# BLM210 PROG. LAB. II PROJE 1

1st Deniz Özcan
*200202106*

2st Halitcan Özarabacı
*160202087*

***Özet*—Bu projede C dili kullanılarak txt uzantılı input.txt dosyasından aldığı tam manasıyla çalıştırılabilen bir kodun,ilk önce ekrana bastırma ve daha sonrasında ise kodun Big O yani O(n) notasyonuna göre zaman ve yer karmaşıklıyla birlikte T(n) yani çalışma süresi hesaplamaya çalışılmıştır.Uygulama, uygulama klasörü içinde bulunan text dosyasını otonom olarak açar ve karakter karakter okur.**

## I. GIRIŞ

Big O Notasyonu-O(n): Bir algoritmanın çalışma zamanının veya yerinin üst sınırını temsil eder. Big O Notation'ın rolü, bir algoritmanın yürütülmesi için alabileceği en uzun süreyi veya yeri hesaplamaktır, yani bir algoritmanın en kötü durumunu (worst case) analizi yapmak için kullanılır.Bu mantık esas alınarak süslü-köşeli parantez, virgül,ünlem,eşittir gibi bir çok yapı hesaplamaya dahil edilmiştir.

## II. YÖNTEMLER

input dosyasından kod alınır ve alınan kod aşağıdaki fonksiyonları kullanarak c uzantılı program.c dosyasına aktarılır.Bu fonksiyonlar:

- fileraw()
  - input dosyasından kod alındıktan sonra program.c dosyasına içinde çalışma süresinin hesaplanabilmesi için gerekli olan 3 kod satırı daha yazar. Bunlar: time.h kütüphanesini dahil etmek,clock() fonksiyonunu kullanmak,double değeriyle sonucu hesaplama kodlarıdır. Daha sonrasında ise kodu fonksiyonel bölünme yapılması için fragmentationofcode() fonksiyonuna göndererir
- fragmentationofcode()
  - Kodu fonksiyonel olarak yani void ya da int olarak böldükten sonra her fonksiyon için void'ten void'e, void'ten int'e ya da void ve int'ten sona doğru mantığıyla fragmentationoffunctions() fonksiyonuna bu aralıklarla kodu gönderir. Göndermeden önce fonksiyonun özyinelemeli mi değil mi diye sorgulamasını yapmak amacıyla recursivefinder() fonksiyonuna fonksiyonların adını gönderir.
- recursivefinder()
  - fragmentationofcode() fonksiyonların adını alır ve her fonksiyon için kendi adını kendi içinde aramasını yapar. Özyinelemeli olan fonksiyonların O(n)'i çalıştırılma sayısına yani T(n) ve Master teorem hesabıyla T(n)'e bağlıdır. Burada amaç bunları bulmaktır. Bulabildiği değeri O(n) olarak ekrana basar.
- fragmentationoffunctions()
  - fragmentationofcode() fonksiyonundan aldığı kodları bütün anahtar kelimeleri (for,while,if,else,int,float vb.) aratarak her fonksiyonun kendi içerisinde yer alan bütün kod parçaları için yer ve zaman karmaşıklığı yaratan bütün kod satırlarını iki parçaya ayırır. Yer karmaşıklıklarını anahtar kelimeler (int,float,char vb) ve işaretler(virgül,köşeli parantez,eşittir vb) ile bulur.Yer karmaşıklığı yaratan bütün kodları spacecomplexity() fonksiyonuna gönderiririr.Zaman karmaşıklıklarını anahtar kelimeler (for,while,switch,case,if,else if, else vb) ve işaretler(süslü parantez,parantez vb) ile bulur. Zaman karmaşıklığı yaratan bütün kodları da timecomplexity() fonksiyonuna gönderir.Bu iki fonksiyondan aldığı doneler ile calculator struct yapısıyla adder() fonksiyonunu kullanarak bağlı liste oluşturur.
- spacecomplexity()
  - fragmentationoffunctions() fonksiyonundan sadece yer karmaşıklığı oluşturan kod satırlarını alır ve bu fonksiyon da aldığı kodun yer karmaşıklığını hesaplatarak geri gönderir.
- timecomplexity()
  - fragmentationoffunctions() fonksiyonundan sadece zaman karmaşıklığı oluşturan kod bloklarını alır ve bu fonksiyon da aldığı kodun zaman karmaşıklığını hesaplatarak geri gönderir.
- insidecontrol(
  - timecomplexity() fonksiyonundan aldığı kodun parantezler arası <,>,=,++,–,*=,/= gibi atamaları bularak fonksiyonun hangi aralıklarda artış hızını da baz alarak kaç kere çalışacağını geri gönderir.
- adder()
  - fragmentationoffunctions() fonksiyonundan zaman ve yer karmaşıklığı yaratan her değeri bağlı liste haline getirir.
- calculator_showing()
  - main() fonksiyonunun çalıştırdığı bu fonksiyon son olarak fragmentationoffunctions() aldığı değerlerin adder() fonksiyonuyla bağlı liste haline geldikten sonra zaman ve yer karmaşıklığında bütün fonksiyonların en büyük zaman ve en büyük yer karmaşıklığını ekrana basar.
- main()
  - fileraw() fonksiyonunu çalıştırır.Bütün hesaplama ve ekrana yazdırmalardan sonra program.c dosyasını

açar ve çalıştırır. Çalıştırdıktan sonra ne kadar zaman aldığını ekrana basar.

- struct calculator
  - fragmentationoffunctions() fonksiyonundan zaman ve yer karmaşıklığı yaratan her değer bağlı liste haline getirilebilmesi için oluşturulmuş struct yapısıdır.

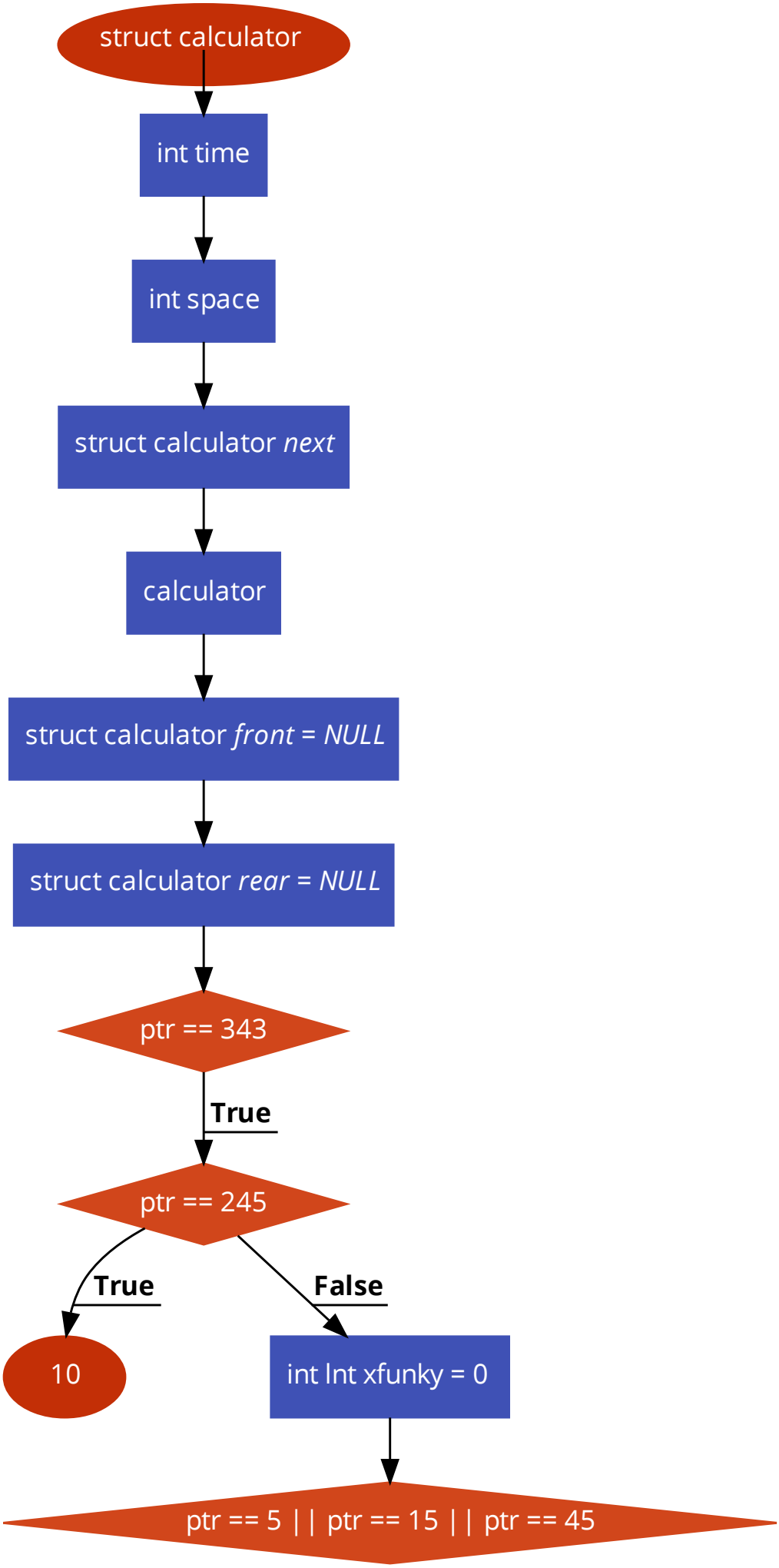
## III. Sonuçlar

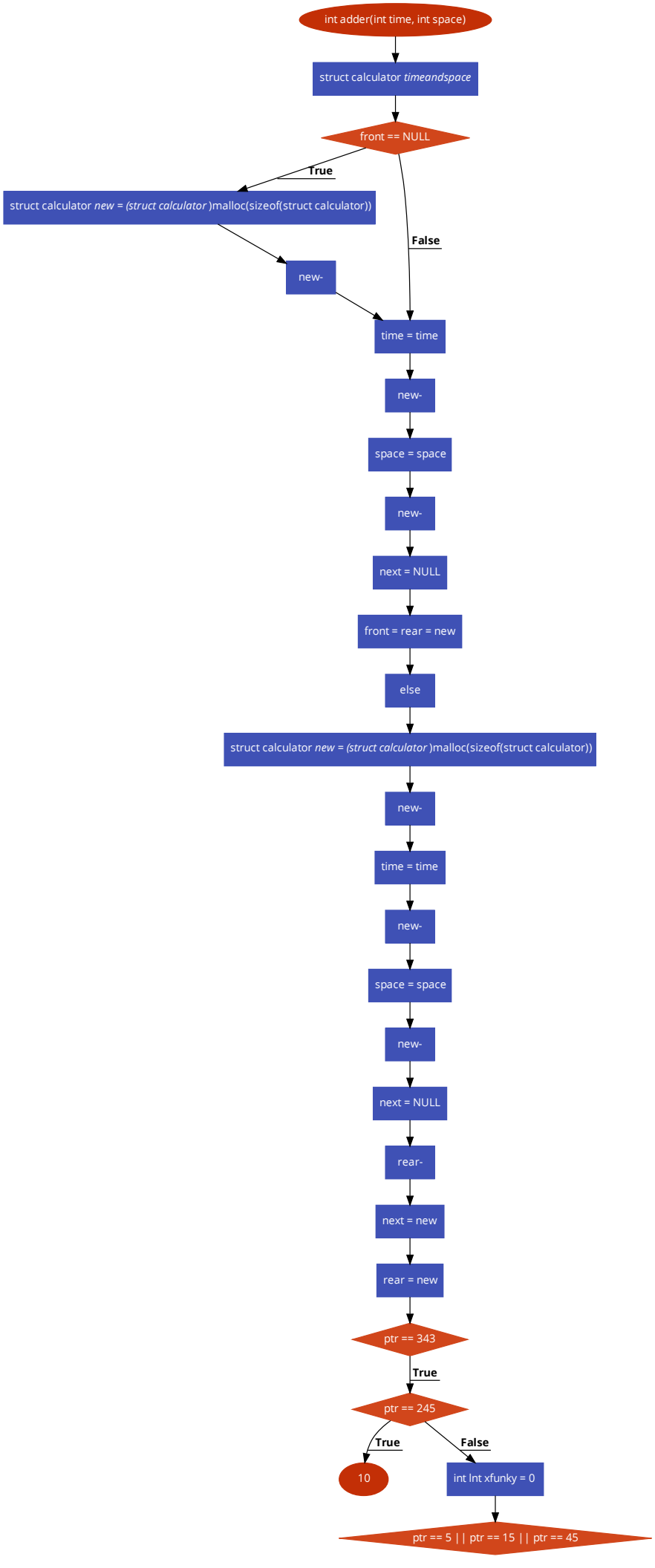
Tüm bu işlemler sonucunda programa input dosyasındaki kodlar ilk önce başarıyla eklendi.Sonra da ekrana kodlar, karmaşıklıklar,çalışma süresi ve hesabı yazdırılırken bir yandan da program.c dosyasının hem içine kod yazıldı hem de kod çalıştırıldı.Olabilecek hatalar için çeşitli kontroller konularak runtime hatalarının önüne geçilmeye çalışıldı.

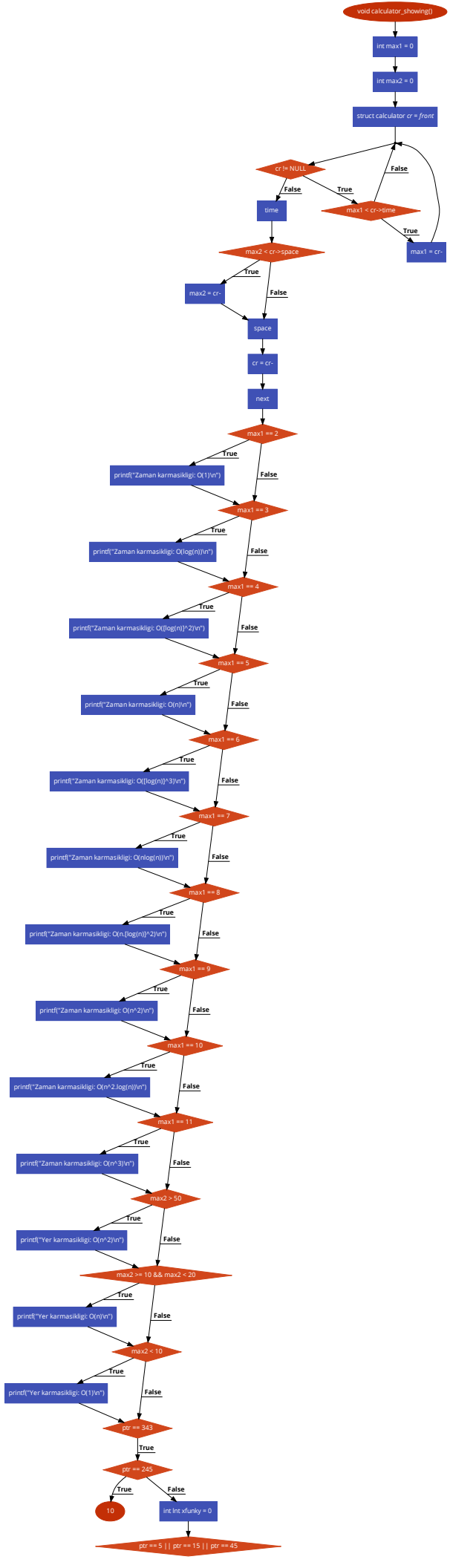
## IV. Çıktılar

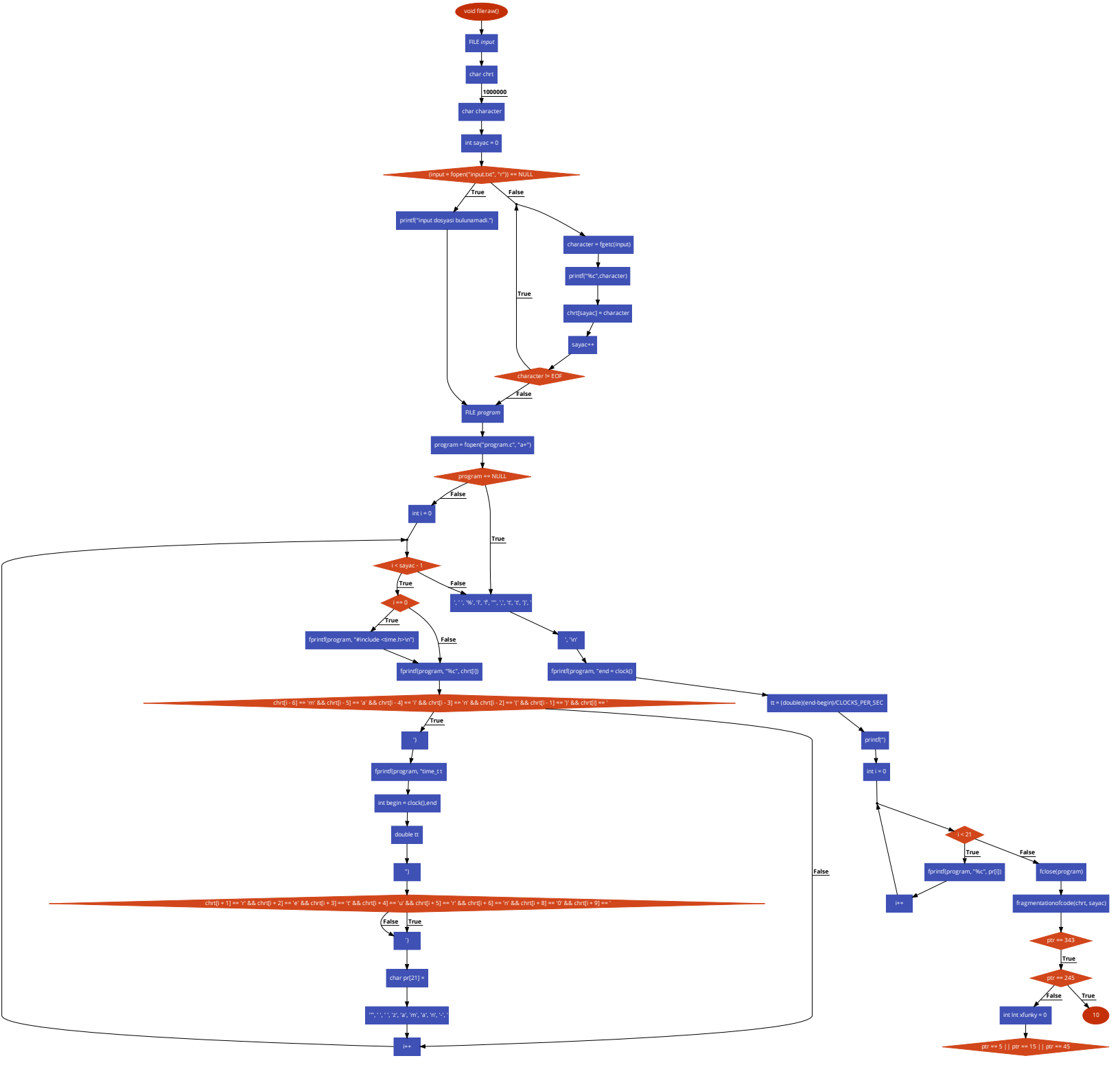
```
#include <stdio.h>
int main(){
        int i,j;
        int sum = 0;
        int n=10;
        int arr[n][n];
        for (i=0;i<n;i++){
                for (j=0;j<n;j++){
                    arr[i][j]=i*j;
                        sum = sum + arr[i][j];
                }
        }
        printf("%d", sum);
        return 0;

}
T(N): 2n+2+n*(2m+2)+n*m*2
Zaman karmasikligi: O(n^2)
Yer karmasikligi: O(n^2)

2025  zaman-> 0.000000
```

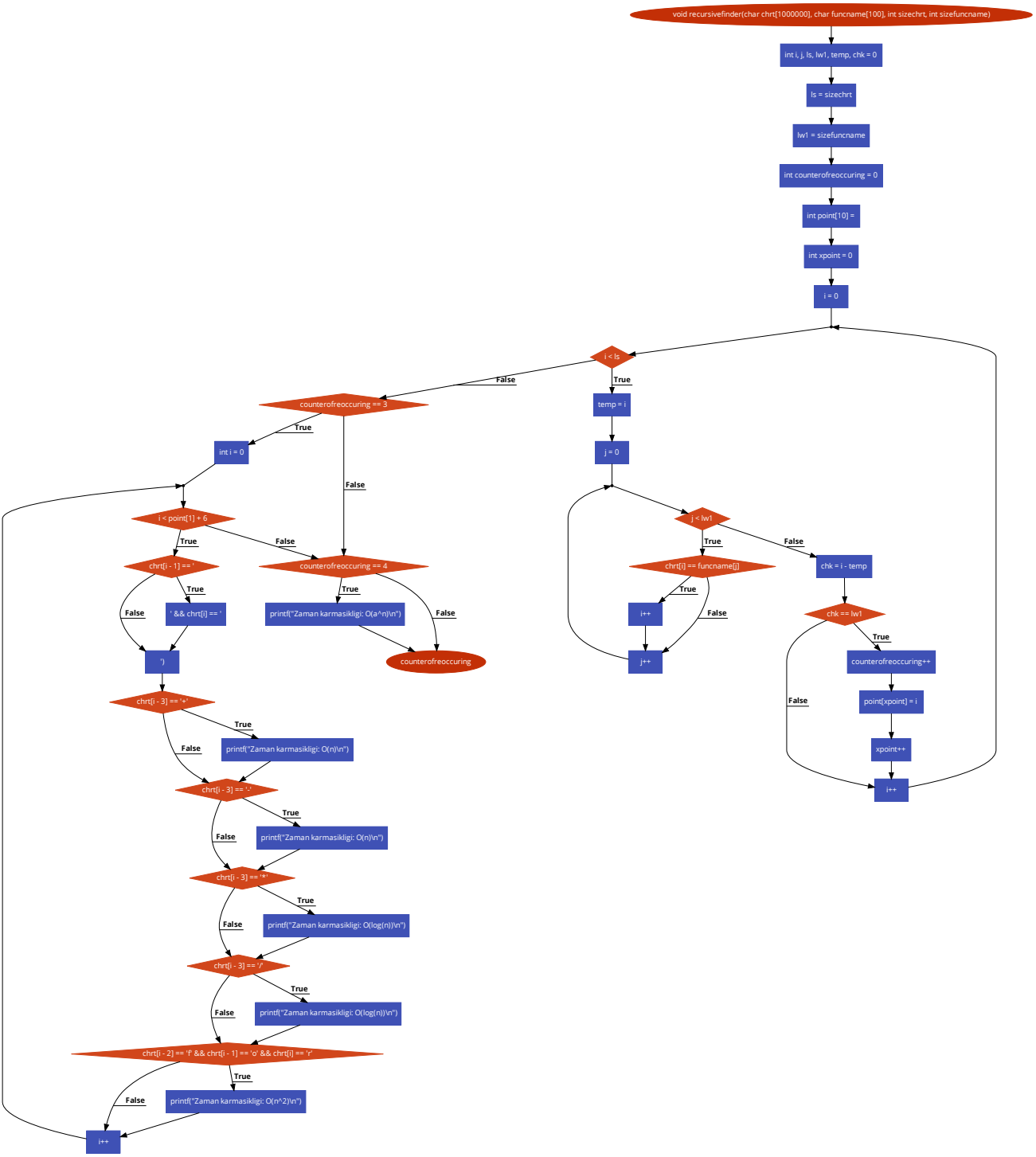
```c
#include <time.h>
#include <stdio.h>
int main(){time_t t;int begin = clock(),end;double tt;
    int i,j;
    int sum = 0;
    int n=10;
    int arr[n][n];
    for (i=0;i<n;i++){
        for (j=0;j<n;j++){
            arr[i][j]=i*j;
            sum = sum + arr[i][j];
        }
    }
        printf("%d", sum);
        end = clock();tt = (double)(end-begin)/CLOCKS_PER_SEC;printf("  zaman-> %lf",tt);
return 0;

}
```

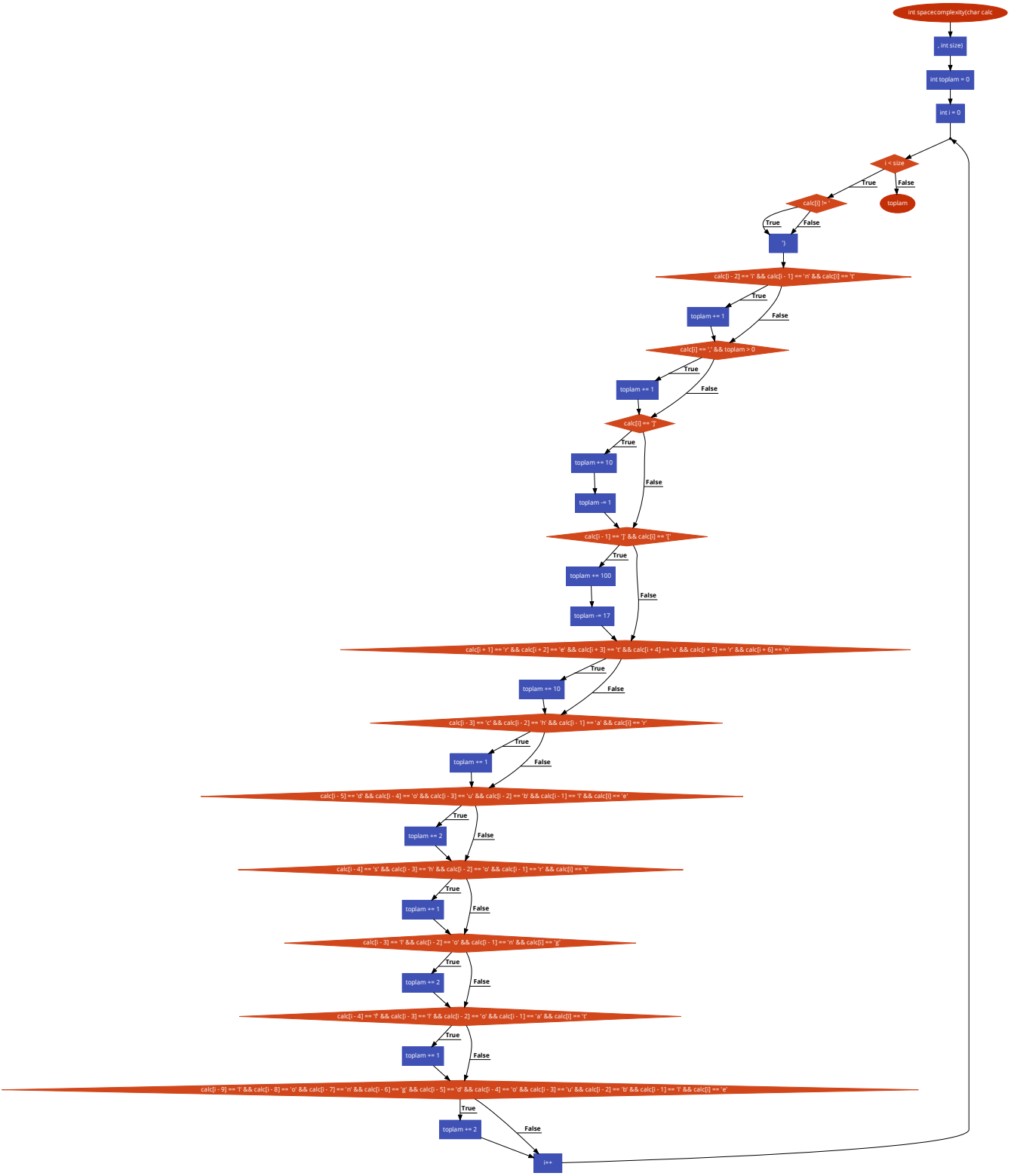

struct calculator
int time
int space
struct calculator *next*
calculator
struct calculator *front* = *NULL*
struct calculator *rear* = *NULL*
ptr == 343
True
ptr == 245
True
False
10
int Int xfunky = 0
ptr == 5 || ptr == 15 || ptr == 45

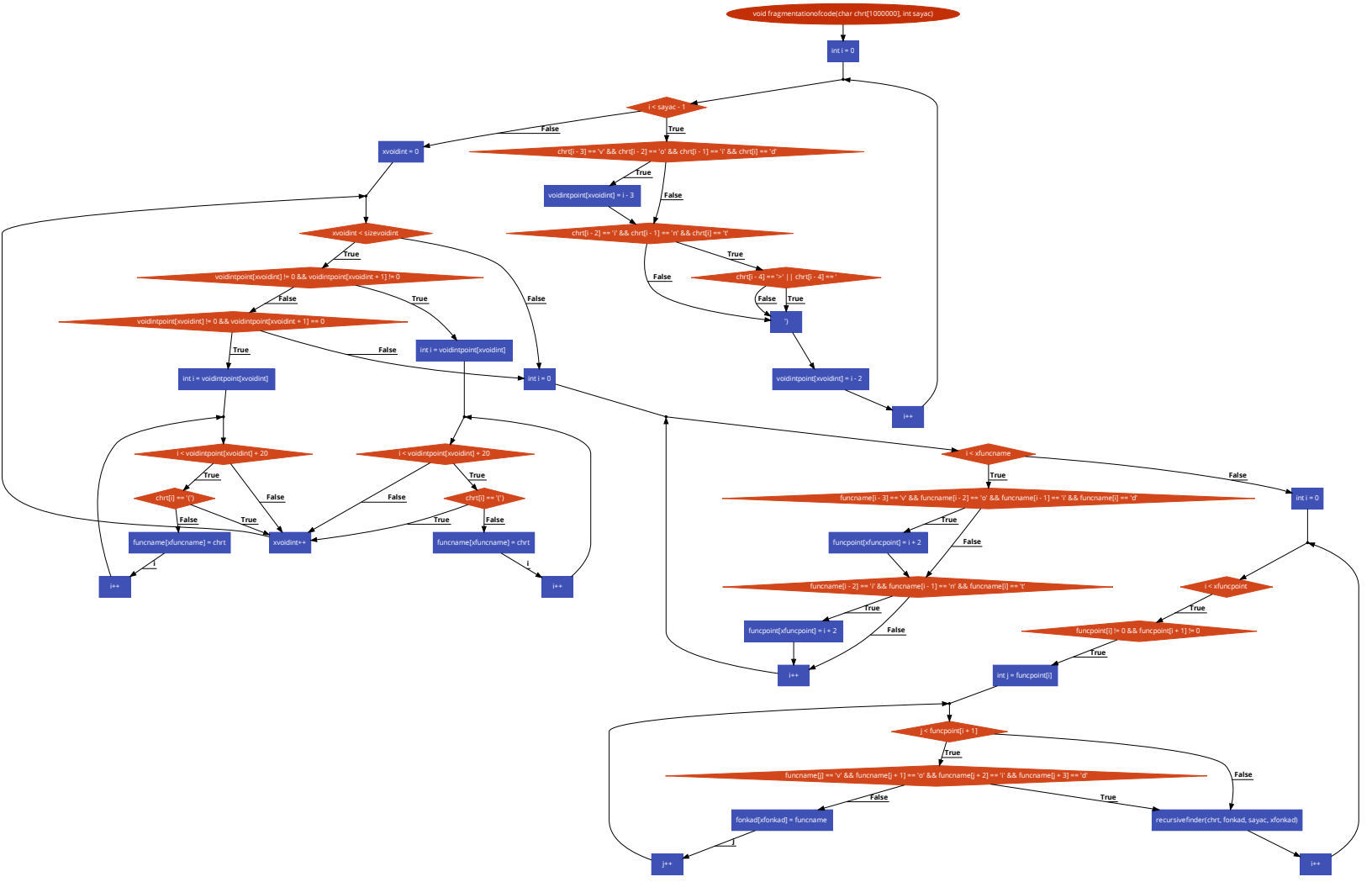

int adder(int time, int space)
struct calculator timeandspace
front == NULL
True
False
struct calculator new = (struct calculator )malloc(sizeof(struct calculator))
new-
time = time
new-
space = space
new-
next = NULL
front = rear = new
else
struct calculator new = (struct calculator )malloc(sizeof(struct calculator))
new-
time = time
new-
space = space
new-
next = NULL
rear-
next = new
rear = new
ptr == 343
True
ptr == 245
True
False
10
int Int xfunky = 0
ptr == 5 || ptr == 15 || ptr == 45

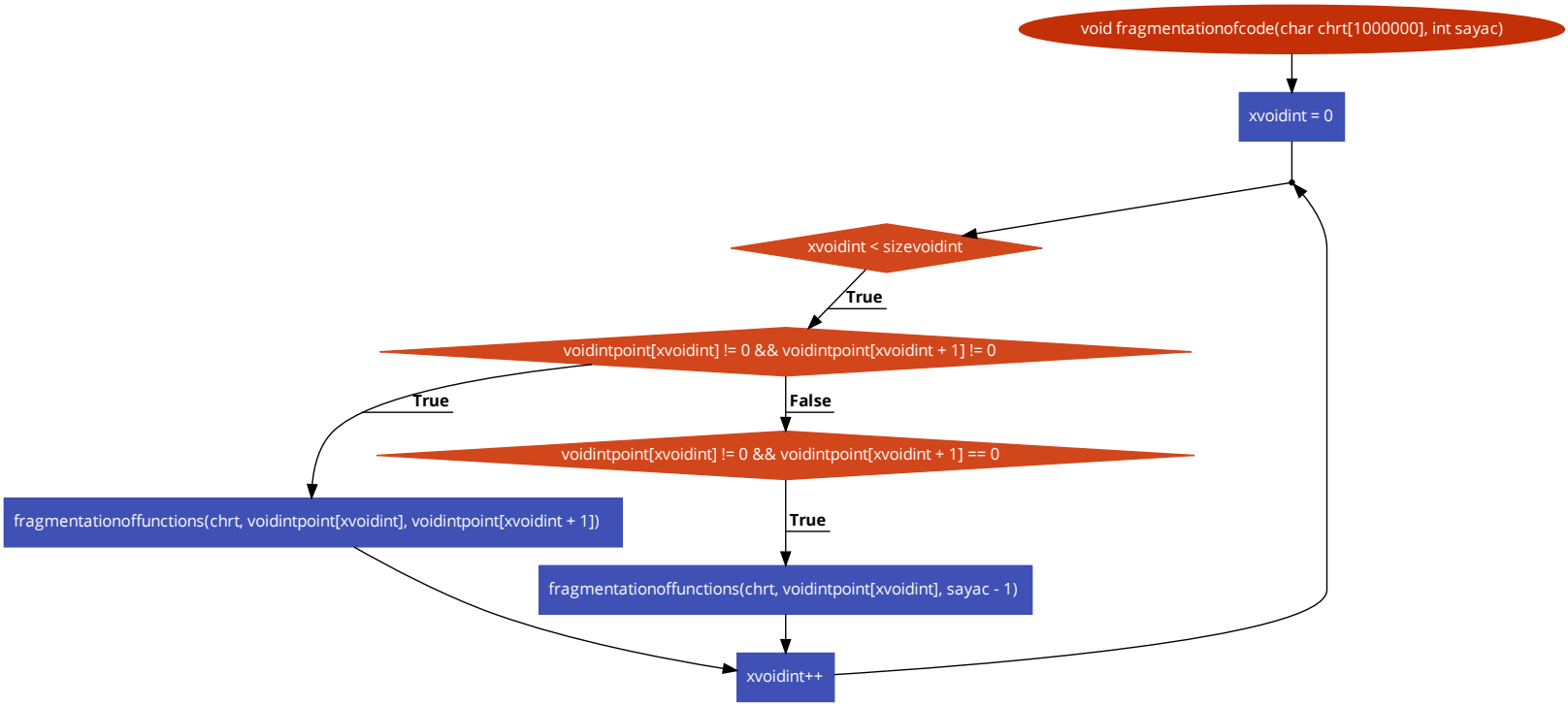

void calculator_showing()
int max1 = 0
int max2 = 0
struct calculator cr = front
cr != NULL
False
True
time
max1 < cr->time
False
True
max1 = cr-
max2 < cr->space
True
max2 = cr-
False
space
cr = cr-
next
max1 == 2
True
printf("Zaman karmasikligi: O(1)\n")
False
max1 == 3
True
printf("Zaman karmasikligi: O(log(n))\n")
False
max1 == 4
True
printf("Zaman karmasikligi: O(log(n)^2)\n")
False
max1 == 5
True
printf("Zaman karmasikligi: O(n)\n")
False
max1 == 6
True
printf("Zaman karmasikligi: O(log(n)^3)\n")
False
max1 == 7
True
printf("Zaman karmasikligi: O(nlog(n))\n")
False
max1 == 8
True
printf("Zaman karmasikligi: O(n.log(n)^2)\n")
False
max1 == 9
True
printf("Zaman karmasikligi: O(n^2)\n")
False
max1 == 10
True
printf("Zaman karmasikligi: O(n^2.log(n))\n")
False
max1 == 11
True
printf("Zaman karmasikligi: O(n^3)\n")
False
max2 > 50
True
printf("Yer karmasikligi: O(n^2)\n")
False
max2 >= 10 && max2 < 20
True
printf("Yer karmasikligi: O(n)\n")
False
max2 < 10
True
printf("Yer karmasikligi: O(1)\n")
False
ptr == 343
True
ptr == 245
True
10
False
int Int xfunky = 0
ptr == 5 || ptr == 15 || ptr == 45

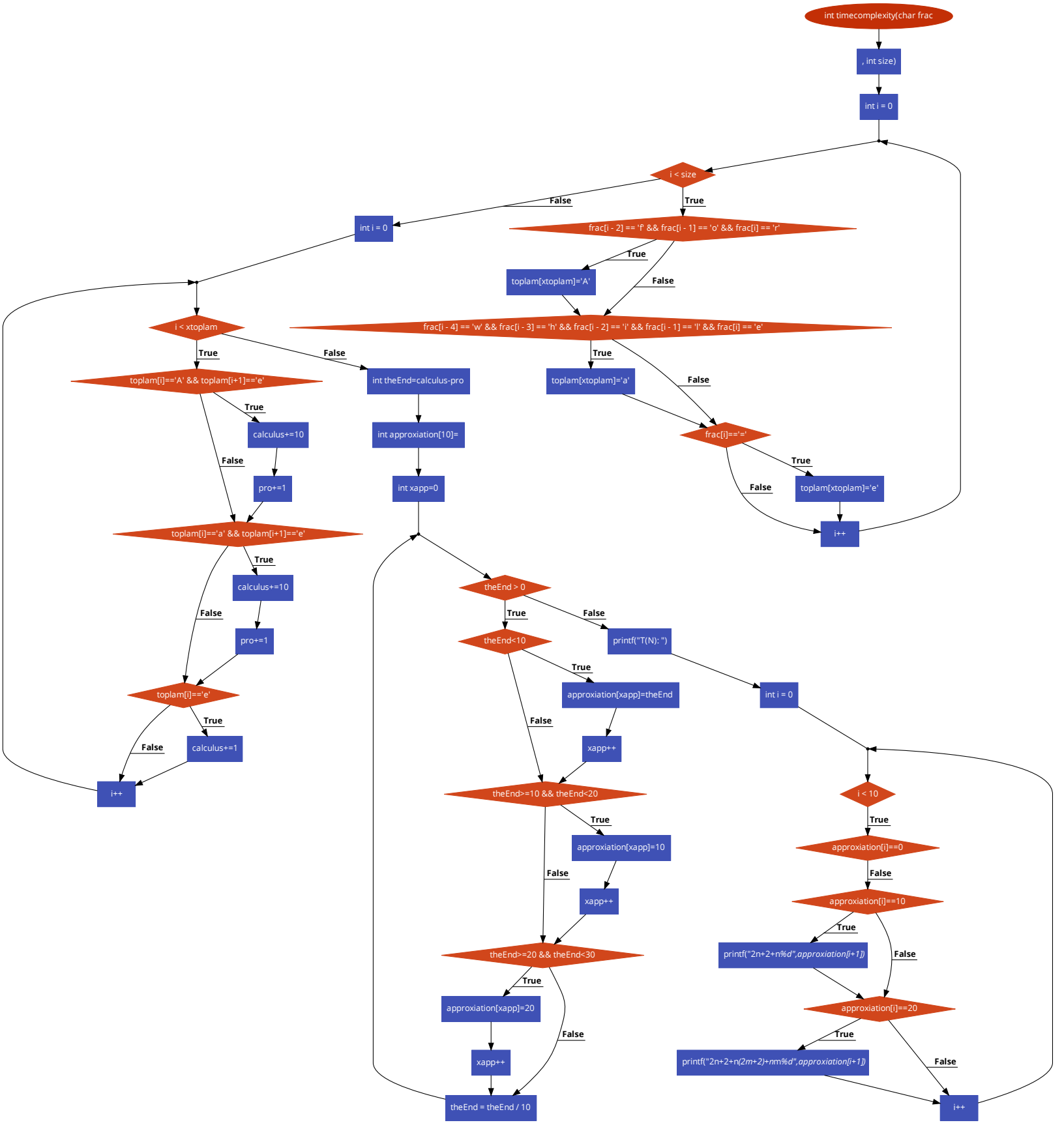

void fileraw()
FILE input
char chrt
1000000
char character
int sayac = 0
(input = fopen("input.txt", "r")) == NULL
True
False
printf("input dosyasi bulunamadi.")
character = fgetc(input)
printf("%c",character)
chrt[sayac] = character
sayac++
True
character != EOF
False
FILE program
program = fopen("program.c", "a+")
program == NULL
False
True
int i = 0
i < sayac - 1
True
False
i == 0
True
False
fprintf(program, "#include <time.h>\n")
fprintf(program, "%c", chrt[i])
', ' ', '%', 'f', 'f', '', '', '', 'e', ')', '
', '\n'
fprintf(program, "end = clock()
chrt[i - 6] == 'm' && chrt[i - 5] == 'a' && chrt[i - 4] == 'i' && chrt[i - 3] == 'n' && chrt[i - 2] == '(' && chrt[i - 1] == ')' && chrt[i] == '
True
')
fprintf(program, "time_t t
int begin = clock(),end
double tt
")
chrt[i + 1] == 'r' && chrt[i + 2] == 'e' && chrt[i + 3] == 't' && chrt[i + 4] == 'u' && chrt[i + 5] == 'r' && chrt[i + 6] == 'n' && chrt[i + 8] == '0' && chrt[i + 9] == '
False
True
')
char pr[21] =
'', ' ', ' ', 'a', 'a', 'm', 'a', 'n', '-', '
i++
False
tt = (double)(end-begin)/CLOCKS_PER_SEC
printf("
int i = 0
i < 21
True
False
fprintf(program, "%c", pr[i])
i++
fclose(program)
fragmentationofcode(chrt, sayac)
ptr == 343
True
ptr == 245
False
True
int int xfunky = 0
10
ptr == 5 || ptr == 15 || ptr == 45

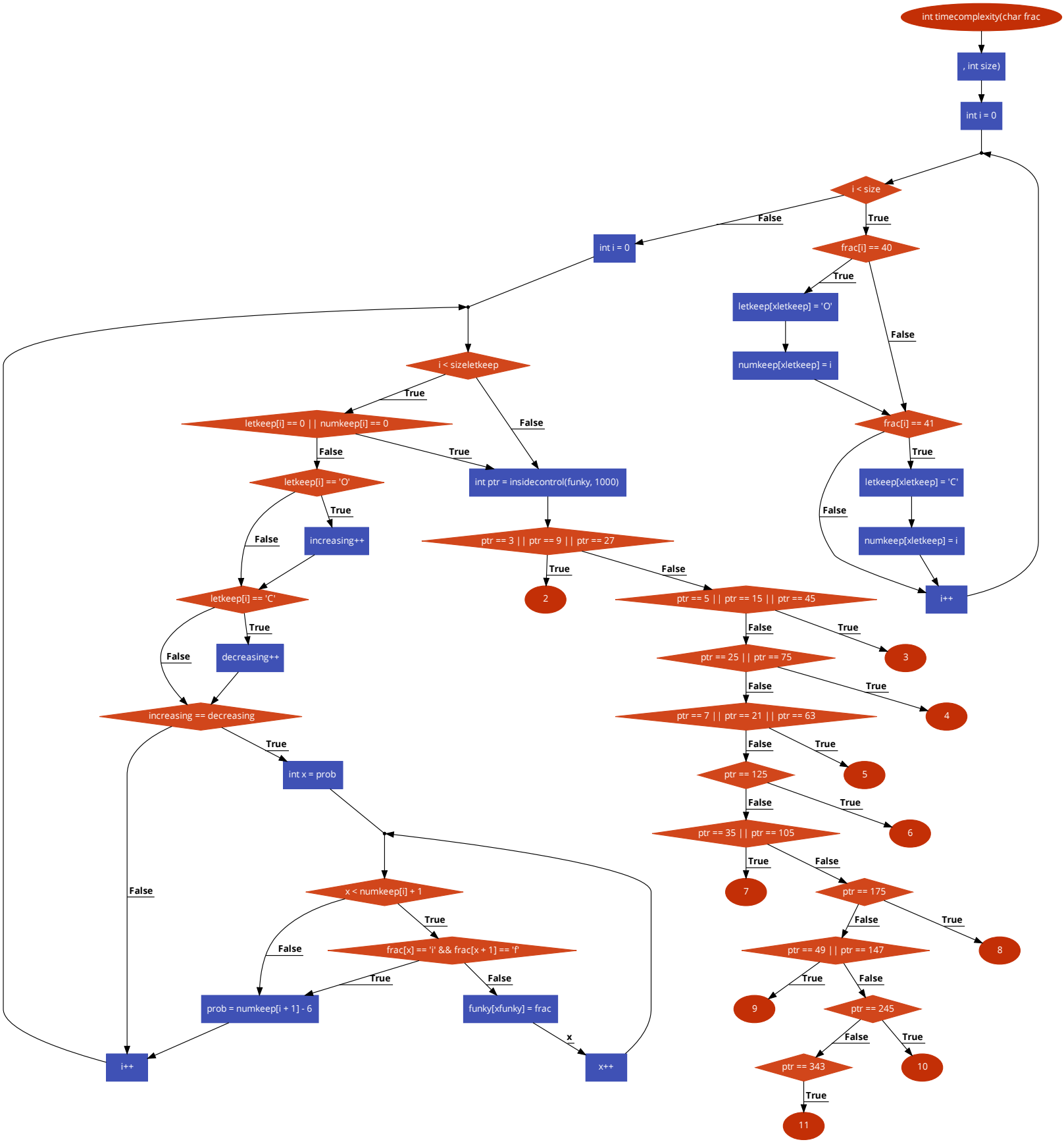

void recursivefinder(char chrt[1000000], char funcname[100], int sizechrt, int sizefuncname)
int i, j, ls, lw1, temp, chk = 0
ls = sizechrt
lw1 = sizefuncname
int counterofreoccuring = 0
int point[10] =
int xpoint = 0
i = 0
i < ls
False
True
counterofreoccuring == 3
True
False
int i = 0
temp = i
j = 0
i < point[1] + 6
True
False
chrt[i - 1] == '
True
False
' && chrt[i] == '
')
counterofreoccuring == 4
True
False
printf("Zaman karmasikligi: O(a^n)\n")
counterofreoccuring
chrt[i - 3] == '+'
True
False
printf("Zaman karmasikligi: O(n)\n")
chrt[i - 3] == '-'
True
False
printf("Zaman karmasikligi: O(n)\n")
chrt[i - 3] == '*'
True
False
printf("Zaman karmasikligi: O(log(n))\n")
chrt[i - 3] == '/'
True
False
printf("Zaman karmasikligi: O(log(n))\n")
chrt[i - 2] == 'f' && chrt[i - 1] == 'o' && chrt[i] == 'r'
True
False
printf("Zaman karmasikligi: O(n^2)\n")
i++
j < lw1
True
False
chrt[i] == funcname[j]
True
False
i++
j++
chk = i - temp
chk == lw1
True
False
counterofreoccuring++
point[xpoint] = i
xpoint++
i++

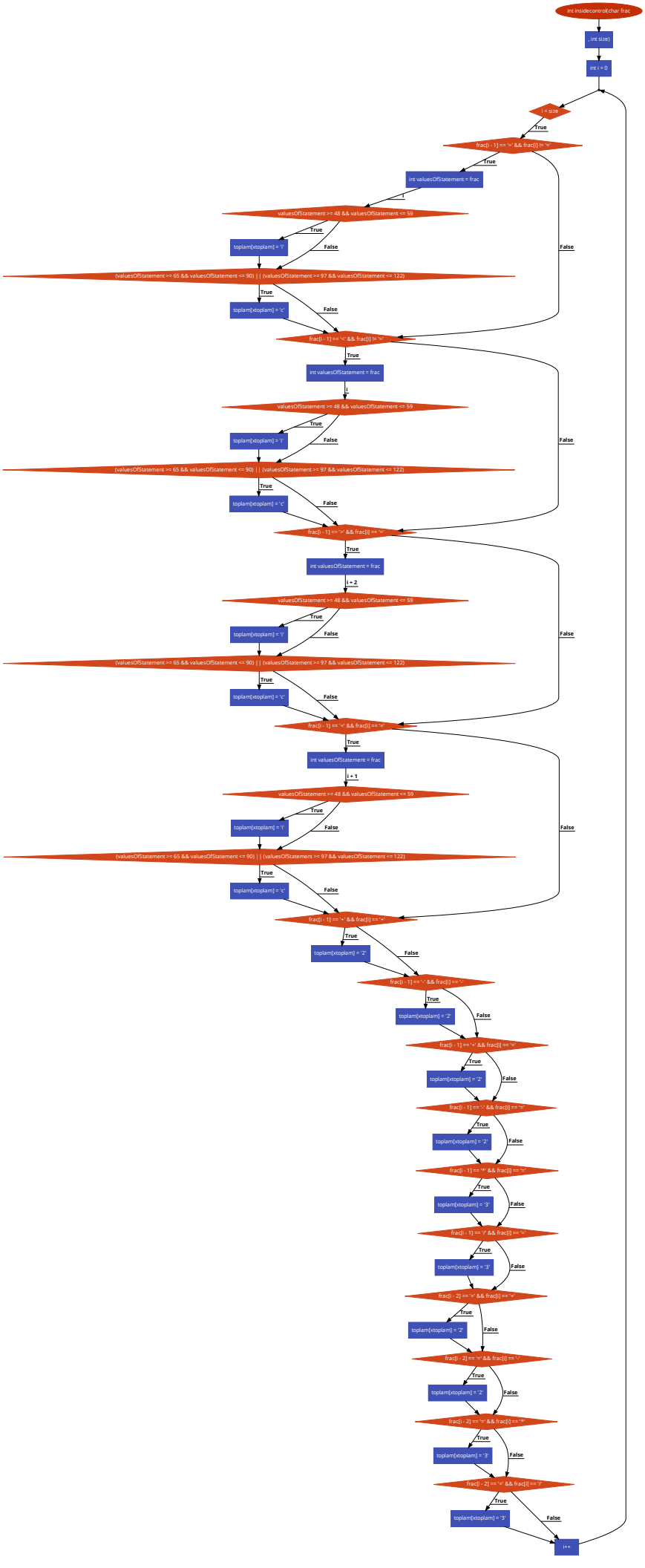

int spacecomplexity(char calc
, int size)
int toplam = 0
int i = 0
i < size
True
False
toplam
calc[i] != '
True
False
')
calc[i - 2] == 'i' && calc[i - 1] == 'n' && calc[i] == 't'
True
toplam += 1
False
calc[i] == ',' && toplam > 0
True
toplam += 1
False
calc[i] == 'j'
True
toplam += 10
toplam -= 1
False
calc[i - 1] == 'j' && calc[i] == 't'
True
toplam += 100
toplam -= 17
False
calc[i + 1] == 'r' && calc[i + 2] == 'e' && calc[i + 3] == 't' && calc[i + 4] == 'u' && calc[i + 5] == 'r' && calc[i + 6] == 'n'
True
toplam += 10
False
calc[i - 3] == 'c' && calc[i - 2] == 'h' && calc[i - 1] == 'a' && calc[i] == 'r'
True
toplam += 1
False
calc[i - 5] == 'd' && calc[i - 4] == 'o' && calc[i - 3] == 'u' && calc[i - 2] == 'b' && calc[i - 1] == 'l' && calc[i] == 'e'
True
toplam += 2
False
calc[i - 4] == 's' && calc[i - 3] == 'h' && calc[i - 2] == 'o' && calc[i - 1] == 'r' && calc[i] == 't'
True
toplam += 1
False
calc[i - 3] == 'l' && calc[i - 2] == 'o' && calc[i - 1] == 'n' && calc[i] == 'g'
True
toplam += 2
False
calc[i - 4] == 'f' && calc[i - 3] == 'l' && calc[i - 2] == 'o' && calc[i - 1] == 'a' && calc[i] == 't'
True
toplam += 1
False
calc[i - 9] == 'f' && calc[i - 8] == 'o' && calc[i - 7] == 'n' && calc[i - 6] == 'g' && calc[i - 5] == 'd' && calc[i - 4] == 'o' && calc[i - 3] == 'u' && calc[i - 2] == 'b' && calc[i - 1] == 'l' && calc[i] == 'e'
True
toplam += 2
False
i++

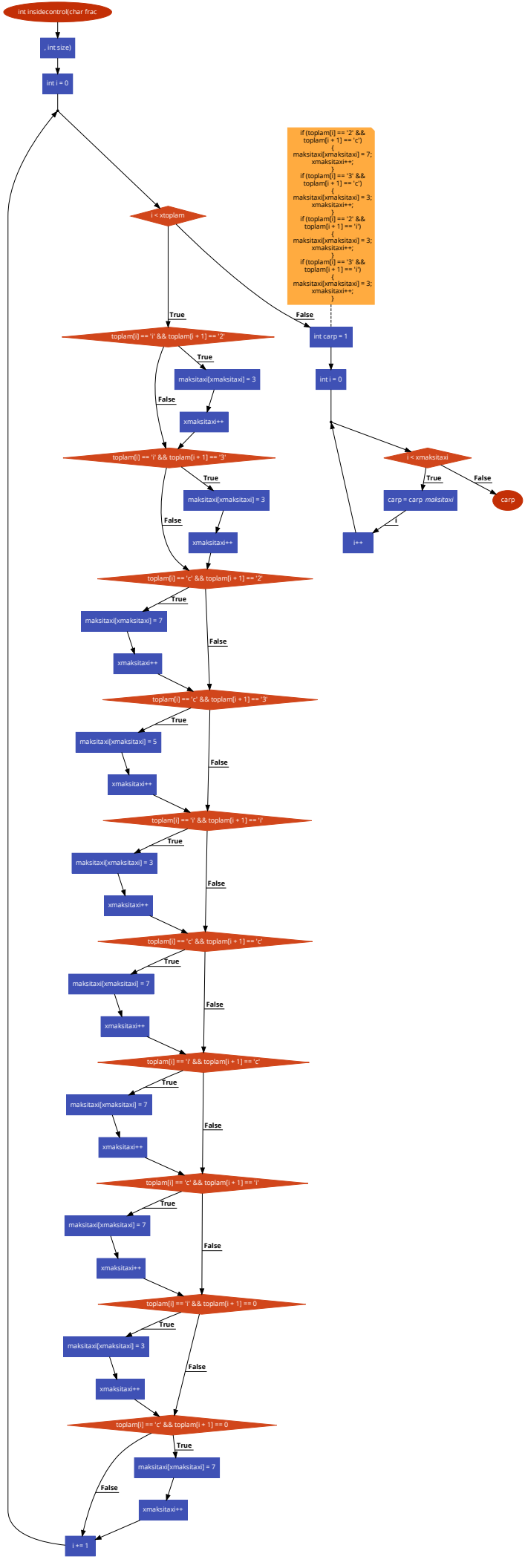

void fragmentationofcode(char chrt[1000000], int sayac)
int i = 0
i < sayac - 1
False
True
xvoidint = 0
chrt[i - 3] == 'v' && chrt[i - 2] == 'o' && chrt[i - 1] == 'i' && chrt[i] == 'd'
True
voidintpoint[xvoidint] = i - 3
False
chrt[i - 2] == 'i' && chrt[i - 1] == 'n' && chrt[i] == 't'
True
False
chrt[i - 4] == '>' || chrt[i - 4] == '
False
True
'
voidintpoint[xvoidint] = i - 2
i++
xvoidint < sizevoidint
True
False
voidintpoint[xvoidint] != 0 && voidintpoint[xvoidint + 1] != 0
False
True
voidintpoint[xvoidint] != 0 && voidintpoint[xvoidint + 1] == 0
True
False
int i = voidintpoint[xvoidint]
int i = voidintpoint[xvoidint]
int i = 0
i < voidintpoint[xvoidint] + 20
True
False
chrt[i] == '{'
True
False
funcname[xfuncname] = chrt
i
i++
xvoidint++
i < voidintpoint[xvoidint] + 20
True
False
chrt[i] == '{'
True
False
funcname[xfuncname] = chrt
i
i++
i < xfuncname
True
False
funcname[i - 3] == 'v' && funcname[i - 2] == 'o' && funcname[i - 1] == 'i' && funcname[i] == 'd'
True
funcpoint[xfuncpoint] = i + 2
False
funcname[i - 2] == 'i' && funcname[i - 1] == 'n' && funcname[i] == 't'
True
funcpoint[xfuncpoint] = i + 2
False
j++
int i = 0
i < xfuncpoint
True
funcpoint[i] != 0 && funcpoint[i + 1] != 0
True
int j = funcpoint[i]
j < funcpoint[i + 1]
True
funcname[j] == 'v' && funcname[j + 1] == 'o' && funcname[j + 2] == 'i' && funcname[j + 3] == 'd'
False
True
fonkad[xfonkad] = funcname
1
j++
recursivefinder(chrt, fonkad, sayac, xfonkad)
i++

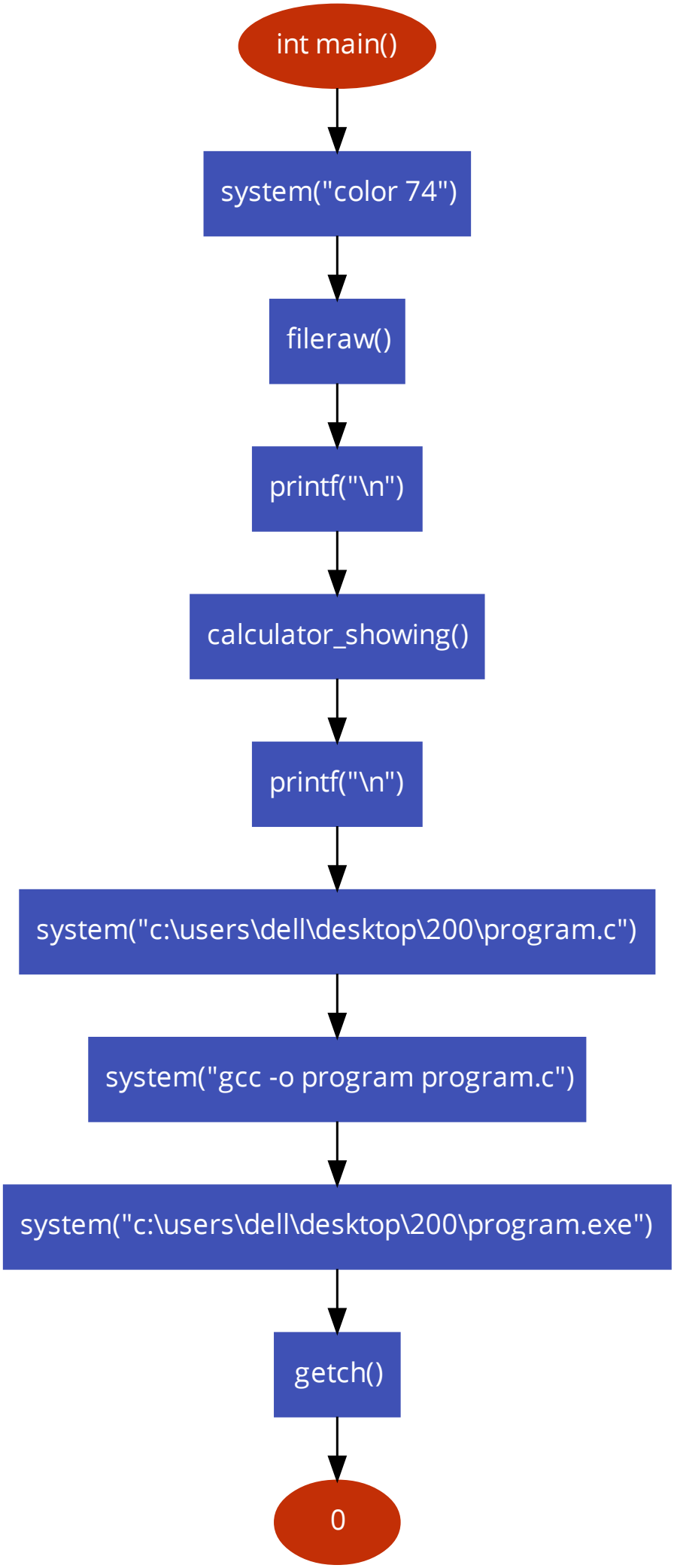

void fragmentationofcode(char chrt[1000000], int sayac)
xvoidint = 0
xvoidint < sizevoidint
True
voidintpoint[xvoidint] != 0 && voidintpoint[xvoidint + 1] != 0
True
False
voidintpoint[xvoidint] != 0 && voidintpoint[xvoidint + 1] == 0
fragmentationoffunctions(chrt, voidintpoint[xvoidint], voidintpoint[xvoidint + 1])
True
fragmentationoffunctions(chrt, voidintpoint[xvoidint], sayac - 1)
xvoidint++


int timecomplexity(char frac
, int size)
int i = 0
i < size
False
True
int i = 0
frac[i - 2] == 'f' && frac[i - 1] == 'o' && frac[i] == 'r'
True
toplam[xtoplam]='A'
False
i < xtoplam
frac[i - 4] == 'w' && frac[i - 3] == 'h' && frac[i - 2] == 'i' && frac[i - 1] == 'l' && frac[i] == 'e'
True
False
toplam[i]=='A' && toplam[i+1]=='e'
int theEnd=calculus-pro
toplam[xtoplam]='a'
False
True
calculus+=10
int approxiation[10]=
frac[i]=='='
False
pro+=1
int xapp=0
True
toplam[xtoplam]='e'
False
toplam[i]=='a' && toplam[i+1]=='e'
i++
True
calculus+=10
theEnd > 0
True
False
False
pro+=1
theEnd<10
printf("T(N): ")
toplam[i]=='e'
True
approxiation[xapp]=theEnd
int i = 0
False
True
xapp++
False
calculus+=1
i++
theEnd>=10 && theEnd<20
i < 10
True
approxiation[xapp]=10
approxiation[i]==0
False
xapp++
False
approxiation[i]==10
theEnd>=20 && theEnd<30
True
True
False
printf("2n+2+n%d",approxiation[i+1])
approxiation[xapp]=20
approxiation[i]==20
False
xapp++
True
printf("2n+2+n(2m+2)+nm%d",approxiation[i+1])
False
theEnd = theEnd / 10
i++


int timecomplexity(char frac
, int size)
int i = 0
i < size
False
True
int i = 0
frac[i] == 40
True
letkeep[xletkeep] = 'O'
numkeep[xletkeep] = i
False
i < sizeletkeep
True
False
letkeep[i] == 0 || numkeep[i] == 0
False
True
letkeep[i] == 'O'
True
increasing++
False
letkeep[i] == 'C'
True
decreasing++
False
increasing == decreasing
True
int x = prob
x < numkeep[i] + 1
True
False
frac[x] == 'i' && frac[x + 1] == 'f'
True
False
prob = numkeep[i + 1] - 6
funky[xfunky] = frac
x
x++
False
i++
int ptr = insidecontrol(funky, 1000)
ptr == 3 || ptr == 9 || ptr == 27
True
2
False
ptr == 5 || ptr == 15 || ptr == 45
False
True
3
ptr == 25 || ptr == 75
False
True
4
ptr == 7 || ptr == 21 || ptr == 63
False
True
5
ptr == 125
False
True
6
ptr == 35 || ptr == 105
True
7
False
ptr == 175
False
True
8
ptr == 49 || ptr == 147
True
9
False
ptr == 245
False
True
10
ptr == 343
True
11
frac[i] == 41
True
letkeep[xletkeep] = 'C'
numkeep[xletkeep] = i
False
i++


int insidecontrol(char frac
, int size)
int i = 0
i < size
True
frac[i - 1] == '>' && frac[i] != '='
True
int valuesOfStatement = frac
i
valuesOfStatement >= 48 && valuesOfStatement <= 59
True
toplam[xtoplam] = '3'
False
(valuesOfStatement >= 65 && valuesOfStatement <= 90) | | (valuesOfStatement >= 97 && valuesOfStatement <= 122)
True
toplam[xtoplam] = 'c'
False
frac[i - 1] == '<' && frac[i] != '='
True
int valuesOfStatement = frac
i
valuesOfStatement >= 48 && valuesOfStatement <= 59
True
toplam[xtoplam] = '1'
False
(valuesOfStatement >= 65 && valuesOfStatement <= 90) | | (valuesOfStatement >= 97 && valuesOfStatement <= 122)
True
toplam[xtoplam] = 'c'
False
frac[i - 1] == '>' && frac[i] == '='
True
int valuesOfStatement = frac
i + 2
valuesOfStatement >= 48 && valuesOfStatement <= 59
True
toplam[xtoplam] = '1'
False
(valuesOfStatement >= 65 && valuesOfStatement <= 90) | | (valuesOfStatement >= 97 && valuesOfStatement <= 122)
True
toplam[xtoplam] = 'c'
False
frac[i - 1] == '<' && frac[i] == '='
True
int valuesOfStatement = frac
i + 1
valuesOfStatement >= 48 && valuesOfStatement <= 59
True
toplam[xtoplam] = '1'
False
(valuesOfStatement >= 65 && valuesOfStatement <= 90) | | (valuesOfStatement >= 97 && valuesOfStatement <= 122)
True
toplam[xtoplam] = 'c'
False
frac[i - 1] == '+' && frac[i] == '+'
True
toplam[xtoplam] = '2'
False
frac[i - 1] == '-' && frac[i] == '-'
True
toplam[xtoplam] = '2'
False
frac[i - 1] == '+' && frac[i] == '='
True
toplam[xtoplam] = '2'
False
frac[i - 1] == '-' && frac[i] == '='
True
toplam[xtoplam] = '2'
False
frac[i - 1] == '*' && frac[i] == '='
True
toplam[xtoplam] = '3'
False
frac[i - 1] == '/' && frac[i] == '='
True
toplam[xtoplam] = '3'
False
frac[i - 2] == '=' && frac[i] == '+'
True
toplam[xtoplam] = '2'
False
frac[i - 2] == '=' && frac[i] == '-'
True
toplam[xtoplam] = '2'
False
frac[i - 2] == '=' && frac[i] == '*'
True
toplam[xtoplam] = '3'
False
frac[i - 2] == '=' && frac[i] == '/'
True
toplam[xtoplam] = '3'
False
i++


int insidecontrol(char frac
, int size)
int i = 0
i < xtoplam
if (toplam[i] == '2' && toplam[i + 1] == 'c') { maksitaxi[xmaksitaxi] = 7; xmaksitaxi++; } if (toplam[i] == '3' && toplam[i + 1] == 'c') { maksitaxi[xmaksitaxi] = 3; xmaksitaxi++; } if (toplam[i] == '2' && toplam[i + 1] == 'i') { maksitaxi[xmaksitaxi] = 3; xmaksitaxi++; } if (toplam[i] == '3' && toplam[i + 1] == 'i') { maksitaxi[xmaksitaxi] = 3; xmaksitaxi++; }
True
False
toplam[i] == 'i' && toplam[i + 1] == '2'
int carp = 1
maksitaxi[xmaksitaxi] = 3
int i = 0
xmaksitaxi++
toplam[i] == 'i' && toplam[i + 1] == '3'
i < xmaksitaxi
carp = carp *maksitaxi*
carp
maksitaxi[xmaksitaxi] = 3
i++
xmaksitaxi++
toplam[i] == 'c' && toplam[i + 1] == '2'
maksitaxi[xmaksitaxi] = 7
xmaksitaxi++
toplam[i] == 'c' && toplam[i + 1] == '3'
maksitaxi[xmaksitaxi] = 5
xmaksitaxi++
toplam[i] == 'i' && toplam[i + 1] == 'i'
maksitaxi[xmaksitaxi] = 3
xmaksitaxi++
toplam[i] == 'c' && toplam[i + 1] == 'c'
maksitaxi[xmaksitaxi] = 7
xmaksitaxi++
toplam[i] == 'i' && toplam[i + 1] == 'c'
maksitaxi[xmaksitaxi] = 7
xmaksitaxi++
toplam[i] == 'c' && toplam[i + 1] == 'i'
maksitaxi[xmaksitaxi] = 7
xmaksitaxi++
toplam[i] == 'i' && toplam[i + 1] == 0
maksitaxi[xmaksitaxi] = 3
xmaksitaxi++
toplam[i] == 'c' && toplam[i + 1] == 0
maksitaxi[xmaksitaxi] = 7
xmaksitaxi++
i += 1


int main()
system("color 74")
fileraw()
printf("\n")
calculator_showing()
printf("\n")
system("c:\users\dell\desktop\200\program.c")
system("gcc -o program program.c")
system("c:\users\dell\desktop\200\program.exe")
getch()
0

## KAYNAKLAR


[1] https://www.faceprep.in/data-structures/space-complexity/
[2] https://www.faceprep.in/data-structures/time-complexity/
[3] https://www.geeksforgeeks.org/sorting-algorithms/
[4] https://www.geeksforgeeks.org/fundamentals-of-algorithms/?ref=shm
[5] https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/
[6] https://www.youtube.com/watch?v=3bhBo9YCTpo&t=4s
[7] https://bilgisayarkavramlari.com/2010/06/17/
karmasiklik-siniflari-complexity-classes/?highlight=big%20o
[8] https://bilgisayarkavramlari.com/2008/12/22/
en-kotu-durum-analizi-worst-case-analysis/?highlight=big%20o
[9] https://bilgisayarkavramlari.com/2010/09/24/
algoritma-analizi-analysis-of-algorithms/?highlight=big%20o