linked list: A list of nodes (entries). Each entry has 2 parts, data part, and link part to the next node. The link part is an address pointer.

singly linked list: each node contains 1 link

list header node: ALL lists have a header node that is a pointer to the first node of list.

last node is represented by: /\ or 0000000 as pointer

what is an empty list?

avail stack - singly linked list of available nodes

```
LISTHDR  DC (0)              This points to a list which nodes are put into

AHDR     DC A(AVAIL)         This points to the avail list
AVAIL    DC CL4' ',A(*+4)
         DC CL4' ',A(*+4)
         DC CL4' ',A(0)       3 nodes are in this sample list
```

see program

```
                                  $NODE DSECT   dsect for avail list
                                  $DATA DS F
                                  $LINK  DS A
popping node from avail stack
 X=AHDR                           assembler code:
IF X = /\                         L  R3,AHDR    get contents of header
  Underflow                      USING $NODE,R3      put dsect over this node
                                  LTR R3,R3
                                  BNZ ELSE
ELSE                                 Print underflow

   AHDR= X --> LINK
ENDIF                             ELSE
                                    MVC  AHDR(4),$LINK   set AHDR  to next node
                                  ENDIF
```

(Assume r3 is "x" value to pop off avail stack)

PUSHING node back to top of stack:

(X (usually a register with a pointer) points to node to return

X --> NODELINK = AHDR move AHDR pointer to be this nodes pointer
AHDR = X Move X (pointer) to AHDR pointer

P.3

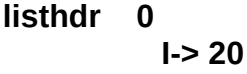
INSERTING INTO SORTED LINK LIST:

I - points to node to insert
S,T - pointer variables to nodes, T will find the insert point, S will trail

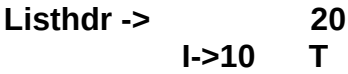
```
*Assume I points to node that has been popped to

T  = LISTHDR

IF (T = NULL    OR    IKEY < TKEY)    CASE 1 -list empty OR
     I--> LINK  = T                    need to insert before node 1
     LISTHDR= I
ELSE
  (CASE 2-insert after node 1 in list, can be middle or end)
  DO WHILE ( T NE NULL )  and (IKEY > TKEY)
    S = T                    set S to previous node
    T = T --> LINK           go to next node
  ENDDO


                        insert in middle or end
   S --> LINK = I   insert to left of node T,
   I --> LINK = T  insert after S



ENDIF
```

CASES FOR INSERTING A NODE IN SORTED LIST

CASE 1a, empty list:



CASE 1b: Need to insert before first node in list, since key of I < key of T



CASE 2, insert key of 40 after node 1, in middle

CASE 2 insert key of 60 at end

DELETE CASES IN SORTED LIST

DELETIONS: delete first node, key of 10

listhdr -> 10

dkey=10

delete node in middle of list, key of 30

listhdr -> 10 20 30 40
S T
dkey=20

DELETING FROM SORTED LINKED LIST (assumes list not empty):

DKEY - key to delete
S,T - pointer variables

```
T = LISTHDR
IF DKEY = TKEY
    LISTHDR–>LINK = T --> LINK       delete node 1
    return node pointed to by T to AVstack
ELSE
    DO WHILE ( T NE NULL)  and  (DKEY NE TKEY)
      S = T
      T = T--> LINK
    ENDDO
    IF DKEY = TKEY              found match
      S--> LINK =  T-->LINK
      return node pointed by T to AV stack
    ELSE
      print 'node not in list'
    ENDIF
ENDIF
```

STACKS

LIFO linear list, add/delete order is what is important

COUNT
variable that tells how many values in stack,
init set to zero
MAX- how many entries can fit in stack

---

proc PUSH

Diagram assume: 4 entries
First entry is 1, second is 2, etc
Max=4, count=0

```
IF COUNT = MAX
  overflow action
ELSE
  COUNT=COUNT+1
  STACK(COUNT) = entry
ENDIF
```

---

proc POP

```
IF COUNT = 0
  underflow
ELSE
  entry = STACK (COUNT)
  COUNT= COUNT - 1
ENDIF
```

---

func(Sfull)

```
IF COUNT=MAX
  return true
ELSE
  return false
ENDIF
```

---

func(Sempty)

```
 IF COUNT= 0
  return true
ELSE
  return false
ENDIF
```

FIFO linear list
insert at rear, pop from front

Front/Rear

set R behind F initially (empty), also first entry is “1"

insertion logic (bump r, place entry, bump r, place entry, etc..)

when F is a distance of 1 from R, queue is empty…but if you fill up queue with 4 elements, it is still a distance of 1!!!!

Solution:leave slot behind f empty, thus full is when distance between f/r is 2

---

note: QSIZE in the following logic is the physical max-4, even though we will only put in 3 entries as the logical max (QMAX)

---

proc Push,Append ,Insert: adds entry to rear of Q

```
IF (Q IS NOT FULL)          call to  Qfull
    R = R + 1           bump
    IF Rear > QSIZE         check for wrap against physical max
      Rear=1
    ENDIF
    Q(Rear)=ITEM        place item

ELSE
    print qfull message or action
ENDIF
```

---

proc Pop,Serve : removes entry from front of Q

```
IF (Q NOT EMPTY)            call to Qempty
    ITEM=Q(Front)           pass item

    Front=Front + 1         bump

    IF Front > QSIZE        test for wrap against physical max
      Front=1
    ENDIF
 ELSE
    print "Q EMPTY" or action
ENDIF
```

function Qempty: returns true/false

```
QR = (R MOD QSIZE) + 1        (QR is just a work value)
IF QR = F
  return TRUE              empty 1  2  3 4
ELSE                                R F
  return FALSE
ENDIF
```

---

function Qfull: returns true/false (checks pos. of r rel to f)

```
QR = (R MOD QSIZE) + 2        2 is for blank entry

IF QR > QSIZE             full     A    B     C
  QR = 1                            1    2    3   4
ENDIF                               F         R

IF QR = F
  return TRUE             full     C        A   B
ELSE                                1   2    3   4
  return FALSE                      R        F
ENDIF
```

---

function SIZE: returns number of entrys in Q

```
IF R > = F
  NUM= (R - F) + 1                Case full 1 above
ELSE
  NUM = QSIZE - (F-R) + 1         Case full 2 above
ENDIF
```
