1. tA2008

function type = triangle(sideLengths)

A = sideLengths(1); % First side

B = sideLengths(2); % Second side

C = sideLengths(3); % Third side

if ((A+B > C) && (B+C > A) && (C+A > B)) % Branch # 1

if ((A ~= B) && (B ~= C) && (C ~= A)) % Branch # 2

type = ‘Scalene’;

else

if (((A == B) && (B ~= C)) || ((B == C) && (C ~= A)) || ((C == A) && (A ~= B))) % Branch # 3

type = ‘Isosceles’;

else

type = ‘Equilateral’;

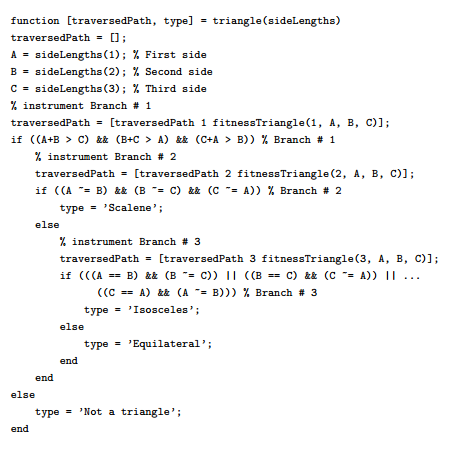
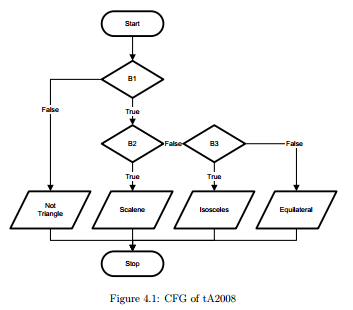
end

end

else

type = ‘Not a triangle’;

end



1. mmA2008

function miniMaxi = minimaxi(num)

numLength = length(num);

mini = num(1);

maxi = num(1);

idx = 2;

while (idx <= numLength) % Branching #1

if maxi < num(idx) % Branching #2

maxi = num(idx);

end

if mini > num(idx) % Branching #3

mini = num(idx);

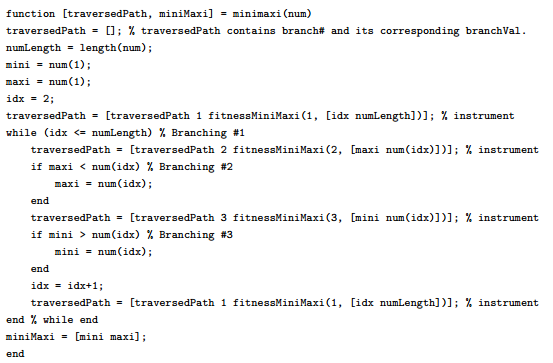
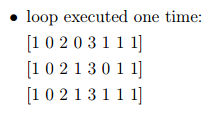
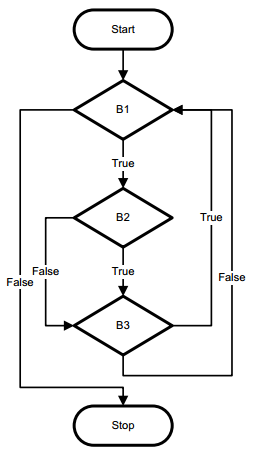
end

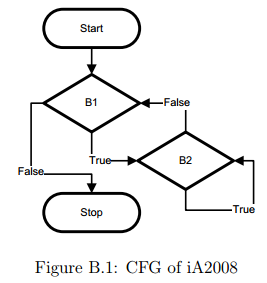
idx = idx+1;

end % while end

miniMaxi = [mini maxi];

end



1. iA2008

function sortedArray = insertion(anyArray)

k = 1; % The smallest integer increment

n = length(anyArray);

i = 2;

for i=2:n

x = anyArray(i);

j = i - 1;

while ((j > 0) & (anyArray(j) > x)),

anyArray(j+1) = anyArray(j);

j = j - 1;

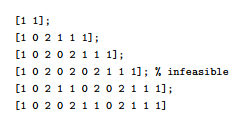
end

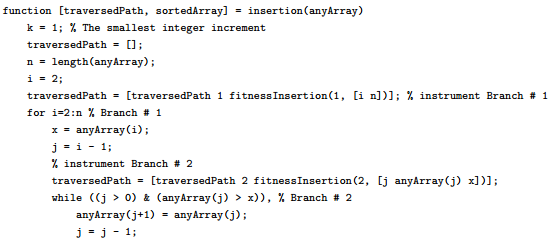
anyArray(j+1) = x;

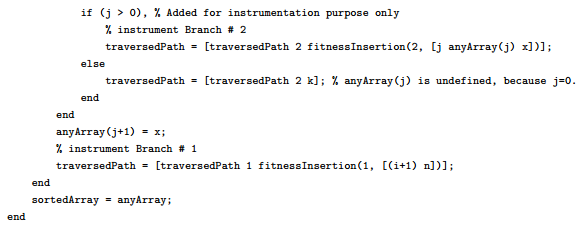
end

sortedArray = anyArray;

end

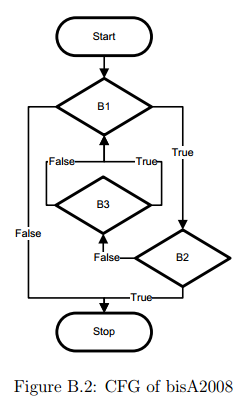






1. bisA2008

function roots = bisection(input)

EPS\_ABS = 1e-2; % constant

EPS\_STEP = 1e-2; % constant

a = input(1);

b = input(2);

c = NaN;

if (f(a) \* f(b)) >= 0,

return;

end

while (b-a >= EPS\_STEP || (abs(f(a)) >= EPS\_ABS && abs(f(b)) >= EPS\_ABS))

c = (a + b)/2;

if (f(c) == 0)

roots = c;

return;

else

if (f(a)\*f(c) < 0)

b = c;

else

a = c;

end

end

end

roots = c;

end

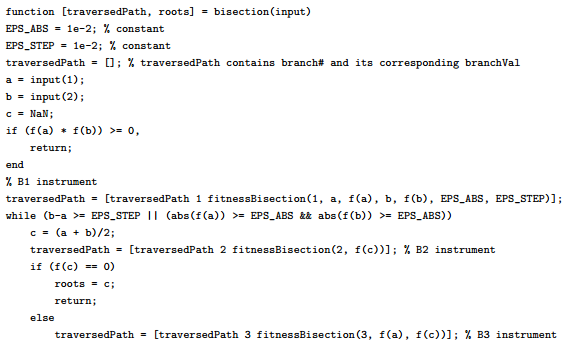
% find the root of the following function y = 3\*x^2 + 10\*x - 3

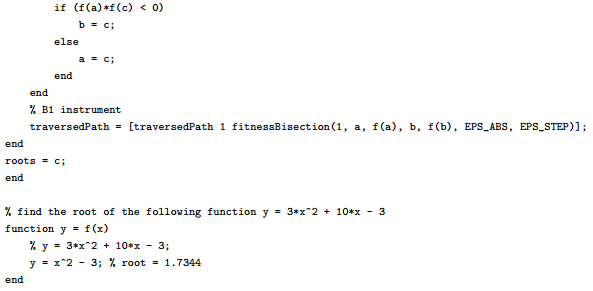
function y = f(x)

% y = 3\*x^2 + 10\*x - 3;

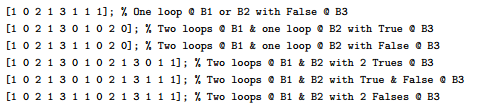
y = x^2 - 3; % root = 1.7344

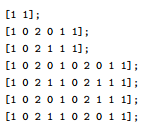
end









1. binA2008

function itemIndex = binary(itemNumbers)

item = itemNumbers(1);

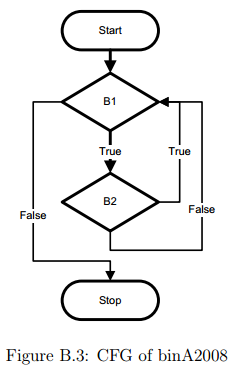
numbers = itemNumbers(1,2:end);

lowerIdx = 1;

upperIdx = length(numbers);

while (lowerIdx ~= upperIdx), % Branch # 1

temp = lowerIdx + upperIdx; % additional statement

 if (mod(temp, 2) ~= 0),

temp = temp - 1;

end % additional statement

idx = temp / 2;

if (numbers(idx) < item), % Branch # 2

lowerIdx = idx + 1;

else

upperIdx = idx;

end

end

% Additional code that returns -1 if the item is not found

if (item == numbers(lowerIdx)),

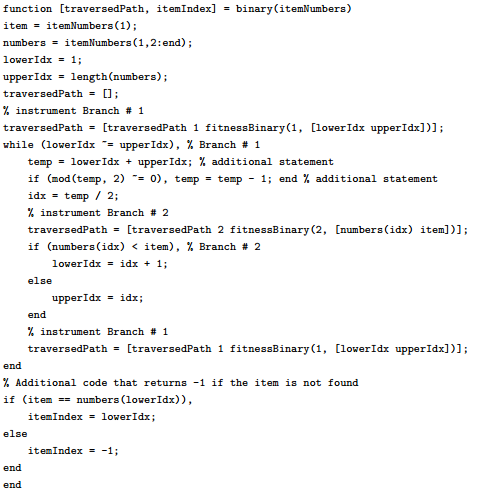
temIndex = lowerIdx;

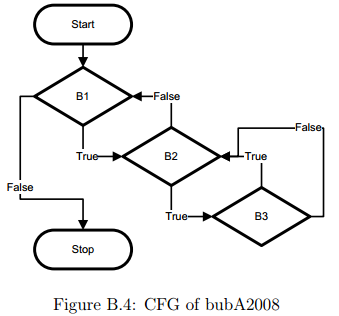
else

itemIndex = -1;

end

end



1. bubA2008

function sortedArray = bubble(anyArray)

sorted = 0; % 0 means false

i = 1; n = length(anyArray);

while ((i <= (n-1)) && ~sorted), % Branch # 1

sorted = 1;

j = n;

for j=n:-1:i+1 % Branch # 2

if (anyArray(j) < anyArray(j-1)) % Branch # 3

%exchange(anyArray(j), anyArray(j-1));

temp = anyArray(j);

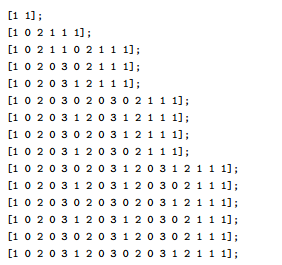
anyArray(j) = anyArray(j-1);

anyArray(j-1) = temp;

sorted = 0;

end

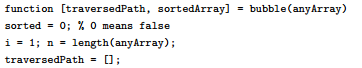
end

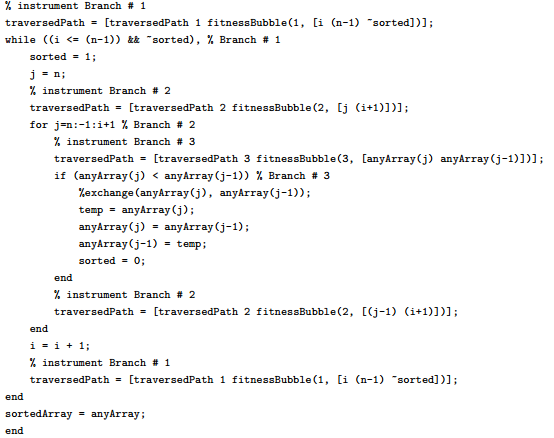
 i = i + 1;

end

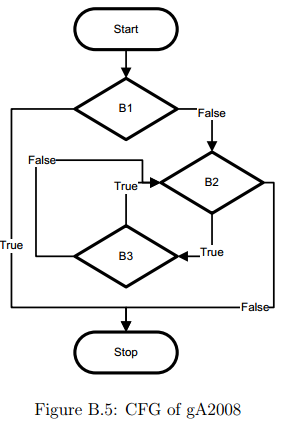
sortedArray = anyArray;

end





1. gA2008

function y = gcd(number)

a = number(1);

b = number(2);

if (a == 0),

y = b;

else

while b ~= 0

if a > b

a = a - b;

else

b = b - a;

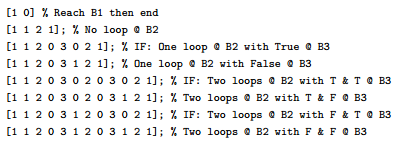
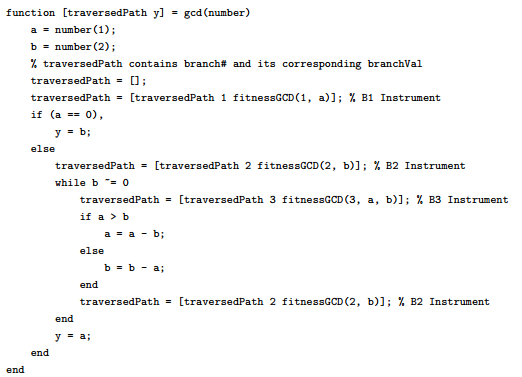
end

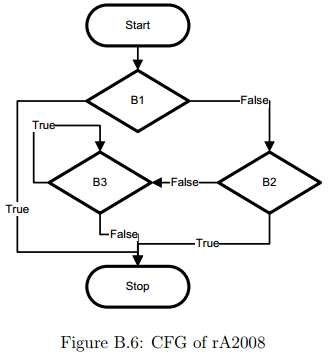
end

y = a;

end

end



1. rA2008

function y = remainder(input)

a = input(1);

d = input(2);

if d == 0 % divisor can not be zero

y = NaN;

else

if a < d

y = a;

else

while a >= d

a = a - d;

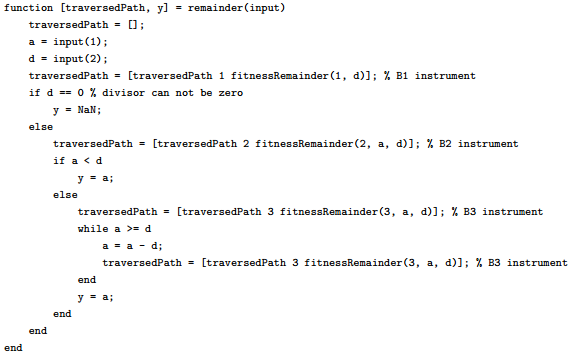
end

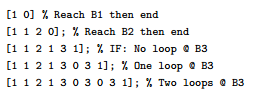
y = a;

end

end

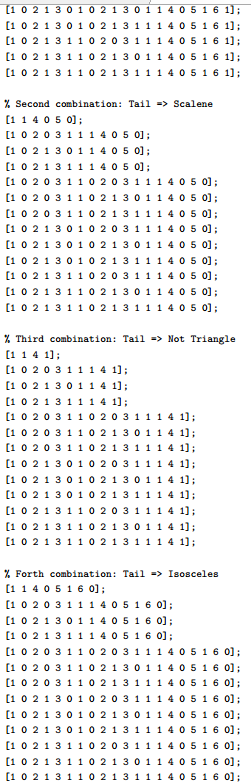
end





1. mtA2008

function [minimaxi, type] = mmTriangle(num)

 numLength = length(num);

mini = num(1);

maxi = num(1);

idx = 2;

while (idx <= numLength)

if maxi < num(idx) % Branching #2

maxi = num(idx);

end

if mini > num(idx) % Branching #3

mini = num(idx);

end

idx = idx+1;

end

minimaxi = [mini maxi];

A = num(1); % First side

B = num(2); % Second side

C = num(3); % Third side

if ((A+B > C) & (B+C > A) & (C+A > B))

if ((A ~= B) & (B ~= C) & (C ~= A)) % Branch # 5

type = ‘Scalene’;

else

if (((A == B) & (B ~= C)) | ((B == C) & (C ~= A)) | ((C == A) & (A ~= B)))

type = ‘Isosceles’;

else

type = ‘Equilateral’;

end

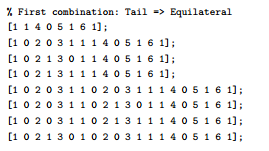
end

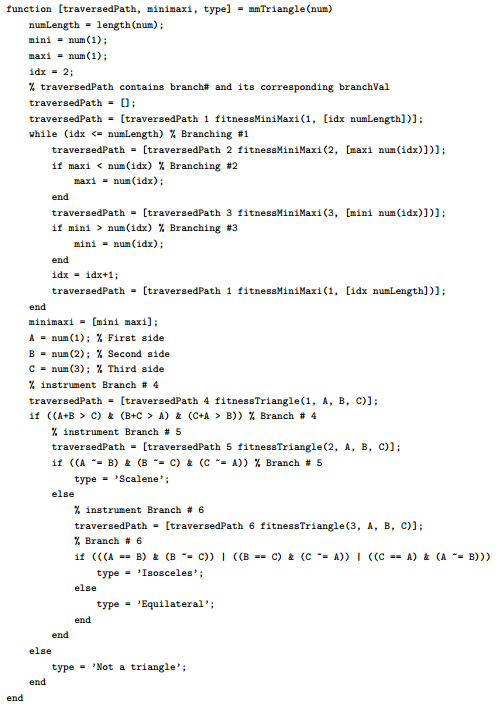
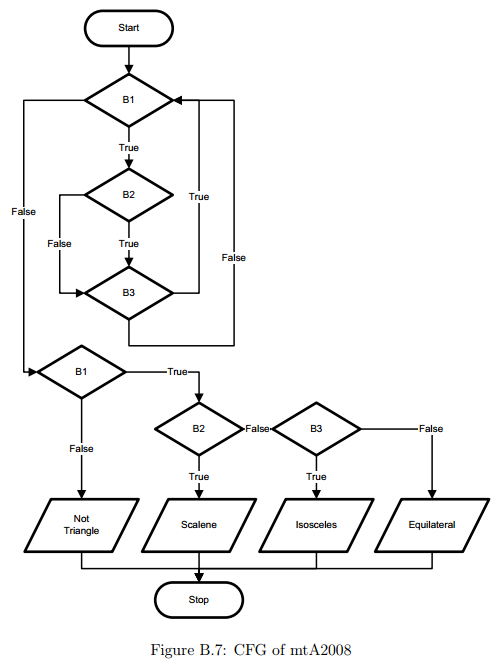
else

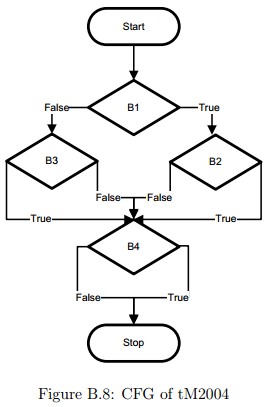
type = ‘Not a triangle’;

end

end





1. tM2004

function type = triangleMansour2004(sideLengths)

A = sideLengths(1); % First side

B = sideLengths(2); % Second side

C = sideLengths(3); % Third side

type = ‘Scalene’;

if (A == B)

if (B == C)

type = ‘Equilateral’;

else

type = ‘Isosceles’;

end

else

if (B == C)

type = ‘Isosceles’;

end

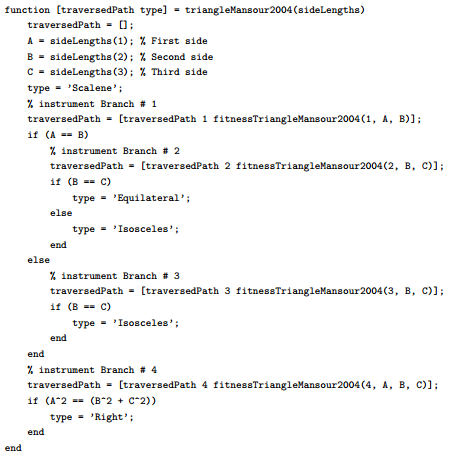
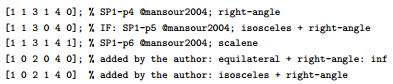
end

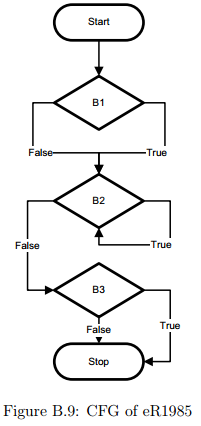
if (A^2 == (B^2 + C^2))

type = ‘Right’;

end

end



1. eR1985

function Z = expintRapps1985(integers)

X = integers(1);

Y = integers(2);

if (Y >= 0)

power = Y;

else

power = -Y;

end

Z = 1;

while (power ~= 0)

Z = Z \* X;

power = power - 1;

end

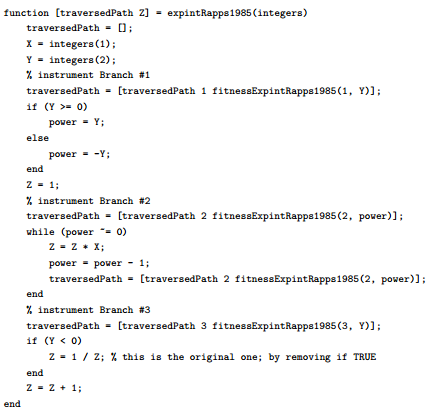
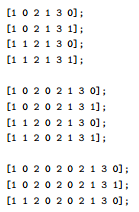
if (Y < 0)

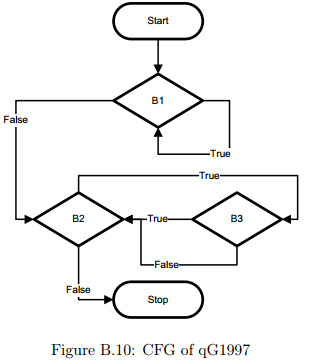
Z = 1 / Z; % this is the original one; by removing if TRUE

end

Z = Z + 1;

End

1. qG1997

function [ q r] = quotientGallagher1997(integers)

q = 0; % q: quotient

r = integers(1); % r: remainder; integers(1): nominator

t = integers(2); % integers(2): denominator

while (r >= t)

t = t \* 2;

end

while (t ~= integers(2))

q = q \* 2;

t = t / 2;

if (t <= r)

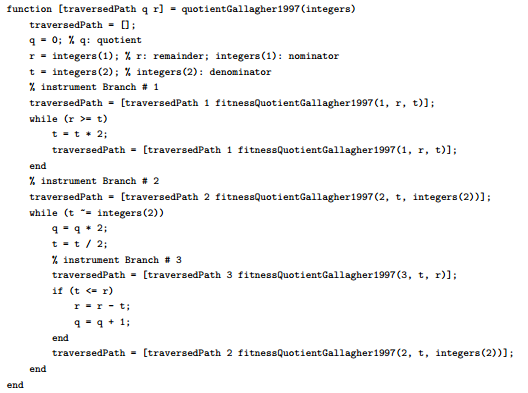
r = r - t;

q = q + 1;

end

end

end



1. tB2002

function [type, area] = tritypeBueno2002(side)

a = side(1);

b = side(2);

c = side(3);

if ((a < b) || (b < c))

type = ‘Invalid input. Input must be ordered a >= b >= c’;

area = 0;

elseif ( a >= (b + c) )

type = ‘Not a triangle’;

area = 0;

elseif ( (a ~= b) && (b ~= c) ) % /\* escaleno \*/

as = a\*a;

bs = b\*b;

cs = c\*c

if (as == bs + cs) % /\* retangulo \*/

type = ‘Rectangle’;

area = b \* c / 2.0;

else

s = (a+b+c) / 2.0;

area = sqrt(s\*(s-a)\*(s-b)\*(s-c));

if ( as < bs + cs )

type = ‘Agudo’; % /\* agudo \*/

else

type =‘Obtuso’; % /\* obtuso \*/

end

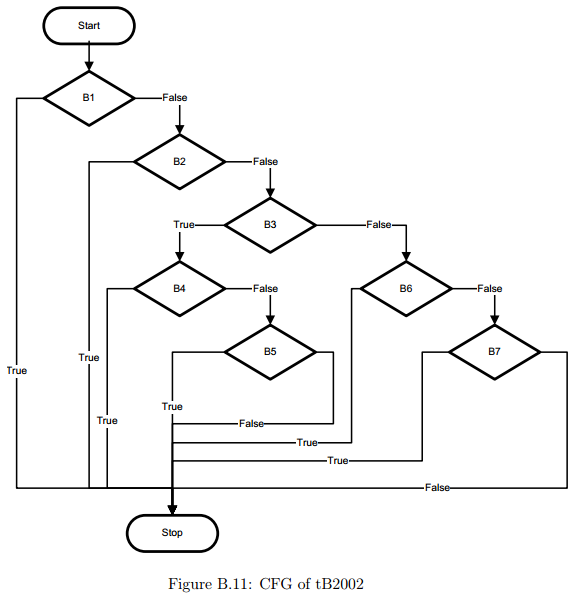
end

elseif ( (a == b) && (b == c) )

type = ‘Equilateral’; % /\* equilatero \*/

area = a\*a\*sqrt(3.0)/4.0;

else

 type = ‘Isosceles’; % /\* isoceles \*/

if ( a == b )

area = c\*sqrt(4\*a\*b-c\*c)/4;

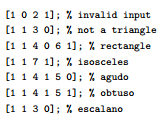
else

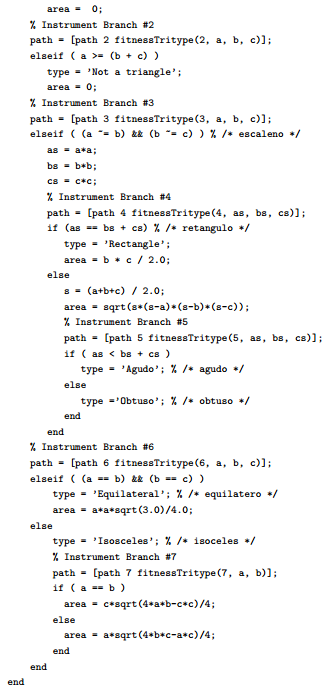
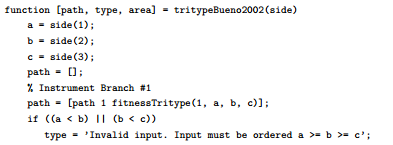
area = a\*sqrt(4\*b\*c-a\*c)/4;

end

end

end





1. eB2002

function result = expintBueno2002(numbersIn)

n = numbersIn(1); % integer

x = numbersIn(2); % floa

MAXIT = 100;

EULER = 0.5772156649;

FPMIN = 1.0e-30;

EPS = 1.0e-7;

nm1 = n - 1;

if (n < 0 || x < 0.0 || (x == 0.0 && (n == 0.0 || n==1)))

result = 0;

% disp(‘bad arguments in expintBueno2002’);

elseif (n == 0)

result = exp(-x)/x;

elseif (x == 0.0)

result = 1.0/nm1; % strangy: what is nm1?

elseif (x > 1.0)

b = x + n;

c = 1.0 / FPMIN;

d = 1.0 / b;

h = d;

for i=1 : MAXIT

a = -i \* (nm1 + i);

b = b + 2.0;

d = 1.0 / (a\*d+b);

c = b + a / c;

del = c \* d;

h = h \* del;

if (abs(del-1.0) < EPS) % abs is fabs in C

result = h \* exp(-x);

return;

end

end

disp(‘continuated fraction failed in expint’);

else

% ans = (nm1!=0 ? 1.0/nm1 : -log(x)-EULER);

% is interpreted as follows

if (nm1 ~= 0)

result = 1.0 / nm1;

else

result = -log(x)-EULER;

end

fact = 1.0;

for i = 1 : MAXIT

fact = fact \* (-x / i);

if (i ~= nm1)

del = -fact / (i - nm1);

else

psi = -EULER;

for ii = 1 : nm1

psi = psi + (1/ii);

end

del = fact \* (-log(x) + psi);

end

result = result + del;

if (abs(del) < abs(result) \* EPS) % abs is fabs in C

return;

end

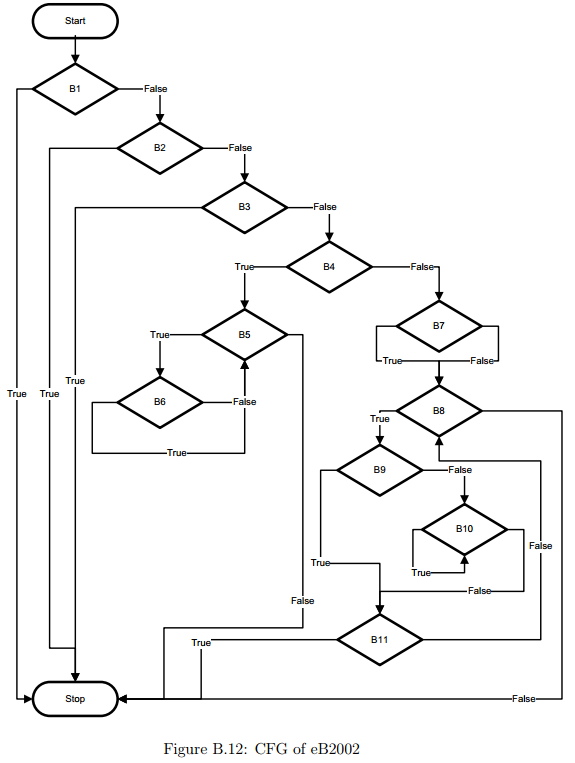
end

disp(‘series failed in expint’);

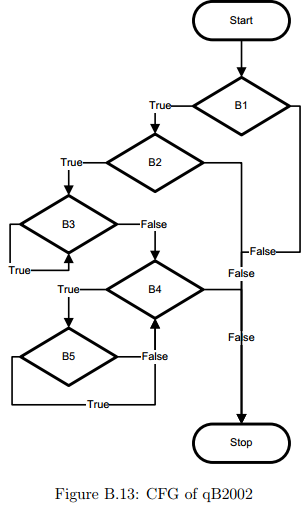
end

end

hasilnya liat aja di halaman 250



1. qB2002

function [q, r] = quotientBueno2002(operands)

n = operands(1); % First number

d = operands(2); % Second number

q = 0;

if (d ~= 0)

if ( (d > 0) && (n > 0) )

q = 0;

r = n;

t = d;

while (r >= t)

t = t \* 2;

end

while (t ~= d)

q = q \* 2;

t = t / 2;

if (t <= r)

r = r - t;

q = q + 1;

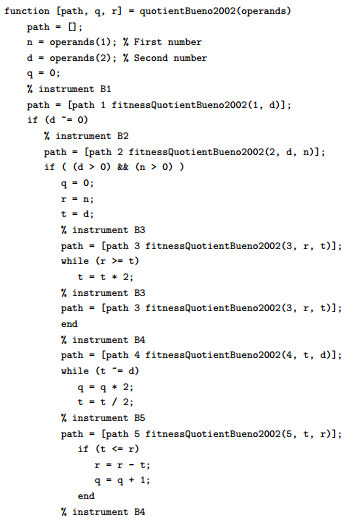
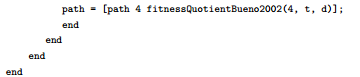
end

end

end

end

end



1. scB2002

function result = strcompBueno2002(strin)

result = ‘ ‘;

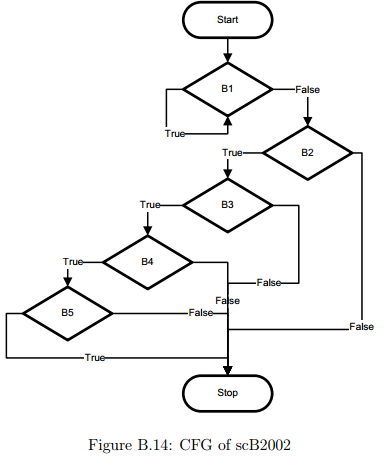
i = 1;

% strin is an array of integers (double) with length 8.

str = char(strin);

while ((str(i) ~= ‘ ‘) && (i <= 5))

i = i + 1;

 end

if (~strcmp(str(1:5),’test1’))

if (str(6) == ‘a’)

if (str(7) == ‘b’)

if (str(8) < ‘c’)

result = ‘Gotcha’;

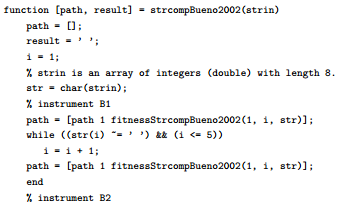
end

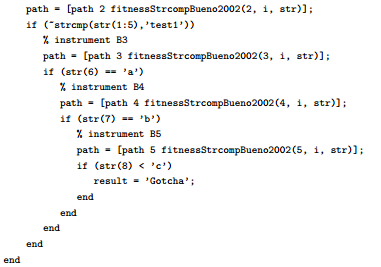
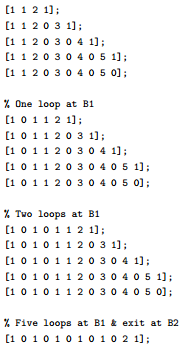
end

end

end

end





1. fcB2002

function result = floatcompBueno2002(floats)

f1 = floats(1); % First number

f2 = floats(2); % Second number

f3 = floats(3); % Third number

result = ‘ ‘;

if (f3 > f2) % B1

if (f2 > f1) % B2

result = ‘f3 > f2 > f1’;

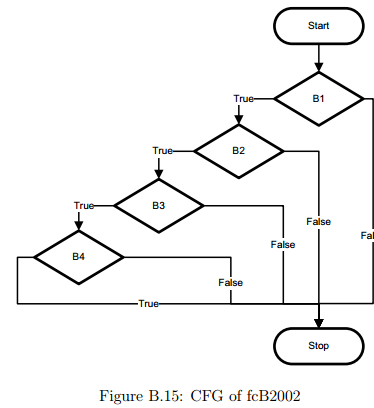
t = f1 + f2;

if (t < f3)

result = ‘f3 > f1 + f2’;

t2 = f1 \* f2;

if (((t2 - f3) <= 5) && ((t2 - f3) >= 0))

 result = ‘(((f1 \* f2) - f3) <= 5) && (((f1 \* f2) - f3) >= 0))’;

end

else

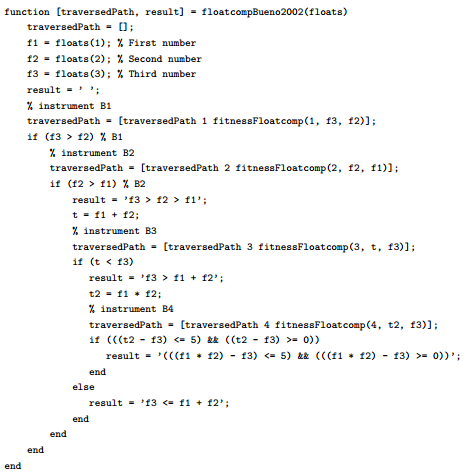
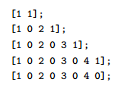
result = ‘f3 <= f1 + f2’;

end

end

end

end



1. fB2002

function a = findBueno2002(numbersIn)

f = numbersIn(1); % key or index

a = numbersIn(2:end); % an array of integers to be re-arranged

% n = length(numbers);

b = 0;

m = 1;

ns = length(a);

% Probe added on 02.09.2010

if f > ns

f = mod(ns,f);

end

i = 1;

while ((m < ns) || b)

if (~b)

i = m;

j = ns;

else

b = 0;

end

if (i > j)

if (f > j)

if (i > f)

m = ns;

else

m = i;

end

else

ns = j;

end

else

while (a(i) < a(f))

i = i + 1

end

while (a(f) < a(j))

j = j - 1 ;

end

if (i <= j)

w = a(i);

a(i) = a(j);

a(j) = w;

i = i + 1;

j = j - 1;

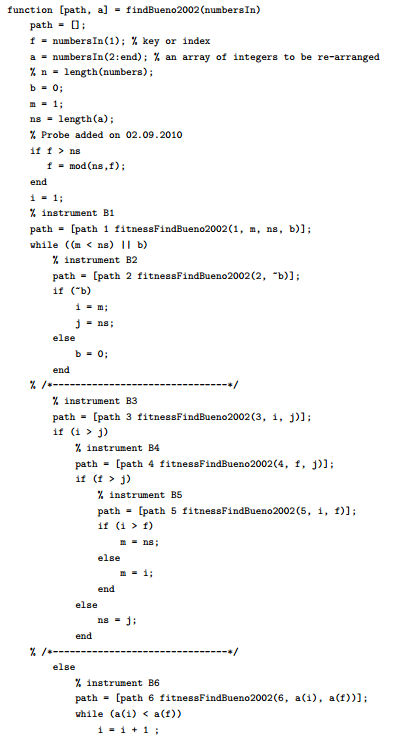
end

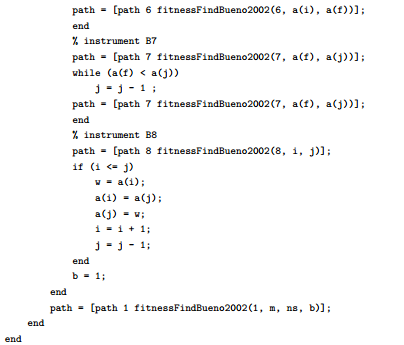
b = 1;

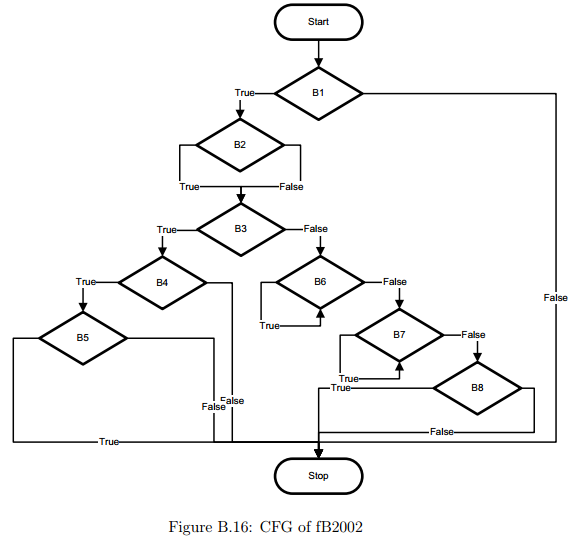
end

end

end







1. bG2011

function pop = bubbleGong2011(depop)

[px,py]=size(depop);

for i=1:px % Branch # 1

q=1;

p=depop(i,:);

for j=1:py-1 % Branch #

for k=j+1:py % Branch # 3

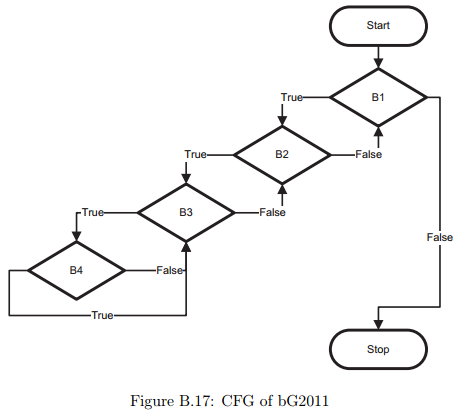
d(q,1)=p(k)-p(j)+0.1;

d(q,1)=1-1.001^(-d(q,1));

d(q,2)=p(j)-p(k);

d(q,2)=1-1.001^(-d(q,2));

if p(j)>p(k) % Branch # 4

 temp=p(j);

p(j)=p(k);

p(k)=temp;

d(q,1)=0;

else

d(q,2)=0;

end

q=q+1;

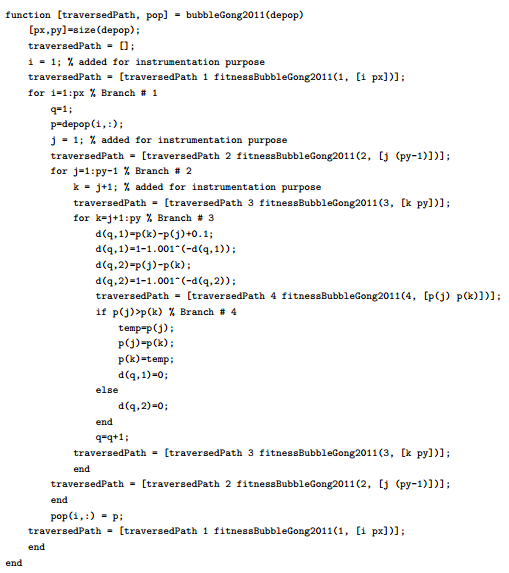
end

end

pop(i,:) = p;

end

end



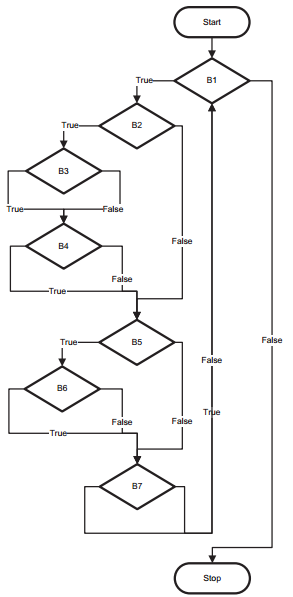
1. fG2011

function pop = flexGong2011(depop)

[px,py]=size(depop);

for i=1:px % Branch # 1

q=1;

 p=depop(i,:);

lex\_compat=p(1);

C\_plus\_plus=p(2);

fulltbl=p(3);

csize =p(4);

unspecified=p(5);

fullspd=p(6);

C\_plus=p(7);

d(1,1)=2;

d(1,1)=1-1.001^(-d(1,1));

d(1,2)=lex\_compat;

d(1,2)=1-1.001^(-d(1,2));

d(1,3)=0;

d(2,1)=2;

d(2,1)=1-1.001^(-d(2,1));

d(2,2)=C\_plus\_plus;

d(2,2)=1-1.001^(-d(2,2));

d(2,3)=0;

d(3,1)=2;

d(3,1)=1-1.001^(-d(3,1));

d(3,2)=fulltbl;

d(3,2)=1-1.001^(-d(3,2));

d(3,3)=0;

d(4,1)=abs( csize-unspecified)+2;

d(4,1)=1-1.001^(-d(4,1));

d(4,2)=2;

d(4,2)=1-1.001^(-d(4,2));

d(4,3)=0;

d(5,1)=2;

d(5,1)=1-1.001^(-d(5,1));

d(5,2)=fullspd;

d(5,2)=1-1.001^(-d(5,2));

d(5,3)=0;

d(6,1)=2;

d(6,1)=1-1.001^(-d(5,1));

d(6,2)=C\_plus;

d(6,2)=1-1.001^(-d(5,2));

d(5,3)=0;

u=3\*ones(1,6);

if (lex\_compat ~= 0) % Branch # 2

d(1,1)=0;

u(1)=1;

if (C\_plus\_plus ~= 0) % Branch # 3

flexerror = ‘Can not use -+ with -l option’;

d(2,1)=0;

else

d(2,2)=0;

end

if (fulltbl ~= 0) % Branch # 4

flexerror=‘Can not use -f or -F with -l option’;

d(3,1)=0;

else

d(3,2)=0;

end

else

d(1,2)=0;

end

if (csize == unspecified) % Branch # 5

d(4,1)=0;

if (fullspd ~= 0) % Branch # 6

csize = ‘DEFAULT\_CSIZE’;

d(5,1)=0;

else

d(5,2)=0;

csize = csize;

end

else

d(4,2)=0;

end

if (C\_plus ~= 0) % Branch # 7

suffix=‘cc’;

d(6,1)=0;

else

d(6,2)=0;

suffix=‘c’;

outfilename = ‘outfile\_path’;

end

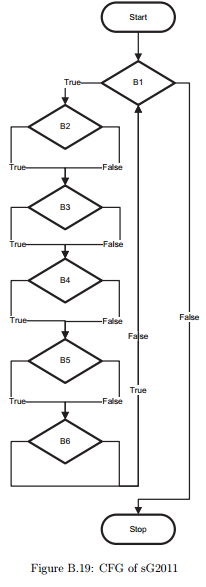
pop(i,:) = p;

end

1. sG2011

% The function accepts a population of 5-number inputs

function pop = spaceGong2011(depop)

 [px,py]=size(depop);

for i=1:px % Branch # 1

q=1;

p=depop(i,:);

unit1=p(1);

unit2=p(2);

unit3=p(3);

error1=p(4);

error2=p(5);

d(1,1)=abs(unit1-1);

d(1,2)=2;

d(1,1)=1-1.001^(-d(1,1));

d(1,2)=1-1.001^(-d(1,2));

d(2,1)=abs(unit2-2);

d(2,1)=1-1.001^(-d(2,1));

d(2,2)=2;

d(2,2)=1-1.001^(-d(2,2));

d(3,1)=abs(unit3-3);

d(3,1)=1-1.001^(-d(3,1));

d(3,2)=2;

d(3,2)=1-1.001^(-d(3,2));

d(4,1)=abs(error1-0);

d(4,1)=1-1.001^(-d(4,1));

d(4,2)=2;

d(4,2)=1-1.001^(-d(4,2));

d(5,1)=abs(error2-0);

d(5,1)=1-1.001^(-d(5,1));

d(5,2)=2;

d(5,2)=1-1.001^(-d(5,2));

u=zeros(1,5);

if unit1 == 1 % Branch # 2

x\_ptr=10;

d(1,1)=0;

else

d(1,2)=0;

end

if unit2 == 2 % Branch # 3

x\_ptr=100;

d(2,1)=0;

else

d(2,2)=0;

end

if unit3 == 3 % Branch # 4

x\_ptr=1000;

d(3,1)=0;

else

d(3,2)=0;

end

if error1 == 0 % Branch # 5

d(4,1)=0;

% return 1

else

d(4,2)=0;

end

if error2 == 0 % Branch # 6

d(5,1)=0;

else

d(5,2)=0;

end

pop(i,:) = p;

end

end

function branchVal = fitnessMiniMaxi(branchNo, predicate)  
k = 1; % the smallest step for integer  
switch (branchNo)  
case 1,  
% branch #1: (idx <= numLength)  
branchVal = predicate(1) - predicate(2);  
case 2,  
% branch #2: (maxi < num(idx))  
branchVal = predicate(1) - predicate(2);  
case 3,  
% branch #3: (mini > num(idx))  
branchVal = predicate(2) - predicate(1);  
end  
if ((branchNo == 2) || (branchNo == 3)),  
if (branchVal < 0)  
branchVal = branchVal - k;  
else  
branchVal = branchVal + k;  
end  
end  
end