Datastructuren najaar 2012 college 1

ADT Stack specificatie

Alle stack elementen zijn van hetzelfde elementaire datatype ElemDT

Relatie: in-> Last-in; Last-in, First-out

Domein: [0,imax]; 0=leeg, imax=vol

Type Stack

Procedure Create(var S: Stack):Boolean

Eindconditie: Create=Y als het reserveren van een ruimte voor S is gelukt, anders N

Procedure Delete(var S:Stack):Boolean

Eindconditie: Delete=Y als ruimte voor stack vrijgegeven, anders N

Operaties op Type Stack:

Procedure Push (var S: Stack, element : ElemDT)

Preconditie: Full(S)=N

Eindconditie: element is top of Stack geworden

Procedure Pop (var S: Stack, var element: ElemDT)

Preconditie: Empty(S)=N

Eindconditie: TOS -> element

Functie Empty(S:Stack):Boolean

Eindconditie: Empty=Y als Stack leeg, anders Empty=N

Functie Full(S:Stack):Boolean

Eindconditie: Full=Y als Stack vol, anders Full=N

ADT Stack: Array implementatie (in Pascal achtige notatie) voor datatype Integer:

```
Const
        Maxsize=nnnn;
Stacktype = record
                Top: 0..Maxsize
                Stack: array [1..Maxsize] of Integer
                end;
Stack   =^Stacktype;
Procedure Push(var S:Stack;i:Integer)
begin
        With S^ do begin
                Top:=Top+1;
                Stack[Top]=i;
        end;
end;
Procedure Pop(var S:Stack;i:Integer)
begin
        With S^ do begin
                I=Stack[Top];
                Top:=Top-1;
        end;
end;
Function Empty (S:Stack): Boolean;
begin
```

```
        Empty := (S^.Top = 0)
end;
Function Full (S:Stack): Boolean;
begin
        Full := (S^.Top = Maxsize)
end;
procedure Create(var S:Stack;var created:Boolean);
begin
        created:=true;
        New(S);
        S^.top :=0;
end;
procedure Delete(var S:Stack);
begin
        Dispose(S)
end;
```

ADT Stack: linked list implementatie in Pascal achtige notatie voor datatype Integer

```
Type
        NodePointer     =       ^Node;
        Stack           =       NodePointer;
        Node            =       record
                                        Int     :       Integer
                                        Next    :       NodePointer
                                end;
procedure Push(var S:Stack;getal:Integer);
var P : NodePointer;
begin
        New(P);
        P^.Int:=getal;
        P^.Next:=S;
        S:=P
end
procedure Pop(var S:Stack;var getal:Integer);
var P:NodePointer;
begin
        getal:=S^.Int;
        P:=S;
        S:=S^.Next;
        Dispose(P)
end;
```

```
function Empty(S:Stack):Boolean;
begin
        Empty:=(S=nil)
end;
function Full(S:Stack):Boolean;
begin
        Full:=false                    #no way in Pascal to test for full, so always not Full
end;
procedure Create(var S:Stack;var created:Boolean);
begin
        S:=nil;
        Created:=True;
end;
procedure Delete(var S:Stack);
var p:NodePointer;
begin
        while S<>nil do begin
                p:=S;
                S:=S^.Next;
                Dispose(S)
        end
end;
```