PROGRAMARE I

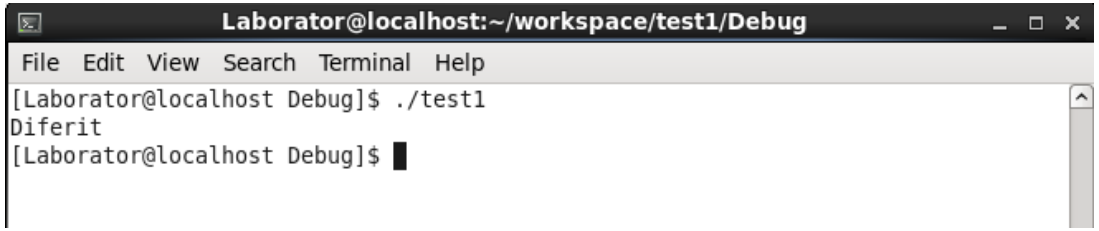
2016

Proba teoretica

6 probleme x 1.5p + 1p oficiu

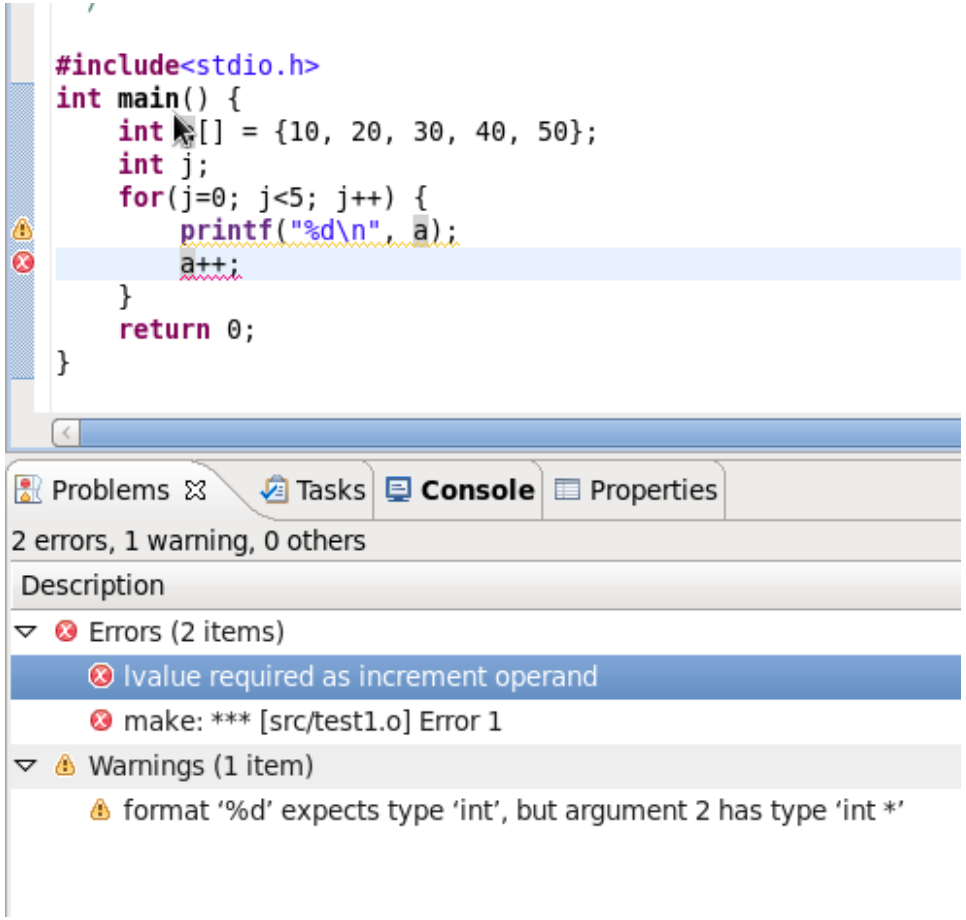
Problema 1

```c
#include <stdio.h>
int main()
{
    char str1[] = "str[]";
    char str2[] = "str[]";
    if(str1 == str2)
    {
        printf("Egal\n");
    }
    else
    {
        printf("Diferit\n");
    }
    return 0;
}
```



Numele unui tablou este un pointer constant care are ca valoare adresa primului element din tablou
str1, str2 = adresele de inceput ale celor doua tablouri.
Sirurile au acelasi continut.
Adresele de inceput sunt insa diferite.

Problema 2



Programul contine erori.

Corect ar fi fost sa scriem:

printf("%p\n",a+j); // afisare adresa

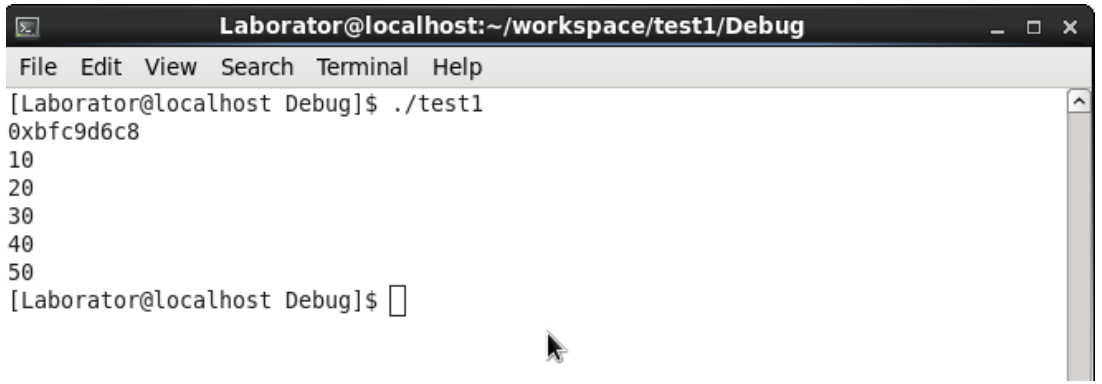
printf("%d\n",*(a+j));//afisare elem

- Numele unui tablou este un pointer constant care are ca valoare adresa primului element din tablou. a++ nu este correct.

Problema 2

```c
#include<stdio.h>
int main() {
    int a[] = {10, 20, 30, 40, 50};
    int j;
    printf("%p\n", a);
    for(j=0; j<5; j++)
    {
        printf("%d\n", *(a+j));
        //a++;
    }
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int a[]={10, 20, 30, 40, 50};
    int j;
    int *b;
    b= a;
    for(j=0;j<5;j++)
    {
        printf("%d\n",*b);
        b++;
    }
    return 0;
}
```

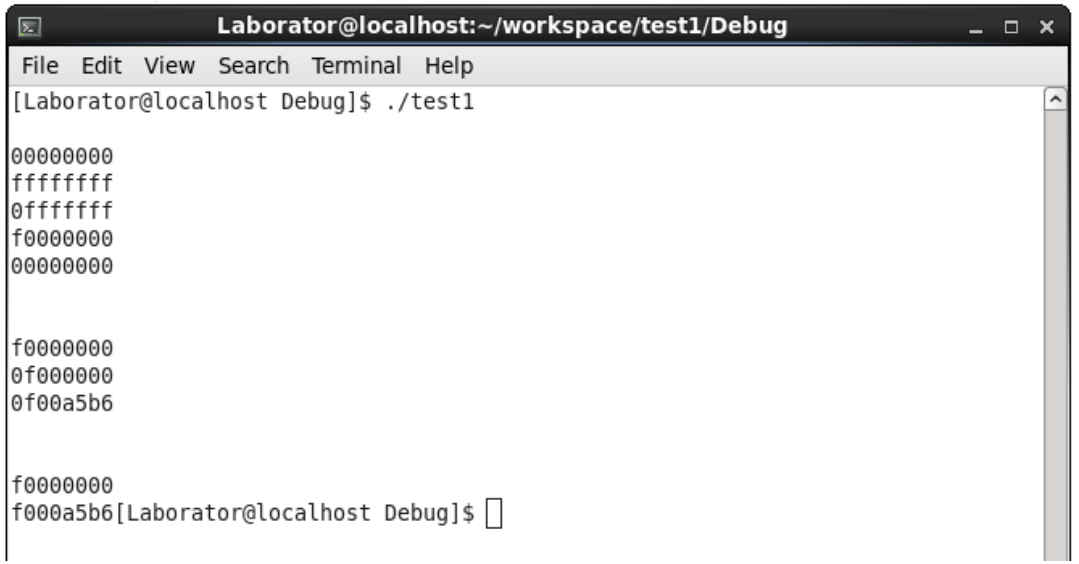


Problema 3

```c
#include<stdio.h>
int main() {
    printf("\n%08x",0u);
    printf("\n%08x",~0u);
    printf("\n%08x",(~0u)>>4);
    printf("\n%08x",~((~0u)>>4));
    printf("\n%08x",~((~0u)>> 4) & 0xa5B6u);

    printf("\n\n");
    printf("\n%08x",~((~0u)>>4));
    printf("\n%08x",(~((~0u)>>4))>>4);
    printf("\n%08x",(~((~0u)>>4))>>4 | 0xA5b6u);

    printf("\n\n");
    printf("\n%08x",~((~0u)>>4));
    printf("\n%08x",~((~0u)>> 4) ^ 0xa5B6u);
    return 0;
}
```



0u = 0 (unsigned)
sizeof(0u) = 4; //4 octeti=32 de biti
0xa5B6u = 0xA5b6u=0x0000A5B6u
~ NOT
& AND
| OR
^ XOR
>> 4 shift dreapta cu 4 pozitii, se completeaza cu 0-uri pozitiile ramase libere
ATENTIE LA:

- Nr paranteze deschise si inchise
- Sa completati in stanga cu 0-uri pana la nr maxim de biti pe care este reprezentat numarul in cazul in care e nevoie
- Sa nu aveti cifra mai mare decat baza de reprezentare

```
int a= 0b1010;//baza 2 0b in fata
int b= 01010;//baza 8 0 in fata
int c= 0x1010;//baza 16 0x in fata
int d = 1010;//baza 10
printf("\n %08x %08x %08x %08x",a,b,c,d);
```

Problema 3

Int => 32 biti => 4octeti

a= 00 00 00 0A

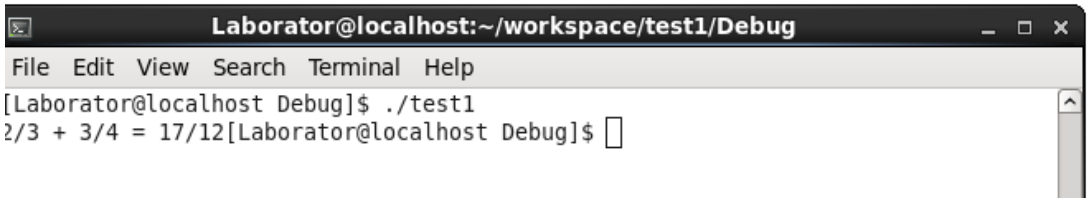
b= 001 000 001 000=00 00 02 08

c= 00 00 10 10

d= 11 1111 0010=00 00 03 f2

Problema 4

```c
#include <stdio.h>
struct _Fractie
{
    int a, b;
};
typedef struct _Fractie Fractie;
Fractie suma(Fractie f1, Fractie f2)
{
    Fractie f;
    f.a = f1.a*f2.b + f1.b*f2.a;
    f.b=f1.b*f2.b;
    return f;
}
int main()
{
    Fractie f, f1, f2;
    f1.a=2;
    f1.b=3;
    f2.a=3;
    f2.b=4;
    f=suma(f1,f2);
    printf("%d/%d + %d/%d = %d/%d",f1.a,f1.b,f2.a,f2.b,f.a,f.b);
    return 0;
}
```



Declaraţia (prototipul) unei funcţii

Declaraţia unei funcţii se găseşte într-un fişier header (sau antet) şi este compusă din antetul funcţiei urmat de semnul de caracterul **;**.

Prototipul unei funcţii nu "spune" cum prelucrază funcţia informaţiile primite, ci indică numai legaăturile sale cu exteriorul.

In cazul nostru:

Fractie suma(Fractie f1, Fractie f2);

Definiţia unei funcţii

Definiţia unei funcţii se va găsi întotdeauna într-un fişier cu cod sursă şi este formată din antetul funcţiei urmat de un bloc în care se găseşte algoritmul, exprimat în limbajul de programare **C**, care descrie modul în care funcţia realizează prelucrările informaţiei primate din exterior. Blocul începe întotdeauna cu caracterul **{** şi se încheie cu **}**.

Fractie suma(Fractie f1,Fractie f2)

{

...

}

Problema 5

```c
#include <stdio.h>
void f1(int *a, int n)
{
    int i;
    printf("n = "); scanf("%d", &n);
    for(i=0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }
}
void f2(int *a, int *n)
{
    int i;
    printf("n = "); scanf("%d", n);
    for(i=0; i<*n; i++)
    {
        scanf("%d", a+i);
    }
}
void afisareVector(int *a, int n)
{
    int i;
    for(i=0;i<n;i++)
        printf("%d ",*(a+i));
}
int main(void)
{
    int a[10], n = 0;
    f1(a,n);
    printf("Vectorul are %d elemente\n", n);
    afisareVector(a,n);

    printf("\n\n\n");
    f2(a,&n);
    printf("Vectorul are %d elemente\n", n);
    afisareVector(a,n);
    return 0;
}
```

```
[Laborator@qualindser7 Debug]$ ./test1
n = 4
3
45
3
15
Vectorul are 0 elemente



n = 4
3
45
3
15
Vectorul are 4 elemente
3 45 3 15 [Laborator@qualindser7 Debug]$
```

Parametrii transferati unei functii in C pot fi valori sau adrese. Prima variant este numita si apel prin valoare, copie valoarea unui arg (param formal) al unei fctii. Modificarile efectuate asupra param nu au effect asupra argumentului.

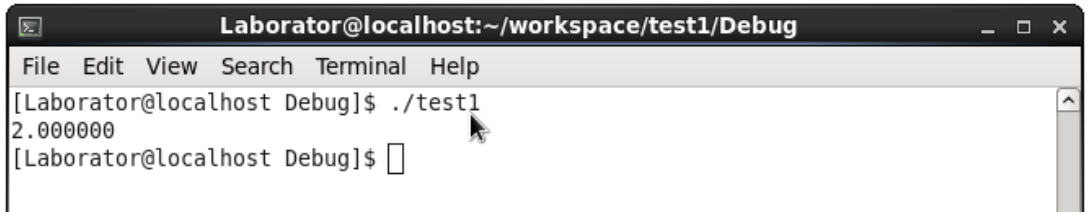
Cazul lui f1 n=0;

Al doilea mod de transfer este cel prin referinta(prin adresa). In acest caz se copie adresa argumentului, iar in interiorul functiei aceasta adresa este folosita pentru a avea acces la argumentul folosit efectiv la apelare, deci modificarile asupra argumentului il vor afecta.

Cazul lui f2 n se modifica din 0 in 4.

Problema 6.

```c
#include <stdio.h>
int main (void)
{
    double x;
    int a = 7, b = 3;
    x = (int)1.5L * (double)(a/b);
    printf("%lf\n", x);
    return 0;
}
```



(int)1.5L = 1
(a/b)=2
(double)(a/b)=2.000000

u (unsigned),
l (lowercase L) (long),
ll (long long),
ul (unsigned long),
ull (unsigned long long),
lu (long unsigned),
llu (long long unsigned)
and uppercase versions.