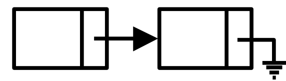
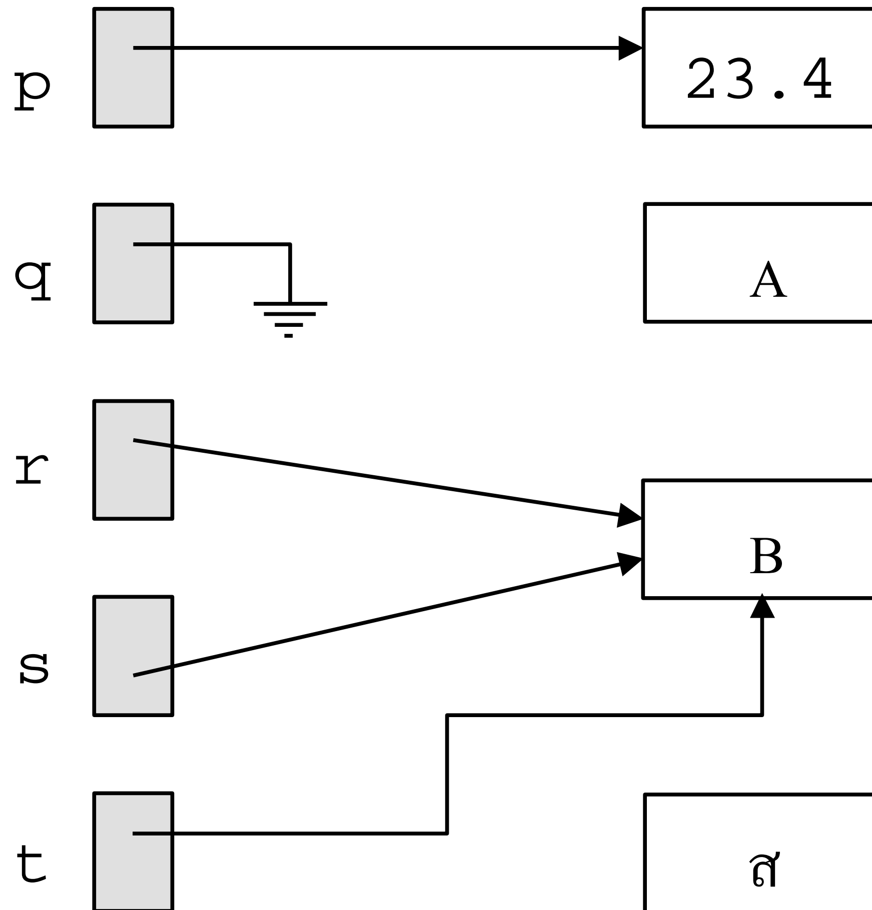
LINKED LISTS



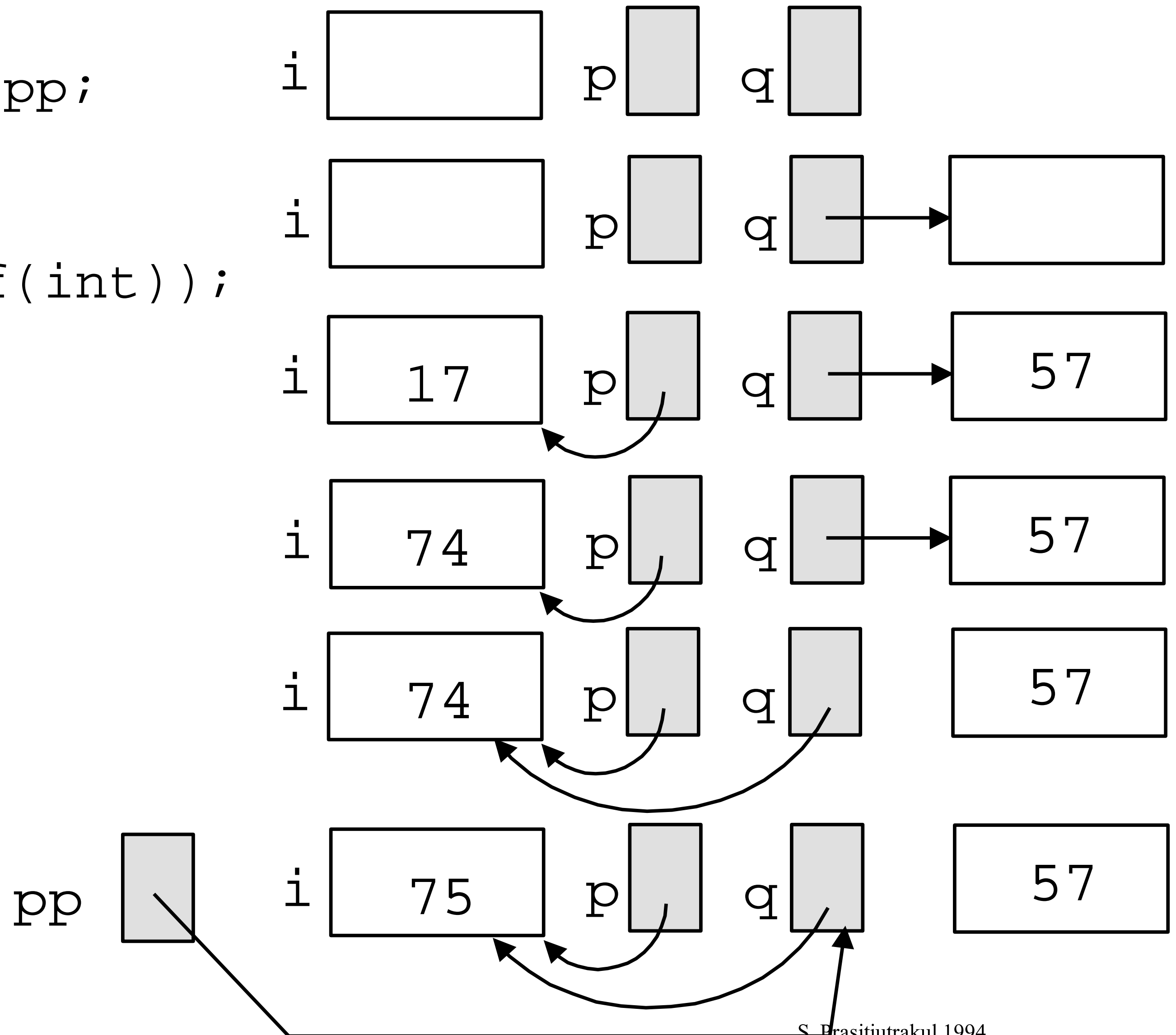
Pointers




```
float   *p;
char    *q, *r;
char    *s, *t;
.
.
.
printf("%f %s\n", *p,*t);
printf("%c %s\n", *r,*s);
```

Pointers

```
int      i;
int      *p, *q, **pp;

q = (int *)
     malloc(sizeof(int));
i  = 17;
p  = &i;
*q = 57;
*p = i + *q;
q  = p;
pp = &q;
(**pp)++;
```



S. Prasitjutrakul 1994

Call-by-Value & Call-by-Reference

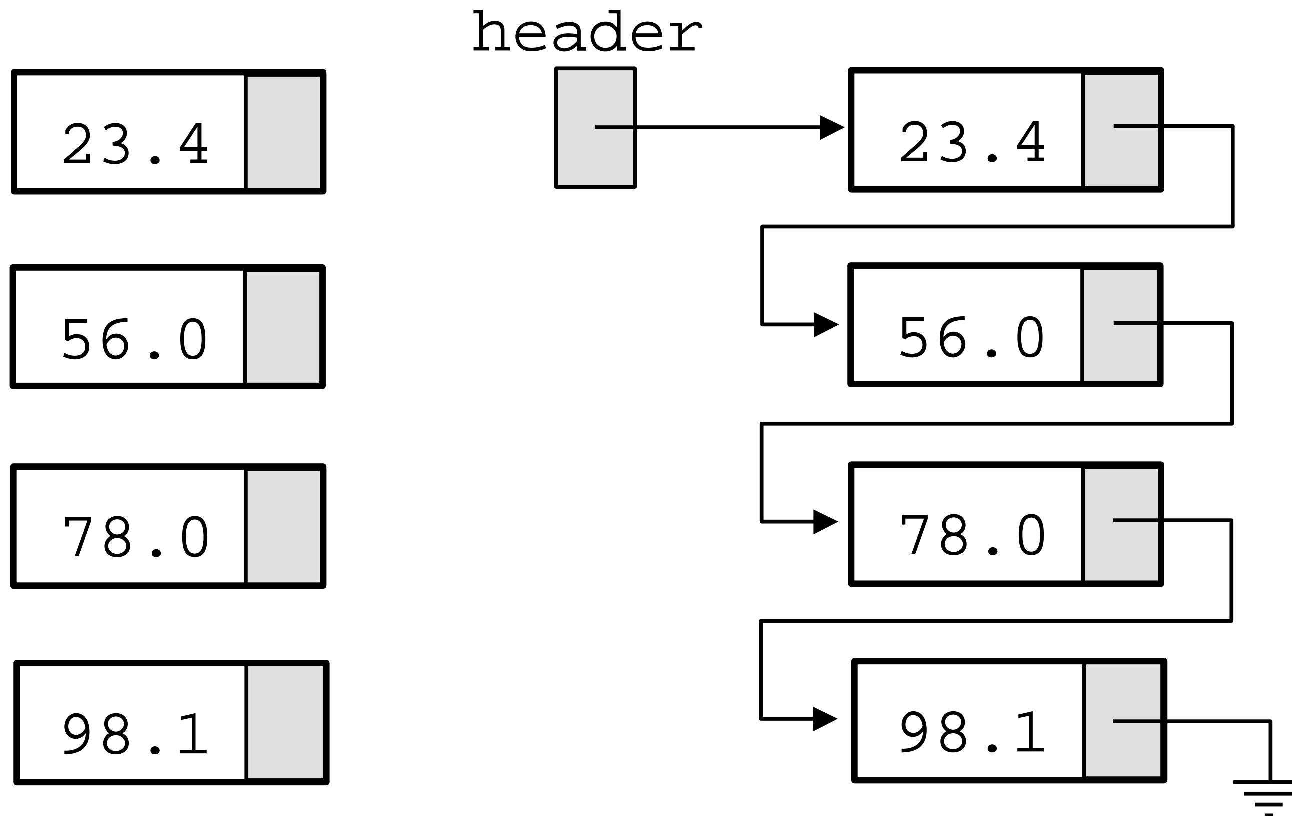
```
void Swap( int p, int q )
{
  int     t;

  t = p;
  p = q;
  q = t;
}
```

```
a = 1;
b = 2
Swap( a, b );

p ←a; q ←b;
t ←p; p ←q; q ←t;
```

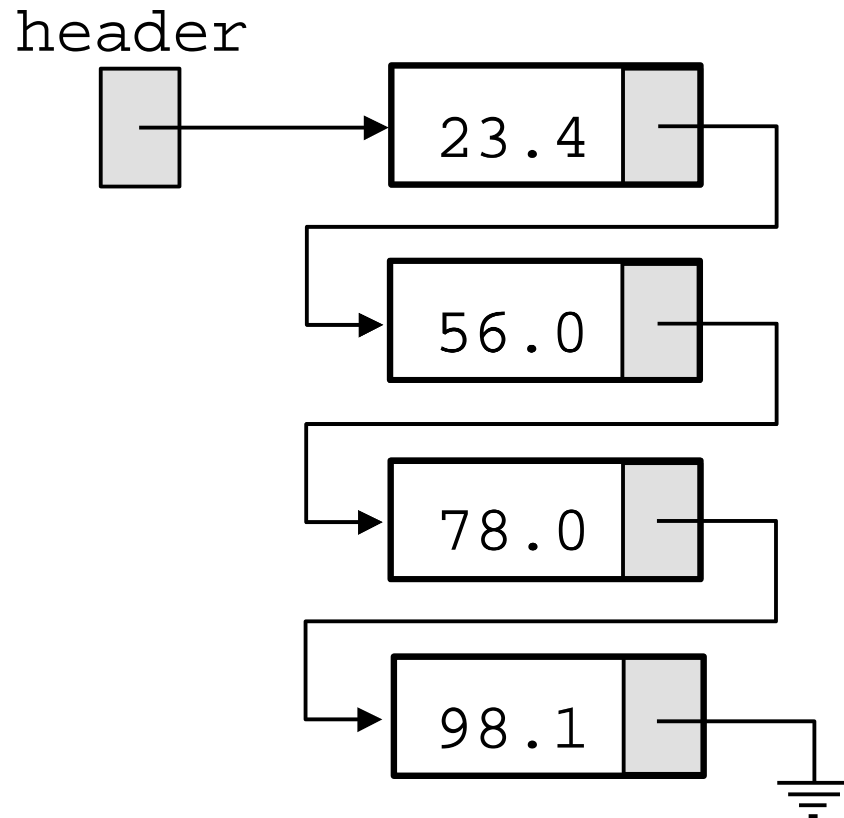
```
void Swap( int *p, int *q )
{
  int     t;

  t  = *p;
  *p = *q;
  *q = t;
}
```

```
a = 1;
b = 2
Swap( &a, &b );

p ←&a; q ←&b;
t ←*p; *p ←*q; *q ←t;
t ←*a; *a ←*b; *b ←t;
```

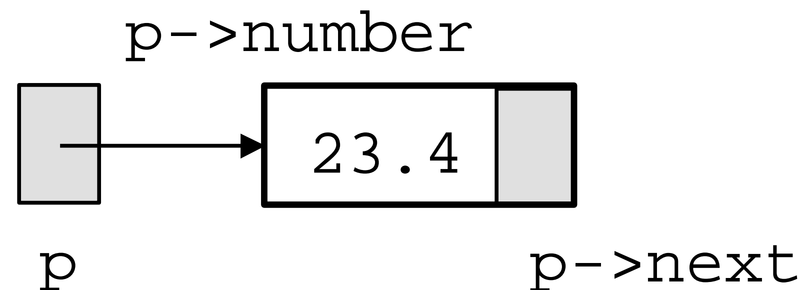
Linked Lists



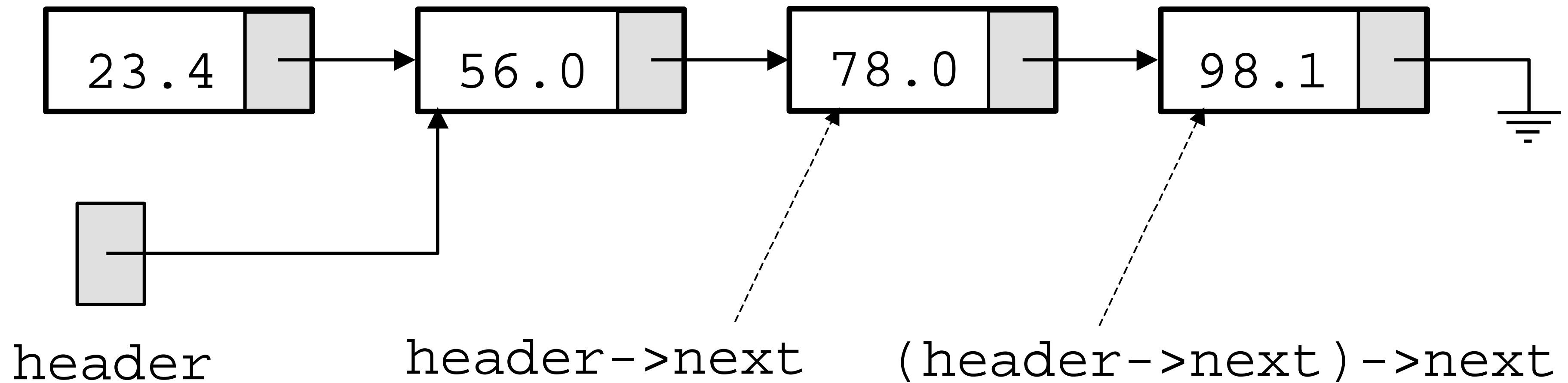
Linked List Structures




```
typedef struct NodeTag {
        float              *number
        struct NodeTag *next;
} NodeType;

NodeType    *header;
```

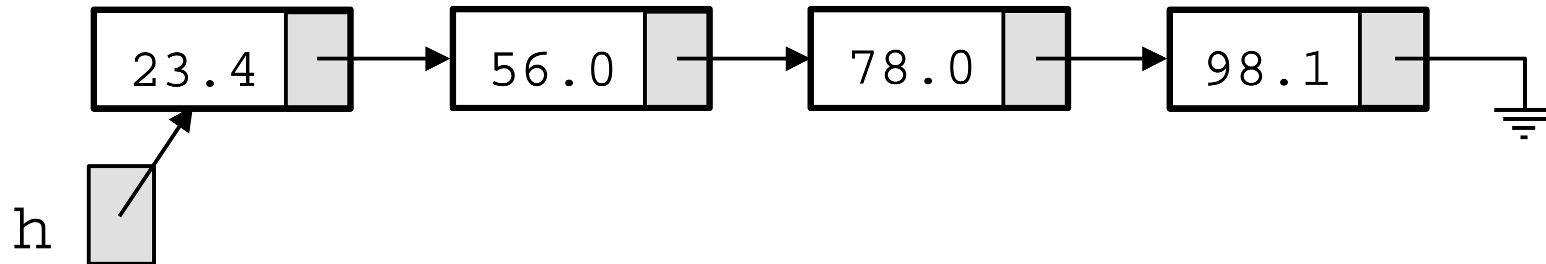


Linked Lists




```
printf("%f\n",header->number);
printf("%f\n",(header->next)->number);
header = header->next;
header = header->next;
printf("%f\n",header->number);
header = header->next;
```

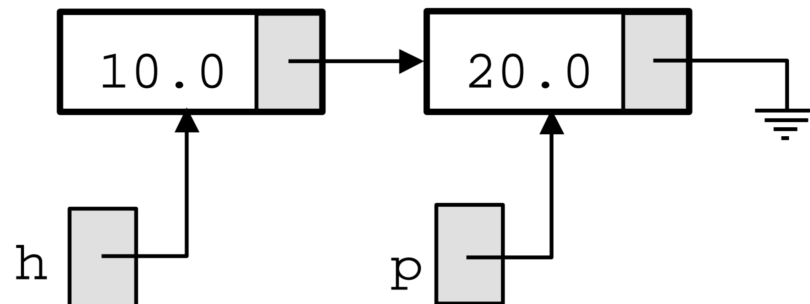
Example: List Size



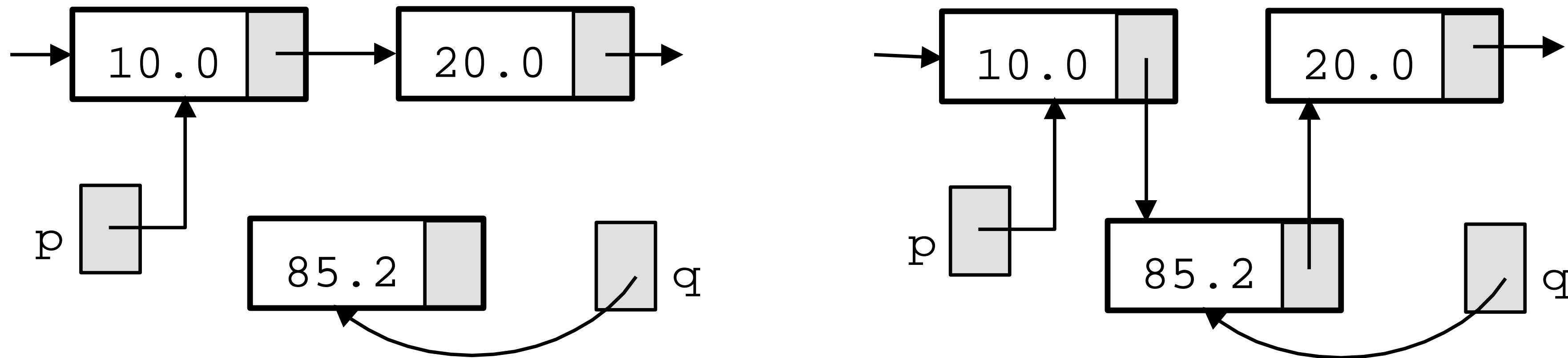

```
int  Size( NodeType *h )
{
  int     size = 0;
  while ( h != NULL ) {
    size++;
    h = h->next;
  }
  return( size );
}
```

Dynamic Memory Allocation

```
typedef struct NodeTag {
     float            *number;
     struct NodeTag     *next;
} NodeType;

NodeType     *h, *p;

h = (NodeType *) malloc( sizeof(NodeType) );
p = (NodeType *) malloc( sizeof(NodeType) );
h->number = 10.0;
p->number = 20.0;
h->next   = p;
p->next   = NULL;
```

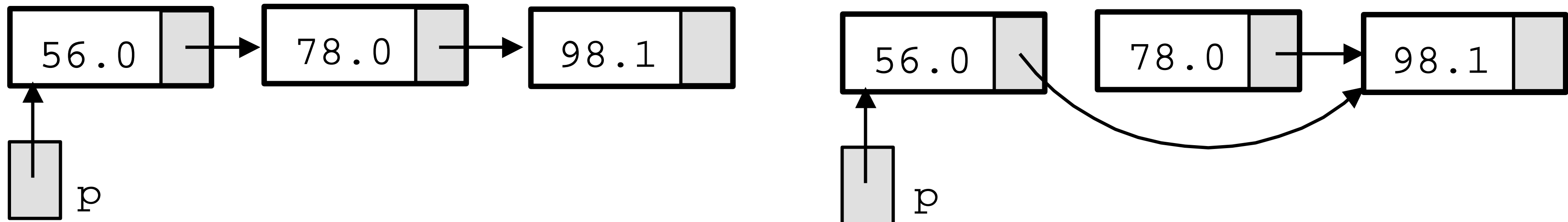


Insert after a Node



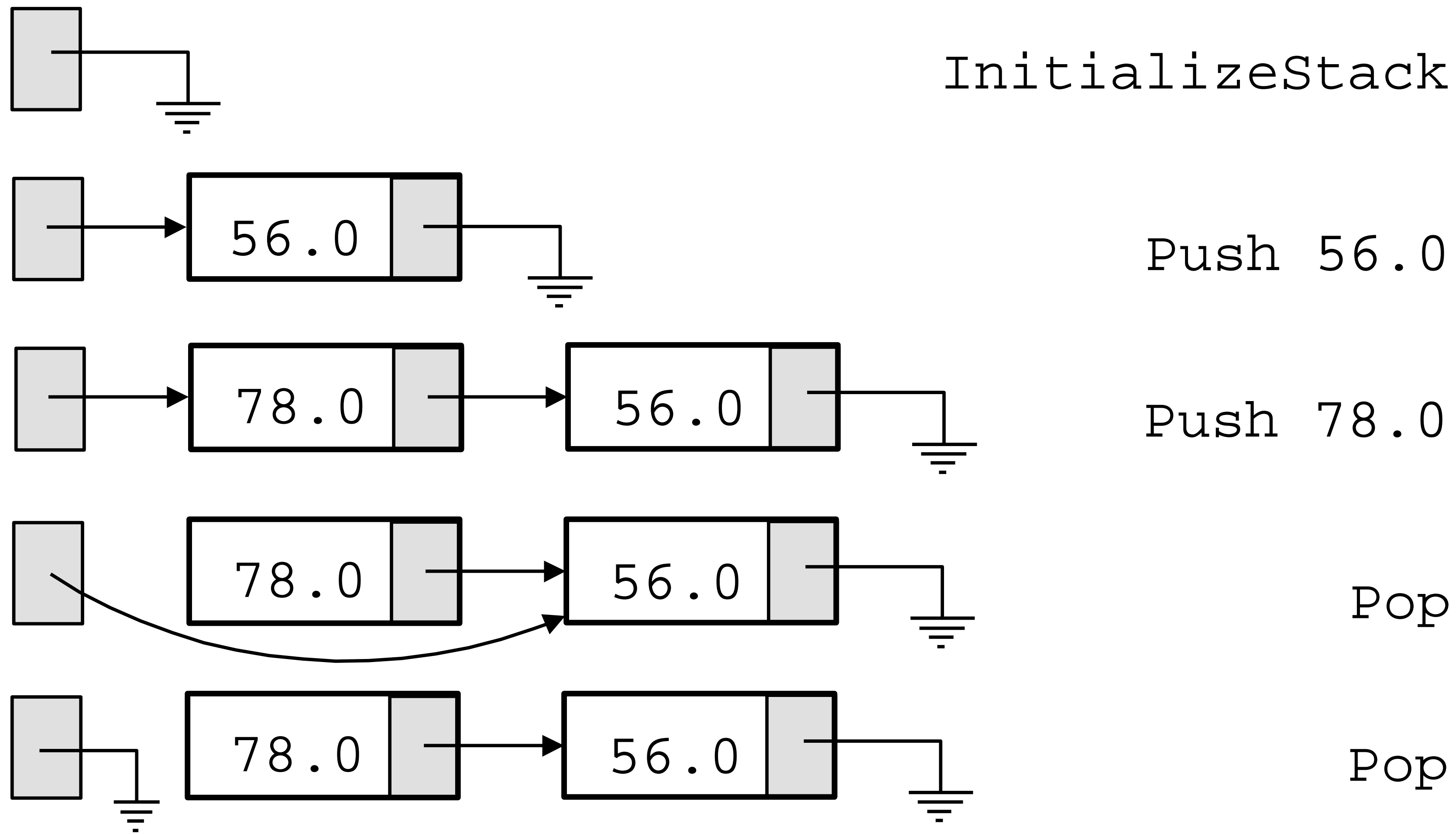

```
void InsertAfter( NodeType *p, NodeType *q )
{
     if ( p == NULL || q == NULL )
     Error( "At lease one of the nodes is NULL" );
   else {
     q->next = p->next;
     p->next = q;
   }
}
```

Delete a Node



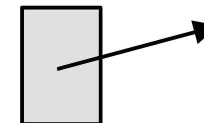

```
void DeleteAfter( NodeType *p )
{
     if ( p == NULL || p->next == NULL )
     Error( "The node is nonexistent" );
   else
     p->next = p->next->next;
}
```

Pointer Implementation of Stacks



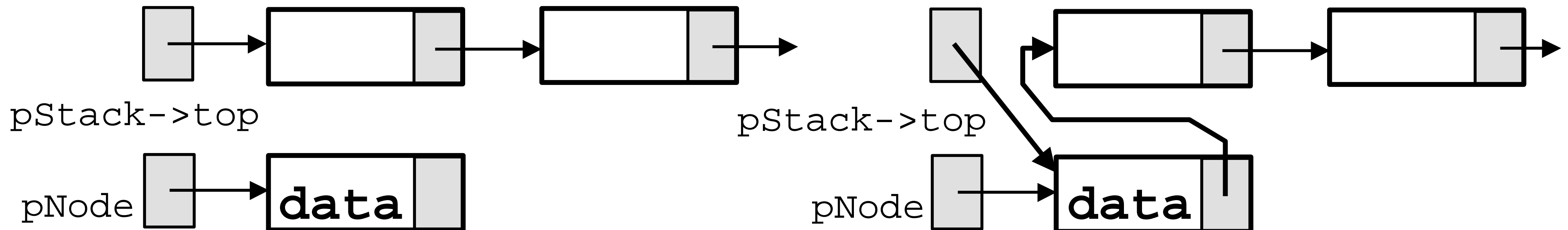
Pointer Implementation of Stacks

```
typedef  struct NodeTag {
   ItemType           info;
     struct NodeTag  *next;
} NodeType;

typedef struct StackTag {
     NodeType             *top;
} StackType;

StackType    stack;
StackType    *pStack = &stack;
```



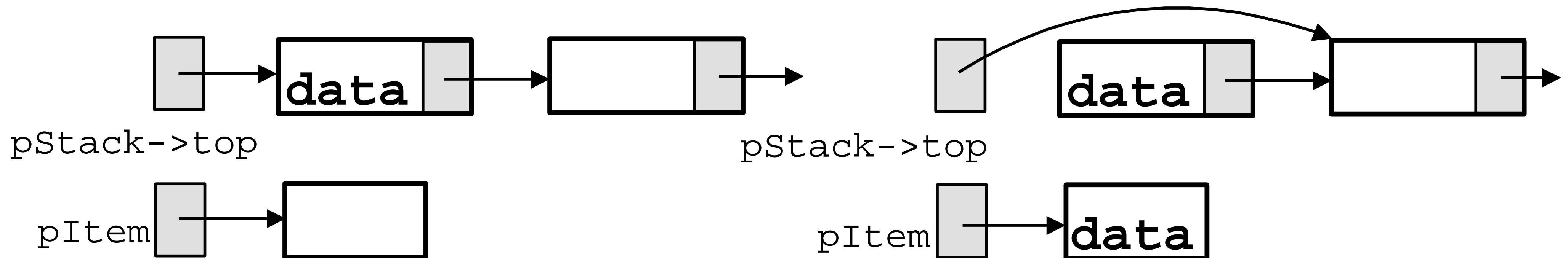
```
stack.top
pStack->top
```

Stack: Pushing a Node



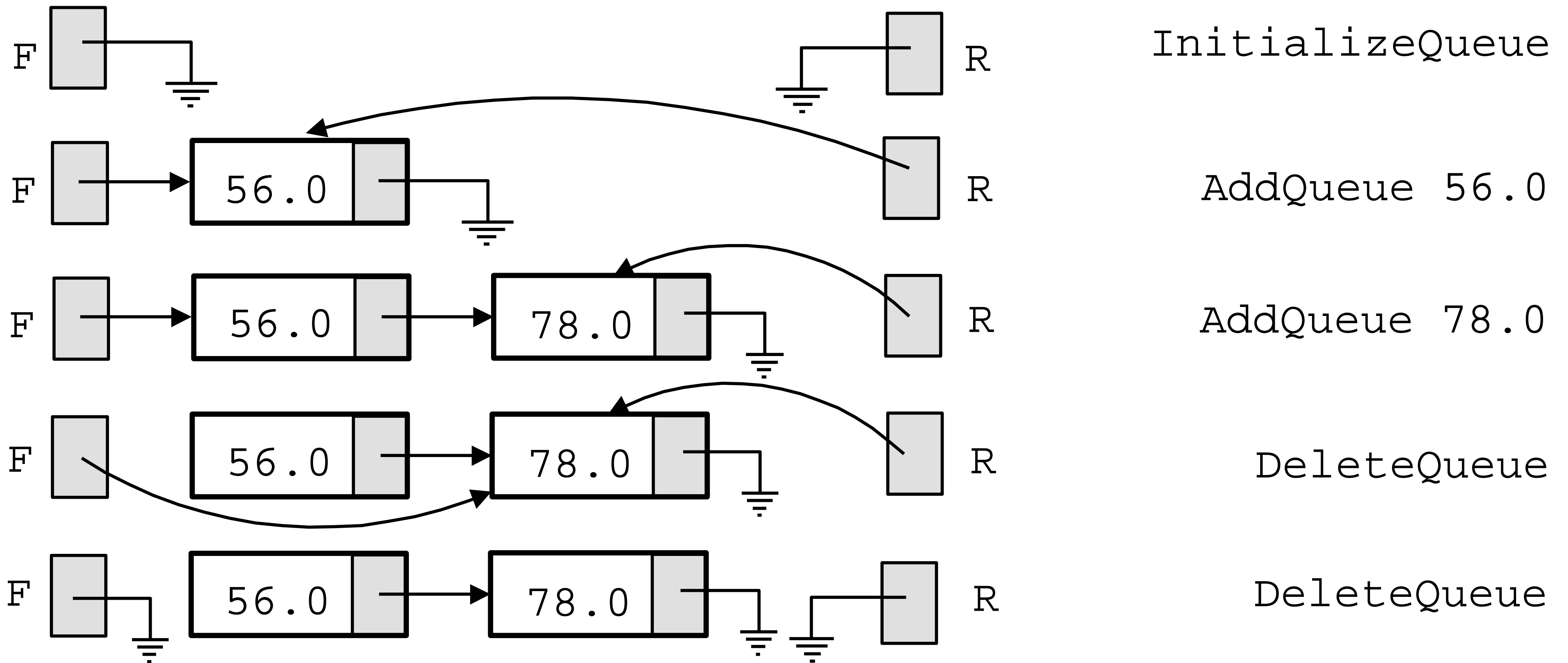

```
void Push( ItemType Item, StackType *pStack )
{
  NodeType  *pNode;

  pNode = (NodeType *) malloc( sizeof( NodeType ) );
  if ( pNode == NULL ) Error("Exhausted memory");
  else {
    pNode->info = Item;
    pNode->next = pStack->top;
    pStack->top = pNode;
  }
}
```

Stack: Poping a Node




```
void Pop( ItemType *pItem, StackType *pStack )
{
  NodeType       *pNode;
  if ( pStack->top == NULL ) Error("Empty stack");
  else {
    pNode       = pStack->top;
    *pItem      = pNode->info;
    pStack->top = pNode->next;
    free( pNode );
  }
}
```

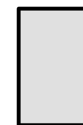
Pointer Implementation of Queues



Pointer Implementation of Queues

```
typedef  struct NodeTag {
   ItemType           info;
     struct NodeTag  *next;
} NodeType;
```

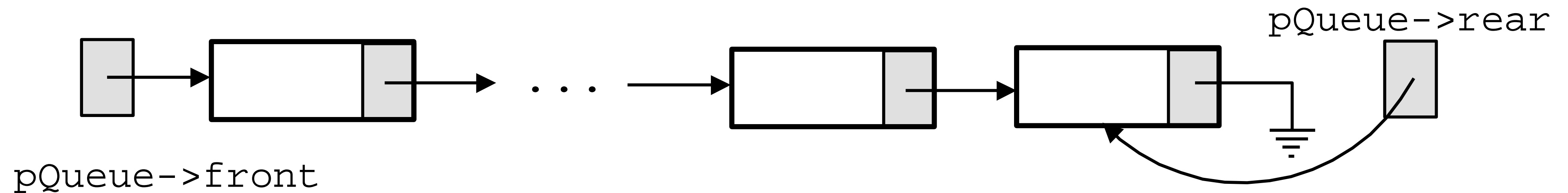
```
typedef struct QueueTag {
     NodeType           *front, *rear;
} QueueType;

QueueType    queue;
QueueType    *pQueue = &queue;
```

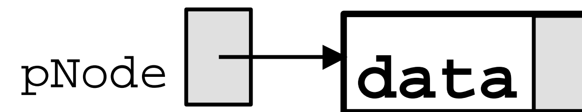


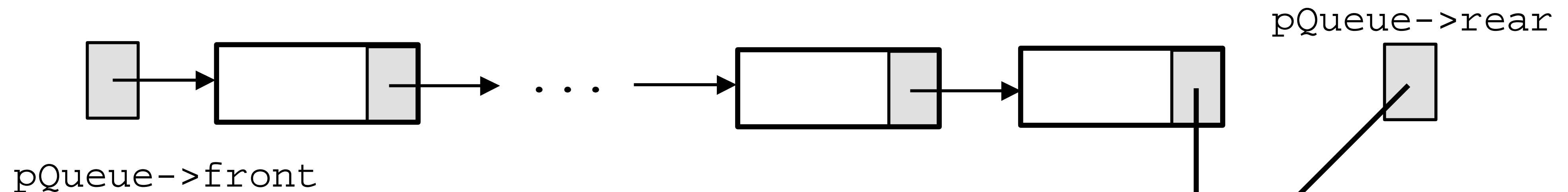
pQueue->front

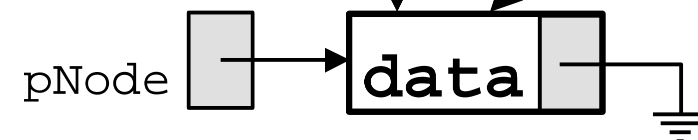
pQueue->rear

Queue: Adding a Node







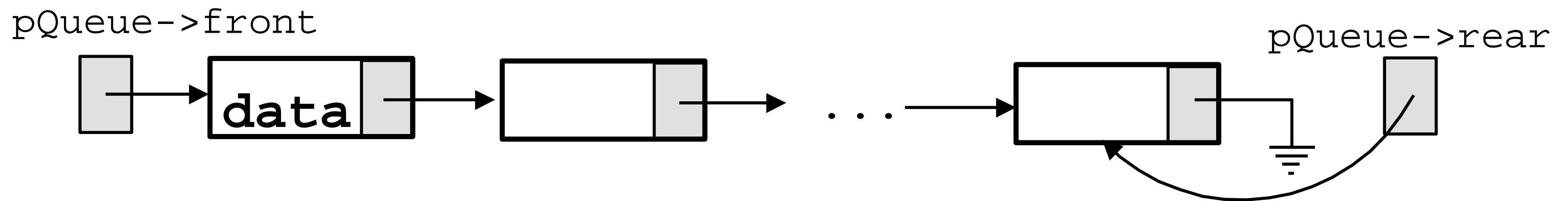


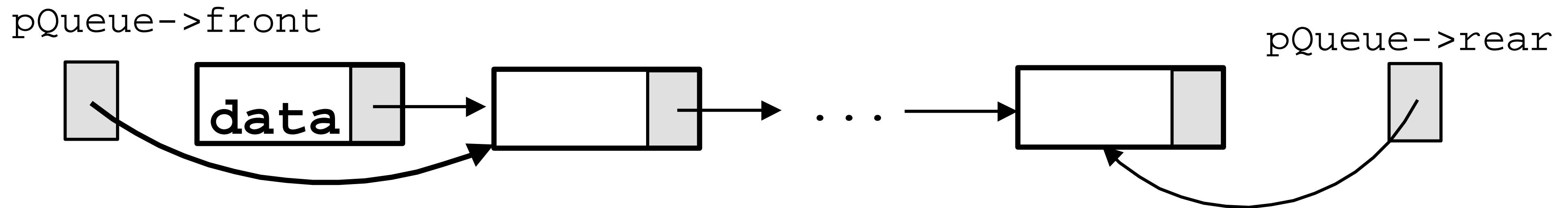

```
pNode->next = NULL;
(pQueue->rear)->next = pNode;
pQueue->rear = pNode;
```

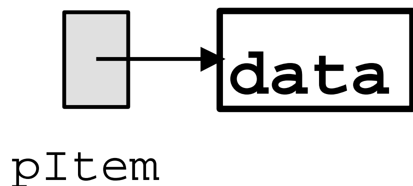
Queue: AddQueue

```
void AddQueue( ItemType Item, QueueType *pQueue )
{
  NodeType  *pNode;

  pNode = (NodeType *) malloc( sizeof( NodeType ) );
  if ( pNode == NULL ) Error("Exhausted memory");
  else {
    pNode->info = Item;
    pNode->next = NULL;
    if ( pQueue->front ) {
      pQueue->front = pQueue->rear = pNode;
    } else {
      (pQueue->rear)->next = pNode;
      pQueue->rear         = pNode
    }
  }
}
```

Queue: Deleting a Node





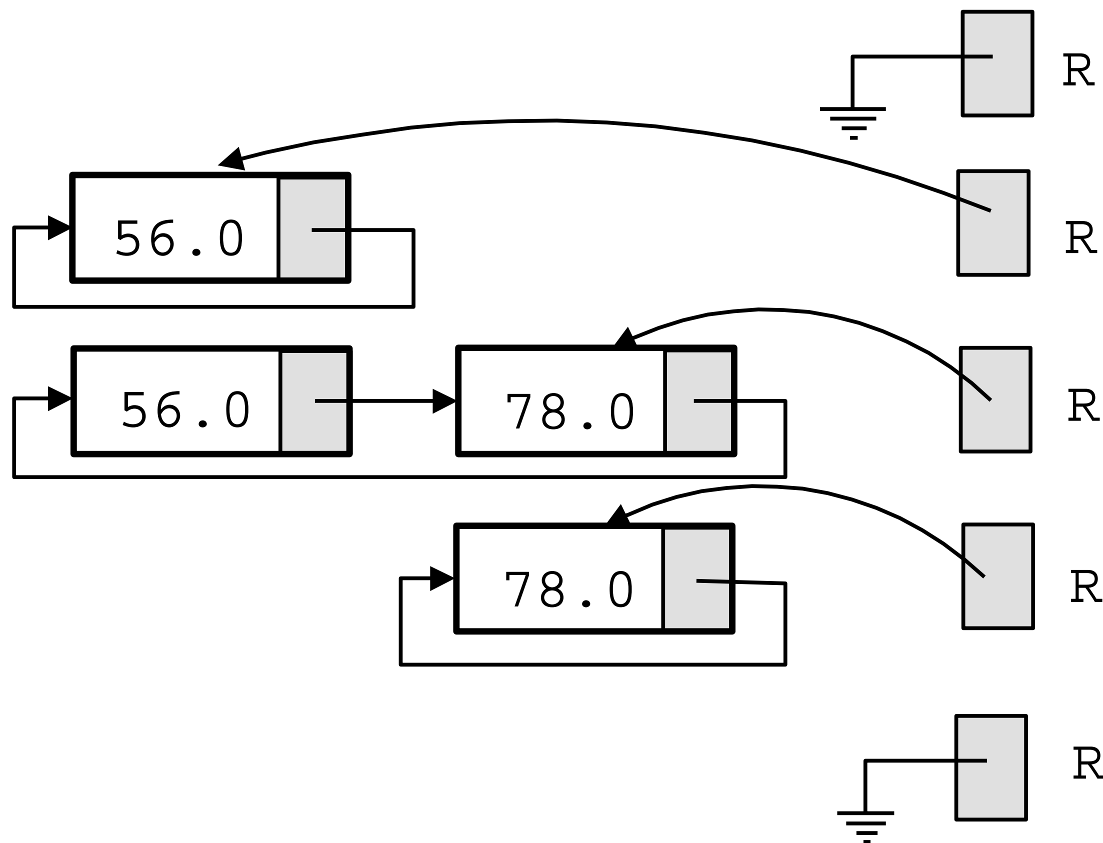



```
*pItem = (pQueue->front)->info;
pQueue->front = (pQueue->front)->next;
```

Queue: DeleteQueue

```
void DeleteQueue( ItemType *pItem, QueueType pQueue )
{
  NodeType      *pNode;

  if ( pQueue->front == NULL )
    Error("Empty Queue");
  else {
    pNode = pQueue->front;
    *pItem = pNode->info;
    pQueue->front = pNode->next;
    free( pNode );
  }
}
```

Pointer Implementation of Queues



InitializeQueue

AddQueue 56.0

AddQueue 78.0

DeleteQueue

DeleteQueue