```pascal
program LinkedList;

uses
  sysUtils;
type
  NodePtr = ^Node;

  Node = record
    data  : Integer;
    next  : NodePtr;
  end;

  LinkedLst = record
    start   : NodePtr;
    finish  : NodePtr;
  end;

  // Add enumerated type to change the operator in the find Node function

procedure CreateList(var list: LinkedLst);
begin
  list.start := nil;
  list.finish := nil;
end;


// Create Node
function CreateNode(data : Integer; next : NodePtr): NodePtr;
begin
  New(result);
  result^.data := data;
  result^.next := next;
end;

function NodeCount(list: NodePtr; value: Integer): Integer;
begin
  result := 0;
  while (Assigned(list)) AND (list^.data <> value) do
  begin
    list := list^.next;
    result := result + 1;
   end;
end;


function FindNode(list: NodePtr; value: Integer): NodePtr;
var
  current : NodePtr;
begin
  result := nil;
  current := list;

  while (Assigned(current)) AND (current^.data <> value) do
  begin
    current := current^.next;
    result := current;
  end;

end;

// new find previous
// function FindPreviousNode(list, ofNode: NodePtr; val: Integer) : NodePtr;
// var
// current, previous : NodePtr;
//
// begin
//   current := list;
//   previous := nil;
//   WriteLn('val in FindPrevNode', val);
//   while (current <> ofNode) and (current <> nil) do
//   begin
//   // pass in the value and it will return the previous node
//    previous := current;
//    current := current^.next;
//   end;
//   result := previous;
// end;



function FindPreviousNode(list: NodePtr; val: Integer) : NodePtr;
var
  i : Integer;
begin
  WriteLn('val in FindPrevNode', val);
  i := 0;
  for i:= 0 to val-2 do
  begin
  // pass in the value and it will return the previous node
    list := list^.next;
    result := list;
  end;
end;

function FindNextNode(list: NodePtr; val: Integer) : NodePtr;
var
  i : Integer;
begin
  WriteLn('val in FindPrevNode', val);
  i := 0;
 for i:= 0 to val-1 do
  begin
  // pass in the value and it will return the previous node
    list := list^.next;
    result := list;
  end;
end;

// Dispose all nodes
procedure DisposeNodes(var start: NodePtr);
begin
  if start <> nil then
  begin
  DisposeNodes(start^.next);
  Dispose(start);
  start := nil;
  end;
end;

// Insert before procedure determined by parameters
procedure InsertBefore(list : NodePtr; beforeVal, val : Integer);
var
  nodeBefore, temp: NodePtr;
begin
  nodeBefore := FindPreviousNode(list, NodeCount(list, beforeVal));
  temp := CreateNode(val, nodeBefore^.next);
  nodeBefore^.next := temp;
end;

// Insert after procedure determined by parameters
procedure InsertAfter(list : NodePtr; beforeVal, val : Integer);
var
  nodeBefore, temp: NodePtr;
begin
  // find the node before
  nodeBefore := FindNextNode(list, NodeCount(list, beforeVal));

  temp := CreateNode(val, nodeBefore^.next);
  nodeBefore^.next := temp;
end;

// Insert At Start of List
procedure PrependNode(var linked: LinkedLst; value: Integer);
var
  temp : NodePtr;
begin
  temp := CreateNode(value, linked.start);
  linked.start := temp;
  if linked.finish = nil then
  begin
    linked.finish := temp;
  end;
end;

procedure AppendNode(var linked: LinkedLst; value: Integer);
var
  temp : NodePtr;
begin
  temp := CreateNode(value, nil);
  if linked.finish <> nil then
  begin
    linked.finish^.next := temp;
  end
  else
  begin
    // list is empty
    linked.start := temp;
  end;

  linked.finish := temp;
end;

// Print Nodes
procedure PrintBackTo(n: NodePtr);
begin
  if (n <> nil) then
  begin
  PrintBackTo(n^.next);
  Write(' <- ', n^.data);
  end
  else
  begin
    Write('nil');
  end;
end;

// Print From Node
// In: NodePtr = list.start
procedure PrintFrom(n: NodePtr);
begin
  if n <> nil then
  begin
  Write(n^.data,' -> ');
  PrintFrom(n^.next);
  end
  else
  begin
  WriteLn('nil');
  end;
    //WriteLn();
end;

// Count nodes
function Count(n: NodePtr):Integer;
var
  current: NodePtr;
begin
  current := n;
  result := 0;
  while (current <> nil) do
  begin
    result +=1;
    current := current^.next;
  end;

end;

procedure Main();
var
  list : LinkedLst;
  find : NodePtr;
begin
  CreateList(list);
  PrependNode(list, 1);
  PrependNode(list, 5);
  PrependNode(list, 10);
  PrependNode(list, 15);
  PrependNode(list, 20);
  AppendNode(list, 99);
  WriteLn('======================');
  PrintFrom(list.start);
  WriteLn('======================');
  WriteLn('Start of the list data ',list.start^.data);
  WriteLn('Finish of the list data ',list.finish^.data);
  PrependNode(list, 30);
  WriteLn('Start of the list data ',list.start^.data);
  PrintFrom(list.start);

  find := FindNode(list.start, 10);

  FindPreviousNode(list.start, 2);
  WriteLn('Node Count :',NodeCount(list.start, 15));


  WriteLn('======================');
  WriteLn('find ', find^.data);

  PrintFrom(list.start);
  WriteLn('======================');

  find :=  FindPreviousNode(list.start, NodeCount(list.start, 10));
  WriteLn('Find Previous Node Function ', find^.data);
  //WriteLn('Find Node ', FindNode(list,20));
  WriteLn('======================');

  InsertBefore(list.start, 15, 18);
  InsertAfter(list.start, 15, 22);
  PrintFrom(list.start);


end;

// Main executable
begin
  Main();
end.
```