Proiecte Laborator PLA
2019 - 2020

Următoarele proiecte sunt pentru studenții din grupele:

- 30211
- 30212, semigrupa 1

Fiecare student alege unul din proiectele de mai jos, pe care sa îl realizeze **individual (grup de 2 persoane este posibil doar pentru cele mai dificile - proiectul 11 – se va discuta înainte partea realizata de fiecare)** in cadrul orelor de laborator si eventual acasă.

Termen limită pentru alegere proiect: 28.04.2020.

Termen limită pentru finalizare si prezentare proiect: 26.05.2020

1. BIBLIOTECĂ DE FUNCŢII DE CRIPTARE/DECRIPTARE PENTRU MEMORII

Cerinţe:

Să se realizeze o bibliotecă de funcţii pentru criptarea si decriptarea datelor stocate într-o memorie. Memoria de date este de 4 KB, respectiv 1024 linii a cate 32 bit fiecare (mărimea informației este de 32 bit).

Algoritm criptare: fiecare linie de date este criptata cu o cheie de criptare. Se va lua cate un bloc de 32 de biti (4 octeti) si se va aplica un XOR cu cheia de criptare (tot 32 bit).
Algoritm decriptare: aplicarea pe datele criptare a unui XOR cu aceeași cheie de criptare.

Cheile de criptare se reţin într-o memorie adiţionala, atașata memoriei de date. Fiecare cheie de criptare poate fi folosita pentru 32 de linii de date. Astfel, vor fi 32 de chei de criptare.

Sa se scrie un program demonstrativ care realizează criptarea/decriptarea unei memorii de date. Se va scrie o procedura pentru criptare si alta pentru decriptare. Se va afișa pe ecran operația de criptare/decriptare: linia de date originala, cheia de criptare si datele criptate.

Datele inițiale din memorie si cheile de criptare vor fi generate aleator la începutul executării programului si se vor memora in fișiere (unul pentru memoria de date si unul pentru cheile de criptare). La final, conținutul memoriei de date va fi salvat in fișier (diferit fata de cele inițial).

2. BIBLIOTECĂ DE FUNCȚII DE COMPRIMARE/DECOMPRIMARE PENTRU FIȘIERE TEXT

Cerinte:

A. Sa se realizeze o biblioteca de funcții pentru comprimarea si decomprimarea fișierelor text (minim 10 cuvinte). Sa se implementeze aceste funcții pentru algoritmul de comprimare explicat mai jos.

Algoritm de comprimare bazat pe dicționar Lempel-Ziv-Welch:

1. Se folosește un dicționar inițial in care fiecare litera A-Z primește un cod binar de 5 biți. De ex. A – 00001, B – 00010, C – 00011, etc. Se poate folosi caracterul # pentru sfârșit de fișier, iar codul atribuit sa fie 00000. Nu se tine cont de litere mari sau mici, astfel, înainte de compresie, textul trebuie transformat in litere mici sau mari (după preferință). De asemenea, spatiile intre litere/cuvinte nu se iau in considerare.
2. Comprimare: Se ia primul si al doilea caracter din text. Se afișează/scrie codul binar pentru primul caracter si se verifica daca cele 2 caractere unite ca *string* apar in dicționar. Daca nu apar, se extinde dicționarul cu un noua înregistrare (<litere, cod>) si se merge la următorul caracter. Daca exista in dicționar aceste doua caractere, se ia si al treilea caracter si se verifica iar daca exista in dicționar acest *string* din 3 caractere. Procedeul se repeta pentru fiecare caracter din text pana când se ajunge la caracterul de final #. Atenție: 5 biți ajung pentru 32 de înregistrări in tabel; se folosesc 6 biți doar daca se depășește aceasta valoare.
3. Decomprimare: Se cunoaște dicționarul inițial, iar cel extins se reconstruiește pe durata procesului de decomprimare. Se ia un cod de 5 sau 6 biti, se verifica daca exista in dicționar si se afișează caracterul sau caracterele corespunzătoare. De asemenea, se reconstruiește dicționarul extins prin crearea unei noi înregistrări bazata pe presupunere. La final, dicționarul extins este recreat, iar textul decomprimat este afișat.

Observație: pentru o mai buna înțelegere a algoritmului, folosiți bibliografia de mai jos.

Exemplu:
Text: TOBEORNOTTOBEORTOBEORNOT#
Text criptat: 10100 01111 00010 00101 01111 10010 001110 001111 010100 011011 011101 011111 100100 011110 100000 100010 000000.

B. Sa se scrie un program demonstrativ care realizează comprimarea/decomprimarea unui fișier text. Utilizatorului i se va cere sa furnizeze de la tastatura calea absoluta către fișier si se vor crea proceduri pentru comprimare, respectiv decomprimare.

Bibliografie:
https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch#Example

3. SIMULARE SISTEM DE MANAGEMENT UTILIZATORI

Cerinte:

Sa se scrie un program care simuleaza un sistem de management utilizatori de tip consola. Se va implementa o baza de date de utilizatori rudimentara care este incarcata in memorie la inceputul executiei programului dintr-un fisier text si la sfarsitul executiei este salvata inapoi in acelasi fisier. Atributele unui utilizator sunt: username, parola, nume si prenume, ID unic, email. Fiecare ID este unic (6 octeti, cod alfanumeric sau hexazecimal) pe durata executiei programului si va fi generat aleator la inceputul executei programului. Parola se va retine in fisierul text in forma criptata (se va folosi instructiunea XOR pentru a cripta parola cu o cheie universala, cheie definita in codul sursa). De asemenea, pe ecran/consola, nu se vor afisa parolele in forma lor text, ci se vor afisa caractere „*”. In timpul executiei programului, baza de date de utilizatori poate fi schimbata. Pot fi adaugati utilizatori noi si pot fi stersi utilizatori existenti.

Utilizatorul programului trebuie sa se logheze cu un username si o parola valabile din baza de date, apoi poate folosi comenzi de consola pentru a vedea lista de utilizatori (**list**), pentru a adauga (**add**), sterge din lista (**del**) si (**find**) pentru a cauta dupa un anumit criteriu (nume, user, email). Comanda **exit** va termina executia.

Se vor prevedea si cazurile in care username si parola nu sunt corecte la logare, sau alte cazuri de eroare. Programul va fi modularizat, iar procedurile vor fi puse intr-o biblioteca.

Exemplu:

> username: user1
> parola: ****
> Logare reusita: Ana Georgescu
> **list**
> Ana Georgescu / **** / 5def6k / em1@mail.com
Dan Pop / **** / 45rg1o / em2@mail.com
Maria Belea / **** / ef782ax / em3@mail.com
>**add**
> username: gigi
> parola: ****
> nume si prenume: George Aron
> email: em4@mail.com
> adaugare reusita
> **list**
> Ana Georgescu / **** / 5def6k / em1@mail.com
Dan Pop / **** / 45rg1o / em2@mail.com
Maria Belea / **** / ef782ax / em3@mail.com
George Aron / **** / cbhs22 / em4@mail.com
> **del** danp
> s-a sters username danp
> **list**
> Ana Georgescu / **** / 5def6k / em1@mail.com
Maria Belea / **** / ef782ax / em3@mail.com
George Aron / **** / cbhs22 / em4@mail.com
>**exit**

4. SIMULARE SISTEM DE PLATA RESTAURANT

Cerințe:

Sa se scrie un program care simulează un sistem de plată rudimentar într-un restaurant. Meniul restaurantului se reține într-un fișier text, care este încărcat la începutul execuției programului. Fiecare produs are un stoc limitat și un preț. Clientul își alege apoi produsul dorit și cantitatea, după care se înregistrează plata în sistem. Apoi, clientul poate să facă o altă comandă sau să iasă din program. Înainte de închiderea aplicației, se înregistrează în sistem suma totală de plată și se reține într-un fișier text. Programul va fi modularizat, procedurile folosite vor fi puse într-o bibliotecă.

Opțional: Se poate folosi și biblioteca ***canvas.dll*** din arhiva ***canvas_example.zip*** pentru a crea o interfață grafică a programului.

Principiul de funcționare a bibliotecii ***canvas.dll*** este următorul: fiecare program va furniza o matrice de pixeli (un pixel ocupă 1 DWORD) și un CALLBACK (o funcție ce va fi apelată de către bibliotecă) care re-desenează imaginea. Biblioteca Canvas.dll va desena fereastra principală și va apela CALLBACK-ul primit la apăsarea fiecărei taste. Un exemplu de utilizare găsim în exemplu.asm. Programul scrie un text în fereastră și re-desenează background-ul la fiecare click de mouse, cu o culoare în funcție de poziția mouse-ului în fereastră.

Link canvas_example.zip

Exemplu:

Meniu restaurant:
1. Orez – 100
2. Burger – 200
3. Supa – 50
4. Legume – 70
Alegeti produsul si cantitatea:
1 2

Produs ales: Orez
Cantitate: 2
Pret total: 200

1.Alta comanda
2.Iesire
1

Meniu restaurant:
1. Orez – 100
2. Burger – 200
3. Supa – 50
4. Legume – 70
Alegeti produsul si cantitatea:
1 3

Ne pare rau, nu mai avem acest produs.

1.Alta comanda
2.Iesire
1

Meniu restaurant:
1. Orez – 100
2. Burger – 200
3. Supa – 50
4. Legume – 70
Alegeti produsul si cantitatea:
2 1

Produs ales: Burger
Cantitate: 1
Pret total: 200

1.Alta comanda
2.Iesire
2

Pret final: 400

5. ALGORITMUL LUI DIJKSTRA

Cerințe:

Sa se scrie un program care simulează algoritmul lui Dijkstra: găsirea drumului de cost minim între două noduri dintr-un graf orientat, în care fiecare muchie are un anumit cost. Datele de intrare sunt date fie de la tastatură, fie se citesc dintr-un fișier text: numărul de noduri din graf și muchiile existente și costul lor. Se va introduce un nod de start și un nod final, iar programul va afișa toate drumurile posibile, respectiv drumul de cost minim. Programul va fi modularizat, procedurile vor fi puse într-o bibliotecă.
Opțional: Se poate folosi și biblioteca ***canvas.dll*** din arhiva ***canvas_example.zip*** pentru a crea o interfață grafică a programului.

Principiul de funcționare a bibliotecii ***canvas.dll*** este următorul: fiecare program va furniza o matrice de pixeli (un pixel ocupă 1 DWORD) și un CALLBACK (o funcție ce va fi apelată de către bibliotecă) care re-desenează imaginea. Biblioteca Canvas.dll va desena fereastra principală și va apela CALLBACK-ul primit la apăsarea fiecărei taste. Un exemplu de utilizare găsim în exemplu.asm. Programul scrie un text în fereastră și re-desenează background-ul la fiecare click de mouse, cu o culoare în funcție de poziția mouse-ului în fereastră.

Link canvas_example.zip

Exemplu:

Noduri: 4
Nume noduri: a b c d
Muchii:
a b 7
a c 9
b c 10
b d 15
c d 11

Nod de start: a
Nod final: d

Ruta 1: a b d 22
Ruta 2: a c d 20
Ruta 3: a b c d 28
Ruta 4: a c b d 34

Drum de cost minim: a c d 20

6. ALGORITMUL LUI WEASEL

Cerințe:

Sa se scrie un program care simulează algoritmul lui Weasel: pornind de la un string de 29 de caractere complet aleatoare, se generează mai multe copii ale stringului initial, pe care apoi se realizează mutații (unele caractere se schimbă cu altul aleator, altele nu), apoi se compară cu un string țintă și se calculează câte caractere din string sunt corecte și în poziția corectă (se acordă un punctaj/scor). Dacă se obține stringul țintă (scor maxim, 29 de puncte), programul se oprește. Altfel, se reia generarea de stringuri, dar de data asta se folosesc acele stringuri care au obținut cele mai mari punctaje. Se folosesc doar literele mari (uppercase) și spațiul. Programul va fi modularizat, procedurile vor fi puse într-o bibliotecă.

Opțional: Se poate folosi și biblioteca ***canvas.dll*** din arhiva ***canvas_example.zip*** pentru a crea o interfață grafică a programului.

Principiul de funcționare a bibliotecii ***canvas.dll*** este următorul: fiecare program va furniza o matrice de pixeli (un pixel ocupă 1 DWORD) și un CALLBACK (o funcție ce va fi apelată de către bibliotecă) care re-desenează imaginea. Biblioteca Canvas.dll va desena fereastra principală și va apela CALLBACK-ul primit la apăsarea fiecărei taste. Un exemplu de utilizare găsim în exemplu.asm. Programul scrie un text în fereastră și re-desenează background-ul la fiecare click de mouse, cu o culoare în funcție de poziția mouse-ului în fereastră.

Link canvas_example.zip

Exemplu:

Fraza tinta: METHINKS IT IS LIKE A WEASEL

Generatia 01: WDLTMNLT DTJBKWIRZREZLMQCO P
Generatia 02: WDLTMNLT DTJBSWIRZREZLMQCO P
Generatia 10: MDLDMNLS ITJISWHRZREZ MECS P
Generatia 20: MELDINLS IT ISWPRKE Z WECSEL
Generatia 30: METHINGS IT ISWLIKE B WECSEL
Generatia 40: METHINKS IT IS LIKE I WEASEL
Generatia 43: METHINKS IT IS LIKE A WEASEL

Bibliografie: https://en.wikipedia.org/wiki/Weasel_program#Example_algorithm

7. SIMULARE CONSOLĂ

Cerințe:

Sa se scrie un program care simuleaza o consola care recunoaste si executa 3 comenzi:

- **ls** (listare director curent)
- **grep** (cautare in directorul curent)
- **cp** (copier fisiere).

1. Comanda **ls** nu are parametri si afiseaza structura directorului curent. Pentru fiecare element se precizeaza daca e director sau fisier si sunt afisate informatii de genul data ultimei modificari sau atributele. Se poate lua ca exemplu comanda **dir** din consola Windows, respectiv comanda **ls** din Linux.

2. Comanda **grep** are un parametru, si anume textul cautat (ex: grep proiect). Vor fi afisate numele tuturor fisierelor aflate in directorul curent in interiorul carora s-a gasit textul precizat ca parametru.

3. Comanda **cp** are 2 parametri calea spre fisierul care va fi copiat si calea spre destinatia unde va fi copiat fisierul (ex: cp f1.txt f2.txt).

Programul va fi modularizat, iar procedurile vor fi puse intr-o biblioteca.

Exemplu:

>**grep** proiect
>p1.txt
p2.txt

8. GENERAREA SI VERIFICAREA CODURILOR HAMMING

Cerinte:

Sa se scrie un program care simuleaza o memorie de calculator in care sunt retinute date pe 32 bit. Valorile din memorie fie se citesc din fisiere de intrare, fie sunt generate aleator. Dimensiunea memoriei de date este de 4 KB, adica 1024 locatii de memorie a cate 32 bit fiecare. Se vor genera codurile Hamming pentru fiecare locatie de memorie si se vor retine intr-o memorie separata.

A. Sa se scrie o procedura de generare a codului Hamming.

B. Sa se scrie o procedura de verificare a codului Hamming. In cazul in care codul Hamming este gresit, se va calcula codul Hamming corect si se va indica unde este eroarea.

Realizati un program ce foloseste procedurile de mai sus, acestea fiind incluse intr-o biblioteca. Rezultatele (continutul celor 2 memorii) vor fi scrise in fisiere text si se vor afisa pe ecran.

Bibliografie:
- codurile Hamming https://en.wikipedia.org/wiki/Hamming_code
- http://users.cs.fiu.edu/~downeyt/cop3402/hamming.html
- http://www.ee.unb.ca/cgi-bin/tervo/hamming.pl?L=7&D=4&X=+Receive+&T=1001010
- http://www.enginmag.uodiyala.edu.iq/uploads/Vol.%208%20No.%203/ADHAM%202.pdf

Exemplu de cod Hamming cu date pe 4 bit:

- Generarea codului Hamming (7,4)

intrare	iesire
1011	0110011
1010	1011010

- Verificarea codului Hamming (7,4)

intrare	iesire
0110011	Corect
1011010	Corect
1001010	Incorect Correct: 1011010 Error on bit 3

9.TIC-TAC-TOE

Cerinte:

Sa se implementeze jocul Tic-Tac-Toe (denumit si „x si 0”). Jocul va afișa grila de 3x3 si va lasa jucătorii sa efectueze mutările pe rând (primul click completează căsuța corespunzătoare cu X, al doilea cu 0, al 3-lea cu X, etc.). Se poate folosi biblioteca ***canvas.dll*** din arhiva ***canvas_example.zip*** pentru a desenarea grilei, calcularea căsuței si a poziției, etc.

Principiul de funcționare a bibliotecii ***canvas.dll*** este următorul: fiecare program va furniza o matrice de pixeli (un pixel ocupă 1 DWORD) și un CALLBACK (o funcție ce va fi apelată de către bibliotecă) care re-desenează imaginea. Biblioteca Canvas.dll va desena fereastra principală și va apela CALLBACK-ul primit la apăsarea fiecărei taste. Un exemplu de utilizare găsim în exemplu.asm. Programul scrie un text în fereastră și re-desenează background-ul la fiecare click de mouse, cu o culoare în funcție de poziția mouse-ului în fereastră.

Link canvas_example.zip

10. WINDOWS CALCULATOR

Cerinte:

Să se redea în mod cât mai fidel, interfața aplicației Calculator (varianta simplă), din Windows. Se va lucra doar cu numere întregi. Se poate folosi biblioteca ***canvas.dll*** din arhiva ***canvas_example.zip*** pentru a desenarea interfeței, calcularea poziției, etc.

Principiul de funcționare a bibliotecii ***canvas.dll*** este următorul: fiecare program va furniza o matrice de pixeli (un pixel ocupă 1 DWORD) și un CALLBACK (o funcție ce va fi apelată de către bibliotecă) care re-desenează imaginea. Biblioteca Canvas.dll va desena fereastra principală și va apela CALLBACK-ul primit la apăsarea fiecărei taste. Un exemplu de utilizare găsim în exemplu.asm. Programul scrie un text în fereastră și re-desenează background-ul la fiecare click de mouse, cu o culoare în funcție de poziția mouse-ului în fereastră.

Link canvas_example.zip

11. ALTE PROIECTE

Cerinte:

Lista este deschisă si pentru alte idei de proiect. Se poate folosi biblioteca ***canvas.dll*** pentru crearea unor aplicații interesante, gen: poker, sudoku, on-screen keyboard, 2048, tetris, memory blocks, etc.

Link canvas_example.zip

A. 2048

Se va realiza o implementare a jocului 2048 (https://gabrielecirulli.github.io/2048/). În loc de săgeți, se va face considera poziția mouse-ului în momentul când s-a făcut click, față de grila de joc.

B. Poker

Se va implementa un joc de poker clasic, cu 5 carti de joc si un singur jucator. Se vor afisa 5 carti aleatoare, iar programul va evidentia (text sau vizual) ce combinatie s-a realizat cu cele 5 carti de joc: 1 pereche, 2 perechi, 3 bucati, quinta, culoare, full house, careu, quinta royala mica sau mare.

C. Sudoku

Se va implementa o interfață simplă de joc Sudoku. În grila de 9x9, cifrele completate inițial se vor colora diferit, iar pentru restul pătrățelelor, prin click, se va modifica cifra conținută, în următoarea cifră validă (de exemplu dacă pătrățelul conține cifra 5, iar cifra 6 nu este validă (există deja pe linie, coloană sau careu), se va trece direct la cifra 7). După cifra 9 urmează pătrățelul alb.

D. On-Screen keyboard

Se va implementa o tastatură virtuală, care să conțină cel puțin litere, cifre, respectiv tastele Backspace și Enter. Pe măsură ce utilizatorul tastează, vor apărea litere pe ecran. La apăsarea tastei Enter, se trece la rândul următor. La apăsarea tastei Backspace, se șterge ultimul caracter apăsat.

E. Tetris

Să se implementeze un joc similar cu tetris. Vor apărea piese în partea de sus a ecranului. Un click în stânga sau în dreapta suprafeței de joc va muta piesa în stânga sau în dreapta cu o poziție, iar un click pe suprafața de joc o va roti.

F. Memory blocks

Se vor desena pe suprafața de joc mai multe cărți întoarse cu fața în jos, în perechi, amestecate. Jucătorul poate face click pe oricare carte, pentru a o întoarce. Atunci când s-au întors două cărți, se așteaptă o secundă, apoi, dacă conțin aceeași imagine vor dispărea, altfel se întorc din nou cu fața în jos.

G. Minesweeper

Se va implementa jocul de Minesweeper, varianta simplificată. Pe o grilă de dimensiuni 10x10 (se pot folosi și alte dimensiuni), se vor pune 10 mine, care sunt ascunse și trebuie găsite de jucător. Minele se ascund pe poziții aleatoare.

H. 15 puzzle

Se va implementa jocul de 15 puzzle. Pe o grilă de dimensiuni 4x4, se vor pune numerele de la 1 la 15 în ordine aleatoare pe grila, iar ultima căsuță rămâne liberă. Jucătorul trebuie să așeze numerele în ordine crescătoare prin mutarea numerelor pe tabla de joc. Link https://en.wikipedia.org/wiki/15_puzzle

I. 9 MEN MORRIS (JOC DE MOARA/TINTAR)

Se va implementa jocul de 9 MEN MORRIS (jocul de moara/tintar). Se va desena suprafata de joc (3 patrate concentrice si conectate pe mijlocul laturilor, avand in total 24 pozitii posibile) pe care cei 2 jucatori pot plasa cele 9 piese (18 in total). Se vor implementa regulile de joc clasice.