Assignment 4



node \*insert\_any(node \*head, int position, int value)

{

    int i;

    node \*new = create(value);

    node \*temp = head;

    if(position == 1)

    {

        new -> next = temp;

        head = new;

        return head;

    }

    for(i = 0;i < position - 2; i++)

    {

        temp = temp -> next;

        if(temp == NULL)

        {

            printf("\nInvalid position!!!");

            return head;

        }

    }

    new -> next = temp -> next;

    temp -> next = new;

    printf("\n\nInsertion successful!!!");

    return head;

}



node \*delete\_beg(node \*head)

{

    if(head == NULL)

    {

        printf("\nList is empty");

    }

    node \*tmp = head;

    head = tmp -> next;

    free(tmp);

    return head;

}



node \*delete\_end(node \*head)

{

    if(head == NULL)

    {

        printf("\nList is empty");

    }

    else if(head -> next == NULL)

    {

        head == NULL;

        free(head);

        printf("\nDeletion success(Empty Node)");

    }

    else

    {

        node \*tmp = head, \*tmp2;

        while(tmp -> next)

        {

            tmp2 = tmp;

            tmp = tmp -> next;

        }

        tmp2 -> next = NULL;

        free(tmp);

        printf("\nLast Node deleted");

    }

    return head;

}



This is done to avoid overflow that may happen in adding two very big integers where the addition result may become greater than the max integer limit and yield weird results.



int ternarySearch(int l, int r, int key, int ar[])

{

    if (r >= l) {

        int mid1 = l + (r - l) / 3;

        int mid2 = r - (r - l) / 3;

        if (ar[mid1] == key) {

            return mid1;

        }

        if (ar[mid2] == key) {

            return mid2;

        }

        if (key < ar[mid1]) {

            return ternarySearch(l, mid1 - 1, key, ar);

        }

        else if (key > ar[mid2]) {

            return ternarySearch(mid2 + 1, r, key, ar);

        }

        else {

            return ternarySearch(mid1 + 1, mid2 - 1, key, ar);

        }

    }

    return -1;

}