



Tema 2 - ACS Cat Invasion

- Deadline SOFT: ~~10.05.2022~~ 12.05.2022, ora 23:55
- Deadline HARD: 15.05.2022, ora 23:55
- Data publicării: 21.04.2022
- Responsabili:
 -  Ilina-Ioana Struțu
 -  Răzvan-Nicolae Virtan
- Actualizări:
 - 21.04.2022 postare tema
 - 22.04.2022 adăugare hint-uri legate de parcurgerea matricilor alocate static
 - 22.04.2022 update checker (fix la task-ul 4, testare individuală a task-urilor)
 - 26.04.2022 adăugare zonă [FAQ](#)
 - 29.04.2022 modificare deadline soft
 - 05.05.2022 actualizare checker task 2 (nu afectează implementarea)
 - 06.05.2022 configurare vmchecker

Enunț

Patru pisici au invadat Facultatea de Automatică și Calculatoare. Țelul lor este să acapareze toate clădirile facultății fără a fi descoperite. Rolul vostru e de a le ajuta să se plimbe prin campus neobservate. Pentru a considera o clădire cucerită, trebuie să le ajutați să rezolve câte o problemă întrucât ele nu știu să scrie cod în limbaj de asamblare.

Structură și detalii de implementare

Tema este formată din 4 exerciții independente. Fiecare task constă în implementarea unei sau mai multor funcții în limbaj de asamblare. Implementarea se realizează în fișierele puse la dispoziție pentru fiecare exercițiu.



Parametrii funcțiilor sunt plasați în registre, în cadrul scheletului.



Scheletul include și macro-ul `PRINTF32`, folosit în laborator, pentru a vă ajuta la depanarea problemelor. Tema finală nu trebuie să facă afișări folosind `PRINTF32`, funcții externe sau apeluri de sistem.



În tema finală este interzisă apelarea funcțiilor externe.

1. Simple cipher - 10p

Sursee dorește să ajungă pe catedra din PR001 așa că trebuie să trimită mesaje criptate studenților că să afle cine o poate ajuta. Ea vrea să folosească *simple cipher* pentru a transmite mesajele.

Acest algoritm de criptare presupune shiftarea la dreapta în cadrul alfabetului a fiecărui caracter de un anumit număr de ori. De exemplu, textul *ANABANANA* se transformă în *BOBCBOBOB* când pasul este 1. Astfel, o criptare cu pasul 26 nu modifică litera, întrucât alfabetul englez are 26 de caractere.

Pentru acest task va trebui să implementați în fișierul **simple.asm** funcția **simple()**, care criptează un string folosind metoda descrisă mai sus.

Antetul funcției este:

```
void simple(int n, char* plain, char* enc_string, int step)
```

Semnificația argumentelor este:

- **n** dimensiunea textului
- **plain** string-ul care trebuie criptat
- **enc_string** adresa la care se va scrie textul criptat
- **step** cu cât se shiftază fiecare caracter



Pentru ușurință se vor folosi doar majusculele alfabetului englez (A-Z), iar shiftarea se realizează strict în cadrul alfabetului englez cu o limită de 26 pentru step.

2. Points - 25p

Diablo, mîncăciosul, vrea să ajungă la cantina din EC pentru a-și pune lăbuțele pe un meniu de mâncare proaspătă. Ajutați-l să găsească cel mai scurt drum de la intrarea în facultate până la cantină.

Prin intermediul acestui task se dorește aprofundarea lucrului cu structuri.

- Anunțuri
- Bune practici
- Calendar
- Catalog
- Feed RSS
- IOCLA Need to Know
- Reguli și notare
- Resurse utile

Cursuri

- Capitol 00: Prezentare
- Capitol 01: Programe și sistemul de calcul
- Capitol 02: Construirea programelor
- Capitol 03: Arhitectura sistemului de calcul
- Capitol 04: Reprezentarea numerelor
- Capitol 05: Interfața hardware - software x86
- Capitol 06: Stiva
- Capitol 07: Funcții
- Capitol 08: Interfața binară a funcțiilor
- Capitol 09: Gestiunea bufferelor
- Capitol 10: Curs ales de titular
- Capitol 11: Optimizări

Laboratoare

- Laborator 01: Reprezentarea numerelor, operații pe biți și lucru cu memoria
- Laborator 02: Operații cu memoria. Introducere în GDB
- Laborator 03: Compilare
- Laborator 04: Toolchain
- Laborator 05: Introducere în limbajul de asamblare
- Laborator 06: Rolul registrelor, adresare directă și bazată
- Laborator 07: Date Structurate. Structuri, vectori. Operații pe siruri
- Laborator 08: Lucrul cu stiva
- Laborator 09: Apeluri de funcții
- Laborator 10: Interacțiunea C-assembly
- Laborator 11: Gestiunea bufferelor. Buffer overflow
- Laborator 12: CTF
- Laborator facultativ: ARM assembly

Teme

- Tema 1 - Momente disperate
- Tema 2 - ACS Cat Invasion
- Tema 3 - Poli Cat Invasion

Table of Contents

- Tema 2 - ACS Cat Invasion
 - Enunț
 - Structură și detalii de implementare
 - 1. Simple cipher - 10p
 - 2. Points - 25p
 - 3. Beaufort Encryption - 25p
 - 4. Spiral Encryption - 30p
 - Precizări suplimentare
 - Trimitere și notare
 - FAQ
 - Resurse

Se dă structura unui punct în coordonate carteziene:

```
struct point{
    short x;
    short y;
};
```



Înainte de rezolvarea acestui task, recomandăm parcurgerea laboratorului 7

<https://ocw.cs.pub.ro/courses/iocla/laboratoare/laborator-07>



- Pentru obținerea punctajului maxim pe acest task trebuie realizate toate exercițiile.
- Punctajul pentru ex 2 nu se acordă fără rezolvarea exercițiului 1, iar pentru exercițiul 3 trebuie rezolvate și exercițiile 1 și 2

a) Points distance - 5p

Pentru această parte a task-ului aveți de implementat funcția **points_distance()** din fișierul **points-distance.asm**, care va calcula distanța dintre două puncte aflate pe o dreapta paralelă cu axele OX sau OY.

Antetul funcției este:

```
void points_distance(struct point *p, int *rez);
```

Semnificația argumentelor este:

- **p** adresa de început a vectorului de puncte
- **rez** adresa unde trebuie scrisă distanța dintre cele două puncte

b) Road - 7.5p

În continuarea exercițiului 1, acum trebuie să implementați funcția **road()** din fișierul **road.asm**, care va calcula distanța dintre punctele dintr-un vector. Astfel, pentru un vector de 4 puncte (A,B,C,D), se vor calcula 3 distanțe: *A-B*, *B-C*, *C-D*. Perechile de puncte se află pe drepte paralele cu axele.

Antetul funcției este:

```
void road(struct point* points, int len, int* distances);
```

Semnificația argumentelor este:

- **points** adresa de început a vectorului de puncte
- **len** numărul de puncte
- **distances** adresa de început a vectorului de distanțe



Lungimea vectorului de distanțe este cu 1 mai mică decât lungimea vectorului de puncte.



Este permisă utilizarea funcției implementate la punctul a pentru realizarea acestui exercițiu.

c) Is square - 12.5p

Ultima parte a task-ului presupune analizarea fiecărei distanțe calculate anterior pentru a determina dacă este pătrat perfect. Trebuie să implementați funcția **is_square()** din fișierul **is_square.asm**.

Antetul funcției este:

```
void is_square(int *dist, int n, int *rez);
```

Semnificația argumentelor este:

- **dist** adresa de început a vectorului de distanțe calculate la exercițiul 2
- **n** lungimea vectorului
- **rez** adresa de început a vectorului rezultat în urmă verificărilor



Un pătrat perfect va fi notat cu 1, iar restul numerelor cu 0.

3. Beaufort Encryption - 25p

Swaro a aflat că cifrul folosit de Sursee a fost spart, așa că vă roagă să îi trimiteți mesaje cifrate folosind *criptarea Beaufort*, pentru a nu îi fi descoperit planul de a se infiltra în corpul ED.

Algoritmul de criptare Beaufort pleacă de la un string în clar (numit și plain text) și un string secundar (numit cheie) și obține un text criptat, de lungime identică cu string-ul în clar, prin înlocuirea caracter cu caracter a literelor din textul inițial.

De exemplu, să zicem că vrem să criptăm textul **DEFENDTHEEASTWALLOFTHECASTLE**, utilizând cheia **FORTIFICATION**.

Regula după care se face înlocuirea folosește o matrice de caractere, numită **tabula recta**, în care fiecare linie și fiecare coloană au asociate câte o literă a alfabetului limbii engleze. Aceasta are formatul de mai jos:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Criptarea fiecărui caracter se realizează aplicând următorii pași:

- asociem fiecărui caracter din textul în clar un caracter din cheie, repetând cheia pentru a acoperi tot string-ul inițial

DEFENDTHEEASTWALLOFTHECASTLE
FORTIFICATIONFORTIFICATIONFO

- mergem în `tabula recta` pe coloana asociată literei pe care vrem să o criptăm
- coborâm pe această coloană până când găsim litera asociată din cheie
- litera asociată liniei pe care am ajuns reprezintă caracterul criptat

Pentru caracterul **D**:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Criptând fiecare caracter folosind pașii de mai sus, obținem string-ul criptat. Pe exemplul nostru:

CKMPVCPVWPINUJOGIUAPVWRINUUK

Pentru acest task, va trebui să implementați în fișierul **beaufort.asm** funcția **beaufort()** care criptează un string în clar, folosind metoda descrisă mai sus.

Antetul funcției este:

```
void beaufort(int len_plain, char *plain, int len_key, char *key, char tabula_recta[26][26], char
```

Semnificația argumentelor este:

- **len_plain** și **plain**: dimensiunea textului în clar și string-ul în clar (adresa primului element)
- **len_key** și **key**: dimensiunea cheii și string-ul care reprezintă cheia (adresa primului element)
- **tabula_recta**: o matrice (**alocată static**) de 26 X 26 de caractere ale limbii engleze, cu conținutul din figurile de mai sus
- **enc**: adresa la care va trebui să scrieți textul criptat

În funcție de cum considerați că vă e mai ușor, puteți evita utilizarea argumentului `*char tabula_recta[26][26]*`, utilizând anumite corelări logice între linia și coloana matricii și conținutul ei. Ambele metode de rezolvare vor fi considerate la fel de corecte.

4. Spiral Encryption - 30p

Îngerăș, care este capul răutăților, vrea să cucerească toate laboratoarele din EG. Știe că are nevoie de o modalitate sigură de a transmite mesaje celorlalte pisici. Ajutați-o să trimită mesaje folosind *spiral encryption*.

O criptare prin substituție (în care înlocuim caracter cu caracter literele din textul în clar) e cu atât mai eficientă cu cât unui atacator îi e mai greu să decodifice mesajul inițial, chiar dacă are acces la cheie și la mesajul criptat. Pentru asta, putem alege moduri cât mai "ezoterice" de utilizare a cheii, iar **spiral encryption** e un algoritm care face tocmai acest lucru.

Vom pleca de la un text în clar de dimensiune $N \times 2$. Cheia în acest caz va fi o matrice de întregi de dimensiune $N \times N$. Un caracter din textul în clar va fi criptat adunând la codul său ASCII valoarea întreagă din cheie care îi corespunde. Spre exemplu, dacă pentru caracterul 'A' avem asociată în cheie valoarea 2, caracterul criptat va fi 'C'. De asemenea, dacă avem caracterul 'Z' cu cheia asociată 1, caracterul criptat va fi '['.

Partea specială la acest algoritm este modul în care asociem caracterelor din textul plain valori din cheie. Pentru asta, vom parcurge textul în clar secvențial (de la primul la ultimul caracter), iar în același timp vom parcurge cheia în spirală, similar cu figura de mai jos:

Textul în clar
AHUQEMOTMVXBNLAMZNATFBFXD →

Cheia

12	13	16	5	14
12	8	20	11	9
3	7	0	5	8
18	2	16	18	14
7	9	6	19	15

Astfel, ținând cont de dimensiunea matricii care reprezintă cheia și de dimensiunea textului (mereu vor fi corelate astfel încât numărul de elemente din matrice să fie egal cu numărul de litere din text) obținem întregul text criptat.

Pe exemplul de mai sus, rezultatul va fi:

```
MUEVSVWb\i^KU^DYbbLYXRH_D
```

Sarcina voastră este să completați funcția *spiral()* din fișierul *spiral.asm*, astfel încât aceasta să creeze un text în clar cu ajutorul unei chei date, folosind algoritmul prezentat mai sus. Antetul funcției este:

```
void spiral(int N, char *plain, int key[N][N], char *enc_string)
```

Semnificația argumentelor este:

- **N**: dimensiunea textului în clar va fi $N \wedge 2$, iar dimensiunea cheii va fi $N * N$
- **plain**: textul în clar, conținând $N \wedge 2$ caractere
- **key**: matricea (**alocată static**) care stochează cheia
- **enc_string**: adresa de start a șirului de caractere în care va trebui să întoarceți rezultatul (textul criptat)

Precizări suplimentare

În schelet este inclus și checker-ul, împreună cu testele folosite de acesta. Pentru a executa toate testele, se poate executa direct scriptul ``checker.sh`` din rădăcina temei:

```
./checker.sh
```

Pentru a testa task-uri individual, folosiți:

```
./checker.sh <număr_task>
```

Pentru a scrie rezolvarea unui task, intrați în directorul asociat task-ului respectiv și scrieți cod în fișierele în limbaj de asamblare indicate în enunț. **NU** modificați alte fișiere C, script-uri etc!

Formatul fișierelor de intrare pentru Simple Cipher este:

```
len_plaintext
step
plaintext
```

Formatul fișierelor de intrare pentru Points este:

```
nr_points
points
```

Formatul fișierelor de intrare pentru Beaufort Encryption este:

```
len_plaintext len_key
plaintext
key
```

Formatul fișierelor de intrare pentru Spiral Encryption este:

```
N
plain
key
```

În cadrul unora din cele 4 task-uri, va trebui să accesați valori de pe o anumită linie și coloană dintr-o matrice. Știți deja de la laborator cum să accesați date dintr-un vector. Accesarea valorilor din matricea alocată static este similară, deoarece chiar dacă noi o gândim ca fiind reprezentată pe linii și coloane, în realitate ea este reprezentată în memorie continuu. Tot ce trebuie să facem e să găsim un mod de a transforma linia și coloana într-un singur index.



Pentru matricea alocată static:

```
1 2 3
4 5 6
7 8 9
```

Avem de fapt în memorie un șir continuu de forma:

```
1 2 3 4 5 6 7 8 9
```

Trimitere și notare

Temele vor trebui încărcate pe platforma vmchecker (în secțiunea IOCLA) și vor fi testate automat.

Arhiva încărcată va fi o arhivă `.zip` care trebuie să conțină:

- fișierele sursă ce conțin implementarea temei: `simple.asm` `points-distance.asm` `road.asm` `is_square.asm` `beaufort.asm` `spiral.asm`
- `README` ce conține descrierea implementării

Punctajul final acordat pe o temă este compus din:

- punctajul obținut prin testarea automată de pe `vmchecker` - 90%
- `README` + coding style - 10%.

Coding style-ul constă în:



- prezența comentariilor în cod
- scrierea unui cod lizibil
- indentarea consecventă
- utilizarea unor nume sugestive pentru label-uri
- scrierea unor linii de cod/`README` de maxim 80-100 de caractere



Pentru detalii despre coding style parcurgeți acest document:
<http://www.sourceforge.com/pdf/asm-coding-standard-brown.pdf>



Temele care nu trec de procesul de asamblare (build) nu vor fi luate în considerare.



Mașina virtuală folosită pentru testarea temelor de casă pe `vmchecker` este descrisă în secțiunea [Mașini virtuale](#) din pagina de resurse.



Vă reamintim să parcurgeți atât secțiunea de [depunctări](#) cât și [regulamentul de realizare a temelor](#).

FAQ

- **Q:** Este permisă utilizarea variabilelor globale?
 - **A:** Da
- **Q:** Este necesară parcurgerea laboratorului 8 pentru rezolvarea temei?
 - **A:** Nu, tema se poate rezolva doar cu materia din laboratoarelor 5,6,7, dar nu este depunctată utilizarea noțiunilor din laboratoarele următoare

Resurse

Scheletul și checker-ul sunt disponibile pe [repository-ul de IOCLA](#).

iocla/teme/tema-2.txt · Last modified: 2022/05/06 11:40 by razvan.virtan

Old revisions

[Media Manager](#) [Back to top](#)