DATA STRUCTURES

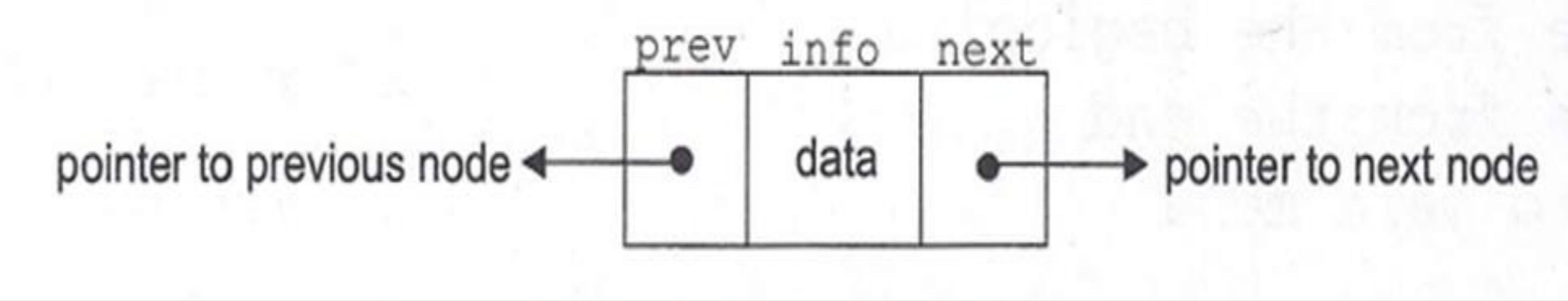
LECTURE-7

LINKED LIST CONT.…

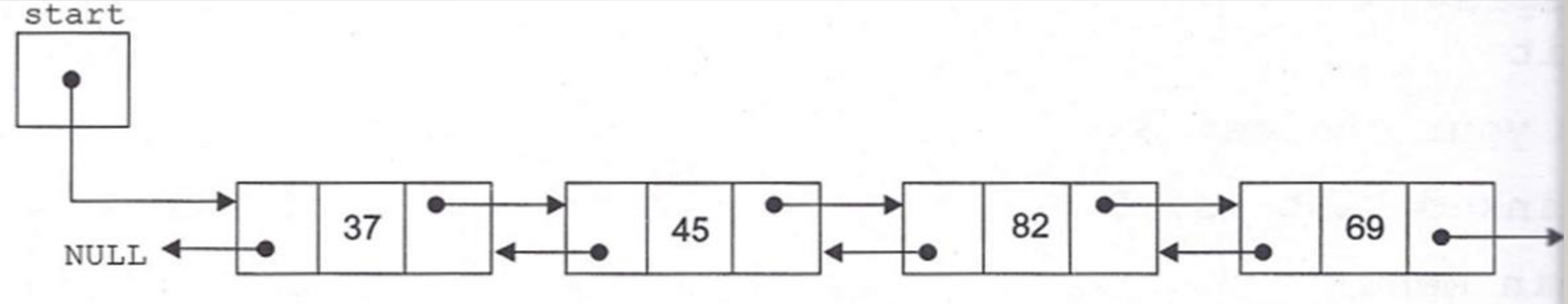
Dr. Sumitra Kisan

DOUBLY LINKED LISTS

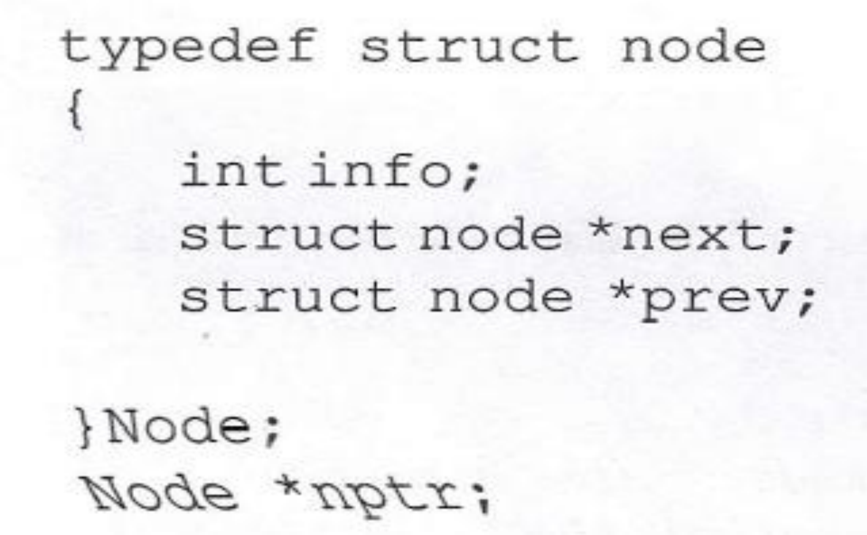
- A singly linked list, each node contains a pointer to the next node and it has no information about its- previous node. Thus, we can traverse only in one direction, that is, from beginning to end.
- However, sometimes it is required to traverse in the backward direction that is, from end to beginning. This can be implemented by maintaining an additional pointer in each node of the list that points to the previous node. Such type of linked list is called **doubly linked list.**
- Each node of a doubly linked list consists of three fields: prev, info and next. The info field contains the data, the prev field contains the address of the previous node and the next field contains the address of the next node.



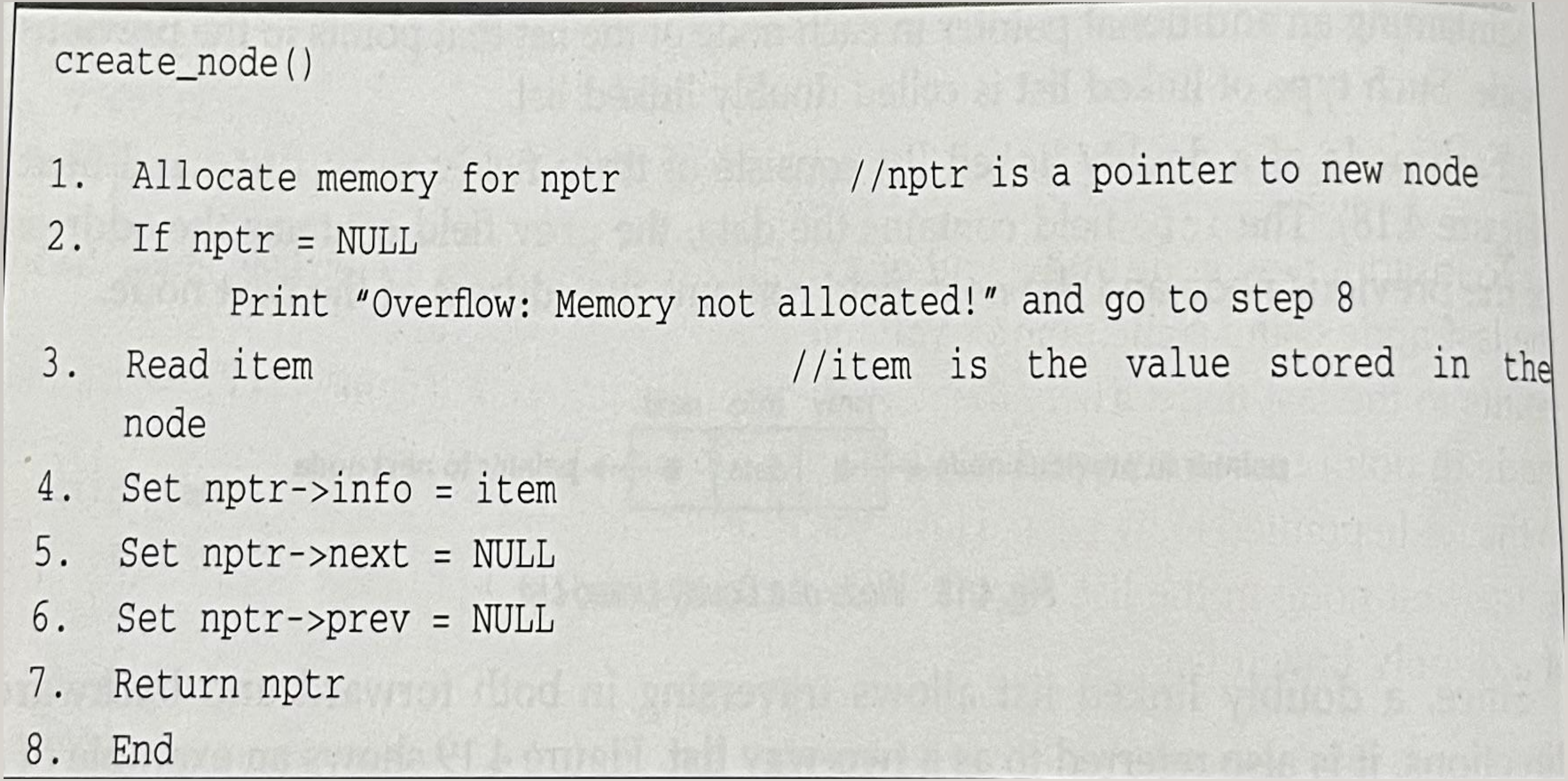
Example of Doubly linked lists



The structure of a node of doubly linked list is shown here:

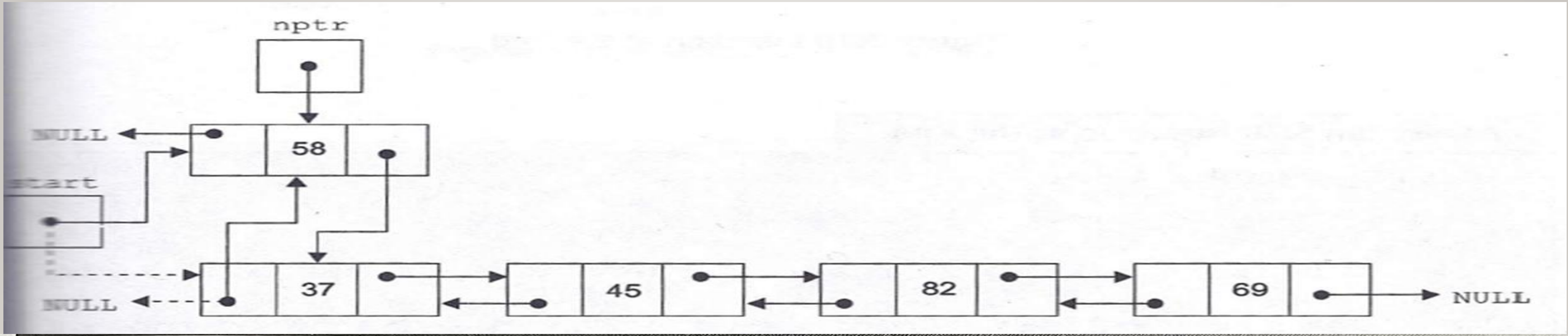


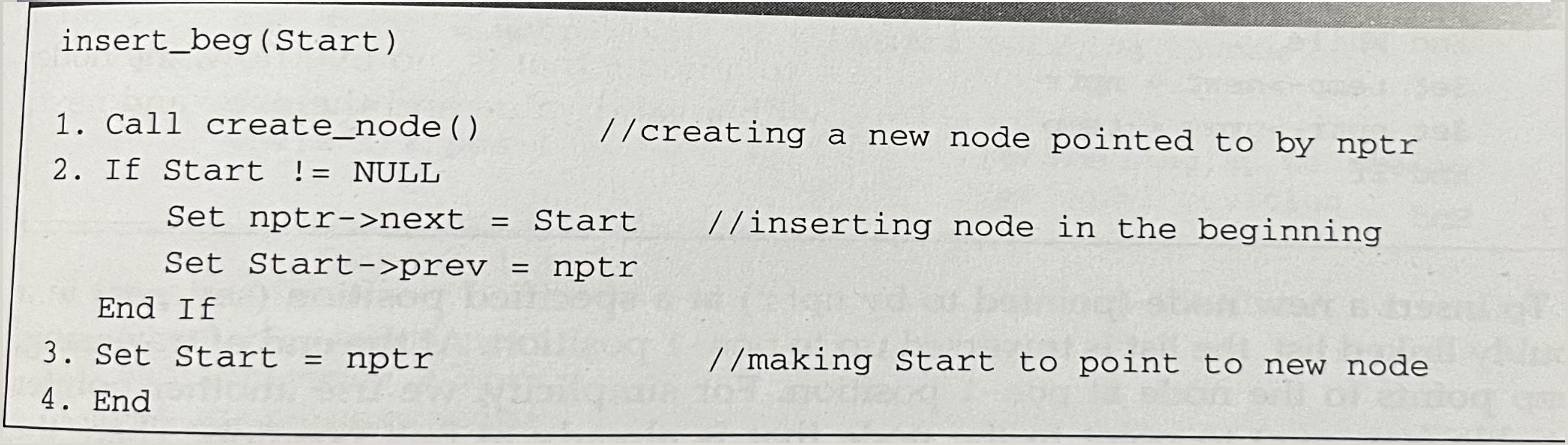
```
typedef struct node
{
    int info;
    struct node *next;
    struct node *prev;

}Node;
 Node *nptr;
```



```
create_node()

1.  Allocate memory for nptr            //nptr is a pointer to new node
2.  If nptr = NULL
        Print "Overflow: Memory not allocated!" and go to step 8
3.  Read item                          //item is the value stored in the
    node
4.  Set nptr->info = item
5.  Set nptr->next = NULL
6.  Set nptr->prev = NULL
7.  Return nptr
8.  End
```

Insertion in Beginning



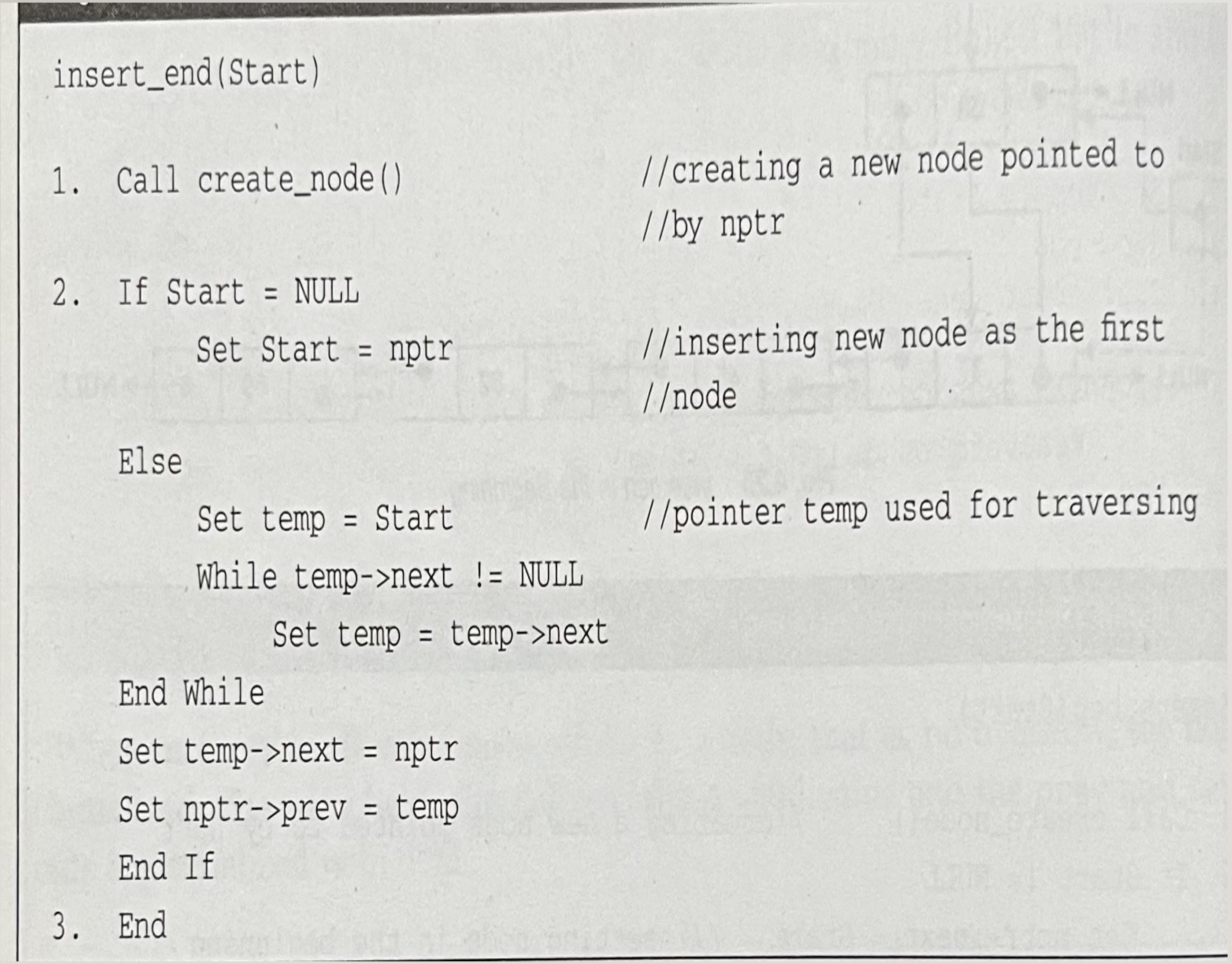


```
insert_beg(Start)

1. Call create_node()        //creating a new node pointed to by nptr
2. If Start != NULL
       Set nptr->next = Start    //inserting node in the beginning
       Set Start->prev = nptr
   End If
3. Set Start = nptr              //making Start to point to new node
4. End
```

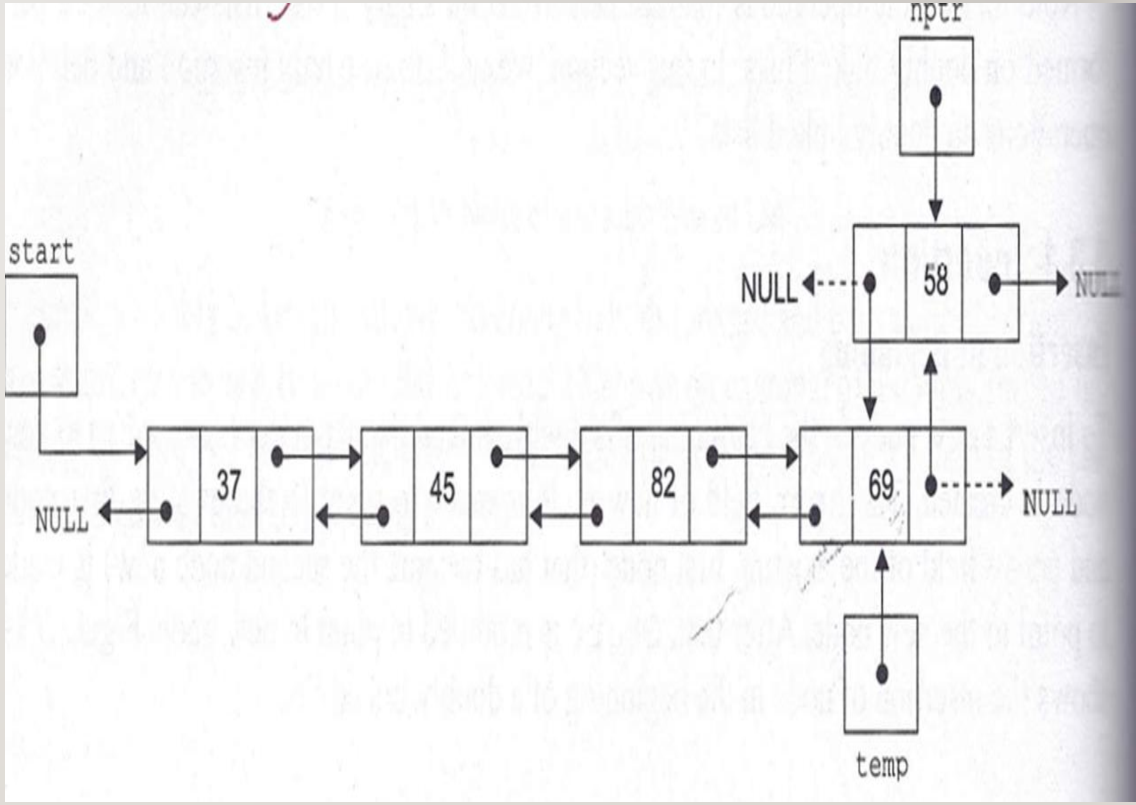
```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next; };
struct Node* createNode(int data) {
    struct Node* newNode =
      (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
struct Node* insertBegin(struct Node* head, int data) {
struct Node* temp = createNode(data);
    temp->next = head;
    if (head != NULL) {
        head->prev = temp;
    }
    return temp;}
```

```
void printList(struct Node* head) {
    struct Node* curr = head;
    while (curr != NULL) {
        printf("%d\n", curr->data);
        curr = curr->next;
    }
}
int main() {
  struct Node* head = createNode(10);
    struct Node* temp1 = createNode(20);
    struct Node* temp2 = createNode(30);
    head->next = temp1;
    temp1->prev = head;
    temp1->next = temp2;
    temp2->prev = temp1;
head = insertBegin(head, 5);
printList(head);
 return 0;
}
```

Insertion at End



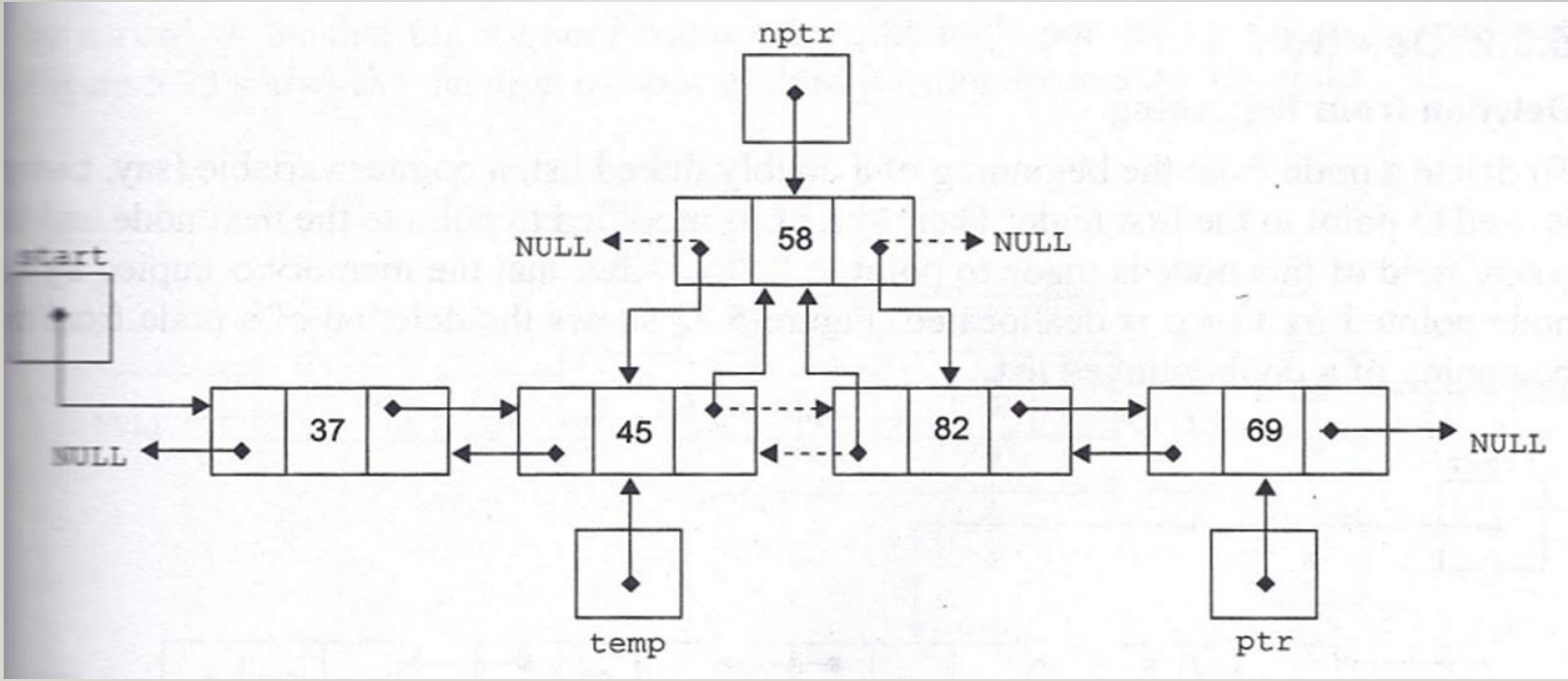
```
insert_end(Start)

1.  Call create_node()              //creating a new node pointed to
                                    //by nptr
2.  If Start = NULL
        Set Start = nptr            //inserting new node as the first
                                    //node
    Else
        Set temp = Start            //pointer temp used for traversing
        While temp->next != NULL
            Set temp = temp->next
    End While
    Set temp->next = nptr
    Set nptr->prev = temp
    End If
3.  End
```

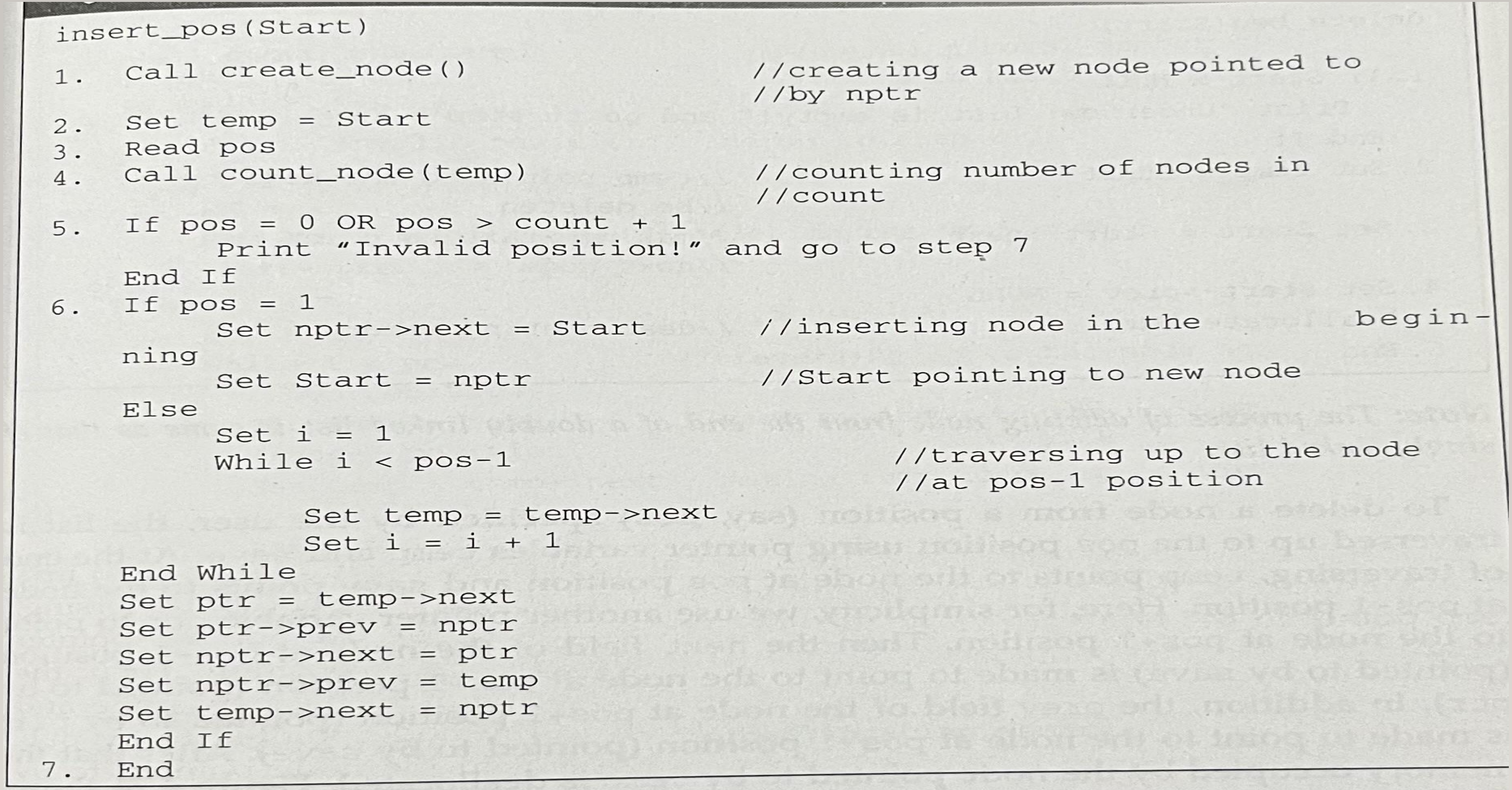


```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next; };
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;  }
struct Node* insertEnd(struct Node* head, int data) {
    struct Node* temp = createNode(data);
if (head == NULL) return temp;
  struct Node* curr = head;
    while (curr->next != NULL)
        curr = curr->next;
    curr->next = temp;
    temp->prev = curr;
    return head;}
```

```
void printList(struct Node* head) {
    struct Node* curr = head;
    while (curr != NULL) {
        printf("%d ", curr->data);
        curr = curr->next;
    }
    printf("\n");
}
int main() {
  struct Node* head = createNode(10);
    struct Node* temp1 = createNode(20);
    struct Node* temp2 = createNode(30);
    head->next = temp1;
    temp1->prev = head;
    temp1->next = temp2;
    temp2->prev = temp1;
head = insertEnd(head, 40);
printList(head);
     return 0;
}
```

Insertion at specified position



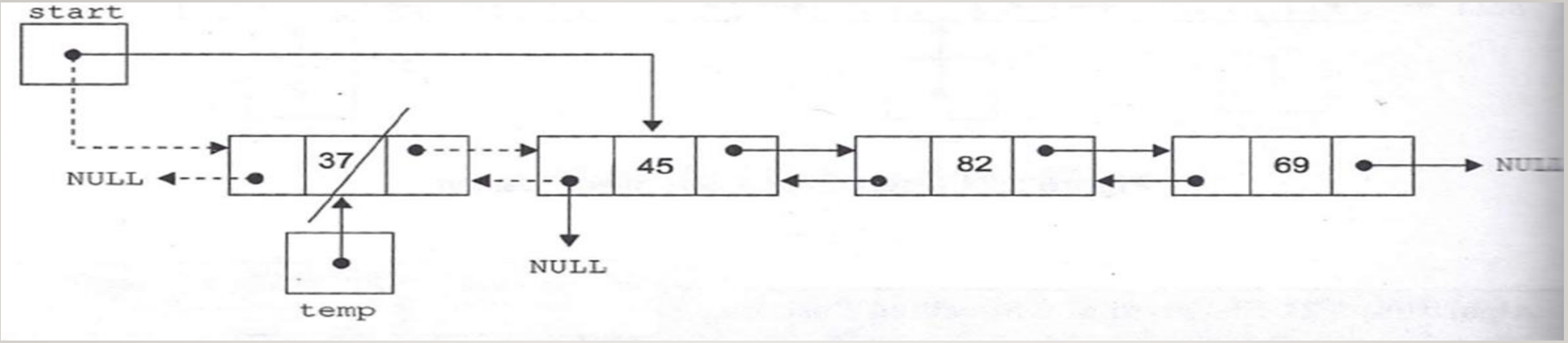


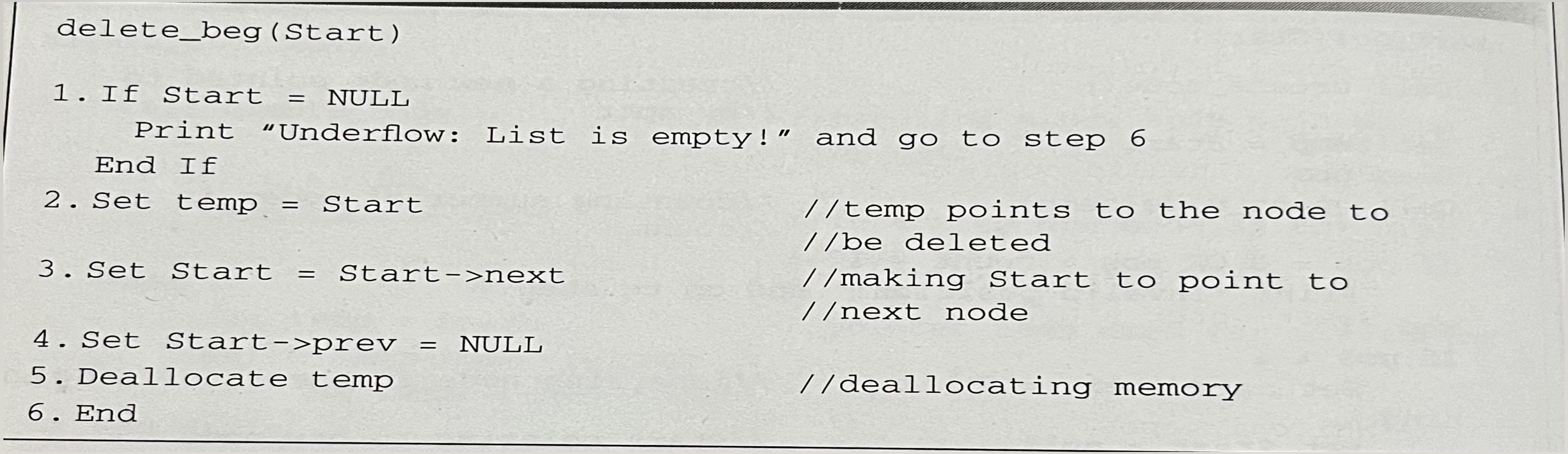
```
insert_pos(Start)
1.  Call create_node()                  //creating a new node pointed to
                                        //by nptr
2.  Set temp = Start
3.  Read pos
4.  Call count_node(temp)               //counting number of nodes in
                                        //count
5.  If pos = 0 OR pos > count + 1
        Print "Invalid position!" and go to step 7
    End If
6.  If pos = 1
        Set nptr->next = Start          //inserting node in the beginning
        Set Start = nptr                //Start pointing to new node
    Else
        Set i = 1
        While i < pos-1                            //traversing up to the node
                                                   //at pos-1 position
            Set temp = temp->next
            Set i = i + 1
    End While
    Set ptr = temp->next
    Set ptr->prev = nptr
    Set nptr->next = ptr
    Set nptr->prev = temp
    Set temp->next = nptr
    End If
7.  End
```

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* prev;
    struct Node* next;};
struct Node* createNode(int data) {
    struct Node* newNode =
       (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;}
struct Node* insertPos(struct Node* head, int pos, int data) {
    if (head == NULL) {
        return (pos == 0) ? createNode(data) : head;}
if (pos == 0) {
        struct Node* temp = createNode(data);
        head->prev = temp;
        temp->next = head;
        return temp; // New node becomes the new head }
struct Node* prev = head;
    for (int i = 0; i < pos - 1; i++) {
        if (prev == NULL) {
            return head;}   prev = prev->next;    }
```

```
struct Node* temp = createNode(data);
  temp->next = prev->next;
    temp->prev = prev;
    if (prev->next != NULL) {
        prev->next->prev = temp; }
    prev->next = temp;
return head;}
void printList(struct Node* head) {
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    } printf("\n");}
    int main() {
    struct Node* head = NULL;
    head = insertPos(head, 0, 10);
    head = insertPos(head, 1, 20);
    head = insertPos(head, 2, 30);
    printList(head);
    head = insertPos(head, 1, 40);
    head = insertPos(head, 2, 50);
    head = insertPos(head, 0, 5);
    printList(head);
    return 0;
}
```

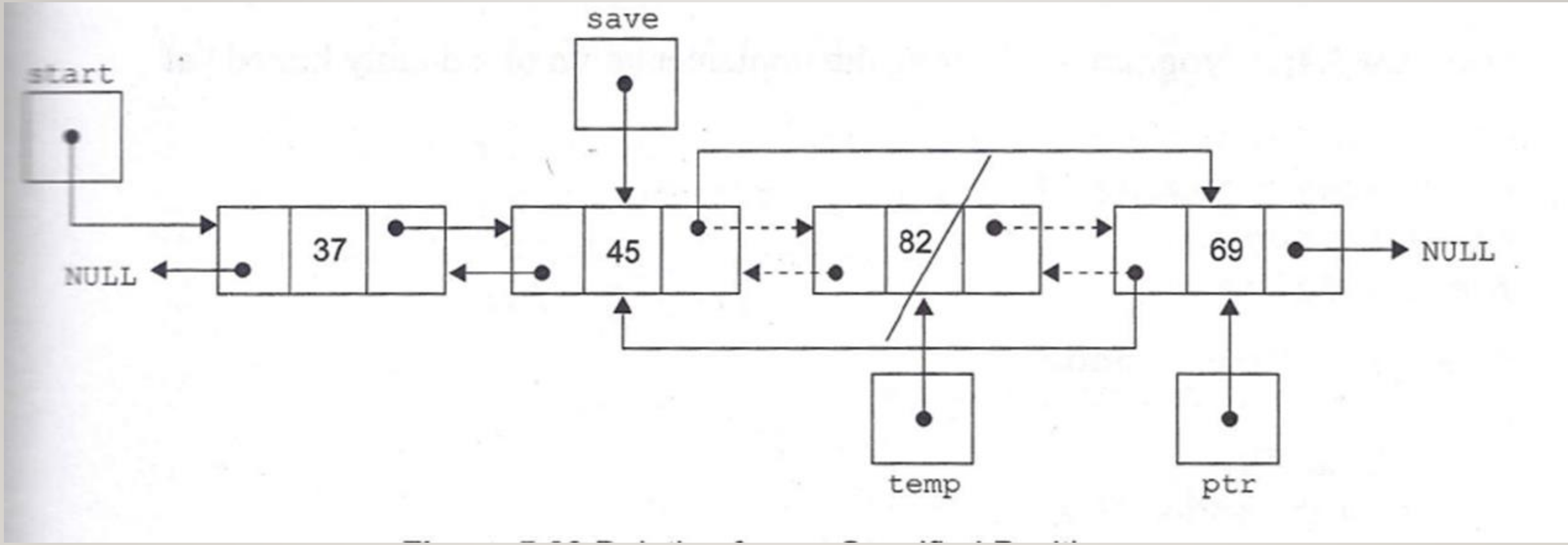
Deletion from Beginning





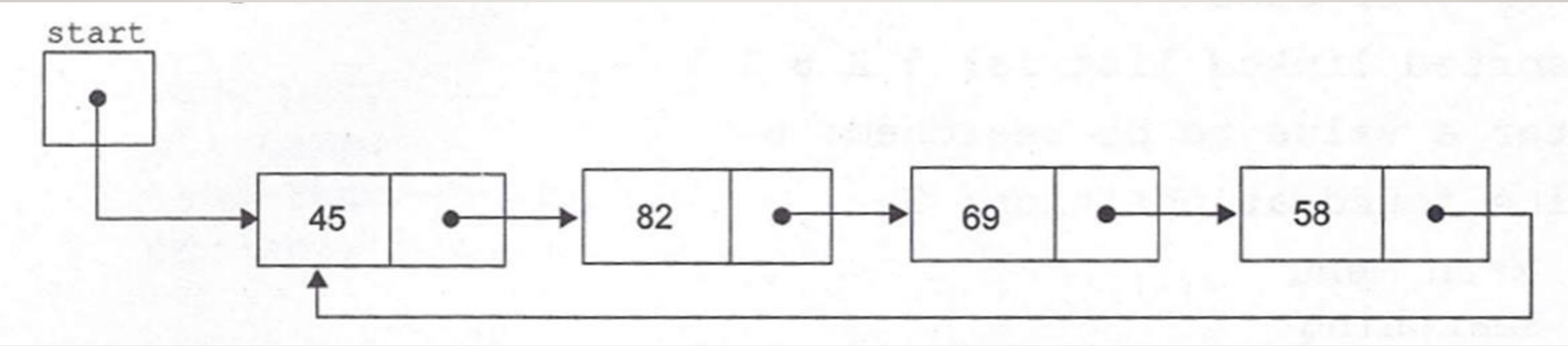
```
delete_beg(Start)

1. If Start = NULL
     Print "Underflow: List is empty!" and go to step 6
   End If
2. Set temp = Start                         //temp points to the node to
                                            //be deleted
3. Set Start = Start->next                  //making Start to point to
                                            //next node
4. Set Start->prev = NULL
5. Deallocate temp                          //deallocating memory
6. End
```

Deletion from a specified position

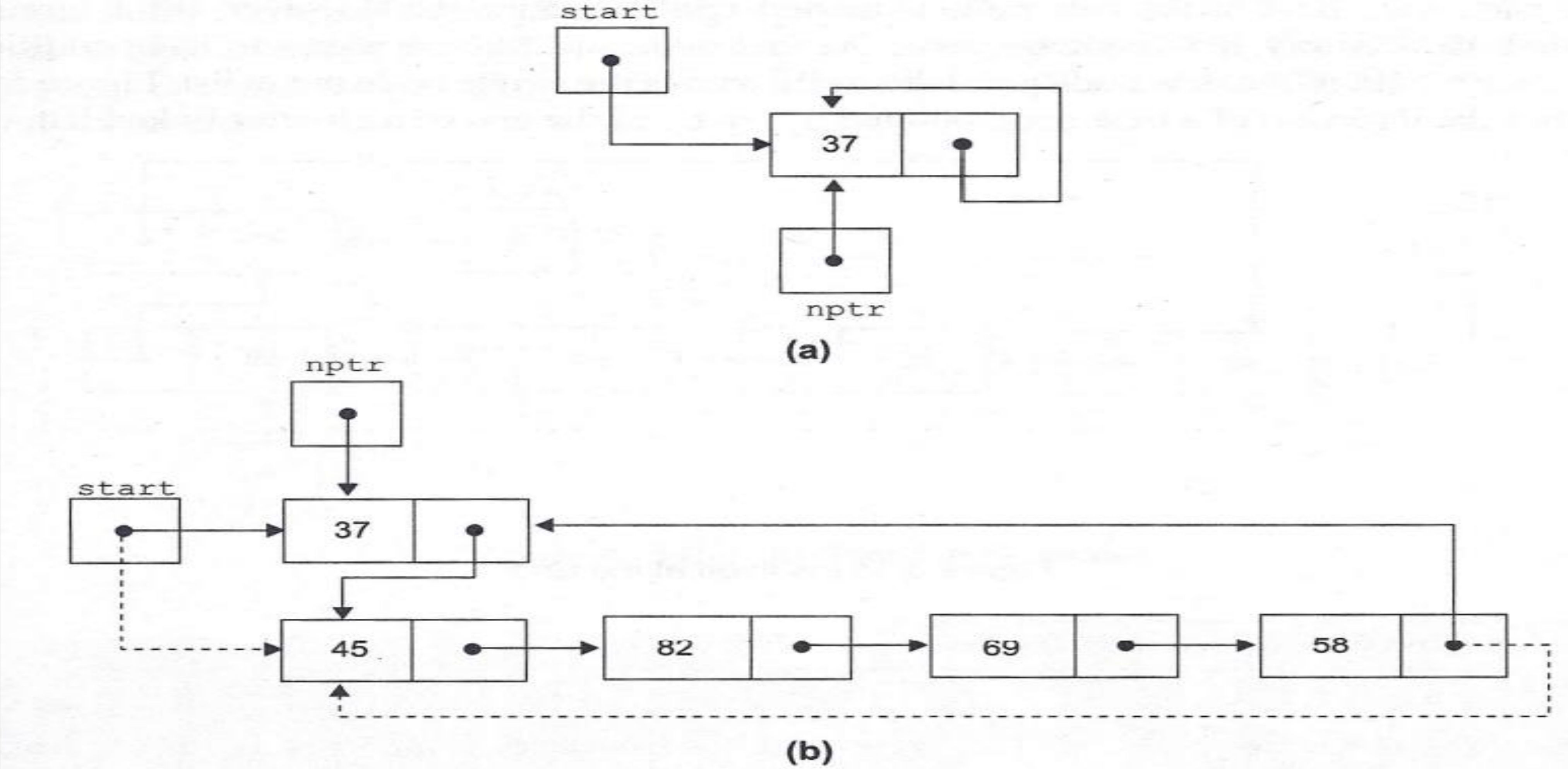


Circular Linked List

A linear linked list, in which the next field of the last node points back to the first node instead of containing NULL, is termed as a circular linked list.

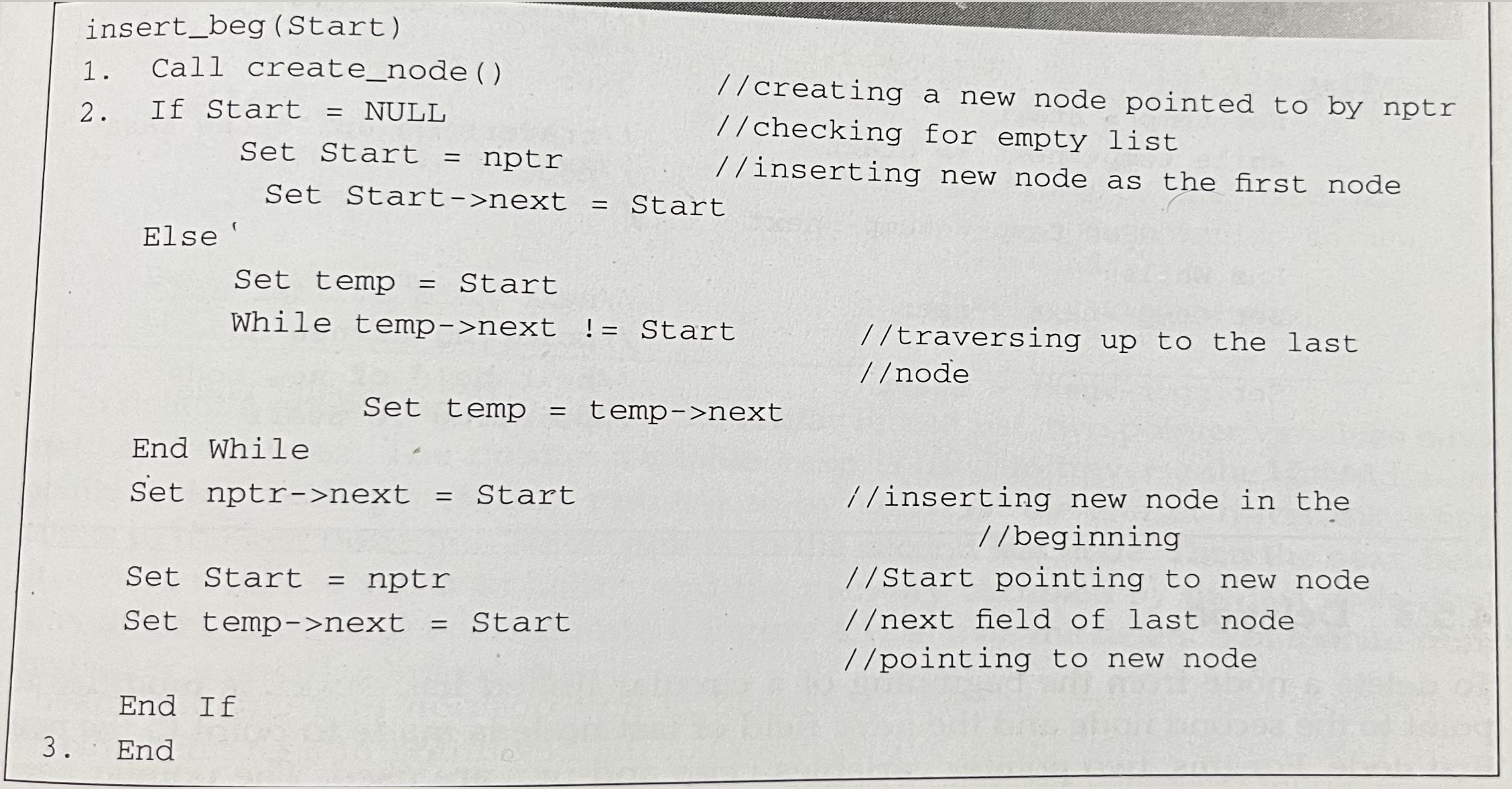


Insertion in Beginning



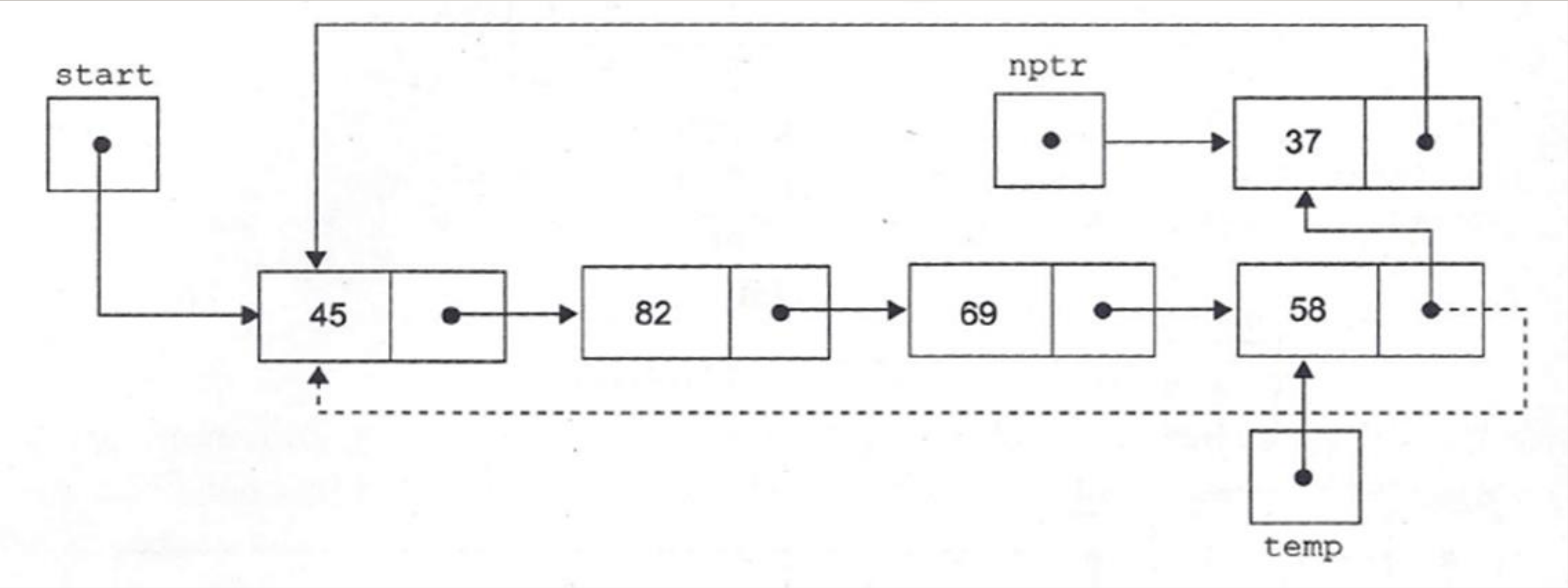
(a)

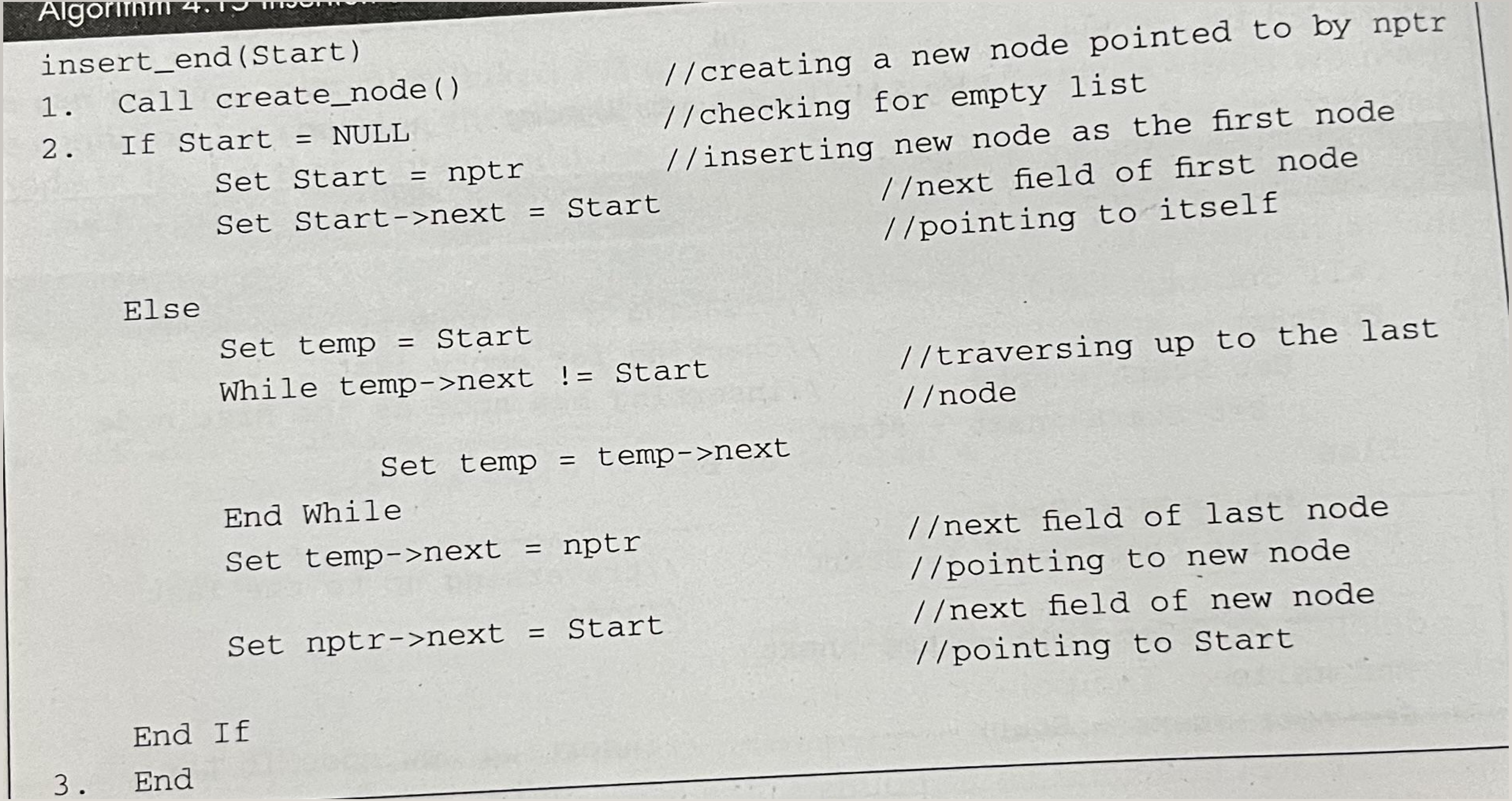
(b)



```
insert_beg(Start)
1.  Call create_node()            //creating a new node pointed to by nptr
2.  If Start = NULL               //checking for empty list
        Set Start = nptr          //inserting new node as the first node
         Set Start->next = Start
    Else
        Set temp = Start
        While temp->next != Start         //traversing up to the last
                                          //node
             Set temp = temp->next
    End While
    Set nptr->next = Start                //inserting new node in the
                                               //beginning
    Set Start = nptr                      //Start pointing to new node
    Set temp->next = Start                //next field of last node
                                          //pointing to new node
    End If
3.  End
```

Insertion at End

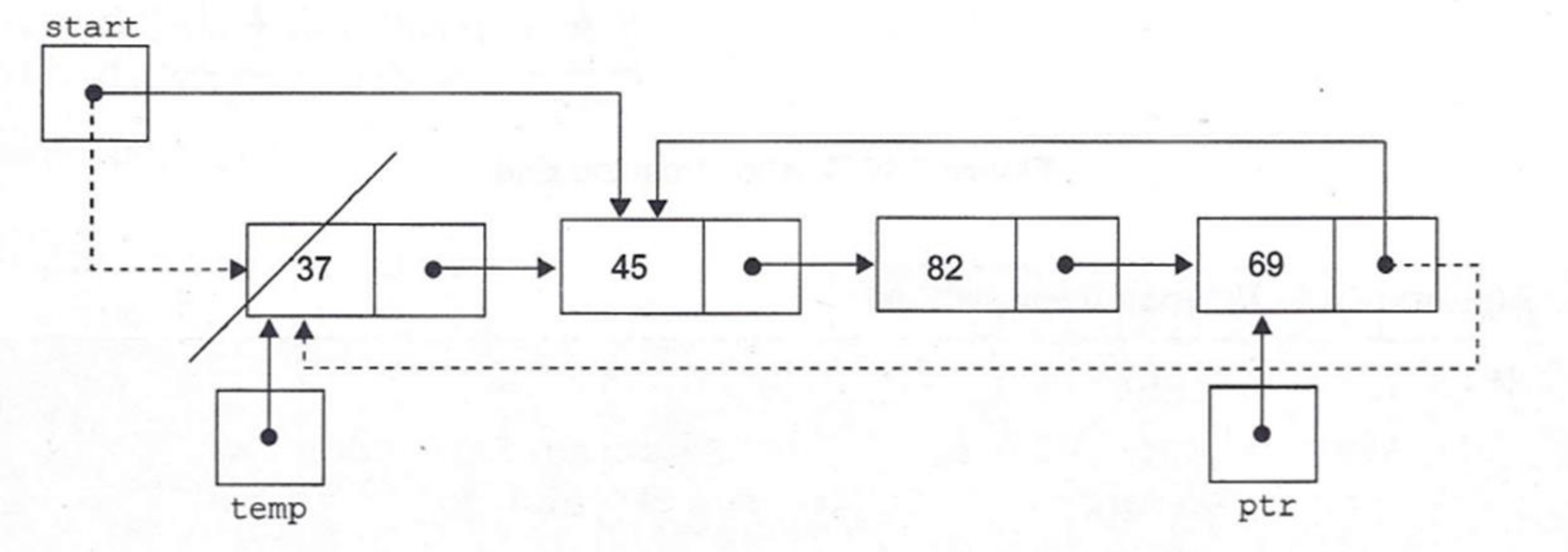


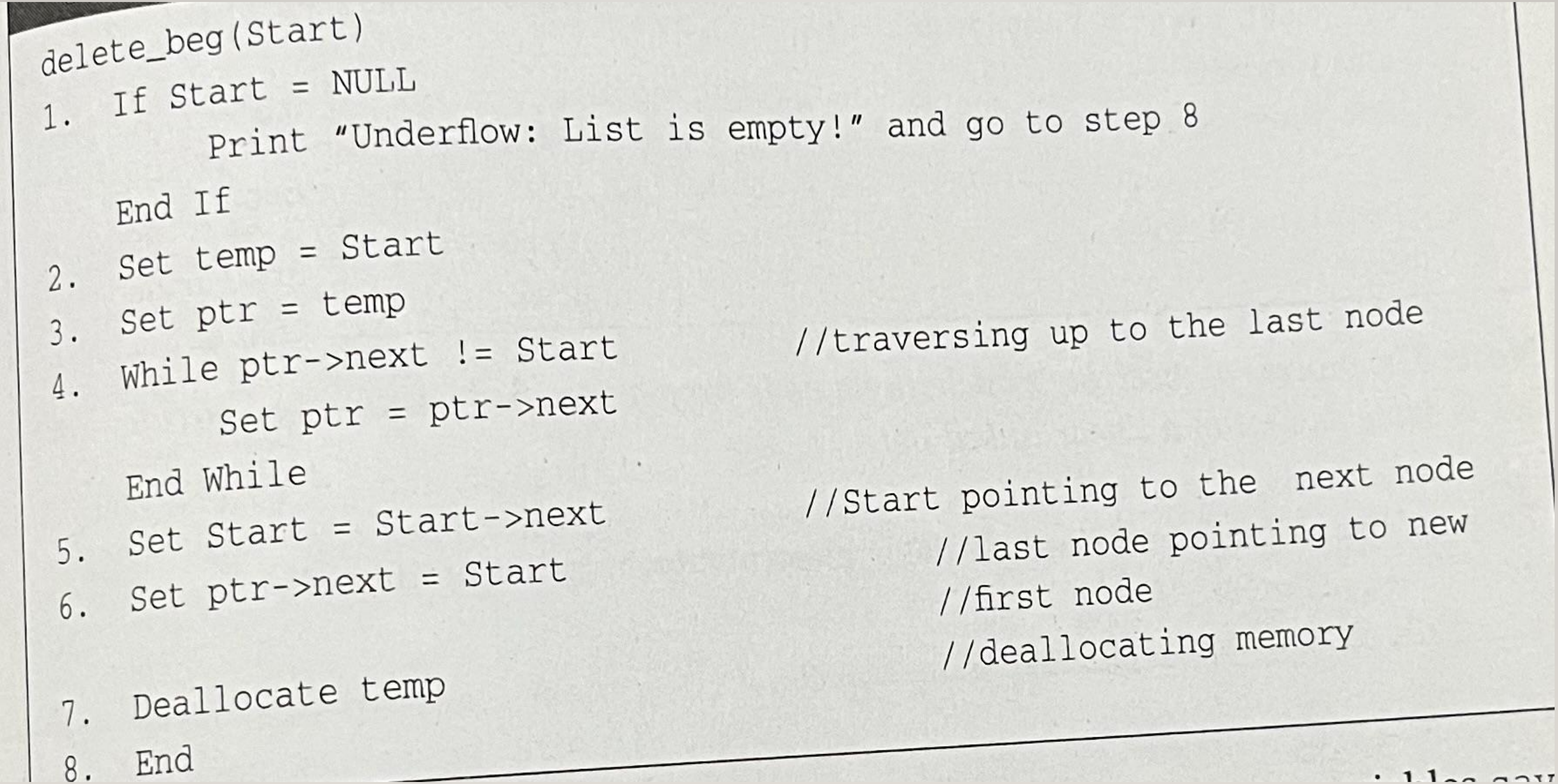


Algorithm 4.15 Insertion

```
insert_end(Start)
1.  Call create_node()              //creating a new node pointed to by nptr
2.  If Start = NULL                 //checking for empty list
         Set Start = nptr           //inserting new node as the first node
         Set Start->next = Start            //next field of first node
                                            //pointing to itself

    Else
         Set temp = Start
         While temp->next != Start          //traversing up to the last
                                            //node
              Set temp = temp->next

         End While                          //next field of last node
         Set temp->next = nptr              //pointing to new node
                                            //next field of new node
         Set nptr->next = Start             //pointing to Start

    End If
3.  End
```

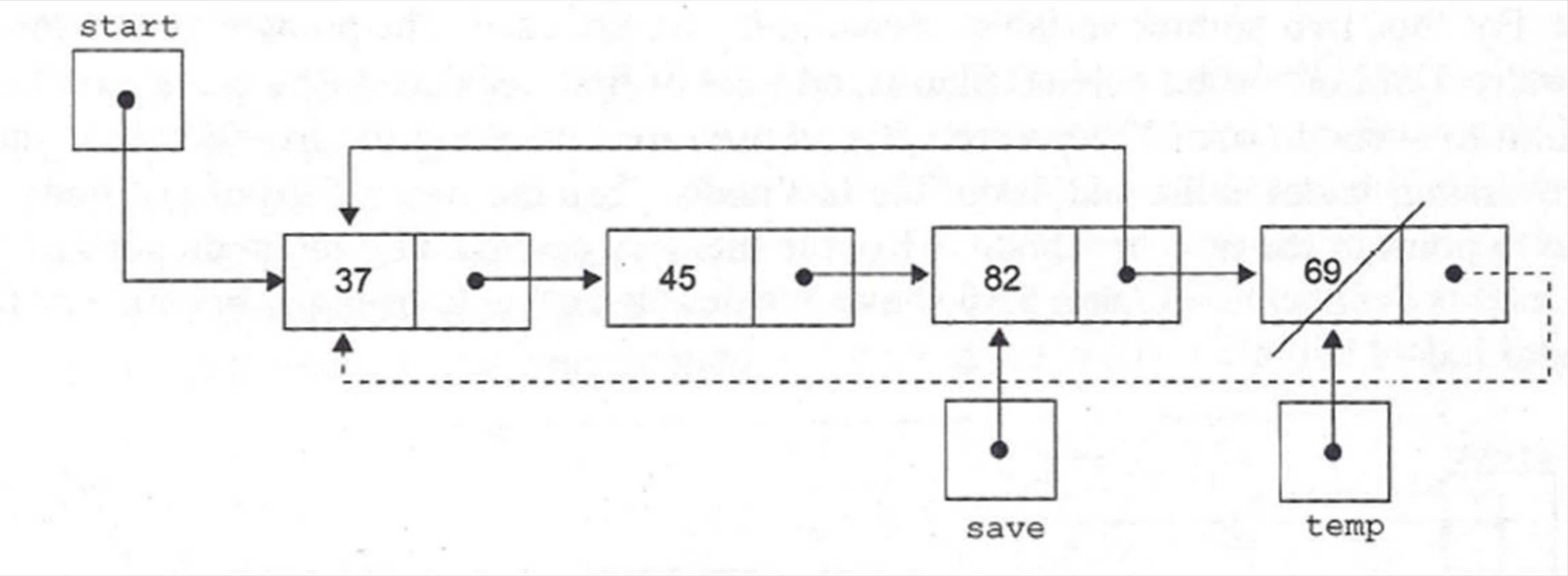
Deletion from Beginning

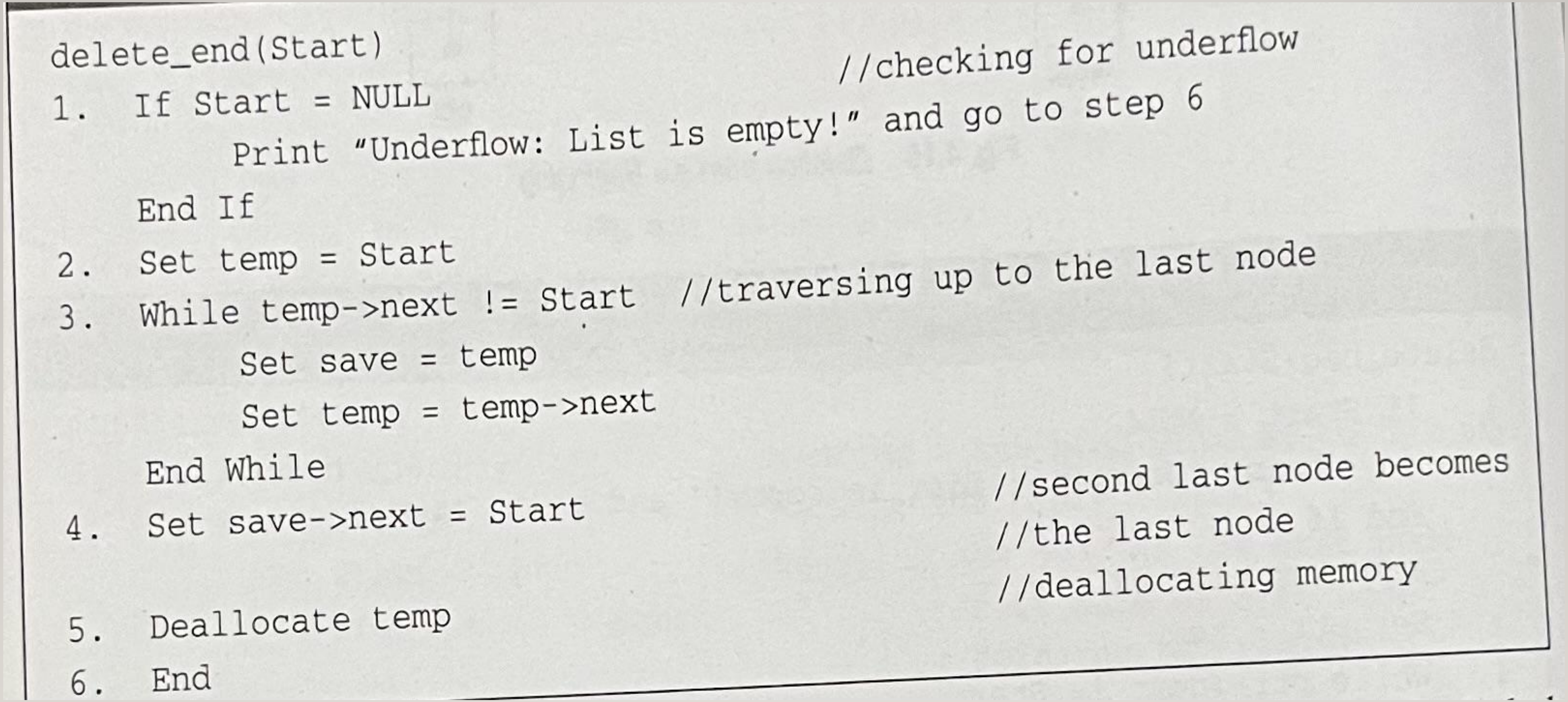




```
delete_beg(Start)
1.  If Start = NULL
        Print "Underflow: List is empty!" and go to step 8
    End If
2.  Set temp = Start
3.  Set ptr = temp
4.  While ptr->next != Start            //traversing up to the last node
        Set ptr = ptr->next
    End While
5.  Set Start = Start->next             //Start pointing to the  next node
6.  Set ptr->next = Start                     //last node pointing to new
                                              //first node
                                              //deallocating memory
7.  Deallocate temp
8.  End
```

Deletion from End





```
delete_end(Start)
1.  If Start = NULL                              //checking for underflow
        Print "Underflow: List is empty!" and go to step 6
    End If
2.  Set temp = Start
3.  While temp->next != Start  //traversing up to the last node
        Set save = temp
        Set temp = temp->next
    End While
4.  Set save->next = Start                       //second last node becomes
                                                 //the last node
                                                 //deallocating memory
5.  Deallocate temp
6.  End
```