```c
#include <stdio.h>
#include <stdlib.h>
#include "../standard_types.h"
typedef struct node
{
    int data;
    struct node* next;
} node;
/*
    create a new node
    initialize the data and next field

    return the newly created node
*/
node* create(u16 data,node* next)
{
    node* new_node = (node*)malloc(sizeof(node));
    if(new_node == NULL)
    {
        printf("Error creating a new node.\n");
        exit(0);
    }
    new_node->data = data;
    new_node->next = next;

    return new_node;
}
/*
    add a new node at the beginning of the list
*/
node* InsertAtStart(node* head,u16 data)
{
    node* new_node = create(data,head);
    head = new_node;
    return head;
}

/*
    add a new node at the end of the list
*/
node* InsertAtEnd(node* head, u16 data)
{
    if(head == NULL)
        return NULL;
    /* go to the last node */
    node *cursor = head;
    while(cursor->next != NULL)
        cursor = cursor->next;

    /* create a new node */
    node* new_node =  create(data,NULL);
    cursor->next = new_node;

    return head;
}

/*
    insert a new node after the prev node
*/
node* insert_after(node *head, u16 data, node* prev)
```

```
{
    if(head == NULL || prev == NULL)
        return NULL;
    /* find the prev node, starting from the first node*/
    node *cursor = head;
    while(cursor != prev)
        cursor = cursor->next;

    if(cursor != NULL)
    {
        node* new_node = create(data,cursor->next);
        cursor->next = new_node;
        return head;
    }
    else
    {
        return NULL;
    }
}
node* creat_list(node *head)
{   u16 n,i=0,x;
    node *tmp;
    printf("enter the number of the element: ");
    scanf("%d",&n);
    printf("enter the first element");
    scanf("%d",&x);
    head=InsertAtStart(head,x);
    for(i=1; i<n; i++)
    {   printf(" Input data for node %d : ", i+1);
        scanf("%d",&x);
        InsertAtEnd(head,x);

    }
    return head;
}
/*
    insert a new node before the nxt node
*/
node* insert_before(node *head, u16 data, node* nxt)
{
    if(nxt == NULL || head == NULL)
        return NULL;

    if(head == nxt)
    {
        head = InsertAtStart(head,data);
        return head;
    }

    /* find the prev node, starting from the first node*/
    node *cursor = head;
    while(cursor != NULL)
    {
        if(cursor->next == nxt)
            break;
        cursor = cursor->next;
    }

    if(cursor != NULL)
    {
```

```
121         node* new_node = create(data,cursor->next);
122         cursor->next = new_node;
123         return head;
124     }
125     else
126     {
127         return NULL;
128     }
129 }
130 
131 /*
132     remove node from the front of list
133 */
134 node* remove_front(node* head)
135 {
136     if(head == NULL)
137         return NULL;
138     node *front = head;
139     head = head->next;
140     front->next = NULL;
141     /* is this the last node in the list */
142     if(front == head)
143         head = NULL;
144     free(front);
145     return head;
146 }
147 
148 /*
149     remove node from the back of the list
150 */
151 node* remove_back(node* head)
152 {
153     if(head == NULL)
154         return NULL;
155 
156     node *cursor = head;
157     node *back = NULL;
158     while(cursor->next != NULL)
159     {
160         back = cursor;
161         cursor = cursor->next;
162     }
163 
164     if(back != NULL)
165         back->next = NULL;
166 
167     /* if this is the last node in the list*/
168     if(cursor == head)
169         head = NULL;
170 
171     free(cursor);
172 
173     return head;
174 }
175 
176 /*
177     remove a node from the list
178 */
179 node* remove_any(node* head,node* nd)
180 {
```

```
181     if(nd == NULL)
182         return NULL;
183     /* if the node is the first node */
184     if(nd == head)
185         return remove_front(head);
186 
187     /* if the node is the last node */
188     if(nd->next == NULL)
189         return remove_back(head);
190 
191     /* if the node is in the middle */
192     node* cursor = head;
193     while(cursor != NULL)
194     {
195         if(cursor->next == nd)
196             break;
197         cursor = cursor->next;
198     }
199 
200     if(cursor != NULL)
201     {
202         node* tmp = cursor->next;
203         cursor->next = tmp->next;
204         tmp->next = NULL;
205         free(tmp);
206     }
207     return head;
208 
209 }
210 /*
211     display a node
212 */
213 void printList(node* head)
214 {
215     node* temp = head;
216     while (temp != NULL) {
217         printf("%d \n ", temp->data);
218         temp = temp->next;
219     }
220 }
221 node* search(node* head,int data)
222 {
223 
224     node *cursor = head;
225     while(cursor!=NULL)
226     {
227         if(cursor->data == data)
228             return cursor;
229         cursor = cursor->next;
230     }
231     return NULL;
232 }
233 
234 /*
235     remove all element of the list
236 */
237 void dispose(node *head)
238 {
239     node *cursor, *tmp;
240 
```

```
    if(head != NULL)
    {
        cursor = head->next;
        head->next = NULL;
        while(cursor != NULL)
        {
            tmp = cursor->next;
            free(cursor);
            cursor = tmp;
        }
    }
}
/*
    return the number of elements in the list
*/
int count(node *head)
{
    node *cursor = head;
    int c = 0;
    while(cursor != NULL)
    {
        c++;
        cursor = cursor->next;
    }
    return c;
}

/*
    reverse the linked list
*/

void reverse(node** head)
{
    node* prev = NULL;
    node * current = *head;
    node* next = NULL;
    while (current != NULL) {
        // Store next
        next = current->next;

        // Reverse current node's pointer
        current->next = prev;

        // Move pointers one position ahead.
        prev = current;
        current = next;
    }
    *head = prev;
}
 void printNthFromLast(node* head, int n)
{
    int len = 0, i;
    node* temp = head;

    // count the number of nodes in Linked List
    len=count(head);

    // check if value of n is not
    // more than length of the linked list
    if (len < n)
```

```
301         return;
302 
303     temp = head;
304 
305     // get the (len-n+1)th node from the beginning
306     for (i = 1; i < len - n + 1; i++)
307         temp = temp->next;
308 
309     printf("the required elemint: %d",temp->data);
310 
311     return;
312 }
313 
314 int main()
315 {
316     node* head=NULL;
317     node* tmp = head;
318     head=creat_list(head);
319     //reverse(&head);
320     head=InsertAtStart(head,1);
321     //InsertAtEnd(head,5);
322    // push(head,5);
323 
324     printList(head);
325 return 0;
326     }
```