Course ASD3 Solution of the final exam year 2025-2026

Exercise 1 — Reverse a linked list using a stack

```
Procedure Reverse_List(Var L : List)
   S : Stack;
   p, last : List;
Begin
   Initialize(S);
   p ← L;
   While p <> NULL Do
      Push(S, p);
      p ← p->next;
   EndWhile;
   If  Is_empty(S) Then
      Return;
   EndIf;
   p ← Top(S);
   Pop(S);
   L ← p;
   last ← p;
   While not Is_empty(S) Do
      p ← Top(S);
      Pop(S);
      last->next ← p;
      last ← p;
   EndWhile;
   last->next ← NULL;
End ;
```

Exercise 2 — Trees

```
Type structure Date :
   day : integer;
   month : integer;
   year : integer;
END ;

Type structure Package :
   PackageCode : integer;
   Weight : real;
```

```
   Destination : string;
   DepositDate : Date;
END ;

Type structure Node :
   Ele : Package;
   LC  : *Node;
   RC  : *Node;
END ;

Type Tree : *Node ;
Function Create_node(x : Package) : Tree
N : Tree;
Begin
   N ← Allocate(Node);
   N->Ele ← x;
   N->LC ← NULL;
   N->RC ← NULL;
   Return(N);
END
Function insert(a : Tree ; x : Package) : Tree
Begin
   If Is_empty(a) Then
      a ← Create_node(x);
   Else
      If x.PackageCode < Content(a).PackageCode Then
         a->LC ← insert(LeftChild(a), x);
      Else
         If x.PackageCode > Content(a).PackageCode Then
            a->RC ← insert(RightChild(a), x);
         EndIf
      EndIf
   EndIf
   Return(a);
END

Procedure Build_Tree(Var a : Tree)
   n, i : integer;
   x : Package;
Begin
```

```
   Read(n);
   a ← NULL;
   For i ← 1 to n Do
      Read(x.PackageCode);
      Read(x.Weight);
      Read(x.Destination);
      Read(x.DepositDate.day);
      Read(x.DepositDate.month);
      Read(x.DepositDate.year);
      a ← insert(a, x);
   Endfor
END
Procedure HeavyList(a : Tree ; p : real ; Var L : List) //recursive
x : Package;
Begin
   If Is_empty(a) Then
      Return;
   EndIf;
   HeavyList(LeftChild(a), p, L);
   x ← Content(a);
   If x.Weight >= p Then
      Add(L, x);     // insertion at head
   EndIf;
   HeavyList(RightChild(a), p, L);
End
Function Oldest(a : Tree) : Date
   dmin, dleft, dright : Date;
   x : Package;
Begin
   x ← Content(a);
   dmin ← x.DepositDate;
   If not Is_empty(LeftChild(a)) Then
      dleft ← Oldest(LeftChild(a));
      If dleft < dmin Then
         dmin ← dleft;
      EndIf;
   EndIf;

   If not Is_empty(RightChild(a)) Then
      dright ← Oldest(RightChild(a));
```

```
      If dright < dmin Then
         dmin ← dright;
      EndIf;
   EndIf;
   Return(dmin);
End
```

Exercise 3 : Correct deletion of first occurrence :

```
Procedure Delete_First(Var L : List ; x : integer)
   p, prec : List;
Begin
   If Is_empty(L) Then
      Write("Empty list");
      Return;
   EndIf;
   p ← L;
   prec ← NULL;
   While (p <> NULL) and (p->Ele <> x) Do
      prec ← p;
      p ← p->next;
   EndWhile;
   If p = NULL Then
      Write("Value not found");
      Return;
   EndIf;
   If prec = NULL Then
      L ← Rest(L);
   Else
      prec->next ← p->next;
   EndIf;
   Disallocate(p);
End
```

Exercise 4 — Palindrome using one stack (queue restored)

```
Function isPalindrome(Var Q : Queue) : boolean
   S : Stack;
   n, i : integer;
   c : char;
   ok : boolean;
Begin
   Initialize(S);
   n ← Size(Q);

   For i ← 1 to n Do
      Dequeue(Q, c);
      Push(S, c);
      Enqueue(Q, c);
   Endfor;
   ok ← True;
   For i ← 1 to n Do
      Dequeue(Q, c);
      If c <> Top(S) Then
         ok ← False;
      EndIf;
      Pop(S);
      Enqueue(Q, c);
   Endfor;
   Return(ok);
End
```

Version with preorder traversal of the tree :

```
Procedure HeavyList_Preorder(a : Tree ; p : real ; Var L : List)
Var x : Package;
Begin
   If Is_empty(a) Then
      Return;
   EndIf;

   x ← Content(a);
   If x.Weight >= p Then
      Add(L, x);        // insertion at head
```

```
   EndIf;

   HeavyList_Preorder(LeftChild(a), p, L);
   HeavyList_Preorder(RightChild(a), p, L);
End
Function Oldest_Preorder(a : Tree) : Date
Var
   dmin, dleft, dright : Date;
   x : Package;

Begin
   // Precondition: not Is_empty(a)

   x ← Content(a);
   dmin ← x.DepositDate;

   If not Is_empty(LeftChild(a)) Then
      dleft ← Oldest_Preorder(LeftChild(a));
      If dleft < dmin Then
         dmin ← dleft;
      EndIf;
   EndIf;

   If not Is_empty(RightChild(a)) Then
      dright ← Oldest_Preorder(RightChild(a));
      If dright < dmin Then
         dmin ← dright;
      EndIf;
   EndIf;

   Return(dmin);
End
```