Y.T.U
C.S.E. DEPARTMENT

MÜMENDİSLİK MATEMATİĞİ

Araştırma No : 08
Konu: Catalan sayılarını kullanarak Stack Permutasyonu yöntemi ile bir dosyann şifrelenmesi.Algoritma, program ve çıktısı.
Hazırlayan : Chasan CHOUSE-CSE-98011021
Tarih : 21 Ocak 2002 Pazartesi

Catalan Sayıları

Bu sayılar binary seviyeye açıldığında düzgün “1” ve “0” lardan oluştuğu görülür. “1” parantez açma ve “0” da parantez kapama için simgelenirse açılan tüm parantezler bu sayılarda kapanır.

$(172)_{10} = (10101100)_2$ Parantez ile ifade edersek ” ( ) ( ) (( )) “ şeklinde olur.

Aynı şekilde,

$(1142920503054772802)_{10} = (1111110111000111100000100100010111110001011100101010010000 10)_2$ olur.

Parantez karşılığı ise “ ((((((()(()))((()))))())()))()((((()))()(()()()()()))))() “ şeklinde olur.

Bu sayıların başka bir kullanım şekli de Stack Permutasyonu ‘dur. Burada “1” PUSH işlemi için, “0” da POP işlemini simgeler. Bu şekilde bir karakter dizisi şifrelenebilir.

n = karakter sayısı

C_n = n adet karakterin kaç değişik şekilde karıştırılabileceği.

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Mesela 30 karakterli dizi $C_{30} = 3.814.986.502.092.304$ değişik şekilde yazılabilir.

Aşağıda Stack Permutasyonu örneği verilmiştir:

3 karakter için $C_3 = 5$ farklı yerleştirme şekli vardır.
Sayı = $(50)_{10} = (110010)_2$.
Karakter dizisi = “ abc ”.

sequence	$a[u]b$	operation
	abc[]	
+	ab[c]	PUSH c
+	a[bc]	PUSH b
-	a[c]b	POP b
-	a[]cb	POP c
+	[a]cb	PUSH a
-	[]acb	POP c

“ abc “ karakter dizisi “ acb ” şeklinde yerleri değişti.

Kullanılan yöntemi şöyle özetleyebiliriz:

Catalan Sayılarının Bulunması:

1. " s " sayısından binary dizi oluşturulur.
2. Binary dizide " 1 " için " counter + 1 " . " 0 " için " counter - 1 "
3. WHILE döngüsü içinde counter ≥ 0 olduğu müddetçe kalınır.
4. WHILE çıkışında counter = 0 ise sayı catalandın.

Encrypt İşlemi:

1. " m " sayısından binary dizi oluşturulur.
2. Binary dizi stack permutasyonuda kullanılarak giriş dizisinden şifrelenmiş dizi oluşturulur.

Decrypt İşlemi:

1. " m " sayısından binary dizi oluşturulur.
2. Binary dizinin her elemanı " 1 "ise " 0 " , " 0 "ise " 1 " yapılır.
3. Binary dizi stack permutasyonuda kullanılarak giriş dizisinden şifrelenmiş dizi oluşturulur.

Not:

60-bit sayı için **3.814.986.502.092.304** adet catalan sayısı vardır.

Tüm sayılar hesaplansaydı, tamamının diskte kaplayacağı alan:

Binary dosya için

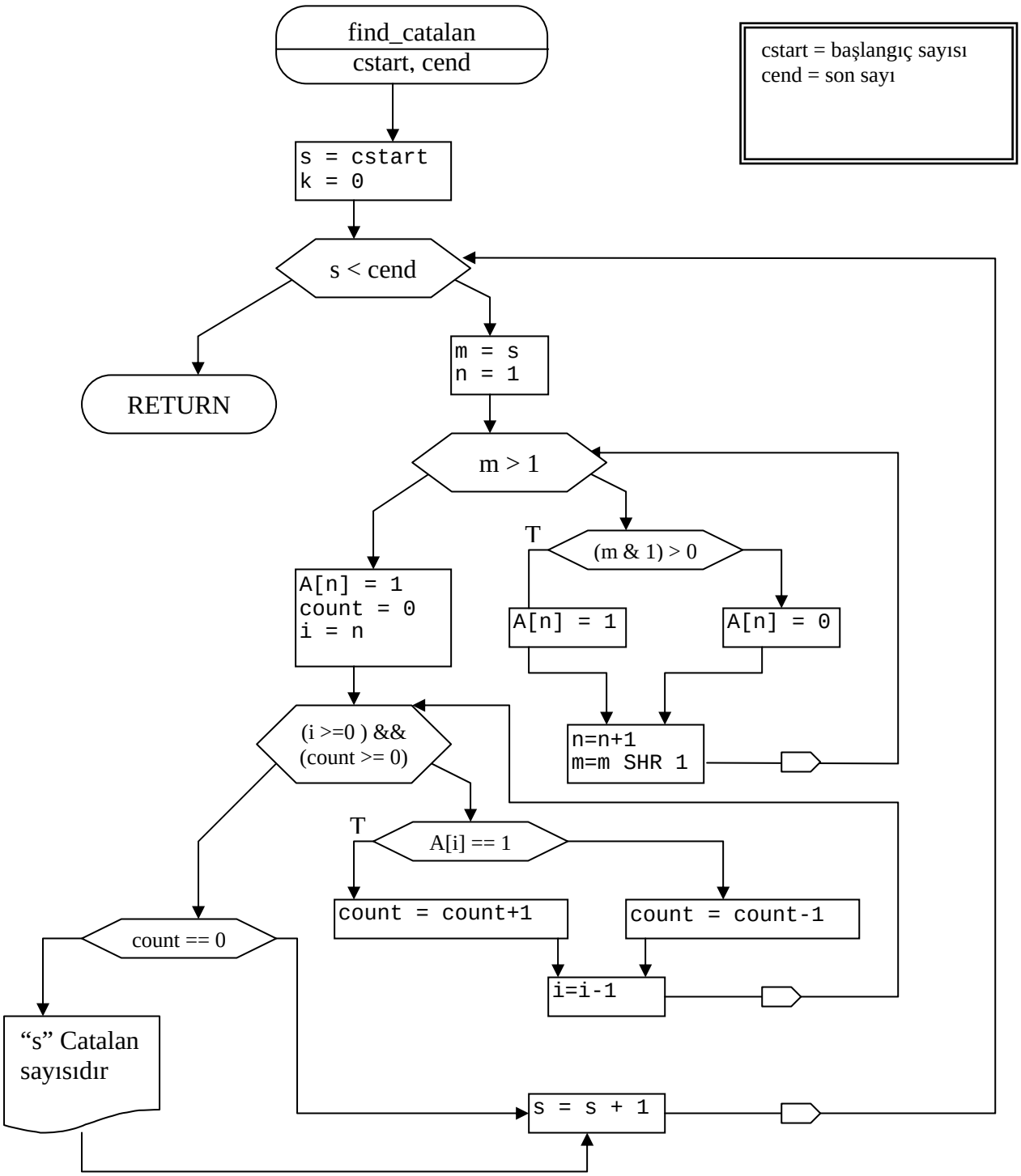
30.519.892.016.738.432 byte
= **29.804.582.047.596** KB
= **29.106.037.155** MB
= **28.423.864** GB
= **27.757** TB yer kaplayacaktı.

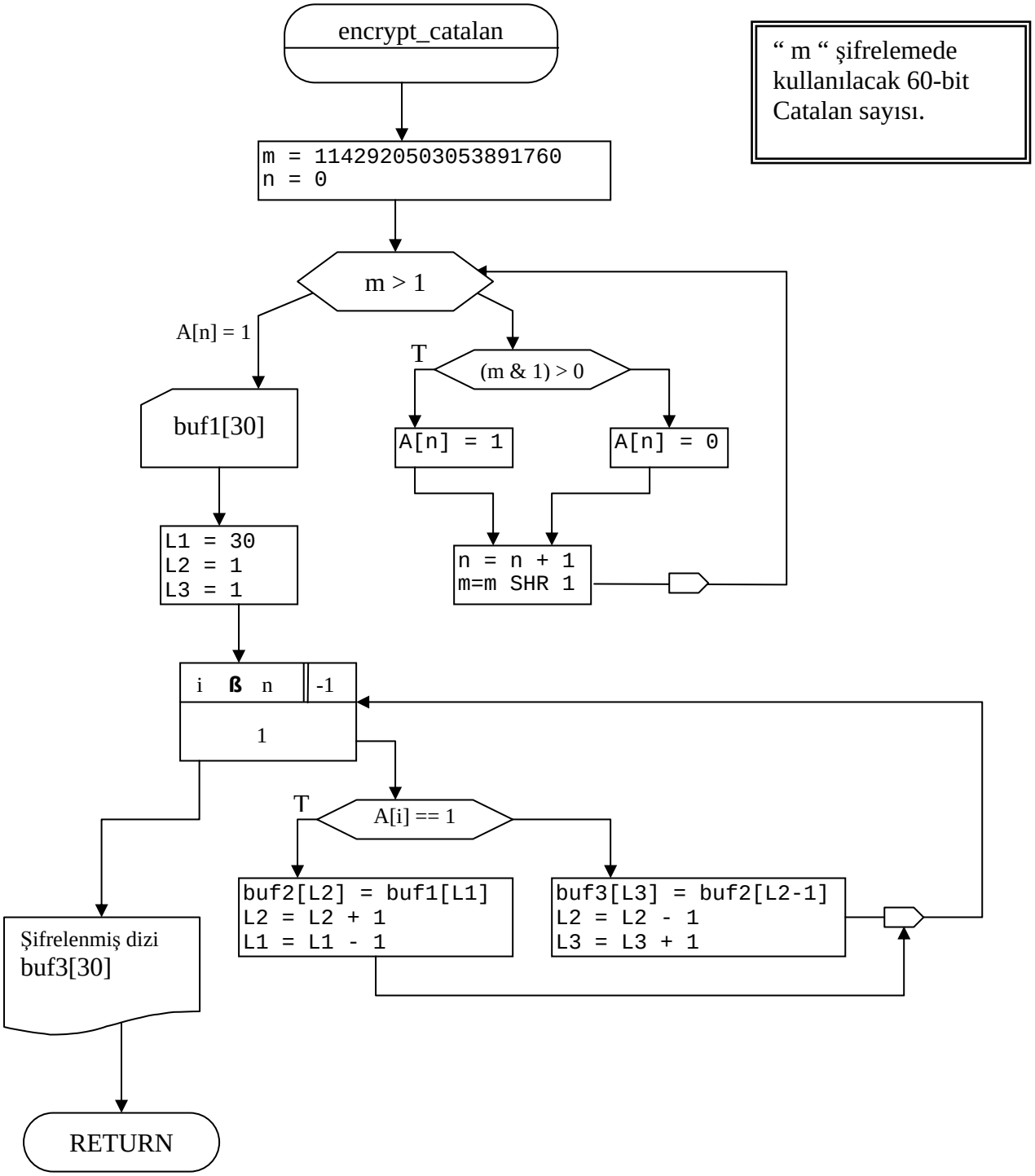
Mesela 60-bit'lik tüm catlan sayılarının hesaplandığını varsayalım.

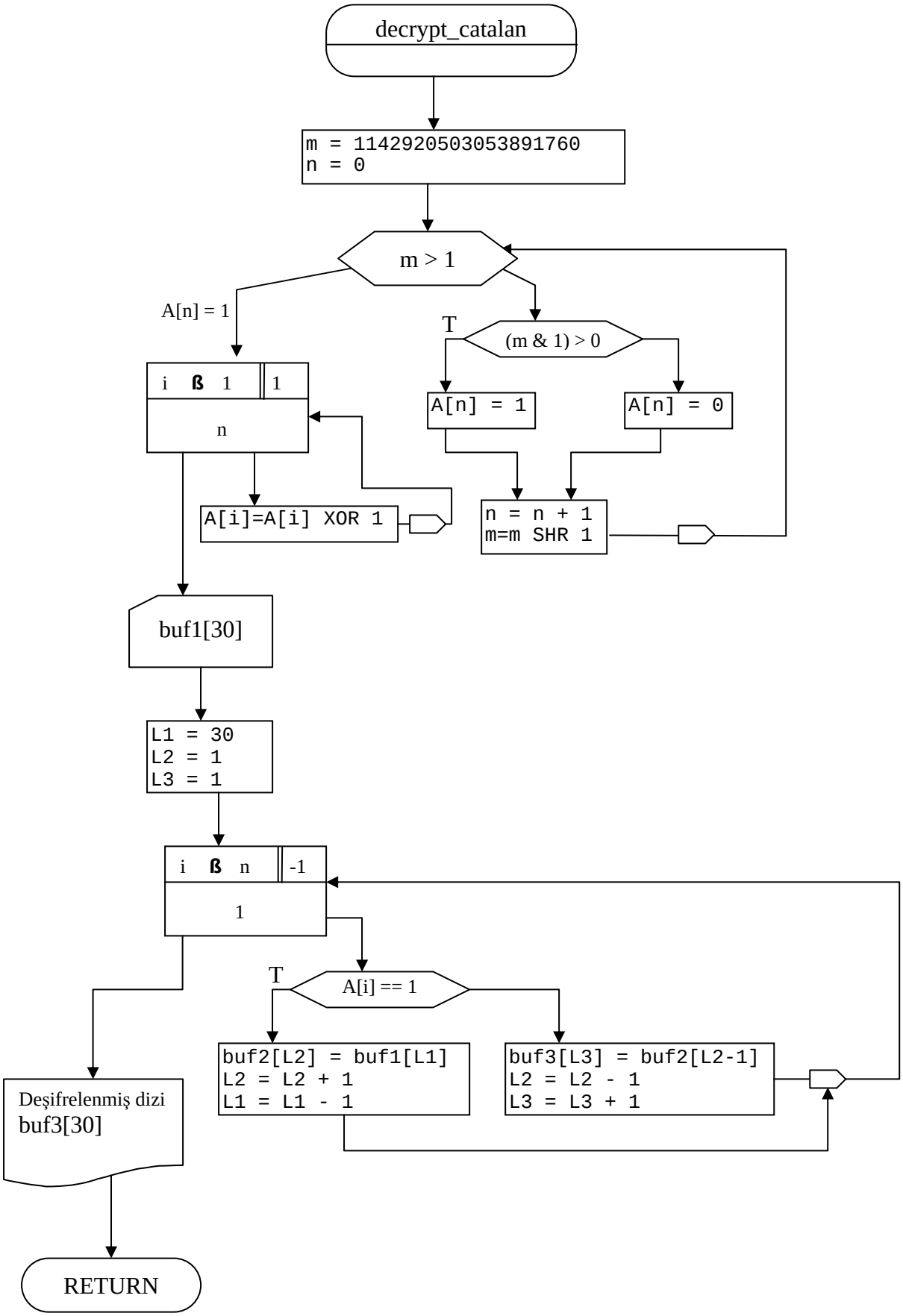
1 elemanın ortalama kontrol edilmesi $\cong$ 1 milisaniye olursa,

tamamının test edilmesi için **120972** yıl gerekir.

Ortalama Süre $\frac{120972}{2} = 60486$ yıl olur.


find_catalan
cstart, cend
cstart = başlangıç sayısı
cend = son sayı
s = cstart
k = 0
s < cend
RETURN
m = s
n = 1
m > 1
T
(m & 1) > 0
A[n] = 1
count = 0
i = n
A[n] = 1
A[n] = 0
n=n+1
m=m SHR 1
(i >=0) &&
(count >= 0)
T
A[i] == 1
count = count+1
count = count-1
i=i-1
count == 0
“s” Catalan
sayısıdır
s = s + 1


encrypt_catalan
" m " şifrelemede kullanılacak 60-bit Catalan sayısı.
m = 1142920503053891760
n = 0
m > 1
A[n] = 1
T
(m & 1) > 0
buf1[30]
A[n] = 1
A[n] = 0
L1 = 30
L2 = 1
L3 = 1
n = n + 1
m=m SHR 1
i ß n -1
1
T
A[i] == 1
buf2[L2] = buf1[L1]
L2 = L2 + 1
L1 = L1 - 1
buf3[L3] = buf2[L2-1]
L2 = L2 - 1
L3 = L3 + 1
Şifrelenmiş dizi
buf3[30]
RETURN


decrypt_catalan
m = 1142920503053891760
n = 0
m > 1
A[n] = 1
T
(m & 1) > 0
i ß 1
1
n
A[n] = 1
A[n] = 0
A[i]=A[i] XOR 1
n = n + 1
m=m SHR 1
buf1[30]
L1 = 30
L2 = 1
L3 = 1
i ß n
-1
1
T
A[i] == 1
buf2[L2] = buf1[L1]
L2 = L2 + 1
L1 = L1 - 1
buf3[L3] = buf2[L2-1]
L2 = L2 - 1
L3 = L3 + 1
Deşifrelenmiş dizi
buf3[30]
RETURN

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>

#define chr_num 30

long filesize(FILE* stream) {
  long curpos, length;
  curpos = ftell(stream);
  fseek(stream, 0L, SEEK_END);
  length = ftell(stream);
  fseek(stream, curpos, SEEK_SET);
  return length;
}

int find_catalan()
{
	short A[67];
	__int64 i, k, m, n, s, count, cstart, cend ;
	FILE *fl;

	cstart = 576460752303423488; // 60-bit için catalan 2^59'dan itibaren bulunabilir.
	cend  = 1152921503533105152;
	k = 0;
	s = cstart; // 60-bit = 30 character.
	while ( s < cend )
	{
		m = s;
		n = 0;
		while(m > 1)
		{
			if ( (m&1) > 0)
				A[n] = 1;
			else
				A[n] = 0;
			n++;
			m = m >> 1;
		}
		A[n] = 1;
		n++;
		count = 0;
		i = n - 1;
		while ( (i>=0) && (count>=0) )
		{
			if (A[i] == 1)
				count++;
			else
				count--;
			i--;
```

```
		}
		if( count == 0 ) // number is catalan
		{
			fprintf(fl,"%I64d\n",s);
		}
		s++;
	}

	fclose(fl);
	return 0;
}

int encrypt_catalan()
{
	short A[67];
	char buf1[chr_num], buf2[chr_num], buf3[chr_num], ch;
	__int64 i, j, k, m, n, s, L1, L2, L3;
	long int count = 0;
	FILE *fl1, *fl2;

	fl1 = fopen("z.txt","rb");
	fl2 = fopen("z2.txt","wb");
	m = 1142920503053891760; // used catalan number.
	n = 0;
	while(m > 1)
	{
		if ( (m&1) > 0)
			A[n] = 1;
		else
			A[n] = 0;
		n++;
		m = m >> 1;
	}
	A[n] = 1;
	count = filesize(fl1) / chr_num;
	for (j=0;j<count;j++)
	{
		fread(&buf1, chr_num, 1, fl1);
		L1 = chr_num-1;
		L2 = L3 = 0;
		for(i=n;i>=0;i--)
			if (A[i] == 1) //  PUSH
			{
				buf2[L2] = buf1[L1];
				L2++;
				L1--;
			}
			else  //  POP
			{
				buf3[L3] = buf2[L2-1];
```

```
                        L2--;
                        L3++;
                        buf2[L2] = 0;
                }
        fwrite(&buf3, chr_num, 1, fl2);
    }

    count = filesize(fl1) - ftell(fl1);
    fread(&buf1, count, 1, fl1);
    fwrite(&buf1, count, 1, fl2);

    fclose(fl1);
    fclose(fl2);
    return 0;
}


int decrypt_catalan()
{
    short A[64];
    char buf1[chr_num], buf2[chr_num], buf3[chr_num], tmp;
    __int64 i, j, k, m, n, s, L1, L2, L3;
    long int count = 0;
    FILE *fl1, *fl2;

    fl1 = fopen("z2.txt","rb");
    fl2 = fopen("z3.txt","wb");
    m = 1142920503053891760; // used catalan number.
    n = 0;
    while(m > 1)
    {
        if ( (m&1) > 0)
            A[n] = 1;
        else
            A[n] = 0;
        n++;
        m = m >> 1;
    }
    A[n] = 1;
    for(i=0;i<=n;i++)
        A[i] =  A[i] ^ 1;
    count = filesize(fl1) / chr_num;
    for (j=0;j<count;j++)
    {
        fread(&buf1, chr_num, 1, fl1);
        L1 = chr_num - 1;
        L2 = L3 = 0;
        for(i=0;i<=n;i++)
            if (A[i] == 1)
            {
```

```
                        buf2[L2] = buf1[L1];
                        L2++;
                        L1--;
                }
                else
                {
                        buf3[L3] = buf2[L2-1];
                        L2--;
                        L3++;
                }

                fwrite(&buf3, chr_num, 1, fl2);
        }

        count = filesize(fl1) - ftell(fl1);
        fread(&buf1, count, 1, fl1);
        fwrite(&buf1, count, 1, fl2);

        fclose(fl1);
        fclose(fl2);
        return 0;
}


void main(void)
{

        find_catalan();
        encrypt_catalan();
        decrypt_catalan();

}
```

---------- ÖRNEK TEXT ----------

30 karakter için 60-bit 'lik sayı kuyllanıldı.
Sayı :
$(1142920503053891760)_{10} = (1111110111000111100000100100010111110000100110111000010110000)_2$

Orijinal Text

TASM now supports aliasing. This means that TASM allows the association
of an alias name with another name (called the substitute name) in a
program. Any time that alias name is encountered it will really refer
to the substitute name.
Aliasing is primarily a linker issue. The alias statement will generate
an alias record setting the alias name equal to the substitute name,
e.g.: 000056 ALIAS '_Set_Coords' = '_SetCoords'
When the linker tries to resolve a reference to a name and it finds an alias
record for that name, it will continue trying to resolve the reference
using the substitute name.

Şifrelenmiş Text

nasi aligs.t rsupp ooM nSTAwTwlloSM asA Tth thasta meshinesliaan a nfamoion
t
isocs aaelcale (emda tnthero ath i wnh n a) i
e
mpraute tntubsse ioaalihat st naey tinm m. agrAm illit wr edaleunteorns eie clutitsubst ee nh
to
trefery alariprimy sa isinga l
A.meilsas aliteahteTue. s ier kinsmi al
anaest ragene rl witenleas naliam ee ehing tted srcotqm natuteei,t
se suhb toluate_ ' SSeAt_I56 A0L0: 0..g0Ch
Wrds'eono tC'_Se t ds'roo=heolv res oat r trie seinkle re it andfeimndao a tneencrfe shr td foarto nc
rsei aln aaaitrynue nigt tnll cio it,mewo
e
rencuesfinethe rvsole reg the substitute name.

Deşifrelenmiş Text

TASM now supports aliasing. This means that TASM allows the association
of an alias name with another name (called the substitute name) in a
program. Any time that alias name is encountered it will really refer
to the substitute name.
Aliasing is primarily a linker issue. The alias statement will generate
an alias record setting the alias name equal to the substitute name,
e.g.: 000056 ALIAS '_Set_Coords' = '_SetCoords'
When the linker tries to resolve a reference to a name and it finds an alias
record for that name, it will continue trying to resolve the reference
using the substitute name.

60-bit bazı Catalan Sayıları

768614336404564650	998914336766534658	1132921503012432770	1142920503052433408
768614336404564652	998914336766534660	1132921503012432772	1142920503052435498
768614336404564658	998914336766534664	1132921503012432776	1142920503052435500
768614336404564660	998914336766534672	1132921503012432784	1142920503052435506
768614336404564664	998914336766534688	1132921503012432800	1142920503052435508
768614336404564682	998914336766534720	1132921503012432832	1142920503052435512
768614336404564684	998914336766534784	1132921503012432938	1142920503052435530
768614336404564690	998914336766534912	1132921503012432940	1142920503052435532
768614336761727106	998914336766535168	1132921503012432946	1142920503052435538
768614336761727108	998914336766535682	1132921503012432948	1142920503052435540
768614336761727112	998914336766535684	1132921503012432952	1142920503052435568
768614336761727120	998914336766535688	1132921503012848690	1142920503052435594
768614336761727136	998914336766535696	1132921503012848692	1142920503052435596
768614336761727168	998914336766535712	1132921503012848696	1142920503052435602
768614336761727234	998914336766535744	1132921503012848714	1142920503052435604
768614336761727236	1102921503000000002	1132921503012848716	1142920503052435608
768614336761727240	1102921503000000004	1132921503012848722	1142920503052435618
768614336762350592	1102921503000000008	1132921503012848724	1142920503052435620
768614336762351618	1102921503000000016	1132921503012848728	1142920503054538050
768614336762351620	1102921503000000032	1132921503012848738	1142920503054538052
768614336762351624	1102921503000000064	1132921503012848740	1142920503054538056
768614336762351632	1102921503000000128	1132921503012848744	1142920503054538064
768614336762351648	1102921503000000256	1132921503012848752	1142920503054538080
768614336762351680	1102921503000000522	1132921503012848778	1142920503054538114
768614336762351744	1102921503000000524	1132921503012848780	1142920503054538116
768614336762351872	1102921503000000530	1132921503012848786	1142920503054538120
768614336762352128	1102921503000000532	1132921503012848788	1142920503054772500
768614336762352640	1102921503000000536	1132921503012848792	1142920503054772504
768614336762353664	1102921503000000546	1132921503012848802	1142920503054772514
768614336762355714	1102921503000000548	1132921503015128594	1142920503054772516
768614336762355716	1102921503000000552	1132921503015128596	1142920503054772520
768614336762355720	1102921503000000560	1132921503015128600	1142920503054772528
768614336762355728	1102921503000000578	1132921503015128610	1142920503054772546
768614336762355744	1102921503000000580	1132921503015128612	1142920503054772548
768614336762355776	1102921503000000584	1132921503015128616	1142920503054772552
768614336762355840	1102921503000000592	1132921503015128624	1142920503054772560
768614336762355968	1102921503000000608	1132921503015128642	1142920503054772576
768614336762356224	1102921503000000642	1132921503015128644	1142920503054772610
998914336762363914	1102921503000000644	1132921503015128648	1142920503054772612
998914336762363916	1102921503000000648	1132921503015128656	1142920503054772616
998914336762363922	1102921503000000656	1132921503015128672	1142920503054772624
998914336762363924	1102921503000000672	1132921503015128706	1142920503054772640
998914336762363928	1102921503000000704	1132921503015128708	1142920503054772672
998914336762363938	1102921503002340194	1132921503015128712	1142920503054772746
998914336762363940	1102921503002340196	1132921503015128720	1142920503054772748
998914336762363944	1102921503002340200	1132921503015128736	1142920503054772754
998914336762363952	1102921503002340208	1132921503015128768	1142920503054772756
998914336762363970	1102921503002340234	1132921503015128834	1142920503054772760
998914336762363972	1102921503002340236	1132921503015128836	1142920503054772770
998914336762363976	1102921503002340242	1132921503015128840	1142920503054772772