**PROBLEME ASM**

**SUMA A DOUA NUMERE**

int a, b, result = 0;

cin >> a >> b;

\_asm {

mov eax, a

mov ebx, b

add eax, ebx

mov result, eax

}

cout << result << endl;

return 0;

**15 – (a+b) =?**

int main()

{

int a, b, c, d;

cin >> a >> b;

\_asm

{

MOV EAX, a // eax = a

MOV EBX, b // ebx = b

ADD EAX, EBX // eax += ebx

MOV c, EAX // c = a + b

MOV ECX, 15 // ecx = 15

MOV EBX, 0 // ebx = 0

\_for: // \*\*\*

CMP EAX, ECX

JGE \_end\_for // %%% cand eax>=ecx

INC EBX // ebx++

INC EAX // eax++

JMP \_for // \*\*\*

\_end\_for :

MOV d, EBX // %%% ebx = d

}

cout << "a = " << a << '\n';

cout << "b = " << b << '\n';

cout << "c = " << c << '\n';

cout << "d = " << d << '\n';

return 0;

}

**DE CATE ORI APARE VOCALA a?**

int main()

{

char \*s = "Maimutica mica";

int number = 0;

\_asm

{

MOV EAX, s // sirul nostru

MOV ECX, 0 // un fel de i

MOV EDX, 0 // nr de vocale

\_while:

MOV BL, [EAX + ECX]

CMP BL, '\0' // daca am ajuns la sfarsitul sirului

JE \_end\_while

CMP BL, 'a' // daca este vocala A

JNE \_este\_a

INC EDX

\_este\_a :

INC ECX

JMP \_while

\_end\_while :

MOV number, EDX

}

cout << number << '\n';

return 0;

}

**ESTE NUMARUL PRIM?**

int main()

{

int numar;

cin >> numar;

int ok = 1; //daca il facem bool imi da error: operand size conflict (la EBX acolo)

int divizor = 2;

\_asm

{

MOV EBX, ok

MOV ECX, divizor

\_for :

MOV EDX, 0

MOV EAX, numar

CMP ECX, EAX // comparam divizorul cu numarul,

JGE \_end\_of\_for // daca e mai mare sau egal divizorul iesim din for

DIV ECX

CMP EDX, 0 // altfel, impartim numarul la divizor si vedem care este restul

JE \_notprim // daca restul este zero inseamna ca numarul nu e prim.

INC ECX

JMP \_for

\_notprim :

MOV EBX, 0 // ok=0;

\_end\_of\_for :

MOV ok, EBX

}

if (ok)

cout<<"Numarul este prim\n";

else

cout<<"Numarul nu este prim\n";

return 0;

}

**MAXIMUL DINTRE 2 NR**

int main()

{

int a, b, maxi;

cin >> a >> b;

\_asm {

mov eax, a

mov ebx, b

cmp eax, ebx

**JG \_else**

mov maxi, ebx

**JMP \_end\_else**

**\_else:**

**mov maxi, eax**

**\_end\_else:**

}

cout << maxi << endl;

return 0;

}

**1 + 2 + ... + n =?**

int main()

{

int n, sum;

cin >> n;

\_asm

{

mov eax, 0; / nr de adaugat

mov ebx, 0; / suma

cat\_timp:

cmp eax, n

je rez // am ajuns la n

add ebx, eax // adaug in suma

inc eax;

jmp cat\_timp

rez:

add ebx, eax // il adaug si pe n

mov sum, ebx // salvez suma in ebx.

}

cout << sum << endl;

return 0;

}

**CMMDC**

int main()

{

int a, b;

cin >> a >> b;

int cmmdc = 0;

\_asm

{

mov eax, a

mov ebx, b

bucla:

cmp eax, ebx

JE EXIT //if(a==b)

cmp eax, ebx

JG E1

sub ebx, eax // ebx = ebx - eax;

jmp bucla

E1: sub eax, ebx // eax = eax - ebx;

jmp bucla

EXIT:

mov cmmdc, eax //am gasit cmmdc

}

cout << cmmdc << endl;

return 0;

}

**SUMA ELEMENTELOR PARE DIN VECTOR**

int sumapare(int\*, int) //pointer catre vector si nr de elemente

{

int suma = 0;

\_asm {

mov ebx, [ebp+8] // primul parametru

mov esi, [ebp+12] // nr de elemente

mov ecx, 0 // contor pentru i

\_for:

cmp ecx, esi // verificam daca am ajuns la sfarsitul forului

jge \_end\_for

mov eax, [ebx + 4\*ecx] // retinem elementul de pe pozitia i din vector

mov edx, 0 // restul

mov edi, 2 // impartim la 2 pentru a verifica paritatea

div edi // impartim eax la edi, adica elementul din vector la 2

cmp edx, 0

jne \_impar // nu se modifica suma

mov eax, [ebx + 4\*ecx] // punem din nou elem in eax pentru ca s-a modificat la impartire!

// ! MEREU in eax ramane catul iar in edx restul, oricare 2 registri am folosi.

add suma, eax

\_impar:

inc ecx

jmp \_for

\_end\_for:

mov eax, suma // echivalent cu return suma;

// ! O functie returneaza valoarea din EAX daca nu are alt return.

}

}

int main()

{

int v[100];

int s = 0;

int n;

cin >> n;

for (int i = 0; i < n; i++)

cin >> v[i];

s = sumapare(v, n);

cout << s << endl;

return 0;

}

**NR DE CONSOANE DINTR-UN SIR DE CARACTERE**

int nrConsoane(char \*) // pointer catre sir

{

\_asm {

mov esi, [ebp+8] // primul parametru

xor eax, eax // nr de vocale

\_while:

mov bl, [esi]

cmp bl, 0

jz \_sfarsit // jz = jump if zero

// am extras o litera

cmp bl, ' ' // daca e format sirul din mai multe cuvinte

jz \_notconsoana

or bl, 32 // o convertim in litera mica

cmp bl, 'a'

jz \_notconsoana

cmp bl, 'e'

jz \_notconsoana

cmp bl, 'i'

jz \_notconsoana

cmp bl, 'o'

jz \_notconsoana

cmp bl, 'u'

jz \_notconsoana

inc eax // daca e consoana

\_notconsoana:

inc esi

jmp \_while

\_sfarsit:

// nu scriem nimic deoarece va returna automat eax

}

}

int main()

{

char \*s = "Dana are mere";

int nr\_consoane = nrConsoane(s);

cout << nr\_consoane << endl;

return 0;

}

**FACTORIAL RECURSIV**

unsigned int FactorialRecursiv (unsigned int nr)

{

\_asm {

mov esi, [ebp + 8] // parametrul functiei il tinem minte in esi

cmp esi, 2 // comparare n cu 2

jge \_apel\_recursiv // Daca n >= 2, se face saltul, altfel:

mov eax, 1 // return 1

jmp gata

\_apel\_recursiv :

dec esi // n--

push esi // punem n pe stiva

call FactorialRecursiv // apelam recursiv FactorialRecursiv

add esp, 4 // eax = FactorialRecursiv(n-1)

mov esi, [ebp + 8] // esi = n

mul esi // return FactorialRecursiv(n-1)\*n

gata:

}

}

int main() {

int n, fact;

cin >> n;

\_asm {

push n

call FactorialRecursiv

add esp, 4

mov fact, eax

}

cout << n << "! = " << fact << endl;

return 0;

}

**ESTE NR PRIM? APEL FUNCTIE**

int EstePrim(int n)

{

int ok = 1; // daca il fac bool : operand size conflict

int divizor = 2;

\_asm {

mov ebx, ok

mov ecx, divizor

\_for:

mov edx, 0 // edx mereu zero pt impartire deoarece in el se pastreaza restul!

mov eax, n

cmp ecx, n // comparam divizorul cu numarul,

jge \_end\_for // daca divizorul >= n, iesim din for

div ecx

cmp edx, 0 // verific daca am gasit un divizor

je \_notprim

inc ecx

jmp \_for

\_notprim:

mov ebx, 0 // nr nu este prim

\_end\_for:

mov ok, ebx

mov eax, ok // astfel incat functia sa returneze 1 daca nr este prim sau 0 in caz contrar

}

}

int main() {

int n, adevarul;

cin >> n;

\_asm {

push n

call EstePrim

add esp, 4

mov adevarul, eax

}

cout << adevarul << endl;

return 0;

}

**SUMA ELEMENTE PRIME DIN VECTOR**

#include <iostream>

using namespace std;

int prim(int)

{

\_asm

{

MOV EBX, [EBP + 8] // numar

// verific mai intai daca nr este 0 sau 1.

CMP ebx, 1

JLE \_neprim

MOV ECX, 2 // contor

\_for :

MOV EAX, EBX // eax -> numarul

CMP ECX, EAX // contor >= numar

JGE \_stopfor

XOR EDX, EDX // intializam restul cu 0

DIV ECX // impartim numarul la contor

CMP EDX, 0

JE \_neprim // daca nu avem rest, numarul nu este prim

INC ECX // creste contorul

JMP \_for

\_stopfor :

MOV EAX, 1 // prim ♥

JMP \_fin

\_neprim : // neprim

MOV EAX, 0

\_fin :

}

}

int suma(int \*, int)

{

\_asm

{

MOV EBX, [EBP + 8] // vectorul

MOV ESI, [EBP + 12] // nr elemente

XOR EDI, EDI // counter

MOV EDX, 0 // suma

\_for:

CMP EDI, ESI // daca counterul depaseste numarul de elemente iese

JGE \_endfor

MOV ECX, [EBX + 4 \* EDI] // muta in ebx elementul curent

// face copii ale registrilor in stiva. Initial am facut la toti si am scos din ei pana nu mai mergea :d

push EDX

push ECX // elementul pe care il foloseste functia prim

call prim

pop ECX

pop EDX

// scoate copiile din stiva ^^^^

CMP EAX, 1 // daca prim != 1 nu face suma

JNE \_nothing

ADD EDX, ECX

\_nothing :

INC EDI

JMP \_for

\_endfor :

MOV EAX, EDX

}

}

**Afisare element din matrice**

int main()

{

int v[100];

int n, vprim = 0;

cout << "Introduceti n \n";

cin >> n;

cout << "Introduceti elementele vectorului \n";

/\*if (prim(n))

cout << "prim\n";

else

cout << "neprim\n"; \*/

for (int i = 0; i < n; i++)

cin >> v[i];

cout << "Suma elementelor prime din vector este: ";

cout << suma(v, n);

cout << endl;

return 0;

}

#include "stdafx.h"

#include <malloc.h>

#include <iostream>

using namespace std;

#define L 10

#define C 20

int f(int m[][C]) {

//returnarea lui m[0][0]

\_asm {

mov eax, [ebp + 8]

mov eax, [eax]

}

}

int main()

{

int m[L][C];

m[0][0] = 1024000;

int x = f(m);//vectorii si matricele se transmit prin referinta

cout << x << endl;

return 0;

}

**RECURSIV SUMA ELEM PANA LA N**

int suma(int n)

{

\_asm {

mov ebx, [ebp + 8] // n

cmp ebx, 1

jne else

mov eax, 1

jmp end\_function

else:

sub ebx, 1 // n--

push ebx

call suma

add esp, 4 // curat stiva

add eax, [ebp + 8] // adun n

end\_function :

}

}

int main()

{

int n, s;

cin >> n;

\_asm {

push n

call suma

add esp, 4 //curat stiva

mov s, eax

}

cout << s << endl;

return 0;

}

**ORDONARE VECTOR CRESC.**

void sortare(int v[], int n)

{

\_asm {

mov ebx, [ebp + 8] // pointer catre vector

mov esi, 1 // i

start\_for1: cmp esi, [ebp + 12] // i<n

jg exit\_for1

mov edi, esi

inc edi // j=i+1

start\_for2 : cmp edi, [ebp + 12] // j<=n

jge exit\_for2

mov ecx, [ebx + esi \* 4] // a[i]

mov edx, [ebx + edi \* 4] // a[j]

cmp ecx, edx

jle increment

mov[ebx + esi \* 4], edx // interschimb

mov[ebx + edi \* 4], ecx

increment : inc edi // j++

jmp start\_for2

exit\_for2 : inc esi // i++

jmp start\_for1

exit\_for1 :

}

}

int main()

{

int v[10] = { 0, 9, 3, 1, 7 }, n = 5, i;

\_asm

{

push n

lea ebx, v

push ebx

call sortare

add esp, 8

}

for (i = 0; i < 5; i++)

cout << v[i] << ' ';

cout << endl;

return 0;

}

**INLOCUIRE ELEMENTE NULE MATRICE**

#include "stdafx.h"

#include <iostream>

#include <windows.h>

#include <string.h>

#include <stdio.h>

using namespace std;

#define newline f1();

void f1()

{

cout << '\n';

}

//i\*nrcol+j

void matrice(int \*, int)

{

\_asm

{

MOV EBX, [EBP + 8] //matricea

XOR ESI, ESI //i

\_fori :

CMP ESI, [EBP + 12]//compara i cu numarul de linii

JGE \_endfori

XOR EDI, EDI//j

\_forj :

CMP EDI, [EBP + 12]//compara j-ul cu nr de coloane

JGE \_endforj

// !!!!!!!!!!!!!!!!!!

MOV EAX, [EBP + 12]//punem nr col in eax

MUL ESI//inmultim cu i

ADD EAX, EDI //adunam j

// !!!!!!!!!!!!!!!!

// i\*nrcol+j ^

MOV EDX, [EBX + 4 \* EAX] //Punem in EDX elementul curent din matrice

CMP EDX, 0

JE \_modifica

JMP \_nomodif

\_modifica :

MOV EDX, 69//face numarul ca fiind 69

MOV[EBX + 4 \* EAX], EDX//pune numarul pe pozitia curenta in matrice

\_nomodif :

INC EDI//creste j-ul

JMP \_forj//reia forul pentru j

\_endforj :

INC ESI//creste i-ul

JMP \_fori

\_endfori :

}

}

int main()

{

int a[3][3];

int n;

cin >> n;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

cin >> a[i][j];

matrice((int \*)a, n);

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

cout << a[i][j] << ' ';

cout << endl;

}

}

**MATRICEA UNITATE – 1 PE DIAGONALE, IN REST 0**

#include "stdafx.h"

#include <iostream>

#include <windows.h>

#include <string.h>

#include <stdio.h>

using namespace std;

#define newline f1();

void f1()

{

cout << '\n';

}

//i\*nrcol+j

void matrice(int \*, int)

{

\_asm

{

mov ebx, [ebp+8] // matrice

mov esi, 0 // i

\_fori:

CMP esi, [ebp+12]

JGE \_exitfori

mov edi, 0

\_forj:

CMP edi, [ebp+12]

JGE \_exitforj

CMP edi, esi

JNE \_zero

mov eax, [ebp+12] // eax -> nr de coloane

mul esi // i \* nr col

add eax, edi // +j

// accesam elementul

// mov ecx, [ebx + 4\*eax] // a[i][j]

mov ecx, 1

mov [ebx + 4\*eax], ecx // daca nu se afla pe diag pr

JMP \_fin

\_zero:

mov edx, esi

add edx, edi

// i+j

mov ecx, [ebp+12]

dec ecx

// n-1, fiindca incepem numerotarea de la 0

CMP edx, ecx

JNE \_diagonala

//daca sunt egale, le facem 1 pentru ca se afla pe diagonala secundara

mov eax, [ebp + 12]

mul esi

add eax, edi

mov ecx, 1

mov[ebx + 4 \* eax], ecx

JMP \_fin

\_diagonala: mov eax, [ebp+12]

mul esi

add eax, edi

mov ecx, 0

mov [ebx + 4\*eax], ecx // daca se afla pe diag pr

\_fin:

inc edi

JMP \_forj

\_exitforj: inc esi

JMP \_fori

\_exitfori:

}

}

int main()

{

int a[5][5];

int n;

/\*cin >> n;

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

cin >> a[i][j];

\*/

matrice((int \*)a, 5); //!!!!!!!!!!!!!!!!

for (int i = 0; i < 5; i++)

{

for (int j = 0; j < 5; j++)

cout << a[i][j] << ' ';

cout << endl;

}

}

**TRANSFORMARE LITERE MICI IN LITERE MARI**

void Transform(char \*s)

{

\_asm {

mov ebx, [ebp+8] // sirul nostru :)

mov ecx, 0

\_while:

mov al, [ebx+ecx]

CMP al, '\0' // daca s-a terminat sirul

JE \_final

CMP al, 'a'

JL \_gata

CMP al, 'z'

JG \_gata

sub al, 32

mov [ebx+ecx], al

\_gata:

inc ecx

JMP \_while

\_final:

}

}

**SUMA ELEMENTELOR DIN MATRICE**

int main()

{

char s[256];

cin.get(s,256);

cin.get();

Transform(s);

cout << s << endl;

return 0;

}

#include <iostream>

using namespace std;

int Suma(int \*, int)

{

\_asm {

mov ebx, [ebp+8] // matricea

push 0 // pastram suma in stiva, ii rezervam un loc

mov edi, 0 // i

\_fori:

CMP edi, [ebp+12]

JGE \_exitfori

mov esi, 0 // j

\_forj:

CMP esi, [ebp+12]

JGE \_exitforj

mov eax, [ebp+12] // nr coloane

mul edi // \*i

add eax, esi // +j

mov ecx, [ebx + 4\*eax]

// acum avem in ecx elementul a[i][j] <3

add [esp], ecx // adaugam in suma pastrata pe stiva

inc esi

JMP \_forj

\_exitforj:

inc edi

JMP \_fori

\_exitfori:

pop eax // ultima valoare introdusa in stiva. care la noi este suma. pt ca e singura.

}

}

int main()

{

int n,i,j,a[3][3],s;

cin>>n;

cout<<endl;

for (i=0;i<n;i++)

{ for(j=0;j<n;j++)

{cin>>a[i][j];}

cout<<endl;}

s = Suma((int\*)a, n);

cout << s << endl;

}

**DISTANTA DINTRE DOUA PUNCTE, CU STRUCT**

struct point{

int x,y;

};

/\*float dist(point \*a,point \*b)

{

return sqrt((a->x-b->x)\*(a->x-b->x)+(a->y-b->y)\*(a->y-b->y));

}\*/

int dist(point \*a,point \*b)

{

int rad;

\_asm{

mov esi, a

mov edi, b

mov eax, [esi] //a.x

sub eax, [edi] //a.x - b.x

imul eax,eax // ^2

mov ecx,eax // salveaza in ecx rezultatul

mov eax, [esi+4] //a.y

sub eax, [edi+4] //a.y - b.y

imul eax,eax //^2

add ecx,eax //formula

mov rad,ecx

}

return sqrt(rad);

}

int main()

{

point a,b;

a.x=2;

a.y=7;

b.x=6;

b.y=6;

cout<<"Distanta este="<<dist(&a,&b)<<endl;

return 0;

}