;
   int k;
   *numPos = 0;
   soma = 0;
   for(k = 0; k <= N-1; k++)
   {
      if(X[k] >= 0)
      {
         *numPos = *numPos + 1;
         soma = soma + X[k];
      }
   }
   if(*numPos > 0)
      *media = soma / *numPos;
   else
      *media = 0;
}
```

5.

a) *(V + 3) = *(110164 + 3 * sizeof(int)) = *(110164 + 12) = *(110176) = **1500**

b) V – 4 = 110164 – 4 * sizeof(int) = 110164 – 16 = **110148**

c) &V[4] = **110180**

d) *(*X – 1) = *(500 – 1) = **ERRO**

e) **W – 2 = 500 – 2 = **498**

f) W + 4 = 100500 + 4 * sizeof(int*) = 100500 + 32 = **100532**

g) *(*W + 3) = *(110168 + 3 * sizeof(int)) = *(110180) = **50**

h) *W – 2 = 110168 – 2 * sizeof(int) = 110168 – 8 = **110160**

i) ***(V – 1)** = *(110164 – 1 * sizeof(int)) = *(110160) = **100**

j) W[0] = **110168**

6.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int  *X, N, num, k;
   FILE *f, *gPos, *gNeg;
   f = fopen("Inteiros.txt", "r");
   N = 0;
   X = (int*) malloc(N * sizeof(int));
   while(!feof(f))
   {
      fscanf(f, "%d", &num);
      if(num != 0){
         N = N + 1;
         X = (int*) realloc(N * sizeof(int));
         X[N-1] = num;
      }
   }
   fclose(f);
   gPos = fopen("Positivos.txt", "w");
   gNeg = fopen("Negativos.txt", "w");
   for (k = 0; k <= N-1; k++)
      if(A[k] > 0)
         fprintf(gPos, "%d\n", A[k]);
      else
         fprintf(gNeg, "%d\n", A[k]);
   fclose(gPos);
   fclose(gNeg);
}
```

7.

```
int  indiceMaior (int A[], int N)
{
   int  ind;
   // caso base/terminal
   if(N == 1)
      return 0;
   // caso geral: n > 1
   ind = indiceMaior(A, N-1);
   if(A[N-1] > A[ind])
      return N-1;
   else
      return ind;
}
```

8.

```
int  removerRepetidos (int *X, int *N)
{
   int  i, j, existe, k;
   k = 0;
   while (k < *N)
   {
      // verificar se o valor de X[k] já existe nas posições anteriores a k
      existe = 0;
      i = 0;
      while (i < k  &&  existe == 0)
      {
         if (X[i] == X[k])
            existe = 1;
         i = i + 1;
      }
      if (existe == 1)
      {     // se existe, remover X[k] do array, atualizar o tamanho e manter o valor de k
         for (j = k; j < *N-1; j++)
            X[j] = X[j+1];
         *N = *N – 1;
      }
      else  // se não existe, avançar k uma posição no array
         k = k + 1;
   }
   X = (int*) realloc(X, (*N) * sizeof(int));   //  realocar memória para X com novo tamanho
}
```