LINKED LIST

4.4.1 Polynomial Representation

- Polynomials: $a = 3x^{14} + 2x^8 + 1$
  $b = 8x^{14} - 3x^{10} + 10x^6$

- Using linked lists

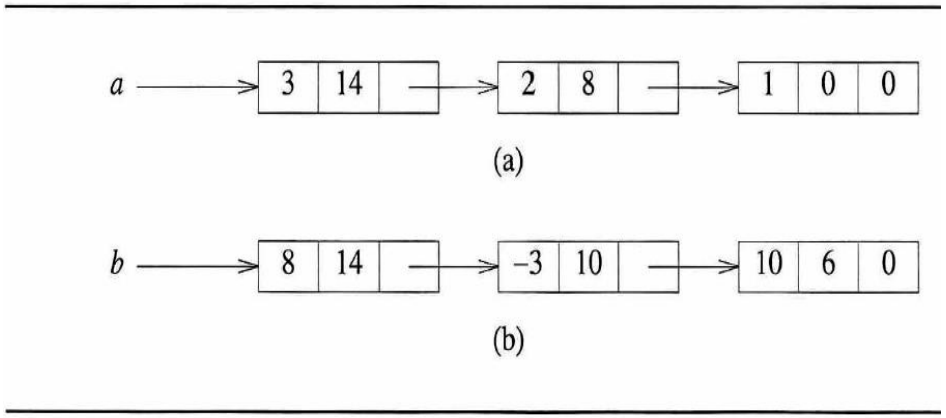


Figure 4.12: Representation of $3x^{14}+2x^8+1$ and $8x^{14}-3x^{10}+10x^6$

- Type declarations:

```
typedef struct polyNode *polyPointer;
struct polyNode {
        int coef;
        int expon;
        polyPointer link;
};
polyPointer a, b;
```

3개의 노드

coef	exp	link

polyNode

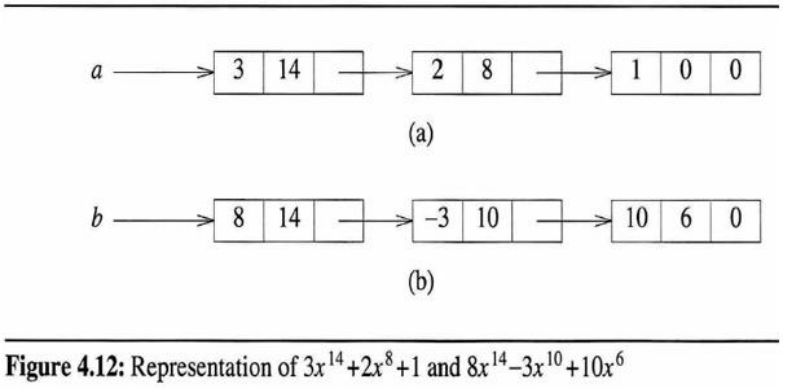


Figure 4.12: Representation of $3x^{14}+2x^{8}+1$ and $8x^{14}-3x^{10}+10x^{6}$

4.4.2 Adding Polynomials: c = a +b

- Examine their terms starting at the nodes pointed to by *a* and *b*

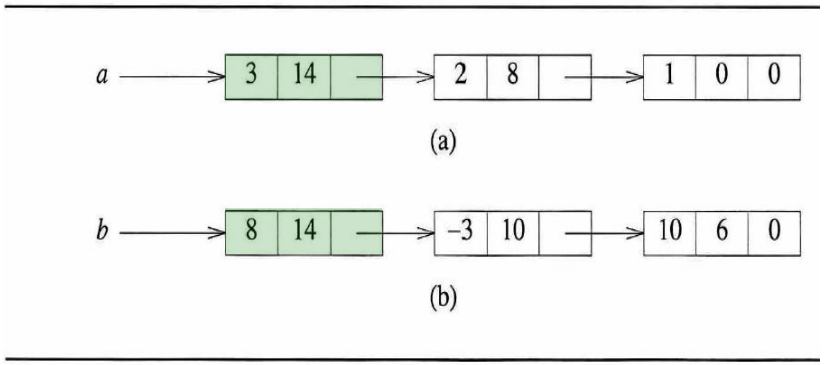
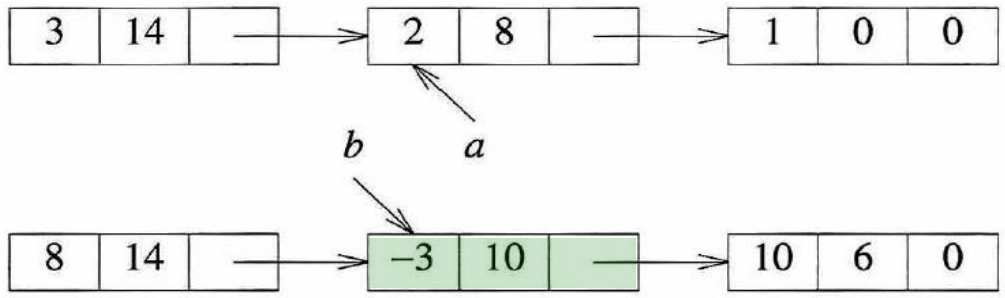


Figure 4.12: Representation of $3x^{14}+2x^{8}+1$ and $8x^{14}-3x^{10}+10x^{6}$

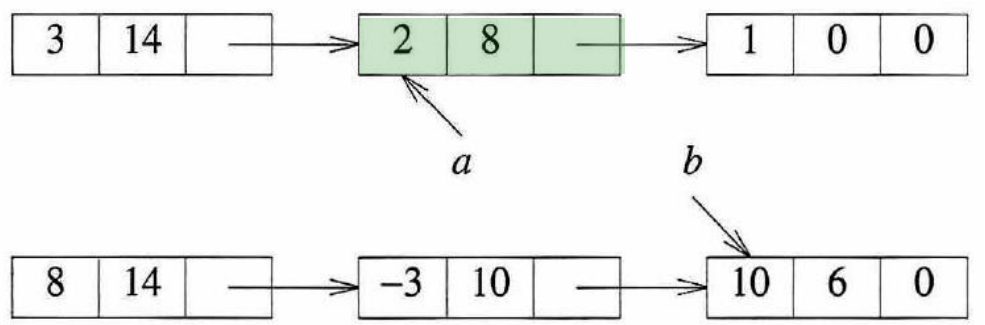
- Case (i): a -> expon == b ->expon
  1) add the two coefficients
  2) create a new term for the result (c)
  3) move the pointers to the next nodes in *a* and *b*

- Case (ii): a->expon < b->expon

  1) create a duplicate term of b
  2) attach this term to the result (c)
  3) advance the pointer to the next term in b

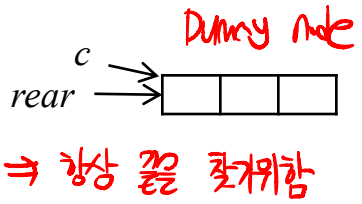


- Case (iii): a->expon > b->expon



```
polyPointer padd(polyPointer a, polyPointer b)
{
   polyPointer c, rear, temp;
   int sum;
   MALLOC(rear, sizeof(*rear));
   c= rear;
   while( a && b )
        switch( COMPARE(a->expon, b->expon) ) {
            case -1:  /* a->expon  <  b->expon */
                      attach(b->coef, b->expon, &rear);
                      b = b->link;
                      break;
            case 0 : /* a->expon == b->expon */
                      sum = a->coef + b->coef;
                      if(sum) attach(sum, a->expon, &rear);
                      a = a->link; b = b->link; break;
            case 1:  /* a->expon > b->expon */
                      attach(a->coef, a->expon, &rear);
                      a = a->link;
        }
   /* copy rest of list a and then list b */
   for(; a; a=a->link) attach(a->coef, a->expon, &rear);
   for(; b; b=b->link) attach(b->coef, b->expon, &rear);
   rear->link = NULL;
   /* delete extra initial node */
   temp = c; c= c->link; free(temp);
   return c;
}
```

```
#define COMPARE (x, y)
( (x) < (y) ? -1 : (x)== (y) ? 0 : 1 )
```



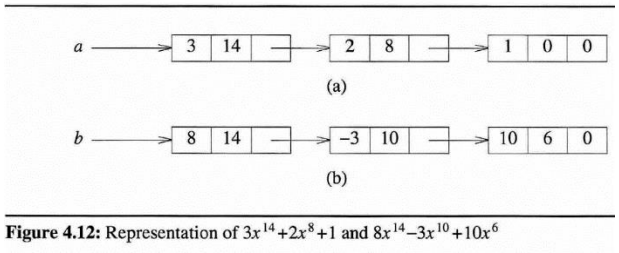
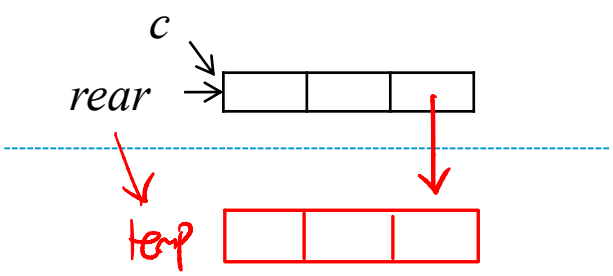


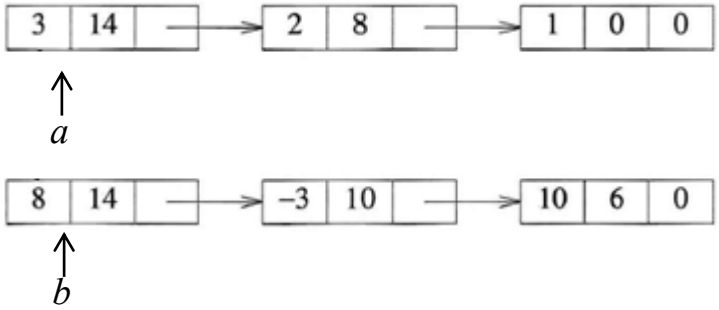
Figure 4.12: Representation of $3x^{14}+2x^8+1$ and $8x^{14}-3x^{10}+10x^6$

Program 4.9:
Add two polynomials

```
void attach(float coefficient, int exponent, polyPointer *ptr)
{
     polyPointer temp;
     MALLOC(temp, sizeof(*temp));
     temp->coef = coefficient;
     temp->expon = exponent;
     (*ptr)->link = temp;
     *ptr = temp;                    /* move ptr to the end of the list */
}
```

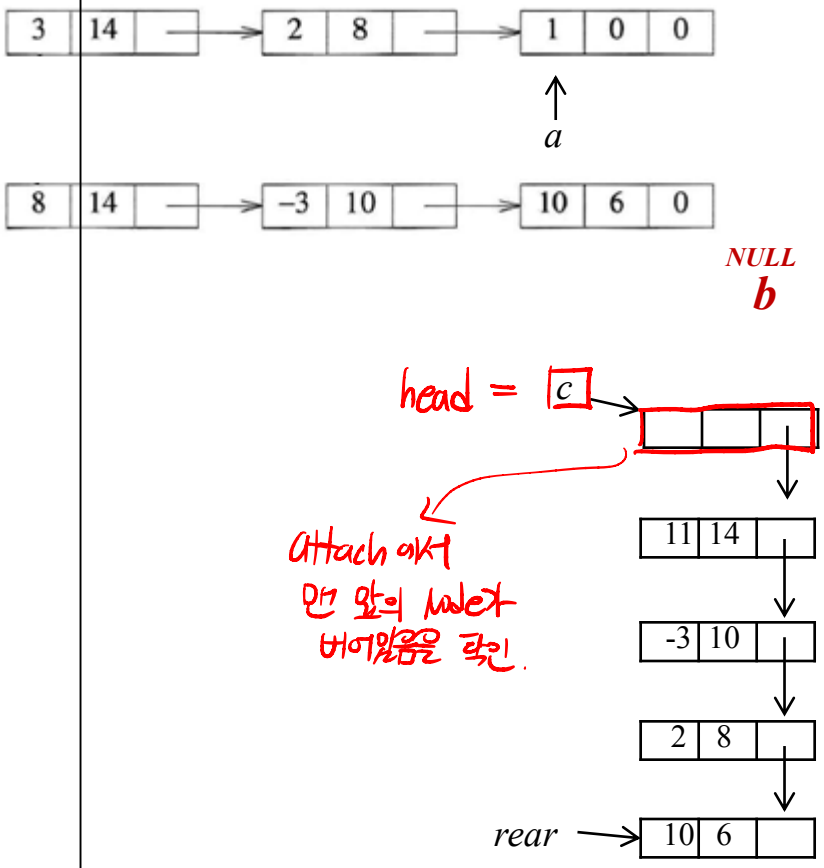
Program 4.10: Attach a node to the end of a list



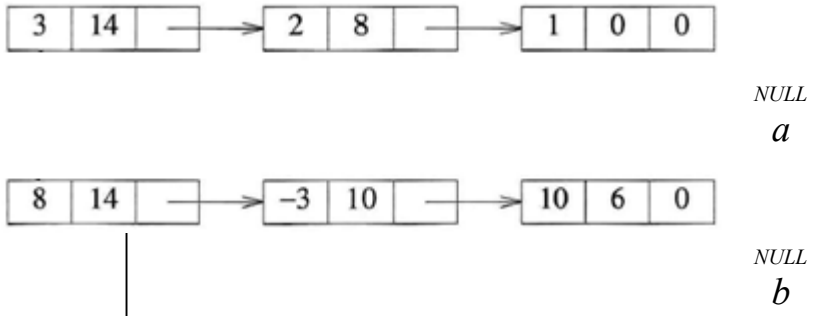


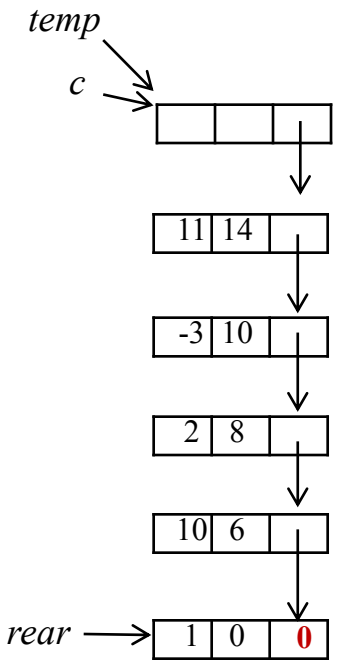
attach(11, 14, &rear)

```
polyPointer padd(polyPointer a, polyPointer b)
{
   polyPointer c, rear, temp;
   int sum;
   MALLOC(rear, sizeof(*rear));
   c= rear;
   while(a && b)
        switch( COMPARE(a->expon, b->expon) ) {
            case -1: /* a->expon  <  b->expon */
                     attach(b->coef, b->expon, &rear);
                     b = b->link;
                     break;
            case 0 : /* a->expon = b->expon */
                     sum = a->coef + b->coef;
                     if(sum) attach(sum, a->expon, &rear);
                     a = a->link; b = b->link; break;
            case 1: /* a->expon > b->expon */
                     attach(a->coef, a->expon, &rear);
                     a = a->link;
        }
   /* copy rest of list a and then list b */
   for(; a; a=a->link) attach(a->coef, a->expon, &rear);
   for(; b; b=b->link) attach(b->coef, b->expon, &rear);
   rear->link = NULL;
   /* delete extra initial node */
   temp = c;   c= c->link;     free(temp);
   return c;
}
```



```
polyPointer padd(polyPointer a, polyPointer b)
{
   polyPointer c, rear, temp;
   int sum;
   MALLOC(rear, sizeof(*rear));
   c= rear;
   while(a && b)
        switch( COMPARE(a->expon, b->expon) ) {
            case -1: /* a->expon  <  b->expon */
                     attach(b->coef, b->expon, &rear);
                     b = b->link;
                     break;
            case 0 : /* a->expon = b->expon */
                     sum = a->coef + b->coef;
                     if(sum) attach(sum, a->expon, &rear);
                     a = a->link; b = b->link; break;
            case 1: /* a->expon > b->expon */
                     attach(a->coef, a->expon, &rear);
                     a = a->link;
        }
   /* copy rest of list a and then list b */
   for(; a; a=a->link) attach(a->coef, a->expon, &rear);
   for(; b; b=b->link) attach(b->coef, b->expon, &rear);
   rear->link = NULL;
   /* delete extra initial node */
   temp = c;   c= c->link;     free(temp);
   return c;
}
```





```
polyPointer padd(polyPointer a, polyPointer b)
{
   polyPointer c, rear, temp;
   int sum;
   MALLOC(rear, sizeof(*rear));
   c= rear;
   while(a && b)
        switch( COMPARE(a->expon, b->expon) ) {
            case -1: /* a->expon  <  b->expon */
                     attach(b->coef, b->expon, &rear);
                     b = b->link;
                     break;
            case 0 : /* a->expon = b->expon */
                     sum = a->coef + b->coef;
                     if(sum) attach(sum, a->expon, &rear);
                     a = a->link; b = b->link; break;
            case 1: /* a->expon > b->expon */
                     attach(a->coef, a->expon, &rear);
                     a = a->link;
        }
   /* copy rest of list a and then list b */
   for(; a; a=a->link) attach(a->coef, a->expon, &rear);
   for(; b; b=b->link) attach(b->coef, b->expon, &rear);
   rear->link = NULL;
   /* delete extra initial node */
   temp = c;   c= c->link;     free(temp);
   return c;
}
```

Analysis of *padd*:…

O(a+b)

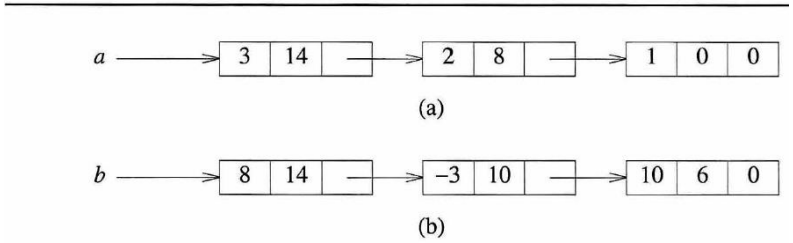
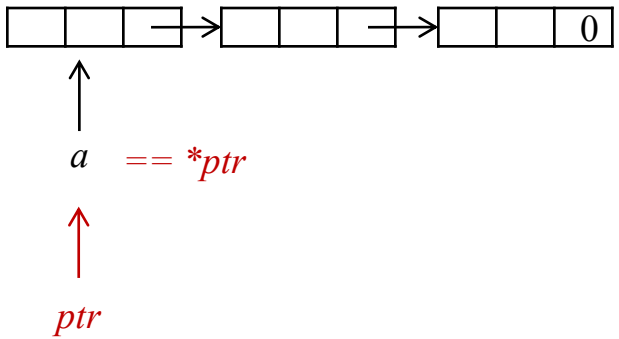


Figure 4.12: Representation of $3x^{14}+2x^8+1$ and $8x^{14}-3x^{10}+10x^6$

4.4.3 Erasing Polynomials

```
void erase(polyPointer *ptr)
{ /* erase the polynomial pointed to by ptr */
    polyPointer temp;
    while(*ptr) {
        temp = *ptr;
        *ptr = (*ptr)->link;
        free(temp);
    }
}
```



erase(&a)

LINKED LIST

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

4.5 Additional List Operations

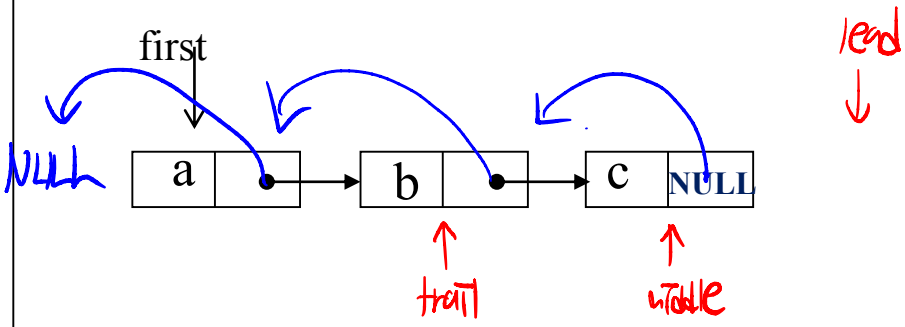
4.7 Sparse Matrixs

4.8 Doubly Linked Lists

4.5.1 Operations For Chains

- Inverting (or reversing) a chain
  - we can do it "in place" if we use three pointers

```
listPointer invert( listPointer lead )
{ /* invert the list pointed to by lead */
   listPointer middle, trail;
   middle = NULL;
   while ( lead ) {
        trail = middle;
        middle = lead;
        lead = lead->link;
        middle->link = trail;
   }
  return middle;
}
```



Program 4.16: Inverting a singly linked list

Time complexity: ...

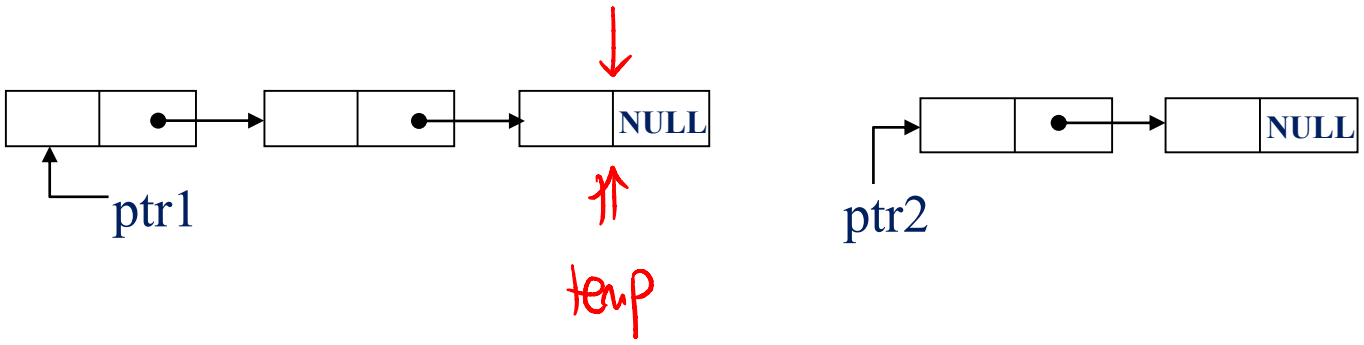
function call:
first = invert(first)

- Concatenating two chains :

```
listPointer concatenate(listPointer ptr1, listPointer ptr2)  {
        listPointer temp;
        /* check for empty lists */
        if(!ptr1) return ptr2;  if(!ptr2) return ptr1;

        /* neither list is empty, find end of first list */
        for(temp = ptr1; temp->link ; temp = temp->link) ;
         /* link end of first to start of second */
        temp->link = ptr2;
        return ptr1;
}
```

시간복잡도 : ptr1의 length

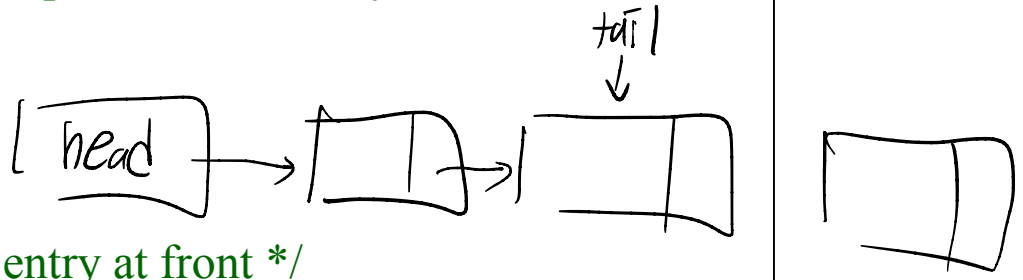
애를 찾아주어야함



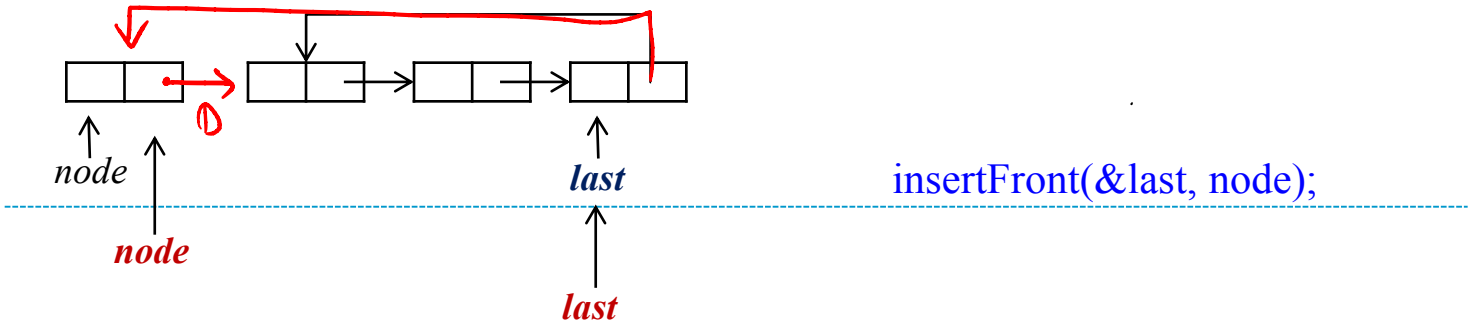
- Time complexity: ...

4.5.2 Operations For Circularly Linked Lists

```
void insertFront(listPointer *last, listPointer node)
{ /* insert node at the front of the circular list whose last node is last */
     if (IS_EMPTY(*last)) {
     /* list is empty, change last to point to new entry */
          *last = node;
           node->link = node;
     }
     else {
    /* list is not empty, add new entry at front */
           node->link = (*last)->link;
           (*last)->link = node;
     }
}
```



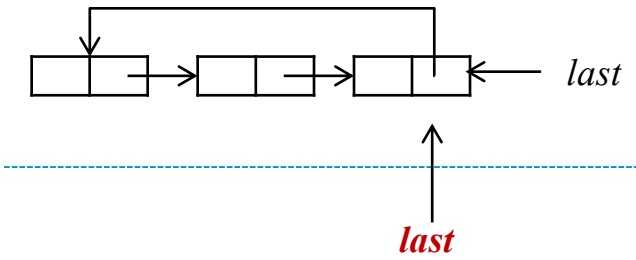
Program 4.18: Inserting at the front of a list



- Finding the length of a circular list

```
int length(listPointer last)
{
    listPointer temp;
    int count = 0;
    if (last) {
        temp = last;
        do {
            count++;
            temp = temp->link;
        } while ( temp != last );
    }
    return count;
}
```

printf(“%d\n”, length(last));

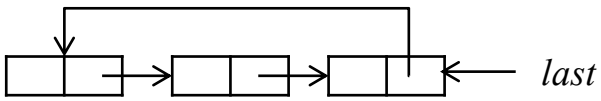


Program 4.19: Finding the length of a circular list

length 함수 버전

```
if ( search(last, x) ) printf("%d\n", x);
```

```
listPointer search(listPointer  last, int x)
{
        listPointer temp = last;
        do {
           if (temp->data == x) return temp;
           temp=temp->link;
        } while  (temp != last);
        return NULL;
}
```



LINKED LIST

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

4.5 Additional List Operations

4.7 Sparse Matrixs

4.8 Doubly Linked Lists

4.7.1 Sparse Matrix Representation

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{bmatrix}$$

Figure 4.18:
5 x 4 sparse matrix *a*

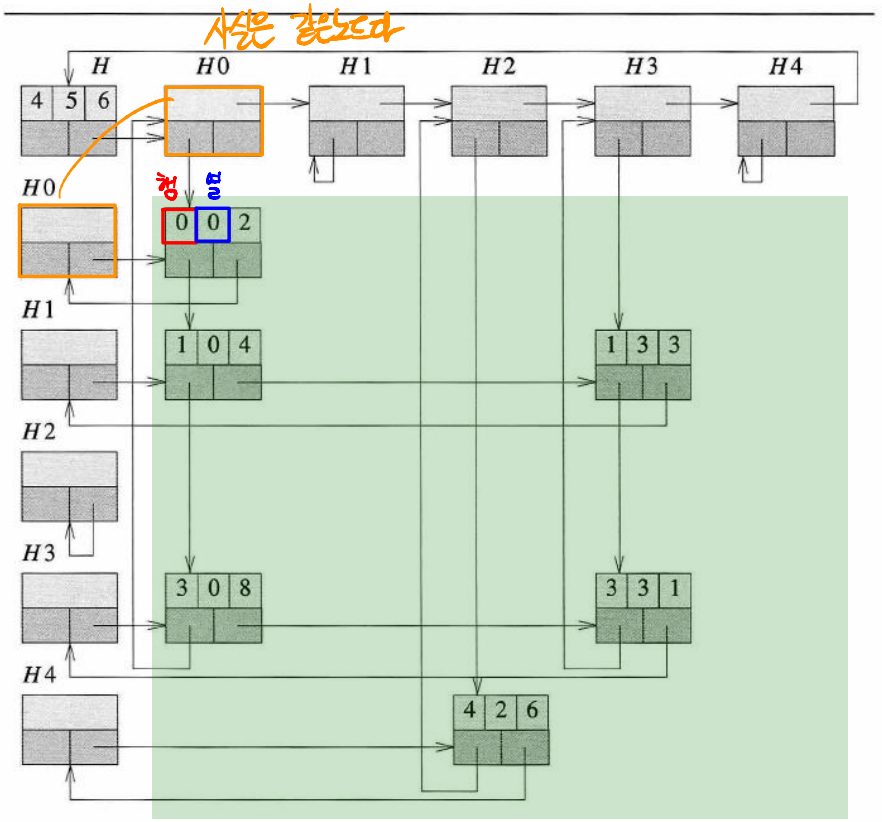


Figure 4.19: Linked representation of the sparse matrix of Figure 4.18 (the *head* field of a node is not shown)

- Node structure for sparse matrices

```
#define MAX_SIZE 50    /* size of largest matrix */
typedef enum {head,entry} tagfield;
typedef struct matrixNode *matrixPointer;
struct entryNode {
        int row;
        int col;
        int value;
        };
struct matrixNode {
        matrixPointer down;
        matrixPointer right;
        tagfield tag;
        union {
              matrixPointer next;
              entryNode entry;
             } u;
        };
matrixPointer hdnode[MAX_SIZE];
```

col

row

< head와 element 구별

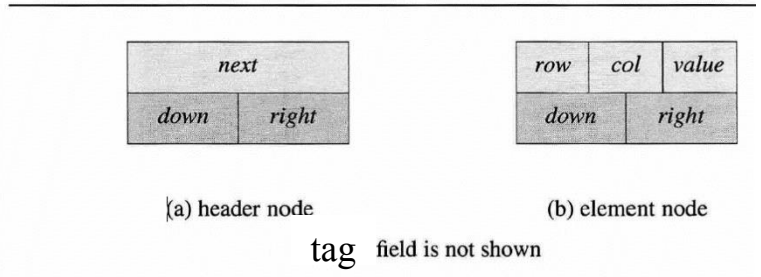


Figure 4.17: Node structure for sparse matrices

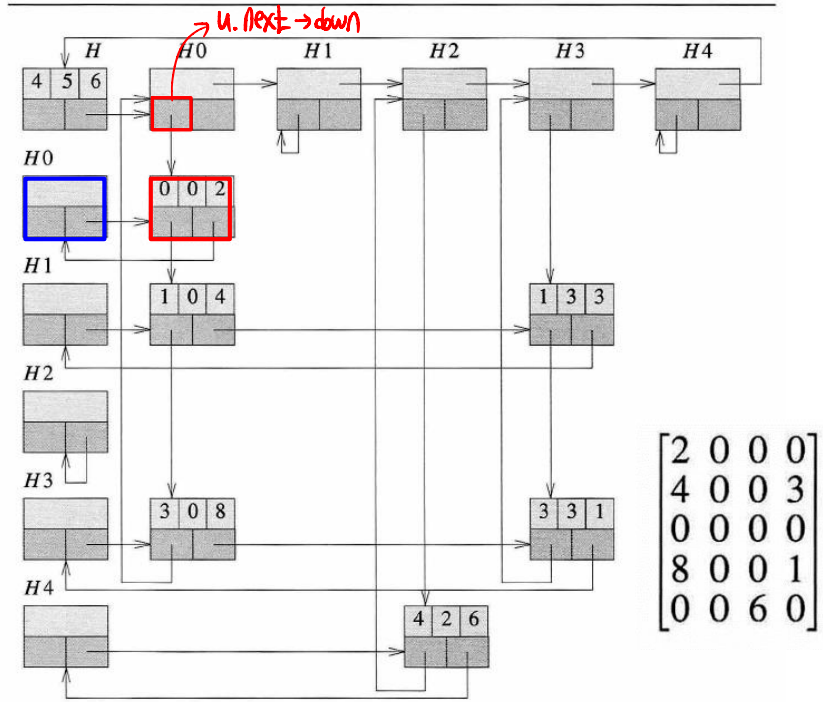
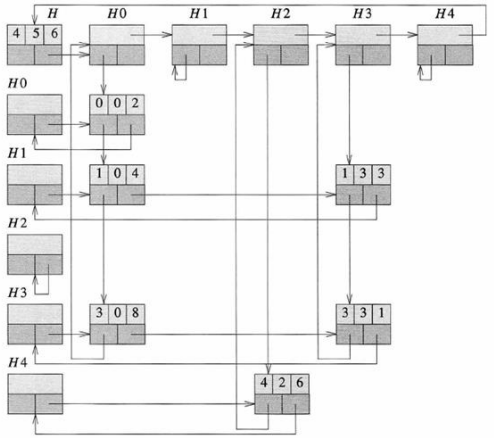


Figure 4.19: Linked representation of the sparse matrix of Figure 4.18 (the *head* field of a node is not shown)

4.7.2 Sparse Matrix Input

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{bmatrix}$$



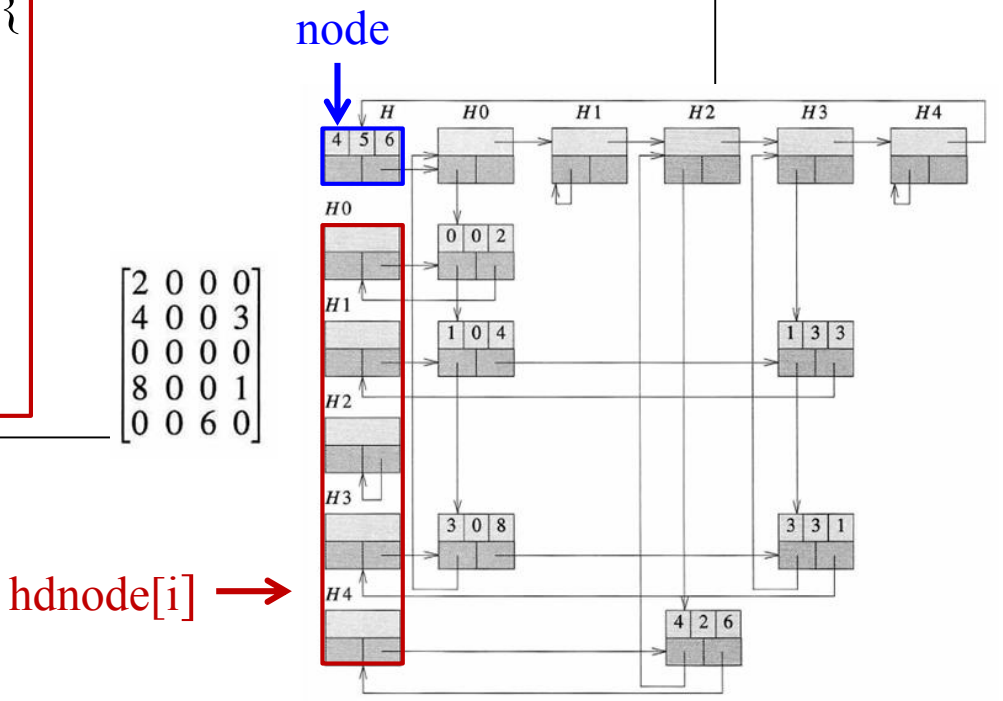

```
matrixPointer mread(void)
{ /* read in a matrix and set up its linked representation. An auxiliary global
array hdnode is used*/
   int numRows, numCols, numTerms, numHeads, i;
   int row, col, value, currentRow;
   matrixPointer temp, last, node;

   printf("Enter the number of rows, columns, and number of nonzero terms: "
   scanf("%d%d%d", &numRows, &numCols, &numTerms);
```

5 4 6

Enter the number of rows, columns, and number of nonzero terms:
5 4 6

```
numHeads = (numCols > numRows) ? numCols : numRows;
/* set up header node for the list of header nodes */
node = newNode();    node->tag = entry;
node->u.entry.row = numRows;
node->u.entry.col = numCols;
if (!numHeads) node->right = node;
else { /* initialize the header nodes */
    for (i = 0; i < numHeads; i++) {
        temp = newNode();
        hdnode[i] = temp;
        hdnode[i]->tag = head;
        hdnode[i]->right = temp;
        hdnode[i]->u.next = temp;
    }
```

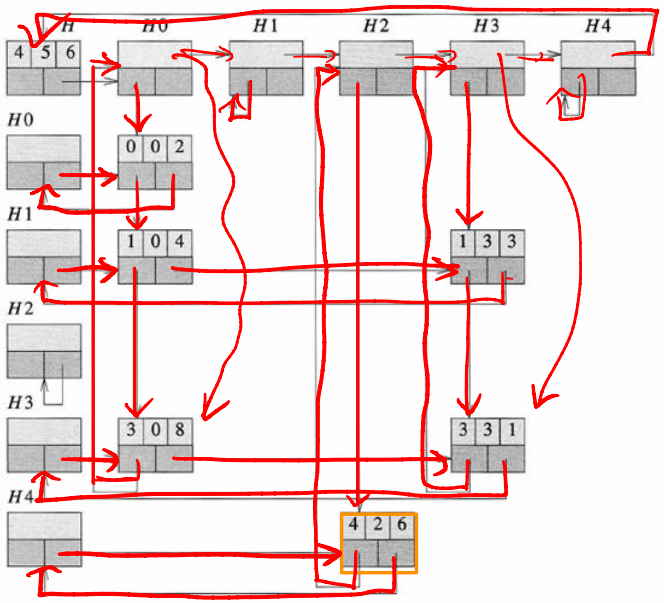
$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{bmatrix}$$



```
currentRow = 0;
last = hdnode[0];  /* last node in current row */
for (i = 0; i < numTerms; i++) {
   printf("Enter row, column and value: ");
   scanf("%d%d%d", &row,&col,&value);

   if (row > currentRow) { /* close current row */
      last->right = hdnode[currentRow];
      currentRow = row;
      last = hdnode[row];
   }
   temp = newNode();
   temp->tag = entry;
   temp->u.entry.row = row;  temp->u.entry.col = col;
   temp->u.entry.value = value;

   last->right = temp; /* link into row list */
   last = temp;
   /* link into column list */
   hdnode[col]->u.next->down = temp;
   hdnode[col]->u.next = temp;
}
/* close last row */
last->right = hdnode[currentRow];
```

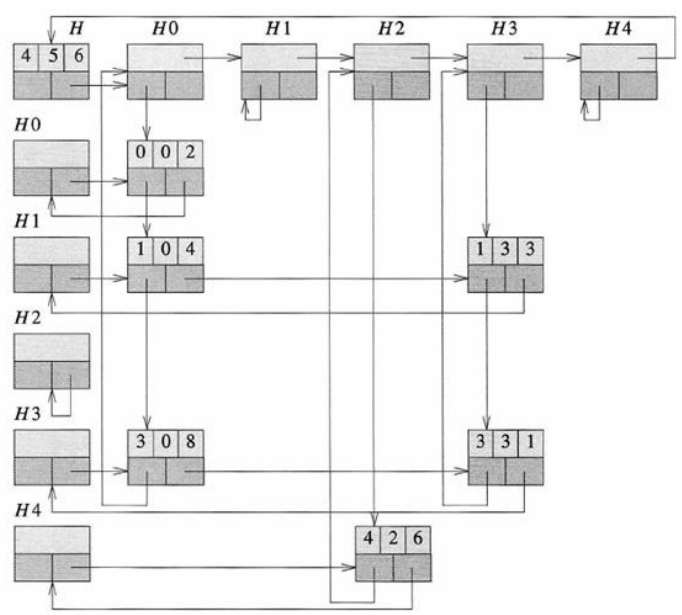
Enter row, column and value:
0 0 2
1 0 4
1 3 3
3 0 8
3 3 1
4 2 6

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{bmatrix}$$



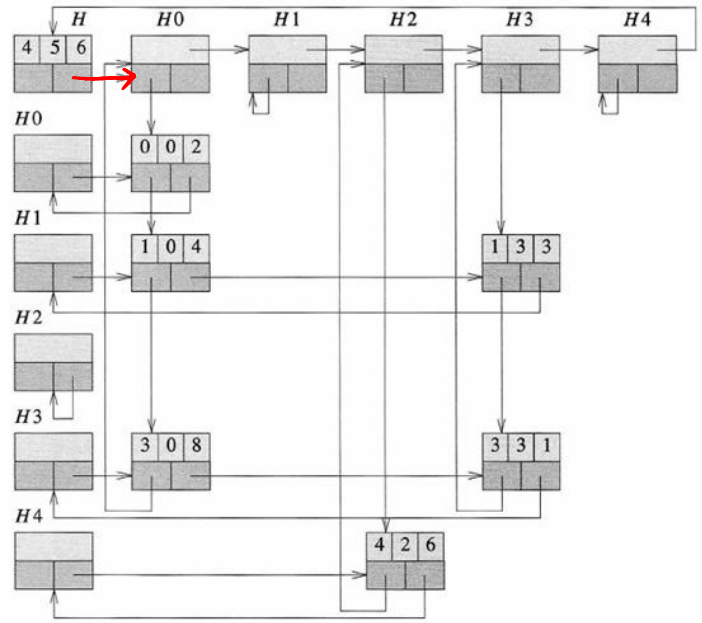
```
    /* close all column lists */
    for (i = 0; i < numCols; i++)
      hdnode[i]->u.next->down = hdnode[i];

    /* link all header nodes together */
    for (i = 0; i < numHeads-1; i++)
      hdnode[i]->u.next = hdnode[i+1];

    hdnode[numHeads-1]->u.next = node;
    node->right = hdnode[0];
  }
  return node;
}
```

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{bmatrix}$$



Program 4.23: Read in a sparse matrix

4.7.3 Sparse Matrix Output

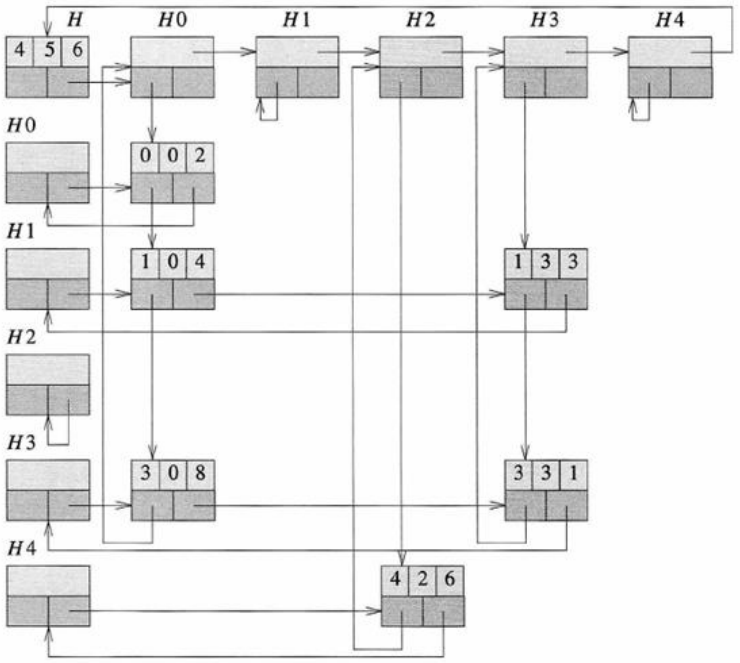



```
void mwrite(matrixPointer node)
{ /* print out the matrix in row major form */
   int i;
   matrixPointer temp,  head = node->right;
   /* matrix dimensions */
   printf("\n numRows = %d, numCols = %d \n",
              node->u.entry.row, node->u.entry.col);

   printf(" The matrix by row, column, and value: \n\n");
   for (i = 0; i < node->u.entry.row; i++) {
   /* print out the entries in each row */
      for ( temp = head->right; temp != head; temp = temp->right )
         printf("%5d%5d%5d \n", temp->u.entry.row,
                                  temp->u.entry.col, temp->u.entry.value);
      head = head->u.next;  /* next row */
   }
}
```

Program 4.24: Write out a sparse matrix

4.7.4 Erasing a Sparse Matrix

```
void merase(matrixPointer *node)
{/* erase the matrix, return the nodes to the heap. */
  matrixPointer x,y, head = (*node)->right;
  int i, numHeads;
  /* free the entry and header nodes by row. */
  for (i = 0; i < (*node)->u.entry.row; i++) {
    y = head->right;
    while ( y != head )
            {x = y;  y = y->right; free(x);}
    x = head;  head = head->u.next;  free(x);
  }
  /* free remaining header nodes. */
  y = head;
  while (y != *node)
            {x = y; y = y->u.next; free(x);}
  free(*node); *node = NULL;
}
```



Program 4.25: Erase a sparse matrix

LINKED LIST

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

4.5 Additional List Operations

4.7 Sparse Matrixs

4.8 Doubly Linked Lists

- Doubly linked lists
  - Move in forward and backward direction
  - A node has at least three fields:
    left link field, data field, right link field

```
typedef struct node *nodePointer;
typedef struct node {
        nodePointer llink;
        element data;
        nodePointer rlink;
        }
```

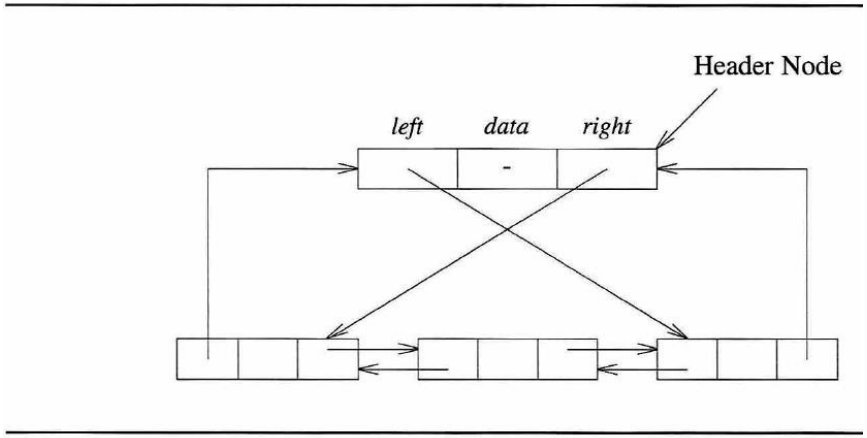


Figure 4.21: Doubly linked circular list with header node

- If *ptr* points to any node in a doubly linked list, then:
  ptr = ptr->llink->rlink = ptr->rlink->llink

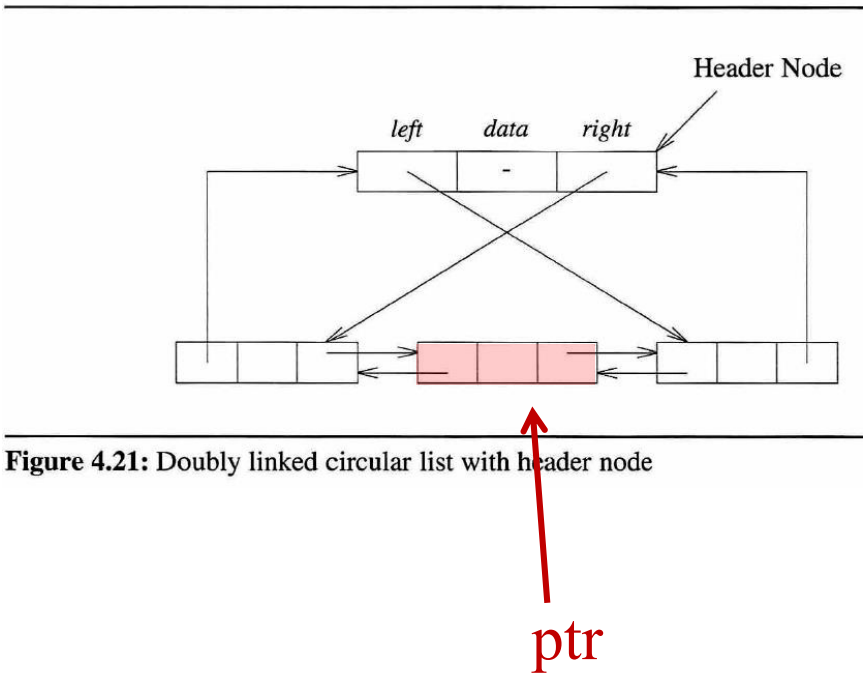


Figure 4.21: Doubly linked circular list with header node

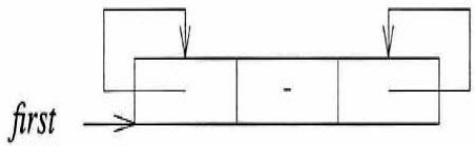


Figure 4.22: Empty doubly linked circular list with header node

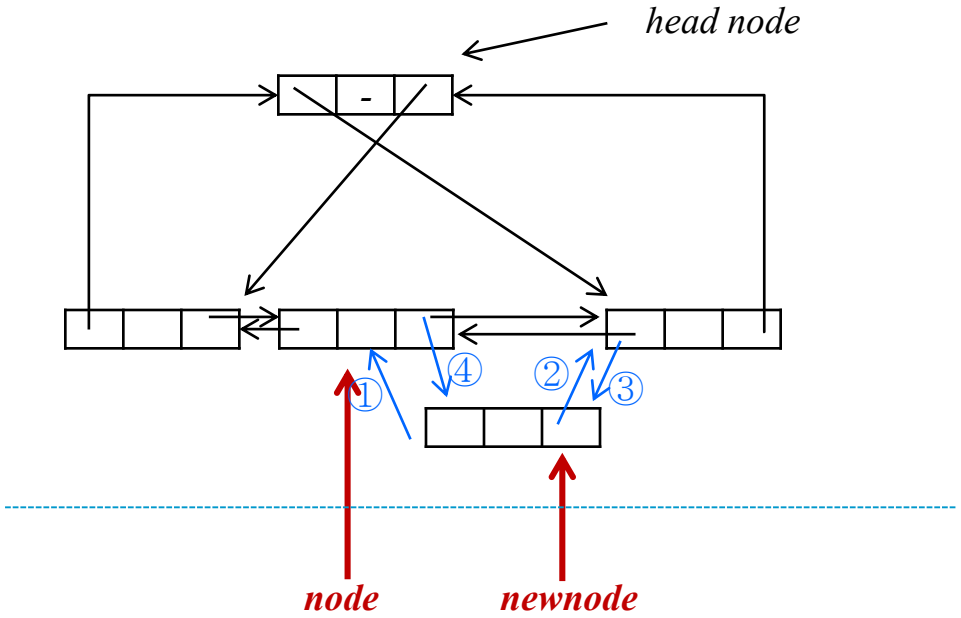
- Insert node 해보기

```
void dinsert(nodePointer node, nodePointer newnode)
{ /* insert newnode to the right of node */
  newnode->llink = node;
  newnode->rlink = node->rlink;
  node->rlink->llink = newnode;
  node->rlink = newnode;
}
```

newnode 부터 연결

그다음 node를 연결

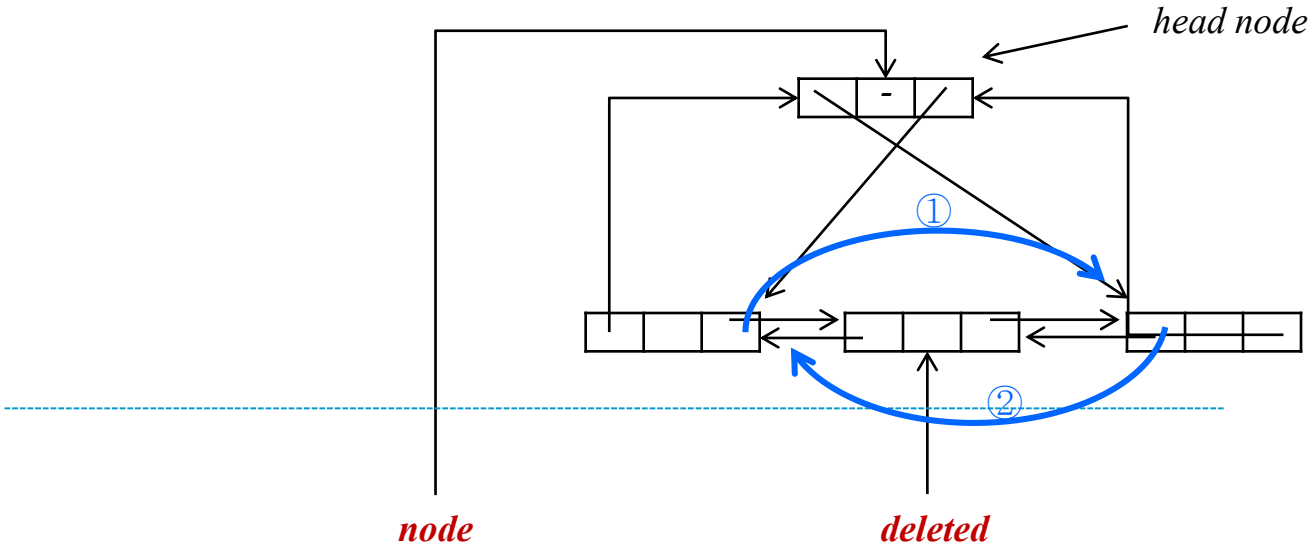
Program 4.26: Insertion into a doubly linked circular list



- Delete node

```
void ddelete(nodePointer node, nodePointer deleted)
{ /* delete from the doubly linked list */
  if (node == deleted)
    printf("Deletion of head node not permitted.\n");
  else {
    deleted->llink->rlink = deleted->rlink;
    deleted->rlink->llink = deleted->link;
    free(deleted);
  }
}
```

Program 4.27: Deletion from a doubly linked circular list



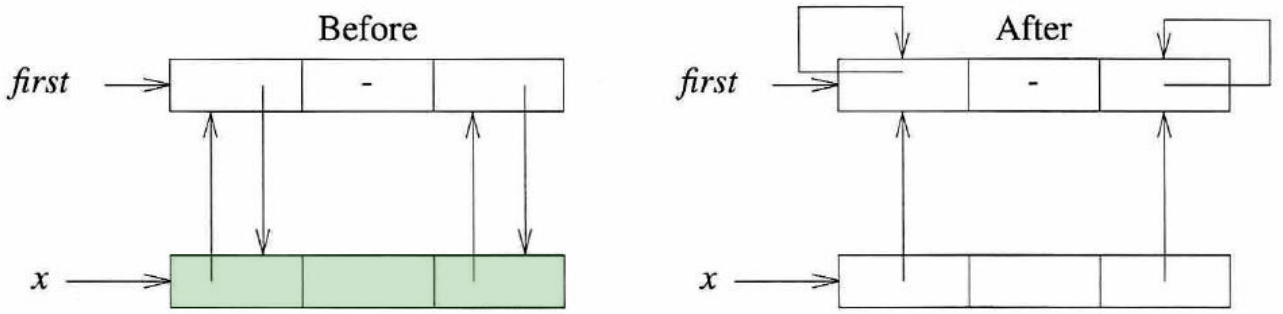


Figure 4.23: Deletion from a doubly linked circular list **with a single node**

```
deleted->llink->rlink = deleted->rlink;
deleted->rlink->llink = deleted->link;
free(deleted);
```

LINKED LIST

4.1 Singly Linked Lists and Chains

4.2 Representing Chains in C

4.3 Linked Stacks and Queues

4.4 Polynomials

4.5 Additional List Operations

4.7 Sparse Matrixs

4.8 Doubly Linked Lists