Single linked:

```
void addatdeglink(int num) {
    NODE* temp = new NODE;
    temp->data = num;
    temp->link = HEAD;
    HEAD = temp;
}

void appendlink(int num) {
    NODE* elem = new NODE;
    elem->data = num;
    if (HEAD == NULL) {
        HEAD = elem;
        elem->link = NULL;
        return;
    }
    NODE* temp = HEAD;
    while (temp->link != NULL) {
        temp = temp->link;
    }
    temp->link = elem;
    elem->link = NULL;
}

void addafterlink(int loc,int num) {//loc=2
    NODE* elem = new NODE;
    elem->data = num;
    NODE* temp = HEAD;
    int i;
    for (i = 1; i < loc; i++) {
        temp = temp->link;
        if (temp == NULL)return;
    }
    elem->link = temp->link;
    temp->link = elem;
}
```

```
void del(int num)
{
    node* temp = head;
    node* prev = NULL;

    while (temp != NULL && temp->data != num)
    {
        prev = temp;
        temp = temp->link;
    }

    if (temp == NULL)
        return;

    if (prev == NULL)
        head = temp->link;
    else
        prev->link = temp->link;

    delete temp;
}
```

```
void displaylink() {
    NODE* temp = HEAD;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->link;
    }
    cout << endl;
}

void reverse() {
    NODE* current = HEAD;
    NODE* prev = NULL;
    NODE* next;
    while (current != NULL) {
        next = current->link;
        current->link = prev;
        prev = current;
        current = next;
    }
    HEAD = prev;
}
```

```
void concat(linklist &L)
{
    NODE* temp;

    if (HEAD == NULL)
    {
        HEAD = L.HEAD;
    }
    else
    {
        if (L.HEAD != NULL)
        {
            temp = HEAD;

            while (temp->link != NULL)
            {
                temp = temp->link;
            }

            temp->link = L.HEAD;
        }
    }

    L.HEAD = NULL;
}
```

Doubly linked:

```
void appendDouble(int num) {
    doubleNODE* elem = new doubleNODE;
    elem->data = num;
    if (doubleHEAD == NULL) {
        elem->next = NULL;
        elem->prev = NULL;
        doubleHEAD = elem;
        return;
    }
    doubleNODE* temp = doubleHEAD;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    elem->next = NULL;
    elem->prev = temp;
    temp->next = elem;
}

void addbegDouble(int num) {
    doubleNODE* elem = new doubleNODE;
    elem->data = num;
    elem->prev = NULL;
    elem->next = doubleHEAD;
    doubleHEAD->prev = elem;
    doubleHEAD = elem;
}

void addafterDouble(int loc,int num) {
    doubleNODE* elem = new doubleNODE;
    elem->data = num;
    int i;
    doubleNODE* temp = doubleHEAD;

    for (i = 1; i < loc; i++) {
        temp = temp->next;
        if (temp == NULL)return;
    }
    elem->next = temp->next;
     temp->next->prev=elem;
     temp->next = elem;
     elem->prev = temp;
}
```

```
void delDouble(int num)
{
    dnode* temp = head;

    while (temp != NULL && temp->data != num)
        temp = temp->next;

    if (temp == NULL)
        return;

    if (temp->prev != NULL)
        temp->prev->next = temp->next;
    else
        head = temp->next;

    if (temp->next != NULL)
        temp->next->prev = temp->prev;

    delete temp;
}
```

```
void displayDouble() {
    doubleNODE* temp = doubleHEAD;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
```

Recursive code:

```
int getlength(node* head)
{
    if (head == NULL)
        return 0;
    return 1 + getlength(head->link);
}

int compare(node* head1, node* head2)
{
    if (head1 == NULL && head2 == NULL)
        return 1;
    if (head1 == NULL || head2 == NULL)
        return 0;
    if (head1->data != head2->data)
        return 0;

    return compare(head1->link, head2->link);
}

void duplicate(node* head, node** newHead)
{
    if (head != NULL)
    {
        *newHead = new node;
        (*newHead)->data = head->data;
        (*newHead)->link = NULL;

        duplicate(head->link, &((*newHead)->link));
    }
}
```