Računarska elektronika

Projektni izveštaj

Grupa 17, projekat broj 8

Tema projekta: Šifrovanje i dešifrovanje primenom *Row Transposition* algoritma

Projekat radili:

Sandra Stanojević 218/12
Miloš Vojinović 415/12

Projektni zadatak:

Napisati program kojim se učitava ulazni fajl koji u prvom redu sadrži

mode ∈ {e, d} - karakter koji definiše da li tekst koji se nalazi u fajlu treba šifrovati (e) ili dešifrovati (d) primenom Row Transposition algoritma.
n - broj koji definiše koliko puta se prilikom šifrovanja ponavlja algoritam koji su razdvojeni jednim znakom razmaka, u drugom redu sadrži ključ koji ima maksimalno 9 cifara i specificira redosled iščitavanja kolona prilikom formiranja šifrovane poruke, a od trećeg reda na dalje nalazi se tekst koji je potrebno obraditi.

Kao rezultat izvršavanja programa, potrebno je u izlazni fajl, prema formatu specificiranom za ulazni fajl upisati mode, parametar n i ključ, kao i rezultat obrade ulaznog teksta (parametar n i ključ su isti kao i u ulaznom fajlu, parametar mode je suprotan od onog koji je definisan u ulaznom fajlu).

U .data sekciji programa definisane su sledeće promenljive:

```
buffer BYTE BUFFER_SIZE DUP(?)
; ulazni bafer sa neinicijalizovanim podacima
buffer1 BYTE BUFFER_SIZE DUP(?)
; bafer koji ima isti sadržaj kao ulazni bafer, ali tekst za šifrovanje odnosno
dešifrovanje nema razmake ni entere
buffer2 BYTE BUFFER_SIZE2 DUP (?)
; bafer koji sadrži kriptovan tekst: sve iteracije sa zaglavljima
buffer3 BYTE BUFFER_SIZE2 DUP (?)
; dekriptovan tekst: sve iteracije
filename BYTE 50 DUP(0)
; prihvata naziv ulaznog fajla koji unosi korisnik
filenameout BYTE "output.txt", 0
; predefinisani naziv izlaznog fajla koji će se kreirati
bytesWritten DWORD ?
; informacija o broju karaktera upisanih u izlazni fajl
stringLength DWORD ?
; dužina stringa
fileHandle  HANDLE ?
; pokazivač na fajl za proveru regularnosti učitavanja fajla
str2 BYTE "Bytes written to file [output.txt]: ",0
; poruka koja će se prikazati na ekranu po izvršenju programa
n DWORD 00000000h
; učitani bajtovi od kojih svaki predstavlja ASCII reprezentaciju cifre broja
iteracija
n_bajtova BYTE 0
; koliko cifara ima broj iteracija
k_length BYTE 00h
; duzina kljuca

broj_n DWORD 00000000h
; decimalni broj iteracija
operand1 word 1000
; operand kojim množimo cifru hiljada iz broja iteracija, a ima vrednost 1000
operand2 BYTE 100
```

```
; operand kojim množimo cifru stotina iz broja iteracija, a ima vrednost 100
operand3 BYTE 10
; operand kojim množimo cifru desetica iz broja iteracija, a ima vrednost 10
duzina WORD 00000000h
; broj karaktera koje sadrži tekst koji treba šifrovati odnosno dešifrovati, ali bez
razmaka i entera
podaci BYTE 05h
; u ovu promenljivu će biti upisan broj elemenata zaglavlja, a ima inicijalnu vrednost
5 koja predstavlja broj bajtova kojima se predstavljaju razmak i dva entera
offbuff DWORD ?
; odakle pocinje novi upis zagavlja i obrađenog teksta
offbuff_prev DWORD ?
; odakle se cita tekst koji se obrađuju i podaci o njegovom zaglavlju
offbuff3 DWORD ?
;konstantan za iteraciju i cuva adresu pocetka kljuca
offbuff4 DWORD ?
;krece od prvog slova ali se uvecava sa svakim skokom jer se odatle skače dalje
ukupna_duzina1 WORD 0001h
; dužina kompletnog ulaznog fajla
numit WORD 0000h
; trenutna iteracija koja se odradjuje
key_digit BYTE 31h
; u ovu promenljivu se smesta cifra kljuca koju trazimo
digit BYTE 00h ; kaze koji kljuc trazimo, ali -1 sluzi da govori koliko kljuceva pre
toga trazimo i koliko puta skacemo
x BYTE 00h
; iz kog puta nalazimo cifru u kljucu
q BYTE 00h
; količnik pri deljenju broja karaktera u tekstu sa dužinom ključa
q1 BYTE 00h
; q uvećan za 1
r BYTE 00h
; ostatak pri deljenju broja karaktera u tekstu sa dužinom ključa
it BYTE 01h
; pomeraj od prve cifre kljuca da bismo dosli na tekucu cifru kljuca / 1, 2 ili 3 za
k_length=4
i BYTE 00h
; koji karakter u bloku uzimamo
brojac WORD 0000h; uveca se kad upisemo jedan karakter i poredi se sa "duzinom"
```

Iz biblioteke *Irvine32.inc* koristimo funkcije: ReadFromFile, WriteToFile, ReadString, OpenInputFile, CreateOutputFile, CloseFile.

Koncept realizacije:

- napomena: sve podebljane reči u daljem tekstu se odnose na originalne nazive promenljivih u kodu.

Ulazni tekstualni fajl se učita u **buffer**. Pod *zaglavljem* ulaznog fajla se podrazumeva svaki karakter koji se nalazi u fajlu do početka teksta za kriptovanje-dekriptovanje:
e(d) razmak n enter ključ enter

Pritom se podrazumeva da je format fajla ispoštovan (detaljnije opisano u prilogu). Učitan fajl je potrebno 'očistiti' od znakova razmaka (ASCII 20h) i entera, odnosno prelazaka u novi red (ASCII 0Dh 0Ah). Prolazi se kroz ulazni bafer i vrši se izbacivanje razmaka i entera*. Ovako sređen tekst se upisuje u **buffer1**, za koji je definisana ista maksimalna veličina kao i za buffer.
Pre samog šifrovanja, vrše se dodatne pripreme, kao što je:

- prebrojavanje karaktera koji čine ključ (definisanje dužine ključa **k_length**)
- upisivanje karaktera koji čine broj iteracija u promenljivu **n**
- određivanje koliko cifara ima n** (**broj_n**)
- nalaženje ukupnog broja karaktera 'očišćenog teksta' (**duzina**)
- nalaženje decimalne vrednosti broja iteracija, **broj_n**

*Zaglavlje zadržava svoj početni oblik.
**Radi jednostavnije manipulacije nad karakterima koji čine broj iteracija n, za šta je preduslov mogućnost smeštanja ovih karaktera u 32-bitni registar, usvojeno je gornje ograničenje broja iteracija na maksimalni četvorocifreni broj, tj. 9999.

Neka je npr, broj iteracija 32. Ovaj broj je smešten u registar na sledeći način 00 00 32 33h. Da bi se dobio decimalni broj iteracija, mora se definisati *'case study'* za slučaj jednocifrenog, dvocifrenog, trocifrenog i četvorocifrenog broja. Pošto je ASCII kod za nulu 30h, onda se decimalna vrednost dobija oduzimanjem 30h od unetih cifara. (00 00 02 03h).
Svaka obrada sadrži poziv odgovarajuće procedure unutar koje se vrši *and*-ovanje sa maskom koja će izdvojiti bajt, odnosno cifru, šiftovati udesno za 0, 8, 16 odnosno 24 mesta i pomnožiti je sa odgovarajućom težinom (**operand1, operand2, operand3**), i sabrati je u **broj_n**, ili je samo sabrati ako je u pitanju cifra jedinica.

Dakle, na 00 00 02 03h (obrada dvocifrenog broja) će se najpre primeniti maska 000000FFh. Rezultat operacije je 00 00 00 03h. Ovaj broj se množi (neoznačeno) sa **operand3** koji ima vrednost 10dec i prebaci u **broj_n**. Zatim se na polaznu vrednost primeni maska 0000FF00h, sa rezultatom 00 00 02 00h, ovaj broj se šiftuje za 8 mesta udesno: 00 00 00 02h i sabere sa već postojećom vrednosti u promenljivoj **broj_n**. Najkompleksnija obrada je za četvorocifreni broj, dok se za jednocifreni obrada praktično svodi na operaciju mov.

Kada su podaci pripremljeni u **buffer1**, odnosno detektovan je NULL u **buffer,** skače se na labelu **ed**, koja vrši proveru prvog karaktera u **buffer1** i u zavisnosti od toga da li je e ili d skače na sekciju koda koja vrši šifrovanje, odnosno dešifrovanje, respektivno.

Logika šifrovanja se sastoji iz sledećeg: pozicioniramo se na poziciju prve cifre ključa. Ideja je da se zaustavimo najpre na cifri 1, pa 2, itd. Pretraživanje se, dakle, ponavlja **k_length** puta. Kada se nađemo u **buffer1** na poziciji cifre *i*, to praktično znači da svi karakteri unutar bafera koji se nađu na celobrojnom umnošku odstojanja **k_length** od cifre *i*, sve do kraja bafera (i tim redosledom)

predstavljaju kolonu ispod cifre ključa *i* koja se prva iščitava i upisuje u **buffer2**. Ovaj bafer je znatno veći od **buffer** i **buffer1**, jer je ideja da se on virtuelno izdeli na blokove koji će sukcesivno sadržati rezultat svake naredne iteracije (dubine šifrovanja).
Postoje tri problema koja se javljaju u ovoj realizaciji. Prvi je da *prvi* skok koji pravimo od utvrđene pozicije cifre ključa nije dužine **k_length** nego **k_length +** 2, zbog postojanja enter-a između poslednje cifre ključa i prvog karaktera teksta. Ova dva slučaja su se morala razdvojiti. Drugi problem se tiče jasnog definisanja bafera (segmenta bafera) iz kojeg čitamo, jer je on drugačiji za svaku narednu iteraciju. Prilikom prve iteracije, to je **buffer1,** a u svakoj narednoj iteraciji to je prethodni blok **buffer2** u odnosu na tekući u koji upisujemo. Informacija se čuva u **offbuff_prev**, koji sadrži adresu (pokazuje na) početni karakter prethodnog rezultata šifrovanja. (Napomena: svaka iteracija se tretira kao finalna, u smislu da joj se dodaje zaglavlje i menja karakter e u d i obrnuto). Ista logika će biti primenjena na dešifrovanje. Treći problem se tiče premašenja opsega bafera/bloka u kojem se nalazimo. Tu nam pomaže informacija o ukupnom broju karaktera u **buffer1**, do koje se lako dolazi sabiranjem podataka koje smo prethodno izvukli (**duzina, podaci** (inicijalna vrednost 5: dva entera i razmak), **k_length, n_bajtova**). Promenljiva **duzina** je inicijalizovana na vrednost 1, zbog karaktera e. Ideja je da svaki put kad 'skočimo' proverimo da li smo ispali iz opsega: od trenutne pozicije koja je ili u opsegu bafera iz kojeg čitamo ili je ispala van tog opsega oduzmemo početnu poziciju bafera iz kog čitamo. Ukoliko je razlika te dve pozicije u memoriji veća od ili jednaka od ukupne dužine fajla (sa zaglavljem i tekstom bez razmaka i entera), onda smo ispali iz opsega, pa se trenutna pozicija zanemaruje i prelazi se na obradu nove cifre ključa. U **key_digit** se smešta trenutna cifra ključa koju pretražujemo (ASCII predstava). U slučaju da smo došli do trenutka kad je i poslednja cifra obrađena, u **key_digit** će se naći ASCII karakter za 2 veći od poslednje cifre ključa (cifra ključa je ekvivalentna dužini ključa).
Neka je ključ 1324.
Nakon pretraživanja, bez obzira na ishod, **key_digit** se uveća za 1. U pretposlednjoj iteraciji, kada smo stigli do poslednje cifre u ključu (**key_digit =** 34h), kad se nađe cifra, uvećaće se **key_digit** za 1 (**key_digit** = 35h). U poslednjoj iteraciji, neće se pronaći cifra, ali će se opet **key_digit** uvećati za 1 (**key_digit =** 36h). Stoga se u poređenju koje dovodi do iskakanja na proveru broja iteracija sabira dužina ključa sa 32h, odnosno 36h se poredi sa 4+32h. Provera da li smo obradili poslednju iteraciju se vrši u okviru labele **provera_n**, gde se brojač iteracija **numit** nakon svake izvršene iteracije (čija je posledica skok na tu labelu) uvećava za 1 i poredi sa ukupnim brojem iteracija, odnosno **broj_n**.

Po završenom šifrovanju, ide se na labelu output koja koristi **offbuff** (poziciju početka poslednje iteracije) i vrši ispis u izlazni file output.txt.

racunarska elektronika, 213, primer logike:

2
1
3
r (8)
a (1)
c (15)
u (9)
n (2)
a (16)
r (10)
s (3)
k (17)
a (11)
e (4)
l (18)
e (12)
k (5)
t (19)
r (13)
o (6)
n (20)
i (14)
k (7)
a (21)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
a	n	s	e	k	o	k	r	u	r	a	e	r	i	c	a	k	l	t	n	a

Logika dešifrovanja najpre počinje labelom **deljenje**, koja priprema potrebne parametre za dešifrovanje. Potrebno je dužinu teksta podeliti sa dužinom ključa i sačuvati u količnik **q** i ostatak **r**. Oblast bafera koji sadrži tekst za dešifrovanje je izdeljen na virtuelne blokove, od kojih su neki dužine **q**, a neki za jedan karakter duži. Stoga je bilo zgodno pripremiti i promenljivu **q1** u koju će biti smešten količnik uvećan za jedan. Deo koji se tiče osnovne inicijalizacije i određivanja podataka koji su od interesa je isti kao u sekciji za šifrovanje.
Postupak će biti objašnjen na primeru:

ključ 4123, broj iteracija 1, tekst cretieaaakrjuslrkrnkeaoa (jedanput šifrovan tekst *racunarskaelektronikaje*)

Ako se tekst rasporedi na način na koji je to urađeno u memoriji:
c
r
e
t
i
e
// virtuelni blok 1
a
a
a
k
r
j
// virtuelni blok 2
u
s
l
r
k
// virtuelni blok 3
r
n
k
e
o
a
// virtuelni blok 4

Budući da 23/4 daje **r=**3 i **q**=5 (odnosno **q1=**6), to znači da će virtuelni blokovi koji odgovaraju cifri ključa 4, 2 i 1 (prve tri u ključu) imati 6 karaktera, a preostali blok 5. Blokovi su zapravo kolone za iščitavanje kao u tabeli:

4	2	1	3
r	a	c	u
n	a	r	s
k	a	e	l
e	k	t	r
o	n	i	k
a	j	e	

To znači da najpre čitamo prvo slovo kolone 4 (virtuelnog bloka 4), pa prvo slovo kolone 2 (virtuelnog bloka 2), pa prvo slovo kolone 1 (virtuelnog bloka 1) itd.
Ključan momenat se tiče definisanja koji blok sadrži 5 a koji 6 karaktera. Upišemo cifru ključa u ASCII reprezentaciji u **digit**. Neka je to npr 34h. Zatim oduzmemo 31h, i dobijemo 3. Ovaj broj predstavlja koliko puta treba da izvršimo skok da bismo se pozicionirali na traženom bloku (u ovom slučaju blok4). Pretražujemo sve cifre manje od 4 unutar ključa. To činimo tako što svaki put umanjujemo **digit** za 1. Definiše se promenljiva **x** koja govori iz kog puta smo prolazeći kroz ključ, karakter po karakter, identifikovali trenutnu cifru. To nam daje informaciju o tome koja je pozicija te cifre u ključu. Kada to znamo, poredeći **x** i **r** dobijamo informaciju o tome da li dati blok ima 5 ili 6 karaktera, zato što samo blokovi koji odgovaraju prvih **r** cifara ključa imaju 6 karaktera.
Čitanje karaktera na bilo kojoj narednoj poziciji u bloku se svodi na čitanje prve pozicije uz dodatak da se na kraju pomerimo još za **i** mesta od prvog karaktera u pravom bloku. Svaki put kad tražimo nov karakter polazimo od prvog slova. Postupak se zaustavlja kada promenljiva **brojac**, koja se inkrementira za svaki pronađeni karakter dostigne vrednost **duzina** (broj karaktera teksta). Tada se prelazi na sekciju koda **output** koja vrši ispis u izlazni fajl.
Napomena: za formiranje izlaznog fajla koristi se **buffer3**, koji je takođe znatno veći jer se primenjuje ista logika smeštanja sukcesivnih rezultata iteracija kao u slučaju šifrovanja.
Takođe se javlja slučaj razdvajanja situacije kada se izvršava prva iteracija, gde se referenciramo na **buffer1**, i svake naredne gde čitamo iz prethodnog bloka u **buffer3**.

ZAKLJUČAK

Tokom ovog projekta, uočili smo:

- Za razliku od viših programskih jezika, pisanje u mašinskom jeziku daje POTPUNU kontrolu nad memorijom.
- Ako bi neko bio jako vešt u pisanju programa na mašinskom jeziku, mogao bi da piše optimalne algoritme sa stanovišta broja instrukcija i memorijske kontrole.
- Istini za volju, koliko god dobra bila preciznost i detaljnost koju pisanje u mašinskom jeziku zahteva, programiranje je veoma zahtevno
- Kako viši programski jezici nude fleksibilnost apstrahovanja toga šta se dešava u memoriji računara (u najdetaljnijem smislu, naravno), relativno velike projekte ćemo pisati u višim programskim jezicima, za koje smo uspostavili precizne mehanizme prevođenja u niži programski jezik, koji ćemo takođe težiti da napišemo na optimalan način sa stanovišta broja instrukcija i memorijskog zauzeća.
- Kao inženjeri elektronike, jasno možemo da uočimo značaj nižih programskih jezika, zbog naše uske povezanosti sa hardverom, tako da ostajemo pod utiskom da ćemo možda nekad za neke specijalizovane sisteme, relativno male složenosti, čija brzina i robusnost moraju biti optimalni, koristiti mašinski oblik programiranja.

Dodatne napomene:

Priložen je primer ulaznog fajla t.txt, koji sadrzi dekripciju poruke koja je sifrovana 5 puta sa kljucem specificiranim u fajlu. Originalna poruka je glasila: *istorija je uciteljica zivota*, sto se moze videti u prilozenom output fajlu. Za proveru enkripcije, potrebno je u t.txt umesto d napisati e, kao i originalnu poruku, a kao rezultat ce se dobiti poruka iz originalnog t.txt fajla: ziuilijcsjvaitiojtocteeraa

Kod:

```
; This project performs encryption or decryption of the message from the input file using
; the Row transposition algorithm. Input .txt file is read into a buffer, and then processed.
; User can define whether the message should be encrypted or decrypted and how many times.
; A specific 1-9 digit key is used to alter the characters' positions within the message correspondingly.
; The result is placed in output file, along with the key, number of iterations performed and altered key word (e/d).

; begin code

INCLUDE Irvine32.inc ; this library's functions are used for the purpose reading an input file,
; communication with the user, and writing to an output file
INCLUDE macros.inc

BUFFER_SIZE = 501   ; input buffer size
BUFFER_SIZE2 = 20000 ; velicina bafera u koji se smestaju kriptovani podaci


.data
buffer BYTE BUFFER_SIZE DUP(?) ; original file
buffer1 BYTE BUFFER_SIZE DUP(?); text without space or enter with header
buffer2 BYTE BUFFER_SIZE2 DUP (?); encrypted/decrypted text, all iterations
buffer3 BYTE BUFFER_SIZE2 DUP (?)
filename BYTE 50 DUP(0) ; input file name is inserted by the user and placed here
filenameout BYTE "output.txt", 0 ; this will be the name of the output file
bytesWritten DWORD ? ; used for ouput file declaration
stringLength DWORD ?
fileHandle  HANDLE ?
str2 BYTE "Bytes written to file [output.txt]: ",0 ; displays how many characters (bytes) are
; written in the output file
n DWORD 00000000h; ASCII representations of iteration number digits are placed here in altered order
n_bajtova BYTE 0; how many digits does the iteration number have
k_length BYTE 00h; key length
broj_n DWORD 00000000h; decimal number of iterations
operand1 word 1000; used to multiply by 1000
operand2 BYTE 100; used to multiply by 100
operand3 BYTE 10; used to multiply by 10
duzina WORD 00000000h; number of character that the message contains (where space and enter are not included)
podaci BYTE 05h; beginning without e/d
offbuff DWORD ?; place where we start writing new iteration results
offbuff_prev DWORD ?; place where we start reading the current section being processed
offbuff3 DWORD ? ;remains const within an iteration and keeps the first key digit address
offbuff4 DWORD ? ;starts from the first letter of the first virtual block but jumps sequentially to
;other blocks in order for us to position on the right letter
ukupna_duzina1 WORD 0001h;complete file length (the message is cleared from space and enter)
numit WORD 0000h; current iteration used to compare with broj_n
key_digit BYTE 31h; key digit that is being searched within the key
digit BYTE 00h ; decremented by 1, it helps us determine how many keys are being searched before
;as well as how many jumps do we make before reaching the right block
x BYTE 00h; what is the number of iterations that we need to perform to locate specific digit within the key
q BYTE 00h; quotient-result of the division of duzina and k_length
q1 BYTE 00h; quotient+1 that represents the number of characters in bigger blocks
r BYTE 00h; remainder of the division mentioned above
it BYTE 01h ; how much we should move from the key's first digit in order to reach the current key digit we want
brojac WORD 0000h; we increment this variable each time a new character is found and copied to output buffer
; it also indicates whether we have reached the end of the message, when it becomes equal to duzina
```

```
i BYTE 00h ; which character in the block is processed (first, second...)

.code
; svaka od procedura obradjuje slucaj razlicitog broja cifara u n i pretvara ih u decimalni zapis.
; svaka sadrzi and-ovanje sa maskom da bi se izdvojio odgovarajuci bajt (odnosno cifra) koja se
; zatim dovodi na najnizu poziciju siftovanjem udesno za odgovarajuci broj pozicija, ukoliko je to
; potrebno, mnozi se sa odgovarajucim tezinskim faktorom: 1000, 100 odnosno 10 i sabira u promenljivoj broj_n
case1 PROC c
push eax
push edx
mov eax, n
mov broj_n, eax
pop edx
pop eax
ret
case1 ENDP

case2 PROC c
push eax
push edx
xor eax, eax
mov eax, n
and eax, 000000FFh
mul operand3
add broj_n, eax
xor eax, eax
mov eax, n
and eax, 0000FF00h
shr eax, 8
add broj_n, eax
pop edx
pop eax
ret
case2 ENDP

case3 PROC c
push eax
push edx
xor eax, eax
mov eax, n
and eax, 000000FFh
mul operand2
add broj_n, eax
xor eax, eax
mov eax, n
and eax, 0000FF00h
shr eax, 8
mul operand3
add broj_n, eax
xor eax, eax
mov eax, n
and eax, 00FF0000h
shr eax, 16
add broj_n, eax
pop edx
pop eax
ret
```

```
case3 ENDP

case4 PROC c
push eax
push edx
xor eax, eax
mov eax, n ; u eax smestamo bajtove za obradu
and eax, 000000FFh; izdvajamo poslednji bajt
mul operand1 ; mnozi se sa hiljadu
add broj_n, eax ; doda se u broj iteracija
xor eax, eax
mov eax, n
and eax, 0000FF00h ; isto za stotinu
shr eax, 8
mul operand2
add broj_n, eax
xor eax, eax
mov eax, n
and eax, 00FF0000h ; isto za deseticu
shr eax, 16
mul operand3
add broj_n, eax
xor eax, eax
mov eax, n
and eax, 4278190080 ; moralo ovako :( FF000000h
shr eax, 24
add broj_n, eax
pop edx
pop eax
ret
case4 ENDP

main PROC
; Reading a File

; Opens, reads, and displays a text file using
; procedures from Irvine32.lib.

; Let user input a filename.
	mWrite "Enter an input filename: "
	mov edx,OFFSET filename
	mov ecx,SIZEOF filename
	call  ReadString ; f-ji ReadString su potrebni odg sadrzaji r-ra edx i ecx

; Open the file for input.
	mov edx,OFFSET filename
	call  OpenInputFile
	mov  fileHandle,eax

; Check for errors.
	cmp eax,INVALID_HANDLE_VALUE	; error opening file?
	jne   file_ok				; no: skip
	mWrite <"Cannot open file",0dh,0ah>
	jmp  quit				; and quit
file_ok:

; Read the file into a buffer.
```

```
        mov edx,OFFSET buffer
        mov ecx,BUFFER_SIZE
        call ReadFromFile
        jnc check_buffer_size                   ; error reading?
        mWrite "Error reading file. "           ; yes: show error message
        call WriteWindowsMsg
        jmp close_file

check_buffer_size:
        cmp eax,BUFFER_SIZE                     ; buffer large enough?
        jb buf_size_ok                          ; yes
        mWrite <"Error: Buffer too small for the file",0dh,0ah>
        jmp quit                                ; and quit

buf_size_ok:
        mov buffer[eax],0               ; insert null terminator
        mWrite "File size: "
        call WriteDec                   ; display file size
        call Crlf

; Display the buffer.
        mWrite <"Buffer:",0dh,0ah,0dh,0ah>
        mov edx,OFFSET buffer     ; display the buffer
        call WriteString
        call Crlf

close_file:
        mov eax,fileHandle
        call CloseFile

        mov eax, OFFSET buffer
        mov edx, OFFSET buffer1

        mov ch, 20h


        mov cl, [eax] ; u cl je prvi element originalnog bafera
        mov [edx], cl ; sada taj element prebacimo u bafer1
        add eax, 2; preskacemo space u orig baferu i u eax je adresa prve cifre od n
        add edx, 1; pomerimo se na prvu slobodnu poziciju u baferu1
        mov [edx], ch; stavljamo space u baffer1
        mov ebx, OFFSET n; u ebx je adresa niza u koji upisujemo broj iteracija-pocece da upisuje od najnizeg
        sub ebx, 1; oduzmemo 1 jer u petlji vracamo 1
        input:
        add edx, 1
        add ebx, 1
        mov cl, [eax] ; u cl je prva cifra
        sub cl, 30h ; u cl je prvi broj
        mov [ebx], cl

        mov [edx], cl; u bafer 1 smestam prvu cifru od n

        add eax, 1 ; pomerimo su unutar niza za 1 mesto
        mov cl, [eax] ; u cl ubacimo sledeci karakter iz bafera
        inc n_bajtova
        cmp cl, 0Dh ; pitamo da li smo stigli do ODh
        jne input
```

```
inc edx ; pomeramo se na prvo slobodno mesto u baferu1
mov cx, 0A0Dh; upisujemo enter new line
mov [edx], cx
add edx, 2; sada tu treba da dodje kljuc
add eax, 2 ; nalazimo se u originalnom baferu na poziciji prve cifre kljuca

cmp n_bajtova, 1 ; ako je broj iteracija jednocifren, ide se na obradu
; gde se poziva ogovarajuca procedura za taj slucaj
je obrada1
cmp n_bajtova, 2 ; ako je broj iteracija dvocifren, ide se na obradu
; gde se poziva ogovarajuca procedura za taj slucaj
je obrada2
cmp n_bajtova, 3; ako je broj iteracija trocifren, ide se na obradu
;gde se poziva ogovarajuca procedura za taj slucaj
je obrada3
call case4; ako nije ni jednocifren ni dvocifren ni trocifren, onda je cetvorocifren i poziva se odgovarajuca procedura
jmp petlja
obrada1:
call case1
jmp petlja

obrada2:
call case2
jmp petlja

obrada3:
call case3
jmp petlja

petlja: ; smesta kljuc u bafer1 (tekst bez razmaka) i racuna duzinu kljuca

    mov cl, [eax] ; u cl se nalazi prva cifra kljuca
    mov [edx], cl; prvu cifru kljuca stavljamo u bafer
    add k_length, 1; uvecamo duzinu kljuca

    inc eax
    inc edx
    mov cl, [eax]
    cmp cl, 0Dh
    jne petlja

mov cx, 0A0Dh
mov [edx], cx
add edx, 2
add eax, 2; na prvom slovu teksta u baferu
; sad sledi upis teksta, izostavljaju se razmaci i enter-i, racuna se duzina cistog teksta
tekst:
mov cl, [eax]
inc eax
cmp cl, 20h
je tekst
cmp cl, 0Dh
je tekst
cmp cl, 0Ah
je tekst
cmp cl, 00h ; ako je NULL, stigli smo do kraja teksta, ide se na proveru da li je prvo slovo e ili d
je ed
```

```
mov [edx], cl
add edx, 1
add duzina, 1
jmp tekst
ed: ; proveravamo da li je prvi karakter šifrovanje ili dešifrovanje
mov edx, offset buffer1
mov cl, [edx]
cmp cl, 64h
je deljenje
sifrovanje: ;  pripremaju se odgovarajuce promenljive od informativnog znacaja
xor eax, eax
mov ukupna_duzina1, 01h; karakter e
mov podaci, 05h ; razmak i dva entera
mov al, 31h; prva cifra koja se pretrazuje
mov key_digit, al
xor eax, eax
xor edx, edx
mov al, podaci
add al, n_bajtova
add al, k_length
mov podaci, al
mov al, podaci
add ax, ukupna_duzina1 ; sracunat ukupan broj svih karaktera koji se nalaze u baferu1
;(sa originalnim zaglavljem i porukom ociscenom od razmaka i entera)
add ax, duzina
mov ukupna_duzina1, ax
mov ebx, offset buffer2
mov ax, ukupna_duzina1
mul numit
mov offbuff, edx
shl offbuff, 16
add offbuff, eax
add eax, ebx
mov offbuff, eax
push eax
mov eax, offbuff
xor ecx, ecx
mov cx, ukupna_duzina1
sub eax, ecx
mov offbuff_prev, eax
pop eax
mov edx, offbuff
mov cl, 64h
mov [edx], cl
inc edx
mov eax, offset buffer
inc eax

popunjavanje: ; popunjava se zaglavlje izlaznog bafera
mov cl, [eax]
mov [edx], cl
inc eax
inc edx
push eax
xor eax, eax
mov al, podaci
dec al
```

```
mov podaci, al
pop eax
cmp podaci, 0
jne popunjavanje
xor ecx, ecx
mov cl, k_length
kljuc:
xor eax, eax
mov cx, numit
cmp cx, 0
jne ofbuf
mov al, n_bajtova
add eax, 00000004h
add eax, offset buffer1
mov ebx, offset buffer1
mov offbuff_prev, ebx
xor ecx, ecx
mov cl, k_length
jmp compare
ofbuf: ; definise se bafer iz kojeg se cita
mov al, n_bajtova
add eax, 00000004h
add eax, offbuff_prev
xor ecx, ecx ; inicijalizuje se brojacki registar za petlju compare
mov cl, k_length
compare:; trazi se odgovarajuca cifra kljuca u kljucu
mov bl, [eax]
cmp bl, key_digit
je encryption
inc eax
loop compare

encryption:
push eax
xor eax, eax
add al, key_digit
inc al
mov key_digit, al
xor eax, eax
mov al, k_length
add al, 32h ; 32h se dodaje jer se je u finalnoj iteraciji (kad treba da iskocimo)
;key digit za dva uvecan u odnosu na najveci broj u kljucu
cmp al, key_digit
je provera_n
pop eax
xor ebx, ebx
mov bl, k_length
add eax, 2
add eax, ebx
mov bl, [eax]
mov [edx], bl
inc edx
nizanje_slova:
xor ebx, ebx
mov bl, k_length
add eax, ebx
push eax
```

```
sub eax, offbuff_prev; provera da li smo ispali iz opsega odnosno da li karakter koji smo dohvatili
; pripada baferu iz kojeg se cita
cmp ax, ukupna_duzina1
jnb kljuc
pop eax
mov bl, [eax]
mov [edx], bl
inc edx
jmp nizanje_slova

provera_n: ;svaki put kad se zavrsi iteracija, broj izvrsenih iteracija numit se inkrementira, pa se
; poredi sa trazenim brojem iteracija; ako se ovi brojevi poklapaju, iskace se iz obrade i ide se na ispis
push eax
xor eax, eax
add ax, numit
inc eax
mov numit, ax
cmp eax, broj_n
jne sifrovanje
jmp output

deljenje: ; odavde pocinje dekripcija; podeli se duzina poruke bez razmaka i entera sa duzinom kljuca i
; kolicnik se smesti u promenljivu q, definise se q1 koji je za 1 veci i r koji predstavlja ostatak deljenja
push eax
xor eax, eax
mov ax, duzina
div k_length
mov q, al
mov r, ah
inc al
mov q1, al
pop eax
desifrovanje: ; racunanje potrebnih informacija za desifrovanje
xor eax, eax
mov ukupna_duzina1, 01h
mov podaci, 05h
mov al, 31h
mov key_digit, al
xor eax, eax
xor edx, edx
mov al, podaci
add al, n_bajtova
add al, k_length
mov podaci, al
mov al, podaci
add ax, ukupna_duzina1
add ax, duzina
mov ukupna_duzina1, ax
mov ebx, offset buffer3
mov ax, ukupna_duzina1
mul numit
mov offbuff, edx
shl offbuff, 16
add offbuff, eax
add eax, ebx
mov offbuff, eax
push eax
```

```
mov eax, offbuff
xor ecx, ecx
mov cx, ukupna_duzina1
sub eax, ecx
mov offbuff_prev, eax
pop eax
mov edx, offbuff
mov cl, 65h
mov [edx], cl
inc edx
mov eax, offset buffer
inc eax

popunjavanje1: ;  svaki bafer koji ce predstavljati izlaz tekuce iteracije ce imati najpre upisano zaglavlje sa izmenjenim d u
e
mov cl, [eax]
mov [edx], cl
inc eax
inc edx
push eax
xor eax, eax
mov al, podaci
dec al
mov podaci, al
pop eax
cmp podaci, 0
jne popunjavanje1
xor ecx, ecx
mov cl, k_length
kljuc1:                          ; provera da li citamo iz buffera1 (prva iteracija) ili prethodnog segmenta buffera3 (sve
naredne)
; i postavljanje odgovarajucih vrednosti offbuff_prev
xor eax, eax
mov cx, numit
cmp cx, 0
jne ofbuf1
mov al, n_bajtova
add eax, 00000004h
add eax, offset buffer1
mov ebx, offset buffer1
mov offbuff_prev, ebx
mov offbuff3, eax
add eax, 2
xor ebx, ebx
mov bl, k_length
add eax, ebx
mov offbuff4, eax
mov eax, offbuff3
jmp velika_petlja

ofbuf1:
mov al, n_bajtova
add eax, 00000004h
add eax, offbuff_prev
mov offbuff3, eax
add eax, 2
xor ebx, ebx
```

```
mov bl, k_length
add eax, ebx
mov offbuff4, eax
mov eax, offbuff3
; eax na prvom kljucu a u ofbuf4 cuvamo poz prvog slova

velika_petlja: ; pronalazenje i smestanje jednog karaktera iz kriptovane poruke
; na naredno slobodno mesto u dekriptovanu poruku
xor ecx, ecx
mov cl, 31h
mov key_digit, cl
xor ebx, ebx
mov bl, [eax]
mov digit, bl
xor eax, eax
mov al, digit
sub al, 31h
mov digit, al
mov eax, offbuff3
add eax, 2
xor ebx, ebx
mov bl, k_length
add eax, ebx
mov offbuff4, eax

provera_digita: ; da li je digit stigao do 0
mov bl, digit
cmp bl, 00h
je smesti_slovo
sub bl, 1
mov digit, bl

setovanje_brojaca_loopa:; petlja compare1 ce se ponavljati k_length puta
; ova informacija se mora smestiti u brojacki registar
xor ecx, ecx
mov x, cl
mov cl, k_length
mov eax, offbuff3

compare1:; sluzi za identifikaciju cifre kljuca u kljucu
xor ebx, ebx
mov bl, x
add bl, 1
mov x, bl

mov bl, [eax]
cmp bl, key_digit
je decryption
add eax, 1
loop compare1

decryption:
mov eax, offbuff4
xor ecx, ecx
mov cl, key_digit
add cl, 1
mov key_digit, cl
```

```
xor ecx, ecx
mov cl, x
cmp cl, r
jg      qu
qu1:; skok za q+1 (blokovi koji se odnose na prvih r cifara kljuca)
xor ecx, ecx
mov cl, q1
add eax, ecx
mov offbuff4, eax
jmp provera_digita
qu:; skok za q (preostali blokovi)
xor ecx, ecx
mov cl, q
add eax, ecx
mov offbuff4, eax
jmp provera_digita
smesti_slovo:
xor ecx, ecx
mov cl, i ; tek ovde cemo se pomeriti
add eax, ecx
mov bl, [eax]
mov [edx], bl ; smestanje karaktera
inc edx
mov cx, brojac
inc cx
mov brojac, cx
cmp cx, duzina
jne ofbuf3
jmp provera_n1
ofbuf3:; ako nismo sva slova zavrsili
xor eax, eax
mov al, it
add eax, offbuff3
push eax
xor eax, eax
mov al, it
cmp al, k_length
jne continue
; pronasli smo sve karaktere na istoj poziciji u blokovima, idemo opet od prvog broja kljuca
xor eax, eax
mov al, i
inc al
mov i, al
xor eax, eax
inc al
mov it, al
mov eax, offbuff3
jmp velika_petlja
; jos smo u kljucu
continue:
inc al
mov it, al
mov eax, offbuff3
add eax, 2
xor ebx, ebx
mov bl, k_length
add eax, ebx
```

```
mov offbuff4, eax
pop eax
jmp velika_petlja
provera_n1: ; provera da li smo stigli do poslednje iteracije
push eax
xor eax, eax
add ax, numit
inc eax
mov numit, ax
cmp eax, broj_n
jne resetovanje
jmp output

resetovanje: ; sve relevntne promenljive se resetuju za narednu iteraciju
xor ecx, ecx
mov x, cl
mov i, cl
mov brojac, cx
inc cl
mov it, cl
mov cl, 31h
mov key_digit, cl
jmp desifrovanje

output:      ; vrsi se ispis u izlazni fajl i prikazuje se informacija o broju karaktera upisanih u fajl
; Create new text file
mov edx, offset filenameout
call CreateOutputFile
mov fileHandle, eax
;Check errors:
cmp eax, INVALID_HANDLE_VALUE
jne file_ok1
mWrite <"Ne moze se ispisati izlazni fajl", 0dh, 0ah>
jmp close_file1
file_ok1:
mov eax, fileHandle
mov edx, offbuff
xor ecx, ecx
mov cx, ukupna_duzina1
mov stringLength, ecx
mov ecx, stringLength
call WriteToFile
mov bytesWritten,eax ; save return value
call CloseFile

; Display the return value.
mov edx,OFFSET str2 ; "Bytes written"
call WriteString
mov eax,bytesWritten
call WriteDec
call Crlf

jmp quit
close_file1:
mov eax, fileHandle
call CloseFile
```

```
quit:
invoke ExitProcess, 0
main ENDP

END main
```