Operations on singly linked list

Sort, Reverse, Concat.

```
#include <stdio.h>
#include <conio.h>
#include <process.h>

struct node
{
    int info;
    struct node *link;
};
typedef struct node *NODE;
NODE getnode()
{
    NODE x;
    x=(NODE) malloc (sizeof (struct node));
    if (x==NULL)
    {
        printf ("Memory full\n");
        exit(0);
    }
    return x;
}
NODE insertfront (NODE first, int item)
{
    NODE temp;
    temp= getnode();
    temp->info = item;
    temp->link = NULL;
    if (first==NULL)
        return temp;
```

```
    temp->link = first;
    first = temp;
    return first;
}
NODE deletefront(NODE first)
{
    NODE temp;
    if (first == NULL)
    {
        printf("Empty list. Can't delete\n");
        return first;
    }
    temp = first;
    temp = temp->link;
    printf("Item deleted @ front: %d\n", first->info);
    free(first);
    return temp;
}

NODE IF(NODE second, int item)
{
    NODE temp;
    temp = getnode();
    temp->info = item;
    temp->link = NULL;
    if(second == NULL)
        return temp;
    temp->link = second;
    second = temp;
    return second;
}
NODE IR(NODE second, int item)
{
```

```
NODE temp, cur;
temp = getnode();
temp->info = item;
temp->link = NULL;
if (second == NULL)
    return temp;
cur = second;
while (cur->link != NULL)
    cur = cur->link;
cur->link = temp;
return second;
}
NODE reverse (NODE first)
{
NODE cur, temp;
cur = NULL;
while (first != NULL)
{
    temp = first;
    first = first->link;
    temp->link = cur;
    cur = temp;
}
return cur;
}
NODE ascending (NODE first)
{
NODE prev = first;
NODE cur = NULL;
int temp;
if (first == NULL)
    return;
```

```
else
{
    while (prev != NULL)
    {
        cur = prev -> link;
        while (cur != NULL)
        {
            if (prev -> info > cur -> info)
            {
                temp = prev -> info
                prev -> info = cur -> info;
                cur -> info = temp;
            }
            cur = cur -> link;
        }
        prev = prev -> link;
    }
} return first;
}
NODE descending ( NODE first)
{
    NODE prev = first;
    NODE cur = NULL;
    int temp;
    if ( first == NULL)
        return;
    else
    {
        while ( prev != NULL)
        {
            cur = prev -> link;
            while ( cur != NULL)
            {
```

```
        if (prev->info < cur->info)
        {
            temp = prev->info;
            prev->info = cur->info;
            cur->info = temp;
        }
        cur = cur->link;
    }
    prev = prev->link;
  }
}
return first;
}
NODE concatenate (NODE first, NODE second)
{
    NODE cur;
    if(first==NULL) return second;
    if( second==NULL) return first;
    cur = first;
    while (cur->link != NULL)
    {
        cur = cur->link;
    }
    cur->link = second;
    return first;
}
void display (NODE first)
{
    NODE temp;
    if(first==NULL)
    {
        printf("Empty list\n");
        return;
    }
```

```
printf("List content:");
for(temp=first; temp!=NULL; temp=temp->link)
        printf("%d\n", temp->info);
}
void main()
{
    int item, choice, pos, element, option, choice2, item1, num;
    ;;
    NODE first=NULL;
    NODE second=NULL;
    for(;;)
    {
        printf("\n\n1. insertfront\n2. deletefront
            \n3. Reverselist\n4. sort\n5. Concatenate\n6. Display\n7. Exit\n");

        scanf("%d", &choice);
        switch(choice)
        {
            case 1: pf("Enter item@ front end:");
                scanf("%d", &item);
                first=insertfront(first, item);
                pf("%d inserted @ front", first->info);
                break;

            case 2: first=deletefront(first);
                break;

            case 3: first=reverse(first);
                printf("List is reversed");
                break;

            case 4: printf("1. Ascending\n2. Descending");
```

```
scanf("%d", &option);
if (option == 1)
{
    first = ascending(first);
    pf("List sorted in ascending\n");
}
if (option == 2)
{
    first = descending(first);
    pf("List sorted in descend\n");
}
break;
case 5: pf("create and list\n");
    pf("Enter no of de in and list :");
    scanf("%d", &num);
    for (i=0; i<=num; i++)
    {
        pf("1.insatpoint 2.ins search"); scanf("%d", &choice2);
        if (choice2==1)
        {
            printf("Enter item @ front :");
            scanf("%d", &item1);
            second = IFC(second, item1);
        }
        if (choice2 == 2)
        {
            pf("Enter item @ pos:");
            scanf("%d", &item1);
            second = IR(second, item1);
        }
    }
    first = concatenate(first, second);
    pf("\n lists are concatenated\n");
    break;
case 6: display(first);
    break;
default: exit(0);
}
}
```