Singly Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
};
struct Node * createNode()
{
struct Node *t;
t=(struct Node *)malloc(sizeof(struct Node));
}
struct Node *start;
void addAtEnd(int data)
{
struct Node *t,*j;
t=createNode();
t->data=data;
t->next=NULL;
if(start==NULL)
{
start=t;
}
else
{
j=start;
while(j->next!=NULL) j=j->next;
j->next=t;
}
}
void insertAtTop(int data)
{
struct Node *t;
t=createNode();
t->data=data;
t->next=start;
start=t;
}
```

```
void insertAt(int pos,int data)
{
struct Node *p1,*p2,*t;
int x;
t=createNode();
t->data=data;
t->next=NULL;
if(start==NULL)
{
start=t;
return;
}
if(pos<0)
{
pos=0;
}
for(p1=start,x=1;p1!=NULL && x<=pos;x++)
{
p2=p1;
p1=p1->next;
}
if(p1==NULL)
{
p2->next=t;
return;
}
if(p1==start)
{
t->next=start;
start=t;
return;
}
t->next=p1;
p2->next=t;
}
int removeAt(int pos)
{
struct Node *p1,*p2;
int x;
if(start==NULL || pos<0) return -1;
```

```
for(p1=start,x=1;p1!=NULL && x<=pos;x++)
{
p2=p1;
p1=p1->next;
}
if(p1==NULL) return -1;
if(p1==start)
{
start=start->next;
}
else
{
p2->next=p1->next;
}
free(p1);
}
void traverseTopToBottom()
{
struct Node *t;
for(t=start;t!=NULL;t=t->next) printf("%d\n",t->data);
}
void _traverseBottomToTop(struct Node *t)
{
if(t==NULL) return;
_traverseBottomToTop(t->next);
printf("%d\n",t->data);
}
void traverseBottomToTop()
{
_traverseBottomToTop(start);
}
int main()
{
int ch,num,pos;
start=NULL;
while(1)
{
printf("1. Add at end\n");
printf("2. Insert at top\n");
printf("3. Insert at position\n");
```

```
printf("4. Remove\n");
printf("5. Traverse top to bottom\n");
printf("6. Traverse bottom to top\n");
printf("7. Exit\n");
printf("Enter your choice : ");
scanf("%d",&ch);
fflush(stdin);
if(ch==1)
{
printf("Enter the number to add at end : ");
scanf("%d",&num);
fflush(stdin);
addAtEnd(num);
}else
if(ch==2)
{
printf("Enter the number to insert at top : ");
scanf("%d",&num);
fflush(stdin);
insertAtTop(num);
}else
if(ch==3)
{
printf("Enter the number to insert : ");
scanf("%d",&num);
fflush(stdin);
printf("Enter the position : ");
scanf("%d",&pos);
fflush(stdin);
insertAt(pos,num);
}else
if(ch==4)
{
printf("Enter the position of the number to remove : ");
scanf("%d",&pos);
fflush(stdin);
if(removeAt(pos)==-1)
{
printf("Invalid position %d\n",pos);
}
```

```
}else
if(ch==5) traverseTopToBottom();
else if(ch==6) traverseBottomToTop();
else if(ch==7) break;
else
{
printf("Invalid choice\n");
}
}
return 0;
}
```

SinglyLinkedList Loop Implementation

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
};
struct Node *start=NULL;
struct Node * createNode(int data)
{
struct Node *t;
t=(struct Node *)malloc(sizeof(struct Node));
t->data=data;
t->next=NULL;
return t;
}
void createLinkedList()
{
struct Node *n1,*n2,*n3,*n4;
n1=createNode(10);
n2=createNode(20);
n3=createNode(30);
n4=createNode(40);
start=n1;
n1->next=n2;
n2->next=n3;
n3->next=n4;
```

```
n4->next=n2;
}
int hasLoop(struct Node *b)
{
struct Node *p1,*p2;
if(b==NULL) return 0;  // return false
p1=b->next;
p2=b;
while(p1!=p2)
{
if(p1==NULL || p1->next==NULL) return 0;  // return false
p2=p2->next;
p1=p1->next->next;
}
return 1;  // return true
}
int main()
{
createLinkedList();
if(hasLoop(start))
{
printf("Linked list has a loop");
}
else
{
printf("Linked list does not have a loop");
}
return 0;
}
```

SLL Middle Implementation

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
};
struct Node *start=NULL;
struct Node * createNode(int data)
```

```
{
struct Node *t;
t=(struct Node *)malloc(sizeof(struct Node));
t->data=data;
t->next=NULL;
return t;
}
void createLinkedList()
{
struct Node *n1,*n2,*n3,*n4,*n5;
n1=createNode(10);
n2=createNode(20);
n3=createNode(30);
n4=createNode(40);
n5=createNode(50);
start=n1;
n1->next=n2;
n2->next=n3;
n3->next=n4;
//n4->next=n5;
}
int getMiddle(struct Node *b)
{
struct Node *p1,*p2,*p3;
int count;
if(b==NULL) return 0;
if(b->next==NULL) return b->data;
p1=b->next;
p2=b;
count=1;
while(p1!=NULL)
{
p3=p2;
p2=p2->next;
if(p1->next==NULL)
{
count+=1;
break;
}
p1=p1->next->next;
```

```
count+=2;
}
printf("%d\n",count);
if(count%2==0) return p3->data;
return p2->data;
}
int main()
{
createLinkedList();
printf("%d\n",getMiddle(start));
return 0;
}
```

SLL Palindromic Implementation

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
};
struct Node *start=NULL;
struct Node * createNode(int data)
{
struct Node *t;
t=(struct Node *)malloc(sizeof(struct Node));
t->data=data;
t->next=NULL;
return t;
}
void createLinkedList()
{
struct Node *n1,*n2,*n3,*n4,*n5;
n1=createNode(10);
n2=createNode(20);
n3=createNode(30);
n4=createNode(20);
n5=createNode(10);
start=n1;
n1->next=n2;
n2->next=n3;
```

```
n3->next=n4;
n4->next=n5;
}
void releaseStack(struct Node *b)
{
struct Node *t;
while(b!=NULL)
{
t=b;
b=b->next;
free(t);
}
}
int isPallindrome(struct Node *b)
{
struct Node *p1,*p2,*top,*t;
top=NULL;
int count;
if(b==NULL) return 0;
if(b->next==NULL) return 1;
p2=b;
p1=b->next;
count=1;
// push a node on stack
t=createNode(p2->data);
t->next=top;
top=t;
while(p1!=NULL)
{
p2=p2->next;
// push a node on stack
t=createNode(p2->data);
t->next=top;
top=t;
if(p1->next==NULL)
{
count+=1;
break;
}
count+=2;
```

```
p1=p1->next->next;
}
if(count%2==0)
{
// pop a node from stack
t=top;
top=top->next;
free(t);
}
while(p2!=NULL)
{
if(p2->data!=top->data)
{
releaseStack(top);
return 0;
}
t=top;
top=top->next;
free(t);
p2=p2->next;
}
return 1;
}
int main()
{
createLinkedList();
if(isPallindrome(start)) printf("Pallindrome");
else printf("Not a pallindrome");
return 0;
}
```

SLL Palindromic Implementation

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
};
struct Node *start=NULL;
struct Node * createNode(int data)
```

```
{
struct Node *t;
t=(struct Node *)malloc(sizeof(struct Node));
t->data=data;
t->next=NULL;
return t;
}
void createLinkedList()
{
struct Node *n1,*n2,*n3,*n4,*n5;
n1=createNode(10);
n2=createNode(20);
n3=createNode(30);
n4=createNode(20);
n5=createNode(10);
start=n1;
n1->next=n2;
n2->next=n3;
n3->next=n4;
n4->next=n5;
}
void releaseStack(struct Node *b)
{
struct Node *t;
while(b!=NULL)
{
t=b;
b=b->next;
free(t);
}
}
int isPallindrome(struct Node *b)
{
struct Node *p1,*p2,*top,*t;
top=NULL;
int count;
if(b==NULL) return 0;
if(b->next==NULL) return 1;
p2=b;
p1=b->next;
```

```
count=1;
// push a node on stack
t=createNode(p2->data);
t->next=top;
top=t;
while(p1!=NULL)
{
p2=p2->next;
// push a node on stack
t=createNode(p2->data);
t->next=top;
top=t;
if(p1->next==NULL)
{
count+=1;
break;
}
count+=2;
p1=p1->next->next;
}
if(count%2==0)
{
// pop a node from stack
t=top;
top=top->next;
free(t);
}
while(p2!=NULL)
{
if(p2->data!=top->data)
{
releaseStack(top);
return 0;
}
t=top;
top=top->next;
free(t);
p2=p2->next;
}
return 1;
```

```
}
int main()
{
createLinkedList();
if(isPallindrome(start)) printf("Pallindrome");
else printf("Not a pallindrome");
return 0;
}
```

SLL Remove Implementation

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
};
struct Node * createNode(int data)
{
struct Node *t;
t=(struct Node *)malloc(sizeof(struct Node));
t->data=data;
t->next=NULL;
return t;
}
struct Node * createLinkedList()
{
struct Node *n1,*n2,*n3,*n4,*n5,*n6,*n7,*n8,*n9,*n10,*head;
n1=createNode(10);
n2=createNode(20);
n3=createNode(30);
n4=createNode(40);
n5=createNode(50);
n6=createNode(60);
n7=createNode(70);
n8=createNode(80);
n9=createNode(90);
n10=createNode(100);
head=n1;
n1->next=n2;
n2->next=n3;
```

```
n3->next=n4;
n4->next=n5;
n5->next=n6;
n6->next=n7;
n7->next=n8;
n8->next=n9;
n9->next=n10;
return head;
}
void releaseLinkedList(struct Node *b)
{
struct Node *t;
while(b!=NULL)
{
t=b;
b=b->next;
free(t);
}
}
struct Node * removeElement(struct Node *head,int num)
{
struct Node *p1,*p2;
while(head!=NULL && head->data==num)
{
p1=head;
head=head->next;
free(p1);
}
if(head==NULL) return head;
p1=head->next;
p2=head;
while(p1!=NULL)
{
if(p1->data==num)
{
p2->next=p1->next;
free(p1);
p1=p2->next;
}
else
```

```
{
p2=p1;
p1=p1->next;
}
}
return head;
}
int main()
{
struct Node *head,*t;
head=createLinkedList();
head=removeElement(head,50);
for(t=head;t!=NULL;t=t->next) printf("%d\n",t->data);
releaseLinkedList(head);
return 0;
}
```

SLL ReverseList

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
};
struct Node * createNode(int data)
{
struct Node *t;
t=(struct Node *)malloc(sizeof(struct Node));
t->data=data;
t->next=NULL;
return t;
}
struct Node * createLinkedList()
{
struct Node *n1,*n2,*n3,*n4,*n5,*n6,*n7,*n8,*n9,*n10,*head;
n1=createNode(10);
n2=createNode(20);
n3=createNode(30);
n4=createNode(40);
```

```
n5=createNode(50);
n6=createNode(60);
n7=createNode(70);
n8=createNode(80);
n9=createNode(90);
n10=createNode(100);
head=n1;
n1->next=n2;
n2->next=n3;
n3->next=n4;
n4->next=n5;
n5->next=n6;
n6->next=n7;
n7->next=n8;
n8->next=n9;
n9->next=n10;
return head;
}
void releaseLinkedList(struct Node *b)
{
struct Node *t;
while(b!=NULL)
{
t=b;
b=b->next;
free(t);
}
}
struct Node * reverseLinkedList(struct Node *head)
{
struct Node *t,*nhead;
nhead=NULL;
while(head)
{
t=head;
head=head->next;
t->next=nhead;
nhead=t;
}
return nhead;
```

```
}
int main()
{
struct Node *head,*t;
head=createLinkedList();
head=reverseLinkedList(head);
for(t=head;t!=NULL;t=t->next) printf("%d\n",t->data);
releaseLinkedList(head);
return 0;
}
```

Remove Duplicates

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
};
struct Node * createNode(int data)
{
struct Node *t;
t=(struct Node *)malloc(sizeof(struct Node));
t->data=data;
t->next=NULL;
return t;
}
struct Node * createLinkedList()
{
struct Node *n1,*n2,*n3,*n4,*n5,*n6,*n7,*n8,*n9,*n10,*head;
n1=createNode(10);
n2=createNode(20);
n3=createNode(30);
n4=createNode(30);
n5=createNode(40);
n6=createNode(50);
n7=createNode(50);
n8=createNode(50);
n9=createNode(60);
n10=createNode(70);
head=n1;
```

```
n1->next=n2;
n2->next=n3;
n3->next=n4;
n4->next=n5;
n5->next=n6;
n6->next=n7;
n7->next=n8;
n8->next=n9;
n9->next=n10;
return head;
}
void releaseLinkedList(struct Node *b)
{
struct Node *t;
while(b!=NULL)
{
t=b;
b=b->next;
free(t);
}
}
void removeDuplicates(struct Node *head)
{
struct Node *p2,*p1,*t,*j;
p1=p2=head;
while(p2!=NULL)
{
p1=p2;
while(p1!=NULL && p1->data==p2->data)
{
p1=p1->next;
}
// code to release skipped nodes
t=p2->next;
while(t!=p1)
{
j=t;
t=t->next;
free(j);
}
```

```
p2->next=p1;
p2=p1;
}
}
int main()
{
struct Node *head,*t;
head=createLinkedList();
removeDuplicates(head);
for(t=head;t!=NULL;t=t->next) printf("%d\n",t->data);
releaseLinkedList(head);
return 0;
}
```

SLL Merge Sorted List

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
};
struct Node * createNode(int data)
{
struct Node *t;
t=(struct Node *)malloc(sizeof(struct Node));
t->data=data;
t->next=NULL;
return t;
}
struct Node * createLinkedList(int *x,int size)
{
struct Node *head,*end,*t;
int i;
head=NULL;
end=NULL;
for(i=0;i<size;i++)
{
t=createNode(x[i]);
if(head==NULL)
{
```

```
head=end=t;
}
else
{
end->next=t;
end=t;
}
}
return head;
}
void releaseLinkedList(struct Node *b)
{
struct Node *t;
while(b!=NULL)
{
t=b;
b=b->next;
free(t);
}
}
struct Node * mergeLinkedList(struct Node *head1,struct Node *head2)
{
struct Node *head3,*end3,*t1,*t2,*j;
head3=end3=NULL;
t1=head1;
t2=head2;
while(t1!=NULL && t2!=NULL)
{
if(t1->data<t2->data)
{
j=t1;
t1=t1->next;
}
else
{
j=t2;
t2=t2->next;
}
j->next=NULL;
if(head3==NULL)
```

```
{
head3=end3=j;
}
else
{
end3->next=j;
end3=j;
}
if(t1!=NULL) end3->next=t1;
if(t2!=NULL) end3->next=t2;
}
return head3;
}
int main()
{
struct Node *head1,*head2,*head3,*t;
int x[7]={10,20,30,40,50,60,70};
int y[10]={5,7,19,25,36,54,80,90,120,130};
head1=createLinkedList(x,7);
head2=createLinkedList(y,10);
head3=mergeLinkedList(head1,head2);
for(t=head3;t!=NULL;t=t->next) printf("%d\n",t->data);
releaseLinkedList(head3);
return 0;
}
```