# TUTORIAL DE REZOLVARE SUBIECT TEORETIC PC

ATENȚIE! LA EXAMEN VEȚI PRIMI O SINGURĂ FOAIE A4 PE CARE TREBUIE SĂ VĂ INCADRAȚI REZOLVAREA. NU AVEȚI VOIE CU ALTE FOI ȘI NICI NU MAI PRIMIȚI ALTE FOI ÎN CAZUL ÎN CARE NU MAI AVEȚI SPAȚIU!

1. Fie următorul program:

```
struct course
    {
        int cursNr;
        char *cursNume;
    };
```

```
#include <stdio.h>
#include "s1.h"
int main(void) {
    struct course c[] = { {202, "C#"},
                          {203, "ANSI C"},
                          {204, "C++"}
     };
     printf("%s\t", c[2].cursNume);
     printf("%% %d\n", (c+1)->cursNr);
     return 0;
}
```

Indicați, dacă programul este corect, ce se afișează pe monitor la execuția sa. Dacă există erori, indicați linia (sau liniile) în care apar erorile. Structura din stânga este declarată în fișierul header **s1.h.**

- c este un pointer care indică adresa de început a tabloului de structuri.
- c-> este echivalent cu (*c). sau cu c[0].
- Operatorul. este folosit pentru acces direct la membrii structurii
- operatorul-> este folosit pentru a dereferenția pointerul pentru a accesa zona de memorie indicată de acesta

Programul este corect și afișează:
C++(TAB)% 203(ENTER)

2. Care este rezultatul următoarei secvenţe de cod (codul este corect scris):

```
int p_test (int a, int * b)
{
     int d, *c;
     c = &a;
     *c = *b+5;
     *b = a+*c;
     d = *c**b;
     a = *c + 4;
     return d;
}
```

```
int main (void)
{
     int a=1, b=2, c=3, d=4;
     b = p_test (a, &c);
     printf ("%d %d %d %d", a, b, c, d);
     return 0;
}
```

În main sunt inițializate variabilele a,b,c,d cu următoarele valori
a=1, b=2, c=3, d=4
Următoarea linie de cod reprezintă apelul funcției p_test. Variabila a este transmisă prin valoare și având în vedere că nu se mai efectuează nici o operație asupra ei până la finalul mainului, ea va rămâne a=1. La fel variabila d=4.
Ne îndreptăm atenția către funcția p_test pentru a vedea ce valori vom obține pentru variabilele b și c. În primul rând ne uităm la semnificațiile parametrilor: b-ul din main va fi d-ul din funcție, iar c-ul din main b-ul din funcție.

În funcția p_test avem declarate două variabile locale (vizibile doar în corpul funcției, adică între acoladele funcției și cu durată de viață de la intrarea în funcție până la return) d(reține un număr întreg) și c(reține adresa unui întreg).
Variabilei c din funcția p_test i se atribuie adresa variabilei a. Pentru funcția p_test a este un parametru formal în care s-a copiat argumentul funcției (a-ul din main), deci modifcările efectuate asupra sa nu au efecte pentru argumentul funcției.
Conținutul locației de memorie indicate de c primește valoarea 3+5=8. 3-ul este preluat de la adresa indicată de b, adică adresa c-ului din main.
Conținutul de la adresa indicată de b se modifică în 8+8=16. Primul 8 provine din locația indicată de c, adică aceeași locație ca și a.
Variabila d primește valoarea 16*8=128
Următoarea instrucțiune nu ne mai interesează pentru că nu este implicată în modificarea lui b sau d.
Funcția returneaza d=128 și îl stochează în b-ul din main.
Referitor la b-ul din funcție acesta este declarat ca pointer pentru a permite transmiterea unui parametru prin referință. Transmiterea parametrilor prin referinţă constă în copierea adresei unui argument într-un parametru. În funcţia în care se face apelarea, este folosită adresa respectivă pentru accesarea argumentului folosit efectiv la apelare. În transmiterea prin adresă, modificările efectuate asupra unui parametru formal afectează argumentul cu care se face apelarea funcţiei (parametrul actual). Pentru a folosi transmiterea prin referinţă, argumentele funcţiei trebuie declarate ca pointeri. Variabila b din funcție indică spre locația de memorie a lui c din main și având în vedere că la apel s-a transmis adresa efectivă a lui c, valoarea lui c din main se modifică în 16.
Rezultatele afișate de secvența de cod de mai sus sunt:
1 128 16 4

3. Care este rezultatul următoarei secvenţe de cod. Justificare

```
#include <stdio.h>
int main(void){
    int x=011,i;
    for(i=0; i<x; i+=3){
         printf("Start ");
        continue;
         printf("End");
    }
    return 0;
}
```

Variabila x =011 este inițializată cu valoarea octală 011 = 1*8+1*1=9 în zecimal.
Iterativ se va testa i-ul de la 0 la 9 din 3 în 3 (=>3iterații)
Instrucțiunea continue ne indică faptul că următoarea instrucțiune efectuată va fi i<x.
Programul va afișa:
Start Start Start

4. Care este rezultatul rulării programului din coloana din stânga dacă funcția **f** este declarată în fișierul **s4.h** așa cum este specificat în coloana din dreapta? Justificare.

```
#include <stdio.h>
#include "s4.h"
int main(void){
     int y;
     int i;
     for(i=0; i<0x1a; i+=2) {
          y = f();
          printf("%d ", y);
     }
     return 0;
}
```

```
int f(void)
{
     int x;
     static int i = 1;
     x = i++;
     return x;
}
```

Numărul hexazecimal 0x1a este 16+10=26 în baza 10. Deci contorul i va merge de la 0 la 25 din 2 în 2. O variabilă declarată cu cuvantul cheie static în interiorul unei funcții își va păstra nemodificată valoarea între apelurile funcției. În urma instrucțiunii x=i++; la prima iterație, x va primi valoarea lui i, x=i=1, după care se executa post incrementarea i++, și i va deveni 2. Valoarea 2 se păstrează la următorul apel.

| i | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| y | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |

Programul va afișa:
1 2 3 4 5 6 7 8 9 10 11 12 13

5. Ce se afişează la execuţia următorului program? Justificare.

```
#include<stdio.h>
int main(void)
{
    enum days {MON=-1, TUE, WED=6, THU, FRI, SAT};
    printf("%d, %d, %d, %d, %d, %d\n", MON, TUE, WED, THU, FRI, SAT);
    return 0;
}
```

La declararea tipului enum se poate specifica valoarea unuia sau a mai multor identificatori din listă folosind o instrucțiune de inițializare. Identificatorii din listă care apar după inițializare primesc automat valori consecutive, începând cu următoarea valoare după cea folosită la inițializare.
Programul va afișa:
-1, 0, 6, 7, 8, 9(ENTER)

6. Analizaţi corectitudinea următorului program:

```
#include<stdio.h>
int main() {
    int a[] = {10, 20, 30, 40, 50};
    int *b = a+2;
    printf("%d\n", *b++);
    printf("%d\n", *b);
    return 0;
}
```

Indicaţi ce se afişează pe monitor la rularea programului (dacă programul este corect) sau, dacă programul nu este corect, indicaţi eroarea (sau erorile) care există în program și linia (sau liniile) la care apar aceste erori.

Denumirea tabloului a reprezintă adresa de început a acestuia în memorie.
În b se reține adresa lui a[2] (a+2)

La primul printf pentru *b++ se executa mai intai (*b) apoi (b++)

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|
| 10 | 20 | 30 | 40 | 50 |
| | | b=a+2 | | |
| | | *b=30 | b++ | |
| | | | *b=40 | |

Programul va afișa:
30(ENTER)
40(ENTER)

7.Fie următorul program:

```
#include <stdio.h>
int main()
{
     int a, b, c, d;
     scanf("%d %d %d", &a, &b, &c);
     if(a > b && c < a) {
          d = a << (b & c);
     }
     else {
          d = a | (b >> c);
     }
     d ^= a;
     printf("%04d", d);
     return 0;
}
```

Ce se afişează pe monitor la execuţia acestui program se introduc valorile: 4 1 3? Justificați.

a=4, b=1, c=3
4>1 Și 3<4 (Adevărat)

⇨ d=a<<(b&c)
b=1 = 0000 0001
c=3 = 0000 0011
b&c= 0000 0001

a=4 = 0000 0100
a<<1=0000 1000=16

d=16=0000 1000
a=4 =0000 0100
d^=a=0000 1100=8+4=12

%04d este specificatorul de format utilizat pentru a afișa o variabilă de tip întreg pe un câmp de 4 caractere completând cu 0-uri.
Programul va afișa:
0012

8. Analizaţi corectitudinea următorului program:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
     int *a = 0;
     int i;
     a = (int*)malloc(3*sizeof(int));
     if(a == 0)
     {
          fprintf(stderr, "%s - small memory",
argv[0]);
          exit(EXIT_FAILURE);
     }
     for(i=0; i<argc-1; ++i)
     {
          *(a+i) = 0x10+atoi(argv[i+1]);

          printf("%d \n", *(a+i));
     }
     free(a);
     a=0;
     return 0;
}
```

Indicaţi ce se afişează pe monitor la rularea programului (dacă programul este corect) sau, dacă programul nu este corect, indicaţi eroarea (sau erorile) care există în program indicând și linia (sau liniile) la care apar erorile.

Linia de comanda cu care se lansează în execuție programul este:

./s8 12 15 161

Se alocă a cu 3 locații de tip int folosind funcția malloc. Numărul argumentelor din linia de comandă este argc=4. Argumentele din linia de comandă sunt:

| argv[0] | argv[1] | argv[2] | argv[3] |
|---|---|---|---|
| "s8" | "12" | "15" | "161" |

Constanta hexazecimală 0x10=16+0=16 în baza 10. Funcția atoi convertește un argument de tip șir de caractere într-o valoare numerică. Atunci când conversia nu este posibilă returnează 0.

| a[0] | a[1] | a[2] |
|---|---|---|
| 12+16=28 | 15+16=31 | 161+16=177 |

Programul va afișa:
28(SPAȚIU) (ENTER)
31(SPAȚIU) (ENTER)
177(SPAȚIU)(ENTER)

9. Ce se afişează la execuţia următorului program (uniunea folosită este declarată în fișierul header **s9.h** ca în coloana din stânga), în cazul în care programul este corect? Dacă programul nu este corect indicați eroarea. Justificare.

```
union var {
        int a, b;
     };
```

```
#include <stdio.h>
#include "s9.h"
int main(void) {
    union var v;
    v.a=10;
    v.b=20;
    printf("%d\n", v.a);
    return 0;
}
```

Uniunea este un tip de dată special care permite stocarea unor variabile de diferite tipuri în aceeași locație de memorie. Dacă pentru o structură se alocă memorie pentru fiecare membru în parte în regiuni consecutive, pentru o uniune se aloca memorie pentru cel mai mare dintre membrii ei, iar datele sunt stocate începand de la respectiva adresă. Chiar dacă în definiția uniunii sunt mai mulți membri, doar unul singur va putea fi inițializat cu o anumită valoare la un moment dat.

| v.a=10 | v.b=20 | v.a |
|---|---|---|
| 10 | 20 | 20 |

Programul va afișa:
20