Algorithm for Creating a Doubly LL Node...

Function createDLLNode(KEY)

Given a data value, KEY, this function will create a DLL node will return the address of the node. If the availability stack (AVAIL) is full, then it returns a NULL. NEWW is a DLL pointer variable.

1. Create an empty node.
   NEWW ← AVAIL
2. Configure the Node
   If NEWW <> NULL
   DATA(NEWW) ← KEY
   NEXT(NEWW) ← NULL
   PREV(NEWW) ← NULL
3. Return the Node
   Return NEWW.

DLL Node Structure..

```
struct dllNode {
    int data;
    struct dllNode *next;
    struct dllNode *prev;
};
typdedef struct dllNode* dlist;
```

Algorithm for Inserting a Node at a specific position in a DLL

```
Function insertDLLNodePOS(FIRST, POS, KEY)
```

Given a DLL pointer FIRST referencing the first node of the list, this function inserts a node with data values, KEY at the specified position, POS, in the list. KEY is an input-output variable.
On successful insertion it returns the list. Otherwise it returns a MN_VAL into KEY. NEWW and TEMP are DLL pointer variables. KNT is a node counter. POS is a non-zero positive integer.

NOTE:

POS can take values between 1 to FIRST.LENGTH()+1. This indicates that position of start node is 1. If FIRST.LENGTH()← N, then inserting a node as last node will mean, **POS = N + 1.**

1. Initialize local variables.

```
KNT ← 1
```

2. If not a valid position, raise failure.

```
If POS < 1
    Write ('Failure: Incorrect Position Specified.').
    (KEY) ← MN_VAL
    Return FIRST
```

3. Is this an insertion at beginning?

```
NEWW <== DLLNode(KEY)                    .. Call createDLLNode(KEY)
If POS = 1
    If FIRST <> NULL
        NEXT(NEWW) ← FIRST
        PREV(FIRST) ← NEWW
    Return NEWW.
```

```
4.  Save the List.
        TEMP ← FIRST
5.  Traverse the list, till specified position.
        Repeat While NEXT(TEMP) <> NULL AND KNT < POS
            TEMP ← NEXT(TEMP)
            KNT ← KNT + 1
6.  Insert the node, otherwise raise failure.
        If KNT = POS - 1                          .. Insert as last Node
            NEWW <== DLLNode(KEY)
            NEXT(TEMP) ← NEWW
            PREV(NEWW) ← TEMP
        If KNT = POS                              .. Insert Between 2 Nodes
            NEWW <== DLLNode(KEY)
            PREV(NEWW) ← PREV(TEMP)
            NEXT(NEWW) ← TEMP
        If KNT < POS - 1                          .. POS > N + 1
            Write('Failure: Incorrect Position Specified.').
            (KEY) ← MN_VAL
7.  Return the List
        Return FIRST.
```

ALGORITHM FOR DELETING A NODE AT A SPECIFIC POSITION IN A DLL

```
Function DELETEDLLNODEPOS(FIRST, POS, KEY)
    Given a DLL pointer FIRST referencing the first node of the list, this
    function deletes a node at the specified position, POS in the list. KEY is
    an input-output variable and will contain DATA of deleted node.
    On successful insertion it returns the list. Otherwise it returns MN_VAL
    into KEY. NEWW and TEMP are DLL pointer variables. KNT is a node counter.
    POS is a non-zero positive integer.
    NOTE: POS can take values between 1 to FIRST.LENGTH().

1.  Is the list empty??
        If FIRST = NULL
            Write ('Failure: Empty List.').
            (KEY) ← MN_VAL
            Return FIRST.

2.  If not a valid position, raise failure.
        If POS < 1
        `   Write ('Failure: Incorrect Position Specified.').
            (KEY) ← MN_VAL
            Return FIRST

3.  Initialize local variables.
        KNT ← 1

4.  Is this a deletion at beginning?
        If POS = 1
            TEMP ← FIRST
            If FIRST <> NULL
```

```
            FIRST ← NEXT(FIRST)
            PREV(FIRST) ← NULL
        Else
            FIRST ← NULL
        (KEY) ← DATA(TEMP)
        Restore(TEMP)                          .. Return node to AVAIL
        Return FIRST.
```

5. Save the List.

```
    TEMP ← FIRST
```

6. Traverse the list, till specified position.

```
    Repeat While NEXT(TEMP) <> NULL AND KNT < POS
        TEMP ← NEXT(TEMP)
        KNT ← KNT + 1
```

7. Remove the node, otherwise raise failure.

```
    If KNT = POS
        If NEXT(TEMP) = NULL                   .. Removing the last Node
            NEXT(PREV(TEMP)) ← NULL
            PREV(TEMP) ← NULL
        Else                                   .. Removing an embedded Node
            NEXT(PREV(TEMP)) ← NEXT(TEMP)
            PREV(NEXT(TEMP)) ← PREV(TEMP)
        (KEY) ← DATA(TEMP)
        Restore(TEMP)                          .. Return node to AVAIL
```

```
        If KNT < POS                                    .. POS > N
            Write('Failure: Incorrect Position Specified.').
            (KEY) ← MN_VAL
```

8. Return the List

```
        Return FIRST.
```

ALGORITHM FOR DELETING A NODE WITH SPECIFIC DATA VALUE IN A DLL

Function DELETEDLLNODEVAL(FIRST, KEY)

Given a DLL pointer FIRST referencing the first node of the list, this function deletes a node with data value, KEY in the list. KEY is an input-output variable. On successful insertion it returns the list. Otherwise it returns a MN_VAL into KEY. NEWW and TEMP are DLL pointer variables.

1. Is the list empty??

```
        If FIRST = NULL
            Write ('Failure: Empty List.').
            (KEY) ← MN_VAL
            Return FIRST.
```

2. Is this the first node?

```
        If NEXT(FIRST) = NULL AND DATA(FIRST) = KEY
            (KEY) ← DATA (FIRST)
            Restore(FIRST)
            Return NULL.
```

3. Save the List.

```
        TEMP ← FIRST
```

```
4.  Traverse the list, till the key is located.
        Repeat While NEXT(TEMP) <> NULL AND DATA(TEMP) <> KEY
            TEMP ← NEXT(TEMP)
5.  Remove the node, otherwise raise failure.
        If DATA(TEMP) = KEY
            If NEXT(TEMP) = NULL                    .. Removing the last Node
                NEXT(PREV(TEMP)) ← NULL
                PREV(TEMP) ← NULL
            Else [* NEXT(TEMP) <> NULL *]           .. Removing an embedded Node
                NEXT(PREV(TEMP)) ← NEXT(TEMP)
                PREV(NEXT(TEMP)) ← PREV(TEMP)
            (KEY) ← DATA(TEMP)
            Restore(TEMP)                           .. Return node to AVAIL
        If DATA(TEMP) <> KEY
            Write('Failure: Node with data value, KEY not found.').
            (KEY) ← MN_VAL
6.  Return the List
        Return FIRST.
```

ALGORITHM FOR DELETING A NODE FROM BEGINNING OF A DLL

```
Function DELETEBEGDLL(FIRST, KEY)
    Given a DLL pointer FIRST referencing the list, this function deletes the
    first node in the list. KEY is an input-output variable and will contain
    DATA of deleted node. On successful insertion it returns the list. Otherwise
    it returns a MN_VAL into KEY. TEMP is DLL pointer.

    1.  Is the list empty??
            If FIRST = NULL
                Write ('Failure: Empty List.').
                (KEY) ← MN_VAL
                Return FIRST.

    2.  Remove the node.
            TEMP ← FIRST
            FIRST ← NEXT(FIRST)

    3.  Was it the only node?
            If FIRST <> NULL
                PREV(FIRST) ← NULL

    4.  Return the list
            (KEY) ← DATA(TEMP)
            Restore(TEMP)
            Return FIRST.
```

ALGORITHM FOR DELETING A NODE FROM END OF A DLL

```
Function deleteEndDLL(FIRST, KEY)
    Given a DLL pointer FIRST referencing the list, this function deletes the
    last node in the list. KEY is an input-output variable and will contain DATA
    of deleted node. On successful insertion it returns the list. Otherwise it
    returns a MN_VAL into KEY. TEMP is DLL pointer.
    1.  Is the list empty??
            If FIRST = NULL
                Write ('Failure: Empty List.').
                (KEY) ← MN_VAL
                Return FIRST.
    2.  Save the list.
            TEMP ← FIRST
    3.  Traverse till the last node
            Repeat While NEXT(TEMP) <> NULL
                TEMP ← NEXT(TEMP)
    4.  Was it the only node?
            If PREV(TEMP) <> NULL
                NEXT(PREV(TEMP)) ← NULL
            Else
                FIRST ← NULL
    5.  Return the list
            (KEY) ← DATA(TEMP)
            Restore(TEMP)
            Return FIRST.
```

ALGORITHM FOR INSERTING A NODE AT BEGINNING IN A DLL

```
Function INSERTBEGDLL(FIRST, KEY)
    Given a DLL pointer FIRST referencing the first node of the list, this
    function inserts a node with data values, KEY at the start of the list. KEY
    is an input-output variable. On successful insertion it returns the list.
    Otherwise it returns a MN_VAL into KEY. NEWW is a DLL pointer.

    1.  Create the node.
            NEWW <== DLLNode(KEY)                    .. Call createDLLNode(KEY)

    2.  Insert the Node.
            If FIRST <> NULL                         .. DLL with 2+ nodes
                NEXT(NEWW) ← FIRST
                PREV(FIRST) ← NEWW

    3.  Return the list.
            Return NEWW.
```

Algorithm for Inserting a Node at end in a DLL

```
Function INSERTENDDLL(FIRST, KEY)
    Given a DLL pointer FIRST referencing the first node of the list, this
    function inserts a node with data values, KEY at the end of the list. KEY is
    an input-output variable. On successful insertion it returns the list.
    Otherwise it returns a MN_VAL into KEY. TEMP and NEWW are DLL pointers.

    1. Create the node.
           NEWW <== DLLNode(KEY)                    .. Call createDLLNode(KEY)

    2. Is the list empty?
           If FIRST = NULL
               Return NEWW.

    3. Save the list
           TEMP ← FIRST

    4. Traverse the list till last node
           Repeat While NEXT(TEMP) <> NULL
               TEMP ← NEXT(TEMP)

    5. Insert the node
           PREV(NEWW) ← TEMP
           NEXT(TEMP) ← NEWW

    6. Return the list
           Return FIRST.
```

ALGORITHM FOR PRINTING A DLL

```
Function DISPLAYDLL(FIRST)
    Given a DLL pointer FIRST referencing the first node of the list, this
    function prints the contents of the list.
    1.  Is the list empty?
            If FIRST = NULL
                Write ('Failure: Empty List.').
                Return.
    2.  Traverse the list to print node contents
            Repeat While FIRST <> NULL
                Write (DATA(FIRST), ' -> ').
                FIRST ← NEXT(FIRST)
    6.  Conclude the traversal.
            Write (' NULL ').
```