Răspunsurile corecte cu explicații

1. b

O variabilă de tip structuă poate fi asignată unei alte variabile de tip structură (același structură), dar nu pot fi comparate folosind operatorul ==.

2. d

DeMorgen aplicat corect:

!((x<=−10)||(x>11)) && !((x<−7)||(x>−2))

((x>−10)&&(x<=11) && (x>=−7)&&(x<=−2))

Deci intervalul este [-7, -2]

3. b

În limbajul C/C++ un identificator (numele unei variabile, functii, clase/structuri etc.) există din momentul în care este scris. La compilare, un identificator este cautat in domeniul curent (domeniu este secvența de instrucțiuni delimitată de acolade ..., exceptie facând domeniul global) si dacă nu este găsit va fi căutat în domeniul parinte. În cazul de față există variabila (identificatorul) x global (declarată in domeniul global) și variabila x local (declarată în domeniul funției main). Instrucțiunea int x = x; are ca efect alocare de memorie pentru variabila x local urmată de inițializarea cu ea însăși. Din acest motiv valoarea afișată de program este nedefinită.

4. c

Acest exercițiu se rezolvă prin urmărirea valorilor variabilelor i și j la fiecare pas . Inițial, i = 0. In al doilea for se intră, pentru prima dată, cu i = 0, deci acest for se transformă in $for(j = 5; j > 3; j = -2)$. Deci, instrucțiunea din al doilea for se va executa o singură dată, pentru că, după primul pas, variabila j va lua valoarea -2. Același raționament și pentru i = 2 și pentru i = 4, deci, la final, s va lua valoarea 3.

5. a

Expresia x=3 are ca rezultat valoarea 3, deci va fi valoarea adevărat din punct de vedere logic. Ca urmare, instrucțiunea if (x=3) va fi executată pe ramura corespunzătoare valorii logice ADEVĂRAT pentru condiție.

6. a

Pentru rezolvarea problemei, trebuie verificate cele patru variante de cod pentru un exemplu dat. Pentru exemplul din subiect, prețul pe 16 decembrie trebuie să fie 90 de lei, corespunzător unui preț inițial de 100 de lei si unei perioade de 4 zile până la expirare. Se testează variantele de răspuns cu aceste date și se stabilește răspunsul corect.

7. b

În cazul declarării unui tablou bidimensional a doua dimensiune trebuie cunoscută.

8. b

Cele două *for*-uri parcurg elementele matricii aflate deasupra diagonalei principale și le interschimbă cu elementele aflate sub aceasta. Un element aflat pe poziția $[i][j]$ va fi interschimbat cu elementul aflat pe poziția $[j][i]$.

9. b

Pentru o litera S_i se parcurg literele de la S_0 la S_{i-1} și se verifică dacă $S_j > S_i$, unde $j = 0, \ldots, i-1$.

10. b

```
//algoritm fibonaci
     int a = 0, b = 0, c = 1;
     for (i = 0; i<n; i++)
          for (j = 0; j<m; j++)
          {
               M[i][j] = c; a = b; b = c; c = a + b;
          }

 for (i = 0; i<n; i++)
 {
     for (j = 0; j<m; j++)
          cout << M[i][j] << " ";
     cout << endl;
 }
 for (i = 0; i<m; i++)
 { suma += M[0][i];
suma += M[n - 1][i];
 }
 for (i = 1; i<n - 1; i++)
 {
     suma += M[i][0];
     suma += M[i][m - 1];
 }
 cout << "Suma elementelor de pe marginea matricii este : " << suma << endl;
```

11. d

Suma elementelor vectorului este 1024. Funcția calculează suma pentru fiecare $h = 1024...1$ ($h = 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2,$
11 ori.

```
int f(int v[], int n) {
     int h = n;
     int sum = 0;
     while (h > 0) {
          for (int i = 0; i < n; i++)
               sum = sum + v[i];
          h = h / 2;
     }
     return sum;
}


int main()
{
     int v[1024];
     for (int i = 0; i < 1024; i++)
          v[i] = 1;
     std::cout << f(v, 1024) << std::endl;
}
```

12. c

Pentru $i = 1$, bucla interioară este executată de n ori.
Pentru $i = 2$, bucla interioară este executată de aproximativ $n/2$ ori.
Pentru $i = 3$, bucla interioară este executată de aproximativ $n/3$ ori.

.....
Pentru $i = n$, bucla interioară este executată de aproximativ n/n ori.
Deci complexitatea timpului total al algoritmului de mai sus este $(n + n/2 + n/3 + ... + n/n)$. Care devine $n * (1/1 + 1/2 + 1/3 + ... + 1/n)$. Seria $(1/1 + 1/2 + 1/3 + ... + 1/n)$ este egală cu $\mathcal{O}(\log n)$. Deci complexitatea timp a codului de mai sus este $\mathcal{O}(n \cdot \log n)$.

13. c

c) V3 este gresită Corect era for (j = 0; j < n; j++) for (i = 0; i < n-1; i++)

14. a

V1: B≠ 0; V2:A Cod pentru pseudocod

```
int euclid(int a, int b)
{
    int c;
    while (b)
    {
        c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

15. b

Elevii sunt aşezaţi în ordinea crescătoare a înălţimilor şi schiurile sunt aşezate în faţa lor, în ordinea crescătoare a lungimilor. În faţa fiecărui elev se află o pereche de schiuri. Fiecare elev primeşte perechea de schiuri din faţa sa.

Observație. Problema se poate rezolva rapid folosind metoda greedy. Instructorul îi așează pe elevi în ordinea descrescătoare a raportului dintre înălțimea lor și lungimea schiurilor. Cei care cunosc metoda, vor rezolva problema mai rapid. Ceilalți vor încerca toate variantele posibile.

16. c

Backtracking. Se fixează pe ultima poziție o literă diferită de O și O pe penultima, ceea ce duce la 5! permutări pentru literele din fața lui O. Cum pentru ultima poziție sunt 6 variante, în total avem 5!*6 combinații posibile deci 6!, adică 720.

17. b

Într-un graf complet neorientat cu n noduri, orice permutare de noduri reprezintă un lanț hamiltonian. Prin urmare, răspunsul este $n!$.

18. a

Trebuie calculate lanțurile minime între fiecare pereche de două vârfuri (i, j), unde $i = 1, \ldots, 7$ și $j = (i + 1), \ldots, 8$. Lnțurile minime de lungime 3 sunt între vârfurile: 1-4, 1-8, 2-6, 3-4, 3-5.

19. a

Numărul maxim de muchii se obține cand cele 3 componente conexe au 8, 1, respectiv 1 noduri. Pentru un graf complet cu n noduri, numărul de muchii este $n * (n - 1)/2$. Cand prima componentă conexă este un graf complet, numărul de muchii ale acesteia este: $8 * 7/2 = 28$. Celelalte două coponente conexe, cu câte un nod fiecare, nu au muchii. Prin urmare, numărul maxim de muchii este 28.

20. b

Doar un circuit hamiltonian conţine toate nodurile unui graf şi nodurile nu se repetă.

21. c

inserarea in Heap

```
//fiu, parinte si aux sunt indecsi in V
N ← N + 1;
V[N] ← a;
fiu ← N;
parinte ← N/2;
cât timp parinte >= 1 execută
    dacă V[parinte] < V[fiu] atunci
        aux ← V[parinte];
        V[parinte] ← V[fiu];
        V[fiu] ← au;;
        fiu ← parinte;
        parinte ← parinte/2;
    altfel
        parinte ← 0;
```

22. a

Operații cu membrii unei structuri.

23. c

24. b

$f1(x) \cdot f2(x)$ calculează $x!$

25. b

Funcția calculează numărul de apeluri recursive până când valoarea lui n devine mai mica decât 10. Astfel, după apelul funcție(102304,a,b), n primește valoarea 102304 și apoi funcția va intra pe ramura else. Practic, la fiecare apel se va extrage doar partea intreagă a împărțirii la 10 a numărului n, pentru ca în momentul în care n rămâne doar cu o cifră acesta este asignată numărului a sau b după caz. După aceasta, în ordinea inversă apelului se execută if-ul final, separând pentru fiecare număr cifrele egale cu 0 (se vor adăuga la dreapta numărului a) și cele diferite de 0 (se vor adăuga la dreapta numărului b). Astfel, a=1 inițial, va primi cei 2 de 0 din n, iar la b=0 inițial se vor adăuga toate cifrele diferite de 0 din n. Deci la final a=100, iar b=1234.