Homework

Problems on DH

2.10

```
(1) create a stack S;
(2) point to the first element of A_N;
(3) do the following N-1 times:
    (3.1) push the pointed element on stack S;
    (3.2) point to the next element;
(3) push the pointed element on stack S;
(4) point to the first element in vector P[N];
(5) do the following until all the elements in vector P[N] are visited:
    (5.1) compare the pointed element with the topmost element on stack S;
    (5.2) if they are equal,then pop the stack and erase the pointed element in vector P[N] and
    (5.3) point to the next element in vector P[N];
(6) if stack S is empty,then output "P represents a permutation of A_N";
(7) if stack S isn't empty,then output "P doesn't represent any permutation of A_N";
(8) stop;
```

2.11

```
(1) create a vector P[N],a variable X;
(2) read(X);
(3) call print-permutations X;
(4) stop;
subroutine print-permutations N:
(1) if N is equal to 1,then do the following:
    (1.1) read(X);
    (1.2) P[1] gets X;
(2) if N isn't equal to 1,then do the following:
    (2.1) call print-permutations N-1;
    (2.2) for i going from 1 to N-1,do the following:
        (2.2.1) insert X into the vector before P[i];
        (2.2.2) for j going from 1 to N,do the following:
            (2.2.2.1) print(P[i]);
        (2.2.3) P[N] gets X;
        (2.2.4) for k going from 1 to N,do the following:
            (2.2.4.1) print(P[i]);
(3) return;
```

2.12

(a)

i.

```
(1) create a stack S,a variable X;
(2) do the following 3 times:
    (2.1) read(X);
    (2.2) push(X,S);
(3) do the following 3 times:
    (3.1) pop(X,S);
    (3.2) print(X);
(4) stop;
```

ii.

```
(1) create a stack S,a variable X;
(2) do the following 2 times:
    (2.1) read(X);
    (2.2) push(X,S);
(3) do the following 2 times:
    (3.1) read(X);
    (3.2) print(X);
(4) do the following 2 times:
    (4.1) pop(X,S);
    (4.2) print(X);
(5) stop;
```

iii.

```
(1) create a stack S,a variable X;
(2) do the following 2 times:
    (2.1) read(X);
    (2.2) push(X,S);
(3) do the following 2 times:
    (3.1) read(X);
    (3.2) print(X);
    (3.3) read(X);
    (3.4) push(X,S);
(4) do the following 4 times:
    (4.1) read(X);
    (4.2) print(X);
    (4.3) pop(X,S);
    (4.4) print(X);
(5) stop;
```

(b)

i.

Proof

Since 1 isn't the first on the output,we must push it on the stack S after read it into X. And then we read 2 into X, because 2 isn't the first on the output neither, we must push it on S, too. Then we can

never print 1 on the output before printing 2 on the output.

ii.

Proof

Since we need to print 4 as the first on the output, then we push 1,2,3 on the stack S in sequence. Then after reading and printing 4 and 5 on the output in sequence, we pop(X,S) and print 3 on the output, then we read and push 6 on S to make sure we can read and print 7 on the output just after 3. Now we can never print 2 and 1 on the output before printing 6 on the output.

(c)

12

2.13

Let's suppose the input sequence to be the very permutation *A* of length N,which is to be checked.

```
(1) create a stack S,a variable X,a variable num,a vector P[N];
(2) num gets 0;
(3) for i going from 1 to N do the following:
    (3.1) read(X);
    (3.2) P[i] gets X;
(4) for i going from 1 to N-1 do the following:
    (4.1) for j going from i+1 to N do the following:
        (4.1.1) compare P[i] and P[j];
        (4.1.2) if P[j] < P[i],then add 1 to num;
(5) if num is odd,then goto(7);
(6) print the following steps:
    (6.1) for k going from 1 to N do the following:
        (6.1.1) do the following for N times:
            (6.1.1.1) if is-empty(S) returns true,then goto(6.1.1.5);
            (6.1.1.2) compare the topmost element on S with P[k]
            (6.1.1.3) if the topmost element on S is equal to P[k],then pop(X,S);
            (6.1.1.4) print(X);
            (6.1.1.5) read(X);
            (6.1.1.6) compare X with P[k];
            (6.1.1.7) if X is equal to P[k],then print(X);
            (6.1.1.8) if X isn't equal to P[k],then push(X,S);
    (6.2) stop;
(7) stop;
```

2.14

(a)

i.

With a queue:

```
(1) create a queue Q,a variable X;
(2) do the following 2 times:
    (2.1) read(X);
    (2.2) add(x,Q);
(3) read(X);
(4) print(X);
(5) do the following 2 times:
    (5.1) remove(X,Q);
    (5.2) print(X);
(6) stop;
```

or with two stacks:

```
(1) create a stack S1,a stack S2,a variable X;
(2) read(X);
(3) push(X,S1);
(4) read(X);
(5) push(X,S2);
(6) read(X);
(7) print(X);
(8) pop(X,S1);
(9) print(X);
(10) pop(X,S2);
(11) print(X);
(12) stop;
```

ii.

With a queue:

```
(1) create a queue Q,a variable X;
(2) do the following 3 times:
    (2.1) read(X);
    (2.2) add(X,Q);
(3) do the following 2 times:
    (3.1) read(X);
    (3.2) print(X);
(4) do the following 2 times:
    (4.1) remove(X);
    (4.2) add(X,Q);
(5) remove(X,Q);
(6) print(X);
(7) read(X);
(8) add(X,Q);
(9) read(X);
(10) print(X);
(11) do the following 2 times:
    (11.1) remove(X,Q);
    (11.2) add(X,Q);
    (11.3) remove(X,Q);
    (11.4) print(X);
(12) remove(X,Q);
(13) print(X);
(14) stop;
```

or with two stacks:

```
(1) create a stack S1,a stack S2,a variable X;
(2) do the following 3 times:
    (2.1) read(X);
    (2.2) push(X,S1);
(3) do the following 2 times:
    (3.1) read(X);
    (3.2) print(X);
(4) pop(X,S1);
(5) print(X);
(6) read(X);
(7) push(X,S2);
(8) read(X);
(9) print(X);
(10) do the following 2 times:
    (10.1) pop(X,S1);
    (10.2) print(S);
(11) pop(X,S2);
(12) print(X);
(13) stop;
```

(b)

Proof

Once we read all the elements and add them to a queue Q, now consider the element in the front of Q, if it is the element we want to print on the output, then remove it and print it, otherwise "move" it to

the rear of Q with *add* and *remove*, and consider the present element in the front of Q. Repeat the steps above, we can finally obtain any permutation we want.

(c)

Proof

Once we read all the elements and push them on a stack S1, now consider the topmost element on S1, if it is the element we want to print on the output, then pop it and print it, otherwise "move" it to S2 with *push* and *pop*, repeat the process until the S1 is empty, then change the name of S1 and S2, and start a new traversal.Repeat the steps above, we can finally obtain any permutation we want.

2.15

```
(1) create a stack S,a stack S1,a stack S2,a variable X,a variable num,a vector P[N];
(2) num gets 0;
(3) for i going from 1 to N do the following:
    (3.1) read(X);
    (3.2) P[i] gets X;
(4) for i going from 1 to N-1 do the following:
    (4.1) for j going from i+1 to N do the following:
        (4.1.1) compare P[i] and P[j];
        (4.1.2) if P[j] < P[i],then add 1 to num;
(5) if num is odd,then goto(7);
(6) print the following steps:
    (6.1) for k going from 1 to N do the following:
        (6.1.1) do the following for N times:
            (6.1.1.1) if is-empty(S) returns true,then goto(6.1.1.5);
            (6.1.1.2) compare the topmost element on S with P[k]
            (6.1.1.3) if the topmost element on S is equal to P[k],then pop(X,S);
            (6.1.1.4) print(X);
            (6.1.1.5) read(X);
            (6.1.1.6) compare X with P[k];
            (6.1.1.7) if X is equal to P[k],then print(X);
            (6.1.1.8) if X isn't equal to P[k],then push(X,S);
    (6.2) stop;
(7) print the following steps:
    (7.1) do the following N times:
        (7.1.1) read(X);
        (7.1.2) push(X,S1);
    (7.2) do the following until both is-empty(S1) and is-empty(S2) return true:
        (7.2.1) for r going from 1 to N do the following:
            (7.2.1.1) do the following until is-empty(S1) returns true;
                (7.2.1.1.1) compare the topmost element on S1 with P[r];
                (7.2.1.1.2) if the topmost element is equal to P[r],then pop(X,S1) and print(X)
                (7.2.1.1.3) if the topmost element isn't equal to P[r],then pop(X,S1) and push(
        (7.2.2) for r going from 1 to N do the following:
            (7.2.2.1) do the following until is-empty(S2) returns true;
                (7.2.2.1.1) compare the topmost element on S1 with P[r];
                (7.2.2.1.2) if the topmost element is equal to P[r],then pop(X,S2) and print(X)
                (7.2.2.1.3) if the topmost element isn't equal to P[r],then pop(X,S2) and push(
    (7.3) stop;
(8) stop;
```

2.16

(a)

```
(1) read(X);
(2) take X to be a node N;
(3) do the following until all the integers in the given list are visited:
    (3.1) read(X);
    (3.2) call add-leaf-to N;
(4) stop;
subroutine add-leaf-to N:
(1) compare N with X;
(2) if N is bigger than X:
    (2.1) if N doesn't have left child,then take X to be the left child of N;
    (2.2) if N has a left child,then call add-leaf-to left child of N;
(3) if X is bigger than N:
    (3.1) if N doesn't have right child,then take X to be the right child of N;
    (3.2) if N has a right child,then call add-leaf-to right child of N;
(4) return;
```

(b)

The output would be in the sequence from the biggest to the smallest.