# Scilab Textbook Companion for Data Structures Using C And C++ by Y. Langsam, M. Augenstein And A. M. Tenenbaum[1]

Created by
Dharmesh Majethiya
B.Tech (pursuing)
Computer Engineering
NIT Tiruchirappalli
College Teacher
Mr.Kunwar Singh
Cross-Checked by
Siddharth Jain

July 31, 2019

[1]Funded by a grant from the National Mission on Education through ICT, http://spoken-tutorial.org/NMEICT-Intro. This Textbook Companion and Scilab codes written in it can be downloaded from the "Textbook Companion Project" section at the website http://scilab.in

# Book Description

**Title:** Data Structures Using C And C++

**Author:** Y. Langsam, M. Augenstein And A. M. Tenenbaum

**Publisher:** Prentice - Hall Of India Pvt. Ltd.

**Edition:** 2

**Year:** 2006

**ISBN:** 81-203-1177-9

Scilab numbering policy used in this document and the relation to the above book.

**Exa** Example (Solved example)

**Eqn** Equation (Particular equation of the above book)

**AP** Appendix to Example(Scilab Code that is an Appednix to a particular Example of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means a scilab code whose theory is explained in Section 2.3 of the book.

# Contents

# List of Scilab Codes

# Chapter 1

# Introduction To Data Structures

**Scilab code Exa 1.1** To calcualte Average And Deviation

```
1  //Solved Example 1
2  //:To  calcualte Average And Deviation
3  function[avg]=average(a)
4    i=1;
5    [j,k]=size(a);
6    j=0;
7    for i=1:k
8      j=j+a(i);
9    end
10   avg=j/k;
11   dev=0;
12   disp(avg,"Average =");
13   disp("The deviations are:");
14   for i=1:k
15     dev=a(i)-avg;
16     disp(dev);
17   end
18 endfunction
19 //Calling routine
```

```
20 a=[3 223 212 343]
21 avg=average(a)
```

---

**Scilab code Exa 1.1.4** Decimal form of given no represented variably

```
1  //Exercise1.1 Example.1.1.4
2  //To calculate  Decimal No. of a given Number
3  //Treating them as i)Normal binary nos(ii)Twos
      complemented iii)BCD:
4  function[c]=twos1(a1)
5    [j1,i1]=size(a1)
6    i4=1
7    c=-(a1(i4)*2^(i1-1));
8    i1=i1-1;
9    while(i1>=1)
10     i4=i4+1;
11     c=c+a1(i4)*2^(i1-1);
12     i1=i1-1;
13   end
14   disp(a1,"Decimal form of the Twos Complement
        Number");
15   disp(c," is");
16 endfunction
17 function[d]=binary_dec(a2)
18   [j2,i2]=size(a2);
19   k=modulo(i2,4);
20   d=0;
21   if(k==0)
22     e=i2/4;
23     i3=1
24     while(i3<=i2)
25       l=3
26       m=0
27       while(l>=0)
28         m=m+(a2(i3)*2^l);
```

```
29          l=l-1;
30          i3=i3+1;
31        end
32        if(m>9)
33          d=-1;
34          disp("Cannot be coded in this form")
35         break;
36        end
37        if(m<=9)
38          d=d+m*10^(e-1)
39          e=e-1;
40        end
41      end
42    end
43    disp(a2,"Decimal form of  BCD number");
44    disp(d,"is");
45  endfunction
46  //Given Example:
47  //(A)
48  p1=[1 0 0 1 1 0 0 1];
49  p2=base2dec(['10011001'],2)
50  p2=twos1(p1)
51  p2=binary_dec(p1)
52  //(b)
53  p3=[1 0 0 1];
54  p4=base2dec(['1001'],2)
55  p4=twos1(p3)
56  p4=binary_dec(p3)
57  //(C)
58  p5=[0 0 0 1 0 0 0 1 0 0 0 1];
59  p6=base2dec(['000100010001'],2)
60  p6=twos1(p5)
61  p6=binary_dec(p5)
62  //(d)
63  p7=[0 1 1 1 0 1 1 1];
64  p8=base2dec(['01110111'],2)
65  p8=twos1(p7)
66  p8=binary_dec(p7)
```

```
67 //(e)
68 p9=[0 1 0 1 0 1 0 1];
69 p10=base2dec(['01010101'],2)
70 p10=twos1(p9)
71 p10=binary_dec(p9)
72 //(F)
73 p11=[1 0 0 0 0 0 0 1 0 1 0 1];
74 p12=base2dec(['100000010101'],2)
75 p12=twos1(p11)
76 p12=binary_dec(p11)
```

---

**Scilab code Exa 1.1.5** Add Substract And Multiply binary numbers

```
1  //Exercise 1.1 example 1.1.5
2  //Add,Substract And Multiply binary numbers
3  function[a]=add(b,c)
4    d=base2dec(b,2)
5    e=base2dec(c,2)
6    a=d+e
7    a=dec2bin(a)
8    disp(a,"Result of addition")
9  endfunction
10 function[a]=subtract(b,c)
11   d=base2dec(b,2)
12   e=base2dec(c,2)
13   a=d-e
14   a=dec2bin(a)
15   disp(a,"Result of subtraction")
16 endfunction
17 function[a]=multiply(b,c)
18   d=base2dec(b,2)
19   e=base2dec(c,2)
20   a=d*e
21   a=dec2bin(a)
22   disp(a,"Result  of multiplication");
```

```
23 endfunction
24 //Calling Routine:
25 b="11001";
26 c="10011";
27 a=add(b,c)
28 a=subtract(b,c)
29 a=multiply(b,c)
```

**Scilab code Exa 1.1.7** TO Convert Binary To Ternary

```
1  //Exercise 1.1Example 1.1.7
2  //TO Convert Binary To Ternary
3  function[t]=bin_ter(a)
4    b=0
5    b=base2dec(a,2);
6    disp(b);
7    [j,i]=size(a);
8    t=[];
9    while(b~=0)
10     m=modulo(b,3);
11     t=[t(:,:) m];
12     b=b/3;
13     b=b-modulo(b,10);
14   end
15   disp(t,"Ternary Equivalent");
16 endfunction
17 //Calling Routine:
18 a="100101101110"
19 disp(a,"input string is");
20 b=bin_ter(a)
```

**Scilab code Exa 1.2** String Manipulations

```
1  //Solved Example 2
2  //:String Manipulations
3  funcprot(0)
4  function[l]=strlen(str)
5    i=1;
6    l=0;
7    [j,k]=size(str)
8    for  i=1:k
9      l=l+length(str(i));
10   end
11   disp(l,"string length is");
12 endfunction
13 //Calling Routine:
14 str="Hello World";
15 l=strlen(str)
16 function[c]=strcat1(a,b)
17   disp(strcat([a b]),"After concatination");
18   c=strcat([a b]);
19 endfunction
20 //Calling Routine:
21 a="hello ";
22 b="world";
23 c=strcat1(a,b);
```

---

**Scilab code Exa 1.2.1** Calculate Median And Mode Of an Array

```
1  //Exercise Example 1.2.1
2  //Calculates Median And Mode Of an  Array
3  //(A)
4  function[y]=median1(a)
5    p=mtlb_sort(a);
6    [j,i]=size(a);
7    y=0
8    j=modulo(i,2);
9    if(j==0)
```

```
10      y=((a(i/2)+a(i/2+1))/2);
11    end
12    if(j==1)
13      i=i/2;
14      i=i-modulo(i,10);
15      y=a(i+1);
16    end
17    disp(y,"median is");
18  endfunction
19  //(B)
20  function[z]=mode1(a)
21    p=mtlb_sort(a);
22    disp(p)
23    q=1;
24    r=1;
25    i=1;
26    [j,i1]=size(a);
27    if(i1>1)
28      for i=1:i1-1
29        if(p(i)~=p(i+1))
30        q=[q(:,:) i+1];
31        r=[r(:,:) 1];
32        else
33          [c,d]=size(r);
34          r(d)=r(d)+1;
35        end
36      end
37      q1=mtlb_sort(r);
38      [j,i1]=size(q1)
39      if(q1(i1-1)==q1(i1))
40        z=-1;
41        disp("Mode does not exist");
42        break;
43        else
44          c=q1(i1);
45          k=1;
46          while(r(k)~=c)
47            k=k+1;
```

```
48          end
49          z=p(q(k));
50        end
51      end
52      if(i1==1)
53        z=a(1);
54      end
55      disp(z,"mode is");
56  endfunction
57  a=[223 12 233322 121]
58  y=median1(a);
59  z=mode1(a);
```

---

**Scilab code Exa 1.2.6** Finding the adress in a row major array

```
1  //Exercise1.2 Example 1.2.6
2  //Finding the adress in a row major array
3  function[]=add(m,n)
4    printf("Adress is %d\n",m+n*20);
5  endfunction
6
7  //(a)
8  add(10,0);
9  //(b)
10 add(100,0);
11 //(c)
12 add(0,0);
13 //(d)
14 add(2,1);
15 //(e)
16 add(5,1);
17 //(f)
18 add(1,10);
19 //(g)
20 add(2,10);
```

```
21 //(h)
22 add(5,3);
23 //(i)
24 add(9,19);
```

---

**Scilab code Exa 1.3** Writing name from structure and counting alphabets

```
1  //Solved Example 5:
2  //Writing a name from the given structure and
3  //counting the number of alphabets printed
4  function[l]=strlen(str)
5    i=1;
6    l=0;
7    [j,k]=size(str)
8    for  i=1:k
9      l=l+length(str(i));
10   end
11 endfunction
12 function[count]=writename(name)
13   printf("\n");
14   printf("%s",name.first);
15   printf("%c",'  ');
16   printf("%s",name.midinit);
17   printf("\t");
18   printf("%s",name.last);
19   printf("\n");
20
21   a=string(name.first);
22   count=strlen(a);
23   a=string(name.midinit);
24   count=count+strlen(a);
25   a=string(name.last);
26   count=count+strlen(a);
27   disp(count,"Count is:");
28 endfunction
```

```
29 //Calling Routine
30 name=struct('first','praveen','midinit','rajeev','
      last','chauhan');
31 count=writename(name)
```

**Scilab code Exa 1.3.1** Implementing Complex Numbers by structure

```
1  //Exercise 1.3
2  //Example 1.3.1
3  //Implementing Complex Numbers by structure
4  function[]=complexmanu(x1,x2,x3,x4)
5
6    com1=struct('real',x1,'complex',x2);
7    com2=struct('real',x3,'complex',x4);
8     //adding 2 numbers
9     add=struct('real',x1+x3,'complex',x2+x4);
10    disp(add.complex,"+ i",add.real,"Addition result
         is ");
11    //Substract
12    sub=struct('real',x1-x3,'complex',x2-x4);
13    disp(sub.complex,"+ i",sub.real,"Substraction
         result is ");
14    //Negating
15    neg=struct('real',-x1,'complex',-x2);
16    disp(neg.complex,"+ i",neg.real,"Negation result
         for the first is ");
17    //Multiplication
18    mul=struct('real',x1*x3-x2*x4,'complex',x2*x3+x4*
         x1);
19     disp(mul.complex,"+ i",mul.real,"Multiplication
          result is ");
20  endfunction
21  x1=3;
22  x2=5;
23  x3=5;
```

```
24  x4=6;
25  complexmanu(x1,x2,x3,x4);
```

**Scilab code Exa 1.3.6** Adding Substracting and multiplying Rational Nos

```
1  //Exercise 1.3
2  //Example 1.3.6
3  //Adding,Subtracting and multiplying Rational
       Numbers
4  function[]=rational(x1,x2,x3,x4)
5  rational1=struct('numerator',x1,'denominator',x2);
6  disp(rational1);
7  rational2=struct('numerator',x3,'denominator',x4);
8  disp(rational2);
9  //Add
10 x5=int32([x2 x4]);
11 x5=lcm(x5);
12 x6=x1*(x5/x2)+x3*(x5/x4);
13 rational3=struct('numerator',x6,'denominator',x5);
14 disp(rational3,"After addition");
15 //subtract
16 x6=x1*(x5/x2)-x3*(x5/x4)
17 rational4=struct('numerator',x6,'denominator',x5);
18 disp(rational4,"After Subtraction");
19 //Multiply
20 x7=x1*x3;
21 x8=x2*x4;
22 rational5=struct('numerator',x7,'denominator',x8);
23 disp(rational5,"After multiplication");
24 endfunction
25 x1=43;
26 x2=32;
27 x3=233;
28 x4=33;
29 rational(x1,x2,x3,x4);
```

**Scilab code Exa 1.3.7** Checking Equality Of 2 Rational Numbers

```
1 //Exercise 1.3
2 //Example 1.3.7
3 //Checking Equality Of 2 Rational Numbers Without
     Reducing Them
4 function[]=rational_equal(x1,x2,x3,x4)
5 rational1=struct('numerator',x1,'denominator',x2);
6 disp(rational1);
7 rational2=struct('numerator',x3,'denominator',x4);
8 disp(rational2);
9 if(x1*x4==x2*x3)
10   disp("Equal");
11   break;
12 else
13   disp("Not  Equal");
14   break;
15 end
16 endfunction
17 //Calling Routine:
18 x1=32;
19 x2=45;
20 x3=43;
21 x4=55;
22 rational_equal(x1,x2,x3,x4);
```

**Scilab code Exa 1.4** Raising the salary of employee

```
1 //Solved Example 6
2 //To Raise The salary of an employee
3 function[employee1]=raise(employee,n)//employee is
     the list of employees
```

```
4     for i=1:n
5       if(employee(i)(1).year<=2000)
6         employee(i)(2)=employee(i)(2)*1.1;
7       else
8         employee(i)(2)=employee(i)(2)*1.05;
9       end
10    end
11    employee1=employee;
12    disp("After Raising");
13    for i=1:n
14      printf("Employee no %d\n",i);
15      disp(employee(i)(1));
16      disp(employee(i)(2));
17    end
18
19  endfunction
20  //Calling Routine:
21  datehired=struct('year',1993,'month',12);
22  employee1=list(datehired,14000);
23  datehired=struct('year',1998,'month',12);
24  employee2=list(datehired,17000);
25  datehired=struct('year',2003,'month',12);
26  employee3=list(datehired,25000);
27  datehired=struct('year',2002,'month',12);
28  employee4=list(datehired,35000);
29  datehired=struct('year',2006,'month',12);
30  employee5=list(datehired,13000);
31  employee=list(employee1,employee2,employee3,
      employee4,employee5);
32  employee=raise(employee,5)
```

**Scilab code Exa 1.5** Reducing the given rational number

```
1  //Solved Example 7:
2  //Reducing The Given Rational Number
```

```
funcprot(0)
function[y]=reduce(nm,dn)
rational1=struct('numerator',nm,'denominator',dn)
y=0
if(rational1.numerator>rational1.denominator)
  a=rational1.numerator;
  b=rational1.denominator;
else
  a=rational1.denominator;
  b=rational1.numerator;
end
while(b~=0)
  rem=modulo(a,b);
  a=b;
  b=rem;
end
y=struct('numerator',nm/a,'denominator',dn/a);
disp(y);
endfunction
nm=22;
dn=44;
y=reduce(nm,dn)
```

**Scilab code Exa 1.6** Equality check of 2 rational nos by reduction

```
//Solved Example 8:
//Checking for the equality of 2 rational numbers by
    reducing them
function[]=equal(x1,x2,x3,x4)
  rational1=struct('numerator',x1,'denominator',x2)
 rational2=struct('numerator',x3,'denominator',x4)
 y=0
if(rational1.numerator>rational1.denominator)
  a=rational1.numerator;
  b=rational1.denominator;
```

```
10 else
11   a=rational1.denominator;
12   b=rational1.numerator;
13 end
14 while(b~=0)
15   rem=modulo(a,b);
16   a=b;
17   b=rem;
18 end
19 y=struct('numerator',x1/a,'denominator',x2/a);
20 y1=0
21 if(rational2.numerator>rational2.denominator)
22   a=rational2.numerator;
23   b=rational2.denominator;
24 else
25   a=rational2.denominator;
26   b=rational2.numerator;
27 end
28 while(b~=0)
29   rem=modulo(a,b);
30   a=b;
31   b=rem;
32 end
33 y1=struct('numerator',x3/a,'denominator',x4/a);
34 if(y==y1)
35   disp("Equal")
36   break;
37 else
38   disp("Not  Equal")
39   break;
40 end
41 endfunction
42 x1=5;
43 x2=7;
44 x3=35;
45 x4=49;
46 equal(x1,x2,x3,x4);
```

# Chapter 2

# Stacks

**Scilab code Exa 2.1** To determine the syntacticaly valid string

```
1  //Solved Example 1
2  //To  determine the syntacticaly valid string
3  function[l]=strlen(x)
4    i=1;
5    l=0;
6    [j,k]=size(x)
7    for  i=1:k
8      l=l+length(x(i));
9    end
10 endfunction
11 function[]=stringvalid(str)
12   str=string(str);
13   stack=struct('a','0','top',0);
14   l1=strlen(str);
15   valid=1;
16   l=1;
17   while(l<=l1)
18       if(str(l)=='('|str(l)=='['|str(l)=='{')
19         if(stack.top==0)
20           stack.a=str(l);
21           stack.top=stack.top+1;
```

```
22          else
23            stack.a=[stack.a(:,:) str(l)];
24            stack.top=stack.top+1;
25          end
26        end
27        if(str(l)==')'|str(l)==']'|str(l)=='}')
28          if(stack.top==0)
29            valid=0;
30            break;
31          else
32           i=stack.a(stack.top);
33           stack.top=stack.top-1;
34           symb=str(l);
35           if(((symb==')')&(i=='('))|((symb==']')&(i==
                 '['))|((symb=='}')&(i=='{')))
36          else
37            valid=0;
38            break;
39          end
40        end
41      end
42        l=l+1;
43      end
44      if(stack.top~=0)
45        valid=0;
46      end
47      if(valid==0)
48        disp("Invalid String");
49      else
50        disp("Valid String");
51      end
52    endfunction
53    //Calling Routine:
54  stringvalid(['H' 'E' 'L' 'L' 'O'])
```

**Scilab code Exa 2.1.2** To determine the syntacticaly valid string

```
1  //Solved Example 1
2  //To determine the syntacticaly valid string
3  function[l]=strlen(x)
4    i=1;
5    l=0;
6    [j,k]=size(x)
7    for  i=1:k
8      l=l+length(x(i));
9    end
10 endfunction
11 function[]=stringvalid(str)
12   str=string(str);
13   stack=struct('a','0','top',0);
14   l1=strlen(str);
15   valid=1;
16   l=1;
17   while(l<=l1)
18       if(str(l)=='('|str(l)=='['|str(l)=='{')
19         if(stack.top==0)
20           stack.a=str(l);
21           stack.top=stack.top+1;
22         else
23           stack.a=[stack.a(:,:) str(l)];
24           stack.top=stack.top+1;
25         end
26         disp(stack);
27       end
28       if(str(l)==')'|str(l)==']'|str(l)=='}')
29         if(stack.top==0)
30           valid=0;
31           break;
32         else
33           i=stack.a(stack.top);
34           b=stack.a(1);
35           for i1=2:stack.top-1
36             b=[b(:,:) stack.a(i1)]
```

```
37            end
38            stack.a=b;
39           stack.top=stack.top-1;
40           symb=str(l);
41           disp(stack);
42           if(((symb==')')&(i=='('))|((symb==']')&(i=='['))|((symb=='}')&(i=='{')))
43          else
44            valid=0;
45            break;
46          end
47        end
48      end
49        l=l+1;
50      end
51      if(stack.top~=0)
52        valid=0;
53      end
54      if(valid==0)
55        disp("Invalid String");
56      else
57        disp("Valid String");
58      end
59    endfunction
60    //Calling Routine:
61    stringvalid(['(' 'A' '+' 'B' '}' ')'])
62    stringvalid(['{' '[' 'A' '+' 'B' ']' '-' '[' '(' 'C' '-' 'D' ')' ']'])
63    stringvalid(['(' 'A' '+' 'B' ')' '-' '{' 'C' '+' 'D' '}' '-' '[' 'F' '+' 'G' ']'])
64    stringvalid(['(' '(' 'H' ')' '*' '{' '(' '[' 'J' '+' 'K' ']' ')' '}' ')'])
65    stringvalid(['(' '(' '(' 'A' ')' ')' ')'])
```

**Scilab code Exa 2.2** Implementing Stack using union

```
1  //Solved Example 2:
2  //Implementing Stack using union:
3  function[stack]=sta_union(etype,a)
4    stackelement=struct('etype',etype);
5    [k,l]=size(a);
6  select stackelement.etype,
7  case 'int' then
8    a=int32(a);
9    stack=struct('top',l,'items',a);,
10   case 'float' then
11   a=double(a);
12   stack=struct('top',l,'items',a);,
13   case 'char' then
14   a=string(a);
15   stack=struct('top',l,'items',a);,
16 end
17 disp(stack,"Stack is:");
18 endfunction
19 a=[32 12.34 232 32.322]
20 stack=sta_union('float',a)
21 stack=sta_union('int',a)
22 stack=sta_union('char',a)
```

---

**Scilab code Exa 2.2.3** Check if string is of certain form

```
1  function[l]=strlen(x)
2    i=1;
3    l=0;
4    [j,k]=size(x)
5    for  i=1:k
6      l=l+length(x(i));
7    end
8  endfunction
9  function[]=str(st)
10   stack=struct('a',0,'top',0);
```

```
11    st=string(st);
12    l=1;
13    l1=strlen(st);
14    symb=st(l);
15    valid=1;
16    while(l<l1)
17      while(symb~='C')
18        if(stack.top==0)
19            stack.a=st(l);
20            stack.top=stack.top+1;
21         else
22            stack.a=[stack.a(:,:) st(l)];
23            stack.top=stack.top+1;
24         end
25         l=l+1;
26         symb=st(l);
27       end
28       i=st(l+1);
29      if(stack.top==0)
30          valid=0;
31          break;
32       else
33         symb1=stack.a(stack.top);
34         stack.top=stack.top-1;
35         if(i~=symb1)
36           valid=0;
37           break;
38         end
39       end
40      l=l+1;
41    end
42      if(stack.top~=0)
43        valid=0;
44      end
45      if(valid==0)
46        disp("Not of the given format");
47      else
48        disp("String Of the Given Format");
```

```
49      end
50  endfunction
51  //Calling Routine:
52  st=['A' 'A' 'B' 'A' 'C' 'A' 'B' 'A' 'A']
53  str(st)
54  st=['A' 'A' 'B' 'A' 'C' 'A' 'B' 'A' ]
55  str(st)
```

---

**Scilab code Exa 2.3** Implementing Push And Pop Functions

```
1  //Solved Example 3:
2  //Implementing Push And Pop Functions:
3  function[y,sta1]=empty(sta)
4    y=0;
5    sta1=0;
6    if(sta.top==0)
7      y=0;
8    else
9      y=1;
10   end
11   sta1=sta
12 endfunction
13
14 function[sta]=push(stac,ele)
15   sta=0;
16   if(empty(stac)==0)
17     stac.a=ele;
18     stac.top=stac.top+1;
19   else
20     stac.a=[stac.a(:,:) ele]
21     stac.top=stac.top+1;
22   end
23   disp(stac);
24   sta=stac;
25 endfunction
```

```
26
27 function[ele,sta]=pop(stack)
28   ele='-1';
29   if(empty(stack)==0)
30     disp("Stack Underflow");
31     break;
32   else
33     ele=stack.a(stack.top);
34     stack.top=stack.top-1;
35     if(stack.top~=0)
36       b=stack.a(1);
37     for i2=2:stack.top
38       b=[b(:,:) stack.a(i2)];
39     end
40     stack.a=b;
41   else
42     stack.a='0';
43   end
44   end
45   disp(stack);
46   sta=stack;
47 endfunction
48 global stack
49 //Calling Routine:
50 stack=struct('a',0,'top',0);
51 stack=push(stack,4);
52 stack=push(stack,55);
53 stack=push(stack,199);
54 stack=push(stack,363);
55 [ele,stack]=pop(stack);
56 disp(stack,"After the above operations stack is:");
```

**Scilab code Exa 2.4** Convering an infix expression to a Postfix Express

```
1 //Solved Example 5:
```

```
2  //Convering an infix expression to a Postfix
       Expression:
3  function[sta]=push(stac,ele)
4    sta=0;
5    if(stac.top==0)
6      stac.a=ele;
7      stac.top=stac.top+1;
8    else
9      stac.a=[stac.a(:,:) ele]
10     stac.top=stac.top+1;
11   end
12   disp(stac);
13   sta=stac;
14 endfunction
15
16 function[ele,sta]=pop(stack)
17   ele='-1';
18   if(stack.top==0)
19     disp("Stack Underflow");
20     break;
21   else
22     ele=stack.a(stack.top);
23     stack.top=stack.top-1;
24     if(stack.top~=0)
25       b=stack.a(1);
26     for i2=2:stack.top
27       b=[b(:,:) stack.a(i2)];
28     end
29     stack.a=b;
30   else
31     stack.a='0';
32   end
33   end
34   sta=stack;
35 endfunction
36 function[l]=strlen(x)
37   i=1;
38   l=0;
```

```
39    [j,k]=size(x)
40    for  i=1:k
41      l=l+length(x(i));
42    end
43  endfunction
44  function[p]=pre(s1,s2)
45       i1=0;
46      select s1,
47      case '+' then i1=5;
48      case '-' then i1=5;
49      case '*' then  i1=9;
50      case '/' then i1=9;
51      end
52      i2=0;
53      select s2,
54      case '+' then i2=5;
55      case '-' then i2=5;
56      case '*' then  i2=9;
57      case '/' then i2=9;
58      end
59      p=0;
60      p=i1-i2;
61      if(s1=='(')
62        p=-1;
63      end
64      if(s2=='('&s1~=')')
65        p=-1;
66      end
67      if(s1~='('&s2==')')
68        p=1;
69      end
70
71    endfunction
72  function[a2]=intopo(a1,n)
73    stack=struct('a',0,'top',0);
74    l1=1;
75    l2=strlen(a1(1))
76    for i=2:n
```

```
77      l2=l2+strlen(a1(i))
78    end
79    a2=list();
80    while(l1<=l2)
81      symb=a1(l1);
82      if(isalphanum(string(a1(l1))))
83        a2=list(a2,symb);
84      else
85        while(stack.top~=0&(pre(stack.a(stack.top),
             symb)>=0))
86          [topsymb,stack]=pop(stack);
87          if(topsymb==')'|topsymb=='(')
88            a2=a2;
89          else
90            a2=list(a2,topsymb);
91          end
92        end
93        if(stack.top==0|symb~=')')
94          stack=push(stack,symb);
95        else
96          [ele,stack]=pop(stack);
97        end
98      end
99      l1=l1+1;
100   end
101   while(stack.top~=0)
102     [topsymb,stack]=pop(stack);
103     if(topsymb==')'|topsymb=='(')
104       a2=a2;
105     else
106       a2=list(a2,topsymb);
107     end
108   end
109   disp(a2);
110 endfunction
111 //Calling Routine:
112 a1=['(' '2' '+' '3' ')' '*' '(' '5' '-' '4' ')']
113 a2=intopo(a1,11)
```

# Chapter 3

# Recursion

**Scilab code Exa 3.1** Multiplication of 2 numbers

```
1 //Multiplication of 2 numbers
2 funcprot(0)
3 function[val]=mul(a,b)
4   if(b==1)
5     val=a;
6   else
7     val=a+mul(a,b-1);
8   end
9 endfunction
10 //Calling Routine:
11 a=4;
12 b=3;
13 val=mul(4,3)
14 printf("Product of %d and %d is %d",a,b,val);
```

**Scilab code Exa 3.2** Factorial of a number

```
1 //Function To Caluculate factorial of a given
    number
```

```
2  function[value]=fact(a)
3    value=-1;
4    if(a<0|a>170)
5      disp("Invalid valu.");
6      break;
7    else
8      if(a==1|a==0)
9        value=1;
10     else
11       value=a*fact(a-1);
12     end
13   end
14 endfunction
15 //Calling Routine:
16 a=5;
17 val=fact(a);
18 printf("%d factorial is %d",a,val);
```

**Scilab code Exa 3.3** Fibbonacci series

```
1  function[fib]=fibbo(n)
2    fib=-1;
3    if(n<0)
4      disp("Invalid Entry");
5    else
6     if(n<=1)
7       fib=n;
8     else
9       fib=fibbo(n-1)+fibbo(n-2);
10    end
11  end
12 endfunction
13
14 function[l]=fibbon(n)
15   x=0;
```

```
16    l=(fibbo(0));
17    for x=1:n-1
18     l=[l(:,:),fibbo(x)];
19    end
20   disp(l);
21  endfunction
22  //Calling Routine:
23  n=5;
24  l=fibbon(n)
```

**Scilab code Exa 3.4** Binary Search

```
1   function[b]=bsear(a,l,u,n)
2     if(l>u)
3       b=-1;
4     else
5       mid=int32((l+u)/2);
6       if(n==a(mid))
7         b=n;
8       else
9         if(n>a(mid))
10           mid=int32((l+u)/2);
11          b=bsear(a,mid+1,u,n);
12        else
13           mid=int32((l+u)/2);
14          b=bsear(a,l,mid-1,n);
15        end
16      end
17    end
18  endfunction
19
20  function[b]=bsearc(a,l,u,n)
21    b=bsear(a,l,u,n);
22    if(b==-1)
23      disp("The element is not there");
```

```
24    end
25    if(b==n)
26      disp("The element is there");
27    end
28  endfunction
29  //Calling Routine:
30  a=[12 122 3233 12121]//Must be sorted:
31  b=bsearc(a,1,4,12)
```

**Scilab code Exa 3.5** Tower Of Hanoi

```
1  function[]=towe(n,from,to,aux)
2    if(n==1);
3      disp(to,"to ",from,"Move peg 1 from");
4    else
5      towe(n-1,from,aux,to);
6      disp(to,"to",from,"from",n,"Move Peg");
7      towe(n-1,aux,to,from);
8    end
9  endfunction
10
11 function[]=tower(from,to,aux)
12   n=input("Enter n");
13   towe(n,from,to,aux);
14 endfunction
15 //Calling Routine:
16 n=3//Number of disks
17 towe(n,'a','b','c')
```

**Scilab code Exa 3.6** Prefix To Postfix Conversion

```
1  funcprot(0)
2  function[y]=find1(g)
```

```
3    length1=strlen(g);
4    if(length1==0)
5      y=0;
6    else
7      if(isalphanum(g(1)))
8        y=1;
9      else
10       if(length1<2)
11         y=0;
12       else
13          s=strsplit(g,1);
14          s=s(2);
15          m=find1(s);
16          if(m==0|length1==m)
17            y=0;
18          else
19             e=strsplit(g,m+1);
20             e=e(2);
21             n=find1(e);
22             if(n==0)
23               y=0;
24             else
25               y=m+n+1;
26             end
27           end
28         end
29       end
30     end
31   endfunction
32   function[l]=strlen(x)
33   i=1;
34   l=0;
35   [j,k]=size(x)
36   for  i=1:k
37     l=l+length(x(i));
38   end
39 endfunction
40 function[po]=pr2po(pr)
```

```
41    length1=strlen(pr);
42    if(length1==1)
43      if(isalphanum(pr))
44        po(1)=pr(1);
45      else
46        disp("Invalid string\n");
47      end
48    else
49      s=strsplit(pr,1);
50      g=s(2);
51      m=find1(g);
52      s=strsplit(pr,m+1);
53      g1=s(2);
54      n=find1(g1);
55      f=strsplit(pr,1);
56      c=f(1);
57      if((c~='+'&c~='-'&c~='/'&c~='*')|m==0|n==0|m+n
           +1~=length1)
58        printf("Invalid string\n");
59      else
60        s=strsplit(pr,1);
61        s=strsplit(s(2),m);
62        opnd1=s(1);
63        s=strsplit(pr,m+1);
64        opnd2=s(2);
65        post1=pr2po(opnd1);
66        post2=pr2po(opnd2);
67        post=[post1(:,:) post2(:,:)]
68        f=strsplit(pr,1);
69        c=f(1);
70        post3=[post(:,:) c];
71        po=post3;
72       end
73     end
74  endfunction
75  //Calling Routine:
76
77  s1="+-*abcd";//no spaces between
```

```
78  po=pr2po(s1);
79  disp(po,"postfix is");
80  s1="+-*/+-*/abcdefghi"
81  po=pr2po(s1);
82  disp(po,"postfix is");
```

**Scilab code Exa 3.7** Simulating Factorial By Non recursion

```
1
2  function[]=simu_fact(n);
3    a=1;
4    while(n>0)
5      a=a*n;
6      n=n-1;
7    end
8    disp(a,"Factorial is ");
9  endfunction
10 //Calling Routine:
11 a=9
12 simu_fact(a)
```

# Chapter 4

# Queues and linked list

**Scilab code Exa 4.1** Implementing Singly Connected Linked List

```
1  //SINGLY CONNECTED LINKED LIST:
2  function[link2]=append(ele,link1)
3    link2=list
        (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
        ;
4    if(link1(1)(1).add==0)
5      link1(1)(1).data=ele;
6      link1(1)(1).add=1;
7      link1(1)(1).nexadd=0;
8      link2(1)=link1(1)(1);
9      else
10     if(link1(1)(1).nexadd==0)
11       lin2=link1(1)(1);
12       lin2.data=ele;
13       lin2.add=link1(1)(1).add+1;
14       link1(1)(1).nexadd=lin2.add;
15       lin2.nexadd=0;
16       link2(1)=link1(1)(1);
17       link2(2)=lin2;
18     else
19       lin2=link1(1)(1);
```

```
20        i=1;
21        while(link1(i)(1).nexadd~=0)
22          i=i+1;
23        end
24        j=i;
25        lin2.data=ele;
26        lin2.add=link1(i).add+1;
27        lin2.nexadd=0;
28        link1(i).nexadd=lin2.add;
29        link2(1)=link1(1)(1);
30        i=2;
31        while(link1(i).nexadd~=lin2.add)
32          link2(i)=(link1(i));
33          i=i+1;
34        end
35        link2(i)=link1(i);
36        link2(i+1)=lin2;
37      end
38    end
39  endfunction
40  function[link2]=add(ele,pos,link1);
41      link2=list
           (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
           ;
42      i=1;
43      while(i<=pos)
44        if(link1(i).nexadd==0)
45          break;
46        else
47          i=i+1;
48        end
49      end
50      if(link1(i).nexadd~=0)
51        i=i-1;
52         lin2.data=ele;
53         lin2.add=i;
54         j=i;
55         while(link1(j).nexadd~=0)
```

```
56          link1(j).add=link1(j).add+1;
57          link1(j).nexadd=link1(j).nexadd+1;
58          j=j+1;
59        end
60        link1(j).add=link1(j).add+1;
61        lin2.nexadd=link1(i).add;
62        link1(i-1).nexadd=lin2.add;
63        k=1;
64        while(k<i)
65           link2(k)=link1(k);
66           k=k+1;
67         end
68         link2(k)=lin2;
69         k=k+1;
70         link2(k)=link1(k-1);
71         k=k+1
72         l=k-1;
73         while(k~=j)
74           link2(k)=link1(l);
75           k=k+1;
76           l=l+1;
77         end
78         link2(j)=link1(j-1);;
79         link2(j+1)=link1(j);
80       else
81         if(i==pos&i~=1)
82           k=1;
83           lin2.data=ele;
84           lin2.add=link1(i-1).add+1;
85           link1(i).add=link1(i).add+1;
86           lin2.nexadd=link1(i).add;
87           k=1;
88           while(k<pos)
89             link2(k)=link1(k);
90             k=k+1;
91           end
92           link2(k)=lin2;
93           link2(k+1)=link1(k)
```

```
94          end
95          if(i==pos&i==1)
96            link2=append(ele,link1);
97            return link2;
98          end
99        end
100 endfunction
101 function[link2]=delete1(pos,link1)
102   link2=list
        (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
        ;
103   i=1;
104   while(i<=pos)
105     if((link1(i).nexadd==0))
106       break;
107     else
108       i=i+1;
109     end
110   end
111   if(link1(i).nexadd~=0)
112     i=i-1;
113     j=1;
114     if(i==1)
115       j=1;
116       while(link1(j).nexadd~=0)
117         link2(j)=link1(j);
118         j=j+1;
119       end
120       link2(j)=link1(j);
121     else
122     link1(i-1).nexadd=link1(i+1).add;
123     while(link1(j).nexadd~=link1(i+1).add)
124       link2(j)=link1(j);
125       j=j+1;
126     end
127     if(j~=i-1)
128       link2(j)=link1(j);
129       link2(j+1)=link1(j+1);
```

```
130         k=i+1;
131         l=2;
132       else
133         link2(j)=link1(j);
134         k=i+1;
135         l=1;
136       end
137       while(link1(k).nexadd~=0)
138         link2(j+l)=link1(k);
139         k=k+1;
140         l=l+1;
141       end
142       link2(j+l)=link1(k);
143     end
144     else
145       if(i==pos)
146         j=1;
147         link1(i-1).nexadd=0;
148         while(j<=i-1)
149           link2(j)=link1(j);
150           j=j+1;
151         end
152       end
153     end
154 endfunction
155
156
157
158 //Calling Routine:
159 link1=struct('data',0,'add',0,'nexadd',0);//Creates
      empty list
160 link1=append(4,link1)
161 link1=append(6,link1)
162 link1=add(7,2,link1)
163 link1=append(8,link1)
164 link1=delete1(4,link1)
165 disp(link1,"The linked list after the above
      modifications is:");
```

**Scilab code Exa 4.2** Implementing Queue Operarions

```
1  //Queue  Operations
2  function[q2]=push(ele,q1)
3    if(q1.rear==q1.front)
4      q1.a=ele;
5      q1.rear=q1.rear+1;
6    else
7      q1.a=[q1.a(:,:) ele];
8      q1.rear=q1.rear+1;
9    end
10   q2=q1;
11 endfunction
12 function[ele,q2]=pop(q1)
13   ele=-1;
14   q2=0;
15   if(q1.rear==q1.front)
16     disp("Queue Underflow");
17     return;
18   else
19     ele=q1.a(q1.rear-q1.front);
20     q1.front=q1.front+1;
21     i=1;
22     a=q1.a(1);
23     for i=2:(q1.rear-q1.front)
24       a=[a(:,:) q1.a(i)];
25     end
26     q1.a=a;
27   end
28   q2=q1;
29 endfunction
30 //Calling Routine:
31 q1=struct('a',0,'rear',0,'front',0)
32 q1=push(3,q1)
```

```
33 q1=push(22,q1);
34 q1=push(21,q1);
35 disp(q1,"Queue after insertion");
36 [ele,q1]=pop(q1)
37 disp(ele,"poped element");
38 disp(q1,"Queue after poping");
39 [ele,q1]=pop(q1);
40 [ele,q1]=pop(q1);
41 [ele,q1]=pop(q1);//Underflow Condition
```

**Scilab code Exa 4.3** Implementing Circular Linked List

```
1  //CIRCULAR LINKED LIST
2  function[link2]=append(ele,link1)
3    link2=list
        (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
        ;
4    if(link1(1)(1).add==0)
5      link1(1)(1).data=ele;
6      link1(1)(1).add=1;
7      link1(1)(1).nexadd=1;
8      link2(1)=link1(1)(1);
9      else
10     if(link1(1)(1).nexadd==link1(1)(1).add)
11       lin2=link1(1)(1);
12       lin2.data=ele;
13       lin2.add=link1(1)(1).add+1;
14       link1(1)(1).nexadd=lin2.add;
15       lin2.nexadd=link1(1)(1).add;
16       link2(1)=link1(1)(1);
17       link2(2)=lin2;
18     else
19       lin2=link1(1)(1);
20       i=1;
21       while(link1(i)(1).nexadd~=link1(1)(1).add)
```

```
22          i=i+1;
23        end
24        j=i;
25        lin2.data=ele;
26        lin2.add=link1(i).add+1;
27        lin2.nexadd=link1(1)(1).add;
28        link1(i).nexadd=lin2.add;
29        link2(1)=link1(1)(1);
30        i=2;
31        while(link1(i).nexadd~=lin2.add)
32          link2(i)=(link1(i));
33          i=i+1;
34        end
35        link2(i)=link1(i);
36        link2(i+1)=lin2;
37      end
38    end
39  endfunction
40  function[link2]=add(ele,pos,link1);
41     link2=list
          (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
          ;
42     i=1;
43     while(i<=pos)
44       if(link1(i).nexadd==link1(1)(1).add)
45         break;
46       else
47         i=i+1;
48       end
49     end
50     if(link1(i).nexadd~=link1(1)(1).add)
51       i=i-1;
52        lin2.data=ele;
53        lin2.add=i;
54        j=i;
55        while(link1(j).nexadd~=link1(1)(1).add)
56          link1(j).add=link1(j).add+1;
57          link1(j).nexadd=link1(j).nexadd+1;
```

```
58          j=j+1;
59        end
60        link1(j).add=link1(j).add+1;
61        lin2.nexadd=link1(i).add;
62        link1(i-1).nexadd=lin2.add;
63        k=1;
64        while(k<i)
65           link2(k)=link1(k);
66           k=k+1;
67         end
68         link2(k)=lin2;
69         k=k+1;
70         link2(k)=link1(k-1);
71         k=k+1
72         l=k-1;
73         while(k~=j)
74           link2(k)=link1(l);
75           k=k+1;
76           l=l+1;
77         end
78         link2(j)=link1(j-1);;
79         link2(j+1)=link1(j);
80       else
81         if(i==pos)
82           k=1;
83           lin2.data=ele;
84           lin2.add=link1(i-1).add+1;
85           link1(i).add=link1(i).add+1;
86           lin2.nexadd=link1(i).add;
87           link1(i).nexadd=link1(1)(1).add;
88           k=1;
89           while(k<pos)
90             link2(k)=link1(k);
91             k=k+1;
92           end
93           link2(k)=lin2;
94           link2(k+1)=link1(k)
95         end
```

```
96        end
97
98  endfunction
99  function[link2]=delete1(pos,link1)
100   link2=list
         (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
         ;
101   i=1;
102   j=1;
103   while(i<pos)
104     if((link1(j).nexadd==link1(1)(1).add))
105       j=1;
106       i=i+1;
107     else
108       i=i+1;
109       j=j+1;
110     end
111   end
112   if(link1(j).nexadd~=link1(1)(1).add)
113     k=1;
114     if(j==1)
115       k=2;
116       while(link1(k).nexadd~=link1(1)(1).add)
117         link2(k-1)=link1(k);
118         k=k+1;
119       end
120       link2(k-1)=link1(k);
121       link2(k-1).nexadd=link2(1).add;
122     else
123       lin2=link1(j);
124       link1(j-1).nexadd=link1(j+1).add;
125       k=1;
126       while(link1(k).nexadd~=link1(j+1).add)
127         link2(k)=link1(k);
128         k=k+1;
129       end
130       link2(k)=link1(k);
131       k=k+2;
```

```
132         while(link1(k).nexadd~=link1(1)(1).add)
133           link2(k-1)=link1(k);
134           k=k+1;
135         end
136         link2(k-1)=link1(k);
137       end
138     else
139       link1(j-1).nexadd=link1(1)(1).add;
140       l=1;
141       while(link1(l).nexadd~=link1(1)(1).add)
142         link2(l)=link1(l);
143         l=l+1;
144       end
145       link2(l)=link1(l);
146     end
147 endfunction
148 //Calling Routine:
149 link1=struct('data',0,'add',0,'nexadd',0);
150 link1=append(4,link1);//This will actualy create a
      list and 4 as start
151 link1=append(6,link1);
152 link1=add(10,2,link1);
153 link1=delete1(4,link1);//As the list is circular the
      4'th element refers to actualy the 1'st one
154 disp(link1,"After the above manuplations the list is
     ");
```

**Scilab code Exa 4.4** Implementing Doubly connected Linked List

```
1 //DOUBLE LINKED LIST:
2 function[link2]=append(ele,link1)
3   link2=list
       (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
       ;
4   if(link1(1)(1).add==0)
```

```
5       link1(1)(1).data=ele;
6       link1(1)(1).add=1;
7       link1(1)(1).nexadd=0;
8       link1(1)(1).prevadd=0;
9       link2(1)=link1(1)(1);
10      else
11      if(link1(1)(1).nexadd==0)
12        lin2=link1(1)(1);
13        lin2.data=ele;
14        lin2.add=link1(1)(1).add+1;
15        link1(1)(1).nexadd=lin2.add;
16        lin2.nexadd=0;
17        lin2.prevadd=link1(1)(1).add;
18        link2(1)=link1(1)(1);
19        link2(2)=lin2;
20      else
21        lin2=link1(1)(1);
22        i=1;
23        while(link1(i)(1).nexadd~=0)
24          i=i+1;
25        end
26        j=i;
27        lin2.data=ele;
28        lin2.add=link1(i).add+1;
29        lin2.nexadd=0;
30        link1(i).nexadd=lin2.add;
31        lin2.prevadd=link1(i).add;
32        link2(1)=link1(1)(1);
33        i=2;
34        while(link1(i).nexadd~=lin2.add)
35          link2(i)=(link1(i));
36          i=i+1;
37        end
38        link2(i)=link1(i);
39        link2(i+1)=lin2;
40      end
41    end
42  endfunction
```

```
43 function[link2]=add(ele,pos,link1);
44    link2=list
         (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
         ;
45    i=1;
46    while(i<=pos)
47      if(link1(i).nexadd==0)
48        break;
49      else
50        i=i+1;
51      end
52    end
53    if(link1(i).nexadd~=0)
54      i=i-1;
55       lin2.data=ele;
56       lin2.add=i;
57       j=i;
58       while(link1(j).nexadd~=0)
59         link1(j).prevadd=link1(j).prevadd+1;
60         link1(j).add=link1(j).add+1;
61         link1(j).nexadd=link1(j).nexadd+1;
62         j=j+1;
63       end
64       link1(j).prevadd=link1(j).prevadd+1;
65       link1(j).add=link1(j).add+1;
66       lin2.nexadd=link1(i).add;
67       link1(i).prevadd=lin2.add;
68       lin2.prevadd=link1(i-1).add;
69       link1(i-1).nexadd=lin2.add;
70       k=1;
71       while(k<i)
72          link2(k)=link1(k);
73          k=k+1;
74        end
75        link2(k)=lin2;
76        k=k+1;
77        link2(k)=link1(k-1);
78        k=k+1
```

```
79          l=k-1;
80          while(k~=j)
81            link2(k)=link1(l);
82            k=k+1;
83            l=l+1;
84          end
85          link2(j)=link1(j-1);;
86          link2(j+1)=link1(j);
87        else
88          if(i==pos)
89            k=1;
90            lin2.data=ele;
91            lin2.add=link1(i-1).add+1;
92            link1(i).add=link1(i).add+1;
93            lin2.nexadd=link1(i).add;
94            link1(i).prevadd=lin2.add;
95            lin2.prevadd=link1(i-1).add;
96            k=1;
97            while(k<pos)
98              link2(k)=link1(k);
99              k=k+1;
100           end
101           link2(k)=lin2;
102           link2(k+1)=link1(k)
103         end
104       end
105
106 endfunction
107 function[link2]=delete1(pos,link1)
108   link2=list
        (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
        ;
109   i=1;
110   while(i<=pos)
111     if((link1(i).nexadd==0))
112       break;
113     else
114       i=i+1;
```

```
115       end
116     end
117     if(link1(i).nexadd~=0)
118       i=i-1;
119       j=1;
120       if(i==1)
121         j=1;
122         while(link1(j).nexadd~=0)
123           link2(j)=link1(j);
124           j=j+1;
125         end
126         link2(j)=link1(j);
127       else
128         link1(i-1).nexadd=link1(i+1).add;
129         link1(i+1).prevadd=link1(i-1).add;
130       while(link1(j).nexadd~=link1(i+1).add)
131         link2(j)=link1(j);
132         j=j+1;
133       end
134       if(j~=i-1)
135         link2(j)=link1(j);
136         link2(j+1)=link1(j+1);
137         k=i+1;
138         l=2;
139       else
140         link2(j)=link1(j);
141         k=i+1;
142         l=1;
143       end
144       while(link1(k).nexadd~=0)
145         link2(j+l)=link1(k);
146         k=k+1;
147         l=l+1;
148       end
149       link2(j+l)=link1(k);
150     end
151     else
152       if(i==pos)
```

```
153         j=1;
154         link1(i-1).nexadd=0;
155         while(j<=i-1)
156           link2(j)=link1(j);
157           j=j+1;
158         end
159       end
160     end
161 endfunction
162 //Calling Routine:
163 link1=struct('data',0,'add',0,'nexadd',0);
164 link1=append(4,link1);
165 link1=append(6,link1);
166 link1=add(10,2,link1);
167 link1=delete1(3,link1);
168 disp(link1,"After the above manuplations the list is
    ");
```

---

**Scilab code Exa 4.5** Implementing Stack using circular Linked list

```
1  //STACK USING CIRCULAR LINKED LIST
2  funcprot(0)
3  function[link2]=append(ele,link1)
4    link2=list
        (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
        ;
5    if(link1(1)(1).add==0)
6      link1(1)(1).data=ele;
7      link1(1)(1).add=1;
8      link1(1)(1).nexadd=1;
9      link2(1)=link1(1)(1);
10     else
11     if(link1(1)(1).nexadd==link1(1)(1).add)
12       lin2=link1(1)(1);
13       lin2.data=ele;
```

```
14        lin2.add=link1(1)(1).add+1;
15        link1(1)(1).nexadd=lin2.add;
16        lin2.nexadd=link1(1)(1).add;
17        link2(1)=link1(1)(1);
18        link2(2)=lin2;
19      else
20        lin2=link1(1)(1);
21        i=1;
22        while(link1(i)(1).nexadd~=link1(1)(1).add)
23          i=i+1;
24        end
25        j=i;
26        lin2.data=ele;
27        lin2.add=link1(i).add+1;
28        lin2.nexadd=link1(1)(1).add;
29        link1(i).nexadd=lin2.add;
30        link2(1)=link1(1)(1);
31        i=2;
32        while(link1(i).nexadd~=lin2.add)
33          link2(i)=(link1(i));
34          i=i+1;
35        end
36        link2(i)=link1(i);
37        link2(i+1)=lin2;
38      end
39    end
40  endfunction
41  function[link2]=add(ele,pos,link1);
42      link2=list
           (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
           ;
43      i=1;
44      while(i<=pos)
45        if(link1(i).nexadd==link1(1)(1).add)
46          break;
47        else
48          i=i+1;
49        end
```

```
50      end
51      if(link1(i).nexadd~=link1(1)(1).add)
52        i=i-1;
53         lin2.data=ele;
54         lin2.add=i;
55         j=i;
56         while(link1(j).nexadd~=link1(1)(1).add)
57           link1(j).add=link1(j).add+1;
58           link1(j).nexadd=link1(j).nexadd+1;
59           j=j+1;
60         end
61         link1(j).add=link1(j).add+1;
62         lin2.nexadd=link1(i).add;
63         link1(i-1).nexadd=lin2.add;
64         k=1;
65         while(k<i)
66            link2(k)=link1(k);
67            k=k+1;
68          end
69          link2(k)=lin2;
70          k=k+1;
71          link2(k)=link1(k-1);
72          k=k+1
73          l=k-1;
74          while(k~=j)
75            link2(k)=link1(l);
76            k=k+1;
77            l=l+1;
78          end
79          link2(j)=link1(j-1);;
80          link2(j+1)=link1(j);
81        else
82          if(i==pos)
83            k=1;
84            lin2.data=ele;
85            lin2.add=link1(i-1).add+1;
86            link1(i).add=link1.add+1;
87            lin2.nexadd=link1(i).add;
```

```
88            link1(i).nexadd=link1(1)(1).add;
89            k=1;
90            while(k<pos)
91              link2(k)=link1(k);
92              k=k+1;
93            end
94            link2(k)=lin2;
95            link2(k+1)=link1(k)
96          end
97        end
98
99  endfunction
100 function[link2]=delete1(pos,link1)
101   link2=list
        (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
        ;
102   i=1;
103   if(link1(1)(1).add==0)
104     disp("Invalid");
105   else
106     if(link1(1)(1).nexadd==link1(1)(1).add)
107       link1(1)(1).add=0;
108       link1(1)(1).nexadd=0;
109       link1(1)(1).data=0;
110       link2(1)=link1(1)(1);
111     else
112   while(i<=pos)
113     if((link1(i).nexadd==link1(1)(1).add))
114       break;
115     else
116       i=i+1;
117     end
118   end
119   if(link1(i).nexadd~=link1(1)(1).add)
120     i=i-1;
121     j=1;
122     if(i==1)
123       j=1;
```

```
124         while(link1(j).nexadd~=link1(1)(1).add)
125           link2(j)=link1(j);
126           j=j+1;
127         end
128         link2(j)=link1(j);
129       else
130       link1(i-1).nexadd=link1(i+1).add;
131       while(link1(j).nexadd~=link1(i+1).add)
132         link2(j)=link1(j);
133         j=j+1;
134       end
135       if(j~=i-1)
136         link2(j)=link1(j);
137         link2(j+1)=link1(j+1);
138         k=i+1;
139         l=2;
140       else
141         link2(j)=link1(j);
142         k=i+1;
143         l=1;
144       end
145       while(link1(k).nexadd~=link1(1)(1).add)
146         link2(j+l)=link1(k);
147         k=k+1;
148         l=l+1;
149       end
150       link2(j+l)=link1(k);
151     end
152     else
153       if(i==pos)
154         j=1;
155         link1(i-1).nexadd=link1(1)(1).add;
156         while(j<=i-1)
157           link2(j)=link1(j);
158           j=j+1;
159         end
160       end
161     end
```

```
end
end

endfunction
function[sta]=push(ele,stack)
  if(stack.top==0)
    stack.a=ele;
    stack.top=stack.top+1;
    sta=stack;
  else
    i=1;
    link1=struct('data',0,'add',0,'nexadd',0);
    while(i<=stack.top)
      link1=append(stack.a(i),link1);
      i=i+1;
    end
    link1=append(ele,link1);
    stack.top=stack.top+1;
    a=[stack.a(:,:) link1(stack.top).data];
    stack.a=a;
    sta=stack;
  end
endfunction
function[ele,sta]=pop(stack)
  ele=-1;
  sta=0;
  if(stack.top==0)
    disp("Stack Underflow");
    return;
else
   i=1;
    link1=struct('data',0,'add',0,'nexadd',0);
    while(i<=stack.top)
      link1=append(stack.a(i),link1);
      i=i+1;
    end
    ele=link1(stack.top).data;
    link1=delete1(stack.top,link1);
```

```
200       stack.top=stack.top-1;
201       i=2;
202       a=link1(1)(1).data
203       while(i<=stack.top)
204         a=[a(:,:) link1(i).data];
205         i=i+1;
206       end
207       stack.a=a;
208       sta=stack;
209     end
210 endfunction
211 function[stack]=empty()
212   stack=struct('a',0,'top',0);
213 endfunction
214 //Calling Routine:
215 stack=empty()//Create an empty stack
216 stack=push(4,stack);
217 stack=push(55,stack);
218 stack=push(199,stack);
219 stack=push(363,stack);
220 [ele,stack]=pop(stack);
221 disp(stack,"After the above operations stack is:");
```

**Scilab code Exa 4.6** Implementing Priority Queue Using Lists

```
1 //Implementing Priority Queue Using Lists
2 funcprot(0)
3 function[link2]=insert_pri(ele,link1)
4   link2=list
       (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
       ;
5   if(link1(1)(1).add==0)
6     link1(1)(1).data=ele;
7     link1(1)(1).add=1;
8     link1(1)(1).nexadd=1;
```

```
9       link2(1)=link1(1)(1);
10    else
11      if(link1(1)(1).nexadd==link1(1)(1).add)
12        if(ele>=link1(1)(1).data)
13          t=ele;
14          p=link1(1)(1).data;
15        else
16          t=link1(1)(1).data;
17          p=ele;
18        end
19        link1(1)(1).data=t;
20        lin2=link1(1)(1);
21        lin2.data=p;
22        lin2.add=2;
23        lin2.nexadd=link1(1)(1).add;
24        link1(1)(1).nexadd=lin2.add;
25        link2(1)=link1(1)(1);
26        link2(2)=lin2;
27      else
28        i=1;
29        a=[];
30        while(link1(i).nexadd~=link1(1)(1).add)
31          a=[a(:,:) link1(i).data];
32          i=i+1;
33        end
34        a=[a(:,:) link1(i).data];
35        a=gsort(a);
36        j=1;
37        while(j<=i)
38          link1(j).data=a(j);
39          j=j+1;
40        end
41        k=1;
42        while(link1(k).data>=ele)
43          if(link1(k).nexadd==link1(1)(1).add)
44            break;
45          else
46            link2(k)=link1(k);
```

```
47              k=k+1;
48            end
49          end
50          if(link1(k).nexadd~=link1(1)(1).add)
51            lin2=link1(k);
52            lin2.data=ele;
53            lin2.add=link1(k).add;
54            j=k;
55            y=link1(1)(1).add;
56            while(link1(k).nexadd~=y)
57              link1(k).add=link1(k).add+1;
58              link1(k).nexadd=link1(k).nexadd+1;
59              k=k+1;
60            end
61            link1(k).add=link1(k).add+1;
62            lin2.nexadd=link1(j).add;
63            link2(j)=lin2;
64            j=j+1;
65            while(j<=k+1)
66              link2(j)=link1(j-1);
67              j=j+1;
68            end
69          else
70            lin2=link1(k);
71            lin2.data=ele;
72            lin2.nexadd=link1(1)(1).add;
73            lin2.add=link1(k).add+1;
74            link1(k).nexadd=lin2.add;
75            j=1;
76            while(j<=k)
77              link2(j)=link1(j);
78              j=j+1;
79            end
80            link2(j)=lin2;
81          end
82        end
83      end
84  endfunction
```

```
85  function[ele,link2]=extract_min(link1);
86    link2=list
         (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,,0,0)
         ;
87    i=1;
88    ele=-1;
89    if(link1(1)(1).add==0)
90      disp("Underflow");
91      return;
92    else
93      if(link1(1)(1).nexadd==link1(1)(1).add)
94        link1(1)(1).add=0;
95        link1(1)(1).nexadd=0;
96        ele=link1(1)(1).data;
97        link1(1)(1).data=0;
98        link2(1)=link1(1)(1);
99      else
100       i=1;
101       while(link1(i).nexadd~=link1(1)(1).add)
102         link2(i)=link1(i);
103         i=i+1;
104       end
105       ele=link1(i).data;
106       link2(i-1).nexadd=link2(1).add;
107     end
108   end
109 endfunction
110 //Calling Routine:
111 link1=struct('data',0,'add',0,'nexadd',0);
112 link1=insert_pri(3,link1);
113 link1=insert_pri(4,link1);
114 link1=insert_pri(22,link1);
115 link1=insert_pri(21,link1);
116 link1=insert_pri(11,link1);
117 disp(link1,"List After Insertions");
118 [ele,link1]=extract_min(link1)
119 disp(ele,"Element after the min extraction");
```

# Chapter 5

# Trees

**Scilab code Exa 5.1** Implementing Binary Tree

```
1
2  funcprot(0);
3  function[tree]=maketree(x)
4    tree=zeros(30,1);
5    for i=1:30
6      tree(i)=-1;
7    end
8    tree(1)=x;
9    tree(2)=-2;
10 endfunction
11 function[tree1]=setleft(tree,tre,x)
12   tree1=[];
13   i=1;
14   while(tree(i)~=-2)
15     if(tree(i)==tre)
16       j=i;
17     end
18     i=i+1;
19   end
20   if(i>2*j)
21     tree(2*j)=x;
```

```
22    else
23      tree(2*j)=x;
24      tree(2*j+1)=-2;
25      for l=i:2*j-1
26        tree(i)=-1;
27      end
28    end
29    tree1=tree;
30  endfunction
31  function[tree1]=setright(tree,tre,x)
32    tree1=[];
33    i=1;
34    while(tree(i)~=-2)
35      if(tree(i)==tre)
36        j=i;
37      end
38      i=i+1;
39    end
40    if(i>2*j+1)
41      tree(2*j+1)=x;
42    else
43      tree(2*j+1)=x;
44      tree(2*j+2)=-2;
45      for l=i:2*j
46        tree(i)=-1;
47      end
48    end
49    tree1=tree;
50  endfunction
51  function[x]=isleft(tree,tre)
52    i=1;
53    x=0;
54    while(tree(i)~=-2)
55      if(tree(i)==tre)
56        j=i;
57      end
58      i=i+1;
59    end
```

```
60    if(i>=2*j)
61      if((tree(2*j)~=-1)|(tree(2*j)~=-2))
62        x=1;
63        return 1;
64      else
65        return 0;
66      end
67    else
68      x=0;
69      return x;
70    end
71  endfunction
72  function[x]=isright(tree,tre)
73    i=1;
74    x=0;
75    while(tree(i)~=-2)
76      if(tree(i)==tre)
77        j=i;
78      end
79      i=i+1;
80    end
81    if(i>=2*j+1)
82      if((tree(2*j+1)~=-1)|(tree(2*j+1)~=-2))
83        x=1;
84        return 1;
85      else
86        return 0;
87      end
88    else
89      x=0;
90      return x;
91    end
92  endfunction
93  //Calling Routine:
94  tree=maketree(3);
95  disp(tree,"Tree made");
96  tree=setleft(tree,3,1);
97  disp(tree,"After setting 1 to left of 3");
```

```
tree=setright(tree,3,2);
disp(tree,"After setting 2 to right of 3");
tree=setright(tree,2,4);
tree=setleft(tree,2,5);
tree=setright(tree,1,6);
tree=setright(tree,5,8);
disp(tree,"After above operations:");
x=isright(tree,3);
disp(x,"Checking for the right son of 3 yes if 1
   else no");
x=isleft(tree,2);
disp(x,"Check for left son of 2");
```

**Scilab code Exa 5.2** Tree Trversal Techniques

```
funcprot(0);
function[tree]=maketree(x)
  tree=zeros(30,1);
  for i=1:30
    tree(i)=-1;
  end
  tree(1)=x;
  tree(2)=-2;
endfunction
function[tree1]=setleft(tree,tre,x)
  tree1=[];
  i=1;
  while(tree(i)~=-2)
    if(tree(i)==tre)
      j=i;
    end
    i=i+1;
  end
  if(i>2*j)
    tree(2*j)=x;
```

```
21    else
22      tree(2*j)=x;
23      tree(2*j+1)=-2;
24      for l=i:2*j-1
25        tree(i)=-1;
26      end
27    end
28    tree1=tree;
29  endfunction
30  function[tree1]=setright(tree,tre,x)
31    tree1=[];
32    i=1;
33    while(tree(i)~=-2)
34      if(tree(i)==tre)
35        j=i;
36      end
37      i=i+1;
38    end
39    if(i>2*j+1)
40      tree(2*j+1)=x;
41    else
42      tree(2*j+1)=x;
43      tree(2*j+2)=-2;
44      for l=i:2*j
45        tree(i)=-1;
46      end
47    end
48    tree1=tree;
49  endfunction
50  function[x]=isleft(tree,tre)
51    i=1;
52    x=0;
53    while(tree(i)~=-2)
54      if(tree(i)==tre)
55        j=i;
56      end
57      i=i+1;
58    end
```

```
59    if(i>=2*j)
60      if((tree(2*j)~=-1)|(tree(2*j)~=-2))
61        x=1;
62        return 1;
63      else
64        return 0;
65      end
66    else
67      x=0;
68      return x;
69    end
70  endfunction
71  function[x]=isright(tree,tre)
72    i=1;
73    x=0;
74    while(tree(i)~=-2)
75      if(tree(i)==tre)
76        j=i;
77      end
78      i=i+1;
79    end
80    if(i>=2*j+1)
81      if((tree(2*j+1)~=-1)|(tree(2*j+1)~=-2))
82        x=1;
83        return 1;
84      else
85        return 0;
86      end
87    else
88      x=0;
89      return x;
90    end
91  endfunction
92  funcprot(0);
93  function[]=inorder(tree,p)
94    if(tree(p)==-1|tree(p)==-2)
95      return;
96    else
```

```
97      inorder(tree,2*p);
98      printf("%d\t",tree(p));
99      inorder(tree,2*p+1);
100   end
101 endfunction
102 function[]=preorder(tree,p)
103   if(tree(p)==-1|tree(p)==-2)
104     return;
105   else
106     printf("%d\t",tree(p));
107     preorder(tree,2*p);
108     preorder(tree,2*p+1);
109   end
110 endfunction
111 function[]=postorder(tree,p)
112   if(tree(p)==-1|tree(p)==-2)
113     return;
114   else
115     postorder(tree,2*p);
116     postorder(tree,2*p+1);
117     printf("%d\t",tree(p));
118   end
119 endfunction
120 //Calling Routine:
121 tree=maketree(3);
122 tree=setleft(tree,3,1);
123 tree=setright(tree,3,2);
124 tree=setleft(tree,2,4);
125 tree=setright(tree,2,5);
126 disp("Inorder traversal");
127 inorder(tree,1);
128 disp("Preorder traversal");
129 preorder(tree,1);
130 disp("Postorder traversal");
131 postorder(tree,1);
```

**Scilab code Exa 5.3** Implementing And traversing a Binary Search Tree

```
1  funcprot(0);
2  function[tree]=maketree(x)
3    tree=zeros(1,30);
4    for i=1:30
5      tree(i)=-1;
6    end
7    tree(1)=x;
8    tree(2)=-2;
9  endfunction
10 function[tree1]=setleft(tree,tre,x)
11   tree1=[];
12   i=1;
13   while(tree(i)~=-2)
14     if(tree(i)==tre)
15       j=i;
16     end
17     i=i+1;
18   end
19   if(i>2*j)
20     tree(2*j)=x;
21   else
22     tree(2*j)=x;
23     tree(2*j+1)=-2;
24     for l=i:2*j-1
25       tree(i)=-1;
26     end
27   end
28   tree1=tree;
29 endfunction
30 function[tree1]=setright(tree,tre,x)
31   tree1=[];
32   i=1;
```

```
33    while(tree(i)~=-2)
34      if(tree(i)==tre)
35        j=i;
36      end
37      i=i+1;
38    end
39    if(i>2*j+1)
40      tree(2*j+1)=x;
41    else
42      tree(2*j+1)=x;
43      tree(2*j+2)=-2;
44      for l=i:2*j
45        tree(i)=-1;
46      end
47    end
48    tree1=tree;
49  endfunction
50  function[x]=isleft(tree,tre)
51    i=1;
52    x=0;
53    while(tree(i)~=-2)
54      if(tree(i)==tre)
55        j=i;
56      end
57      i=i+1;
58    end
59    if(i>=2*j)
60      if((tree(2*j)~=-1)|(tree(2*j)~=-2))
61        x=1;
62        return 1;
63      else
64        return 0;
65      end
66    else
67      x=0;
68      return x;
69    end
70  endfunction
```

```
71 function[x]=isright(tree,tre)
72   i=1;
73   x=0;
74   while(tree(i)~=-2)
75     if(tree(i)==tre)
76       j=i;
77     end
78     i=i+1;
79   end
80   if(i>=2*j+1)
81     if((tree(2*j+1)~=-1)|(tree(2*j+1)~=-2))
82       x=1;
83       return 1;
84     else
85       return 0;
86     end
87   else
88     x=0;
89     return x;
90   end
91 endfunction
92 funcprot(0);
93 function[]=inorder(tree,p)
94   if(tree(p)==-1|tree(p)==-2)
95     return;
96   else
97     inorder(tree,2*p);
98     disp(tree(p),"  ");
99     inorder(tree,2*p+1);
100  end
101 endfunction
102 function[]=preorder(tree,p)
103   if(tree(p)==-1|tree(p)==-2)
104     return;
105   else
106     disp(tree(p),"  ");
107     preorder(tree,2*p);
108     preorder(tree,2*p+1);
```

```
109   end
110 endfunction
111 function[]=postorder(tree,p)
112   if(tree(p)==-1|tree(p)==-2)
113     return;
114   else
115     postorder(tree,2*p);
116     postorder(tree,2*p+1);
117     disp(tree(p),"  ");
118   end
119 endfunction
120 function[tree1]=binary(tree,x)
121   p=1;
122   while(tree(p)~=-1&tree(p)~=-2)
123     q=p;
124     if(tree(p)>x)
125       p=2*p;
126     else
127       p=2*p+1;
128     end
129   end
130   i=1;
131   while(tree(i)~=-2)
132     i=i+1;
133   end
134   if(tree(q)>x)
135     if(i==2*q)
136       tree(2*q)=x;
137       tree(2*q+1)=-2
138     else
139       if(i<2*q)
140         tree(i)=-1;
141         tree(2*q+1)=-2;
142         tree(2*q)=x;
143       end
144     end
145
146   else
```

```
147      if(i==2*q+1)
148        tree(2*q+1)=x;
149        tree(2*q+2)=-2;
150      else
151        if(i<2*q+1)
152          tree(i)=-1;
153          tree(2*q+1)=x;
154          tree(2*q+2)=-2;
155        end
156      end
157
158    end
159    tree1=tree;
160  endfunction
161  //Calling Routine:
162  tree=maketree(3);
163  tree=binary(tree,1);
164  tree=binary(tree,2);
165  tree=binary(tree,4);
166  tree=binary(tree,5);
167  disp(tree,"Binary tree thus obtaine by inserting
       1,2,4and5 in tree rooted 3 is:");
```

**Scilab code Exa 5.4** Checking the duplicate number using BST

```
1  function[tree1]=binary(tree,x)
2    p=1;
3    while(tree(p)~=-1&tree(p)~=-2)
4      q=p;
5      if(tree(p)>x)
6        p=2*p;
7      else
8        p=2*p+1;
9      end
10   end
```

```
11    if(tree(q)>x)
12      if(tree(2*q)==-2)
13        tree(2*q)=x;
14        tree(2*q+1)=-2;
15      else
16        tree(2*q)=x;
17      end
18    else
19      if(tree(2*q+1)==-2)
20        tree(2*q+1)=x;
21        tree(2*q+2)=-2;
22      else
23        tree(2*q+1)=x;
24      end
25    end
26    tree1=tree;
27  endfunction
28  funcprot(0);
29  function[tree]=maketree(x)
30    tree=zeros(40,1);
31    for i=1:40
32      tree(i)=-1;
33    end
34    tree(1)=x;
35    tree(2)=-2;
36  endfunction
37  function[]=duplicate1(a,n)
38    tree=maketree(a(1));
39    q=1;
40    p=1;
41    i=2;
42    x=a(i)
43    while(i<n)
44      while(tree(p)~=x&tree(q)~=-1&tree(q)~=-2)
45        p=q;
46        if(tree(p)<x)
47          q=2*p;
48        else
```

```
49          q=2*p+1;
50        end
51      end
52      if(tree(p)==x)
53        disp(x," Duplicate  ");
54      else
55        tree=binary(tree,x);
56      end
57      i=i+1;
58      x=a(i);
59    end
60    while(tree(p)~=x&tree(q)~=-1&tree(q)~=-2)
61        p=q;
62        if(tree(p)<x)
63          q=2*p;
64        else
65          q=2*p+1;
66        end
67      end
68      if(tree(p)==x)
69        disp(x," Duplicate  ");
70      else
71        tree=binary(tree,x);
72      end
73  endfunction
74  //Calling Adress:
75  a=[22 11 33 22 211 334]
76  duplicate1(a,6)
77  a=[21 11 33 22 22 334]
78  duplicate1(a,6)
```

# Chapter 6

# Sorting

**Scilab code Exa 6.1** Bubble Sort

```
function[a1]=bubble(a,n)
  i=1;
  j=1;
  temp=0;
  for i=1:n-1
   for j=1:n-i
      if(a(j)>a(j+1))
        temp=a(j);
        a(j)=a(j+1);
        a(j+1)=temp;
      end
      j=j+1;
    end
    i=i+1;
  end
  a1=a;
  disp(a1,"Sorted array is:");
endfunction
//Calling Routine:
a=[23 21 232 121 2324 1222433 1212]
disp(a,"Given Array");
```

```
22 a1=bubble(a,7)
```

**Scilab code Exa 6.2** Quick Sort

```
1  function[a1]=quick(a);
2    a=gsort(a);//IN BUILT QUICK SORT FUNCTION
3    n=length(a);
4    a1=[];
5    for i=1:n
6      a1=[a1(:,:) a(n+1-i)];
7    end
8    disp(a1,"Sorted array is:");
9  endfunction
10 //Calling Routine:
11 a=[23 21 232 121 2324 1222433 1212]
12 disp(a,"Given Array");
13 a1=quick(a)
```

**Scilab code Exa 6.3** Selection Sort

```
1  function[a1]=selection(a,n)
2    i=n;
3    while(i>=1)
4      large=a(1);
5      indx=1;
6      for j=1:i
7        if(a(j)>large)
8          large=a(j);
9          indx=j;
10       end
11     end
12     a(indx)=a(i);
13     a(i)=large;
```

```
14      i=i-1;
15    end
16    a1=a;
17    disp(a1,"Sorted array is:");
18  endfunction
19  //Calling Routine:
20  a=[23 21 232 121 2324 1222433 1212]
21  disp(a,"Given Array");
22  a1=selection(a,7)
```

**Scilab code Exa 6.4** Insertion Sort

```
1   function[a1]=insertion(a,n)
2     for k=1:n
3       y=a(k);
4       i=k;
5       while(i>=1)
6         if(y<a(i))
7           a(i+1)=a(i);
8           a(i)=y;
9         end
10        i=i-1;
11      end
12    end
13    a1=a;
14    disp(a1,"Sorted array is:");
15  endfunction
16  //Calling Routine:
17  a=[23 21 232 121 2324 1222433 1212]
18  disp(a,"Given Array");
19  a1=insertion(a,7)
```

**Scilab code Exa 6.5** Shell sort

```
function[a1]=shell(a,n,incr,nic)
  for i=1:nic
    span=incr(i);
    for j=span+1:n
      y=a(j);
      k=j-span;
      while(k>=1&y<a(k))
          a(k+span)=a(k);
        k=k-span;
      end
      a(k+span)=y;
    end
  end
  a1=a;
  disp(a1,"Sorted array is:");
endfunction
//Calling Routine:
a=[23 21 232 121 2324 1222433 1212]
disp(a,"Given Array");
incr=[5 3 1]//must always end with 1
a1=shell(a,7,incr,3)
```

**Scilab code Exa 6.6** Merge Sort

```
function[a1]=mergesort(a,p,r)
  if(p<r)
    q=int((p+r)/2);
    a=mergesort(a,p,q);
    a=mergesort(a,q+1,r);
    a=merge(a,p,q,r);
  else
    a1=a;
    return;
  end
  a1=a;
```

```
endfunction
function[a1]=merge(a,p,q,r)
  n1=q-p+1;
  n2=r-q;
  left=zeros(n1+1);
  right=zeros(n2+1);
  for i=1:n1
    left(i)=a(p+i-1);
  end
  for i1=1:n2
      right(i1)=a(q+i1);
  end
  left(n1+1)=999999999;
  right(n2+1)=999999999;
  i=1;
  j=1;
  k=p;
  for  k=p:r
    if(left(i)<=right(j))
      a(k)=left(i);
      i=i+1;
    else
      a(k)=right(j);
      j=j+1;
    end
  end
  a1=a;
endfunction
//Calling Routine:
a=[23 21 232 121 26324 1222433 14212]
disp(a,"Given Array");
a1=mergesort(a,1,7)
disp(a1,"Sorted array is:");
a=[232 11212 3443 23221 123424 32334 12212 2443 232]
disp(a,"Given Array");
a1=mergesort(a,1,9);
disp(a1,"Sorted Array");
```

**Scilab code Exa 6.7** Binary Tree Sort

```
1  function[tree1]=binary(tree,x)
2    p=1;
3    while(tree(p)~=-1&tree(p)~=-2)
4      q=p;
5      if(tree(p)>x)
6        p=2*p;
7      else
8        p=2*p+1;
9      end
10   end
11   if(tree(q)>x)
12       tree(2*q)=x;
13   else
14       tree(2*q+1)=x;
15   end
16   tree1=tree;
17 endfunction
18 funcprot(0);
19 function[tree]=maketree(x)
20   tree=zeros(100,1);
21   for i=1:100
22     tree(i)=-1;
23   end
24   tree(1)=x;
25   tree(2)=-2;
26 endfunction
27 function[]=inorder(tree,p)
28   if(tree(p)==-1|tree(p)==-2)
29     return;
30   else
31     inorder(tree,2*p);
32     printf("%d\t",tree(p));
```

```
33      inorder(tree,2*p+1);
34    end
35  endfunction
36  function[]=binsort(a,n)
37    a1=maketree(a(1))
38    for i=2:n
39      a1=binary(a1,a(i));
40    end
41    disp("Sorted array is:");
42    inorder(a1,1);
43  endfunction
44  //Calling Routine:
45  a=[23 21 232 121 2324 1222433 1212]
46  disp(a,"Given Array");
47  a1=binsort(a,7)
```

# Chapter 7

# Searching

**Scilab code Exa 7.1** Sequential Search

```
function[]=search(a,n,ele)
  i=1;
  j=0;
  for i=1:n
    if(a(i)==ele)
      printf("Found %d AT %d\n",ele,i);
      j=1;
    end
  end
  if(j==0)
    disp("%d NOT FOUND",ele);
  end
endfunction
//Calling Routine:
a=[2 33 22 121 23 233 222]
disp(a,"Given array");
search(a,7,23)
```

**Scilab code Exa 7.2** Sorted sequential search

```
function[]=sortedsearch(a,n,ele)
  if(a(1)&gt;ele|a(n)<ele) disp("not="" in="" the="" list");="" else="" i="1;" j="0;" for="" if(a(i)="ele)" printf("found="" %d="" at="" %d",ele,i);="">ele)
        break;
      end
    end
  end
  if(j==0)
    disp("%d NOT FOUND",ele);
  end
end
endfunction
//Calling Routine:
a=[2 22 23 33 121 222 233]//a should be sorted
disp(a,"Given array");
sortedsearch(a,7,23)
</ele)>
```

**Scilab code Exa 7.3** Binary Search

```
function[]=binsearch(a,n,i)
  l=1;
  h=n;
  while(l&lt;=h)
    mid=int((l+h)/2);
    if(a(mid)==i)
      printf("FOUND %d AT %d",i,mid);
      break;
    else
      if(a(mid)&gt;i)
        h=mid-1;
      else
        l=mid+1;
```

```
14        end
15      end
16    end
17 endfunction
18 //Calling Routine:
19 a=[2 22 23 33 121 222 233]//a should be  sorted
20 disp(a,"Given array");
21 binsearch(a,7,23)
```

# Chapter 8

# Graphs

**Scilab code Exa 8.1** Simple Graph Functions

```
1  //Simple Graph Functions
2  function[]=graph();
3
4    i=1,j=1;
5    adj=zeros(10000);
6    for i=1:n
7    for j=1:n
8
9        adj((i-1)*n+j)=temp;
10     end
11   end
12   for i=1:n
13     for  j=1:n
14       if((adj((i-1)*n+j))==1)
15         printf("Vertex %d is connected to vertex %d\
               n",i,j);
16       end
17     end
18   end
19
20 endfunction
```

**Scilab code Exa 8.2** Finding The Number Of Paths From One VertexToOther

```
1 //Finding The Number Of Paths From One Vertex To
    Another Of A Given Length
2
3 function[b]=path(k,n,adj,i,j)
4   b=0;
5   if(k==1)
6     b=adj((i-1)*n+j);
7   else
8     for c=1:n
9       if(adj((i-1)*n+c)==1)
10        b=b+path(k-1,n,adj,c,j);
11      end
12    end
13  end
14    printf("Number of paths from vertex %d to %d of
        length %d are %d",i,j,k,b);
15  return b;
16 endfunction
17 //Calling Routine:
18 n=3;
19 adj=[0 1 1 0 0 1 0 0 0]
20 b=path(1,n,adj,1,3)
```

**Scilab code Exa 8.3** Finding The Number Of Simple Paths From One Point

```
1 //Finding The  Number Of Simple Paths From One Point
    To Another In A Given Graph
2 funcprot(0)
3 function[]=sim_path(n,adj,i,j);
```

```
4    l=0;
5    m=1;
6    for m=1:n
7      l=l+path(m,n,adj,i,j);
8    end
9    printf("There are %d  simple paths from %d to %d
       in the given graph\n",l,i,j);
10 endfunction
11 function[b]=path(k,n,adj,i,j)
12   b=0;
13   if(k==1)
14     b=adj((i-1)*n+j);
15   else
16     for c=1:n
17       if(adj((i-1)*n+c)==1)
18         b=b+path(k-1,n,adj,c,j);
19       end
20     end
21   end
22   return b;
23 endfunction
24 n=3;
25 adj=[0 1 1 0 0 1 0 0 0];
26 b=sim_path(n,adj,1,3)
```

**Scilab code Exa 8.4** Finding Transitive Closure

```
1  //Finnding Transitive Closure
2  funcprot(0)
3  function[path]=Tranclose(adj,n);
4    i=1,j=1;
5    path=zeros(n*n,1);
6    path=tranclose(adj,n);
7    printf("Transitive Closure Of Given Graph is:\n");
8    for i=1:n
```

```
 9      printf("For Vertex %d\n",i);
10      for j=1:n
11        printf(" %d %d is %d\n",i,j,path((i-1)*n+j));
12      end
13    end
14
15  endfunction
16  function[path]=tranclose(adj,n)
17    adjprod=zeros(n*n,1);
18    k=1;
19    newprod=zeros(n*n,1);
20    for i=1:n
21      for j=1:n
22        path((i-1)*n+j)=adj((i-1)*n+j);
23        adjprod((i-1)*n+j)= path((i-1)*n+j);
24      end
25    end
26    for i=1:n
27      newprod=prod1(adjprod,adj,n);
28      for j=1:n
29        for k=1:n
30          path((j-1)*n+k)=path((j-1)*n+k)|newprod((j
              -1)*n+k);
31        end
32      end
33      for j=1:n
34        for k=1:n
35          adjprod((j-1)*n+k)=newprod((j-1)*n+k);
36        end
37      end
38    end
39  endfunction
40  function[c]=prod1(a,b,n)
41    for i=1:n
42      for j=1:n
43        val=0
44        for k=1:n
45          val=val|(a((i-1)*n+k)&b((k-1)*n+j));
```

```
46        end
47        c((i-1)*n+j)=val;
48      end
49    end
50  endfunction
51  //Calling Routine:
52  n=3;
53  adj=[0 1 0 0 0 1 0 0 0]
54  path=Tranclose(adj,n)
```

---

**Scilab code Exa 8.5** Warshalls Algorithm

```
1  //Warshall's Algorithm
2  funcprot(0)
3  function[path]=transclose(adj,n)
4    for  i=1:n
5      for j=1:n
6        path((i-1)*n+j)=adj((i-1)*n+j);
7      end
8    end
9    for k=1:n
10     for  i=1:n
11       if(path((i-1)*n+k)==1)
12         for j=1:n
13           path((i-1)*n+j)=path((i-1)*n+j)|path((k-1)
                *n+j);
14         end
15       end
16     end
17   end
18   printf("Transitive closure for the given graph is
       :\n");
19   for  i=1:n
20     printf("For vertex %d \n",i);
21     for j=1:n
```

```
22          printf("%d %d is %d\n",i,j,path((i-1)*n+j));
23        end
24      end
25  endfunction
26  //Calling Routine:
27  n=3;
28  adj=[0 1 0 0 0 1 0 0 0]
29  path=transclose(adj,n)
```

**Scilab code Exa 8.6** Depth First Search Traversal

```
1   //Depth First Search Traversal
2   funcprot(0)
3   function[]=Dfs(adj,n);
4     i=1,j=1;
5     colour=[];
6     for i=1:n
7     for j=1:n
8         colour=[colour(:,:) 0];
9       end
10    end
11    disp("The DFS traversal is");
12  dfs(adj,colour,1,n);
13  endfunction
14  function[]=dfs(adj,colour,r,n)
15    colour(r)=1;
16    disp(r,"  ");
17    for i=1:n
18      if(adj((r-1)*n+i)&(colour(i)==0))
19        dfs(adj,colour,i,n);
20      end
21    end
22    colour(r)=2;
23  endfunction
24  //Calling Routine:
```

```
n=4;
adj=[0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0]
Dfs(adj,n)
```

---

**Scilab code Exa 8.7** BFS Traversal

```
////BFS Traversal
funcprot(0)
function[q2]=push(ele,q1)
  if(q1.rear==q1.front)
    q1.a=ele;
    q1.rear=q1.rear+1;
  else
    q1.a=[q1.a(:,:) ele];
    q1.rear=q1.rear+1;
  end
  q2=q1;
endfunction
function[ele,q2]=pop(q1)
  ele=-1;
  q2=0;
  if(q1.rear==q1.front)
        return;
  else
    ele=q1.a(q1.rear-q1.front);
    q1.front=q1.front+1;
    i=1;
    a=q1.a(1);
    for i=2:(q1.rear-q1.front)
      a=[a(:,:) q1.a(i)];
    end
    q1.a=a;
  end
  q2=q1;
endfunction
```

```
30
31 function[]=Bfs(adj,n);
32   i=1,j=1;
33   colour=[];
34   for i=1:n
35   for j=1:n
36       colour=[colour(:,:) 0];
37     end
38   end
39   disp("The BFS Traversal is");
40 bfs(adj,colour,1,n);
41 endfunction
42 function[]=bfs(adj,colour,s,n)
43   colour(s)=1;
44   q=struct('rear',0,'front',0,'a',0);
45   q=push(s,q);
46   while((q.rear)-(q.front)>0)
47     [u,q]=pop(q);
48     disp(u," ");
49     for i=1:n
50       if(adj((u-1)*n+i)&(colour(i)==0))
51         colour(i)=1;
52         q=push(i,q);
53       end
54     end
55     colour(u)=2;
56   end
57 endfunction
58 //Calling Routine:
59 n=4;
60 adj=[0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0]
61 Bfs(adj,n)
```

**Scilab code Exa 8.8** Dijkstras Algorithm

```
1  //Dijkstras Algorithm
2  funcprot(0)
3  function[l]=short(adj,w,i1,j1,n)
4    for i=1:n
5      for j=1:n
6        if(w((i-1)*n+j)==0)
7          w((i-1)*n+j)=9999;
8        end
9      end
10   end
11
12   distance=[];
13   perm=[];
14   for i=1:n
15     distance=[distance(:,:) 99999];
16     perm=[perm(:,:) 0];
17   end
18   perm(i1)=1;
19   distance(i1)=0;
20   current=i1;
21   while(current~=j1)
22     smalldist=9999;
23     dc=distance(current);
24     for i=1:n
25       if(perm(i)==0)
26         newdist=dc+w((current-1)*n+i);
27         if(newdist<distance(i))
28           distance(i)=newdist;
29         end
30         if(distance(i)<smalldist)
31           smalldist=distance(i);
32           k=i;
33         end
34       end
35     end
36     current=k;
37     perm(current)=1;
38   end
```

```
39    l=distance(j1);
40    printf("The  shortest  path  between  %d and %d is %d
        ",i1,j1,l);
41  endfunction
42  //Calling Routine:
43  n=3;
44  adj=[0 1 1 0 0 1 0 0 0]//Adjacency  List
45  w=[0 12 22 0 0 9 0 0 0]//weight list fill 0 for no
      edge
46  short(adj,w,1,3,n);
```