**NAME: KRISHNA KUMAR MAHTO**

**REG.- 16BME0270**

**DIGITAL ASSIGNMENT**

**Solution Problem 1:**

/\*Algorithm:

1. Create a node structure and declare structure pointer variables head, tail and new.

2. Ask for the number of data in sorted manner.

3. Store the entered data in linked list.

4. Define three structure node pointer variables temp1=head,temp2=tail and temp3 (no value, to be used to transfer temp2 to the preceding node).

5. Ask the sum for which pairs is to be determined.

6. Call the find\_sum() function and pass the values x,temp1,temp2,temp3.

7. If the data in the temp1 node summed to the data in temp2 = given sum, then print the node values, assign temp1=temp1->next, temp2=temp2->prev.

Recursively call the function find\_sum() with same arguments-x,temp1,temp2,temp3.

8. If the sum of data in temp1 and temp2 is greater than the given sum, then temp2=temp2->prev. Recursively call the find\_sum().

9. If sum of data in temp1 and temp2 is smaller than the given sum, temp1=temp1->next. Recursively call the function find\_sum().

10. Base case is when temp1->next=temp2.

\*/

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*prev,\*next;

}\*nw,\*head,\*tail;

void display()

{

struct node \*temp;

printf("The nodes are:\n");

for(temp=head;temp->next!=NULL;temp=temp->next)

printf("%d\n",temp->data);

printf("%d\n",temp->data);

}

void creation()

{

int n,item,i;

puts("Enter the number of nodes: ");

scanf("%d",&n);

puts("Enter the values: ");

for(i=0;i<n;i++)

{

scanf("%d",&item);

if(nw->data==NULL)

nw->data=item;

else

{

nw=(struct node\*)malloc(sizeof(struct node));

nw->data=item;

nw->prev=tail;

nw->next=NULL;

tail->next=nw;

tail=nw;

}

}

}

void find\_sum(int x,struct node \*temp1,struct node \*temp2)

{

if((temp1->data+temp2->data)==x)

{

printf("(%d,%d)\n",temp1->data,temp2->data);

temp1=temp1->next;

if(temp1==temp2)

return;

else

temp2=temp2->prev;

if(temp1->next!=temp2)

find\_sum(x,temp1,temp2);

else

printf("(%d,%d)\n",temp1->data,temp2->data);

}

else if(((temp1->data)+(temp2->data))>x)

{

temp2=temp2->prev;

find\_sum(x,temp1,temp2);

}

else if(((temp1->data)+(temp2->data))<x)

{

temp1=temp1->next;

find\_sum(x,temp1,temp2);

}

}

int main()

{

int x;//sum=x

struct node \*temp1,\*temp2;

nw=(struct node\*)malloc(sizeof(struct node));

nw->data=NULL;

nw->prev=NULL;

nw->next=NULL;

head=nw;

tail=nw;

creation();

temp1=head;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*yahan se resume krna h

temp2=tail;

display();

puts("Enter the sum you want to find pairs for: ");

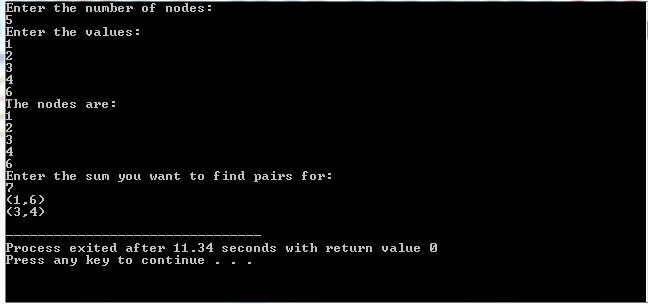
scanf("%d",&x);

find\_sum(x,temp1,temp2);

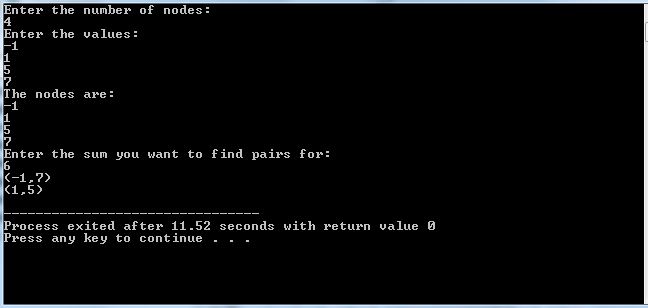
return 0;

}

**TEST CASE 1**



**TEST CASE 2**



**SOLUTION PROBLEM 2**

/\*ALGORITHM:

1. Create node structure.

2. Take number of nodes and the data for nodes. Also ask for the central value around which partition is to be done.

3. Pass the head pointer to the function partition(struct node \*address).

4. If address->data<central\_val, then check if address->next=NULL, if it is true then check if it is the first smaller value found using a flag variable. Depending on if it was the first time, adjust the small\_head and small\_tail pointers. As address->next=NULL, we have reached the end of the list entered. So we stop. If address->next is not NULL, then call recursively the partition(struct node \*address) function with address->next as the argument.

5. If address->data>central\_val, then after checking the same conditions as in step 4, we adjust the large\_head and large\_tail pointers and then either stop or recursively call the function depending on the value of address->next.

6. If address->data=central\_val, similar adjustments are made to the equal\_head and equal\_tail pointers. Then again depending on the value of address->next, the recursive call is either made or not made.

7. Separate variables are used in order to find which values were encountered while going through the list (smaller than, greater than or equal to the central\_val). These variables are found\_smaller,found\_geater,found\_equal. Based on their values, the small\_tail,large\_head, and other equal pointers are joined together to give the final list that satisfies the ask of the question. \*/

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

}\*head,\*tail,\*nw;

int flag=1,central\_val,flag2=1,flag3=1,found\_smaller=0,found\_equal=0,found\_greater=0;

struct node \*small\_head,\*small\_tail,\*large\_head,\*large\_tail,\*temp,\*equal\_head,\*equal\_tail;

void creation(int n)

{

int item,i;

puts("Enter data: ");

for(i=0;i<n;i++)

{

scanf("%d",&item);

nw=(struct node\*)malloc(sizeof(struct node));

nw->data=item;

if(flag==1)

{

head=nw;

tail=nw;

tail->next=NULL;

flag=0;

}

else

{

tail->next=nw;

tail=nw;

tail->next=NULL;

}

}

flag=1;

}

void display()

{

struct node \*temp;

puts("The linked list is: ");

for(temp=head;temp!=tail;temp=temp->next)

printf("%d->",temp->data);

printf("%d\n\n",temp->data);

}

void partition(struct node\* address)

{

if(address->data<central\_val)

{

found\_smaller=1;

if(address->next==NULL)//if true no need to further call the function recursively, this is the base case if the list ends at a number smaller than central val.

{

if(flag2==1)

{

small\_head=address;

small\_tail=address;

flag2=0;

}

else

{

small\_tail->next=address;

small\_tail=address;

}

}

else

{

if(flag2==1)

{

small\_head=address;

small\_tail=address;

flag2=0;

}

else

{

small\_tail->next=address;

small\_tail=small\_tail->next;

}

partition(address->next);

}

}

else if(address->data>central\_val)

{

found\_greater=1;

if(address->next==NULL)

{

if(flag==1)

{

large\_head=address;

large\_tail=large\_head;

flag=0;

}

else

{

large\_tail->next=address;

large\_tail=large\_tail->next;

}

}

else

{

if(flag==1)

{

large\_head=address;

large\_tail=address;

flag=0;

}

else

{

large\_tail->next=address;

large\_tail=large\_tail->next;

}

partition(address->next);

}

}

else if(address->data==central\_val)

{

found\_equal=1;

if(address->next==NULL)

{

if(flag3==1)

{

equal\_head=address;

equal\_tail=equal\_head;

flag3=0;

}

else

{

equal\_tail->next=address;

equal\_tail=address;

}

}

else

{

if(flag3==1)

{

equal\_head=address;

equal\_tail=address;

flag3=0;

}

else

{

equal\_tail->next=address;

equal\_tail=equal\_tail->next;

}

partition(address->next);

}

}

}

int main()

{

int n;

puts("Enter the number of nodes:");

scanf("%d",&n);

creation(n);

display();

puts("Enter the central value: ");

scanf("%d",&central\_val);

partition(head);

if(found\_smaller==1 && found\_equal==1 && found\_greater==1)

{

small\_tail->next=equal\_head;

equal\_tail->next=large\_head;

for(temp=small\_head;temp!=large\_tail;temp=temp->next)

printf("%d->",temp->data);

printf("%d\n",temp->data);

}

else if(found\_smaller==1 && found\_equal==0 && found\_greater==1)

{

small\_tail->next=large\_head;

for(temp=small\_head;temp!=large\_tail;temp=temp->next)

printf("%d->",temp->data);

printf("%d\n",temp->data);

}

else if(found\_smaller==0 && found\_equal==1 && found\_greater==1)

{

equal\_tail->next=large\_head;

for(temp=equal\_head;temp!=large\_tail;temp=temp->next)

printf("%d->",temp->data);

printf("%d\n",temp->data);

}

else if(found\_smaller==1 && found\_equal==1 && found\_greater==0)

{

small\_tail->next=equal\_head;

for(temp=small\_head;temp!=large\_tail;temp=temp->next)

printf("%d->",temp->data);

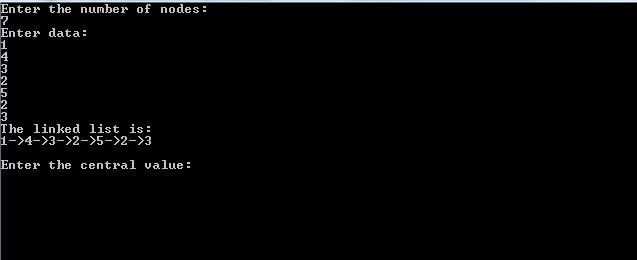
printf("%d\n",temp->data);

}

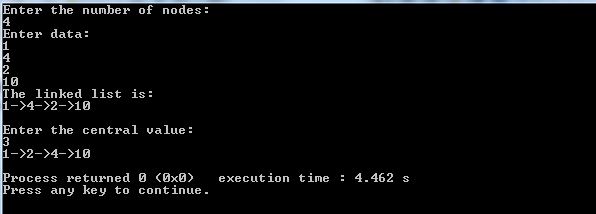
return 0;

}

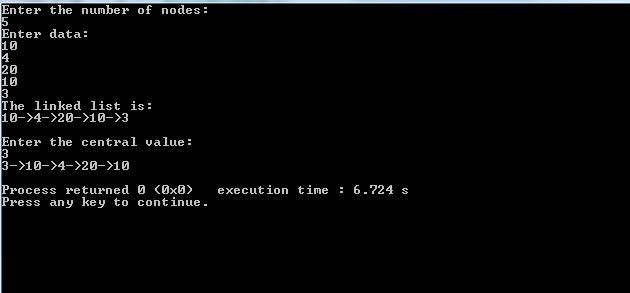
**TEST CASE 1:**

****

**TEST CASE 2:**

****

**TEST CASE 3:**

****

**Solution Problem 3:**

/\*ALGORITHM

1. Create structure node.

2. Take the number of nodes in first and second lists.

3. Create the two lists using creation(int n) routine that takes the number of nodes as formal arguments.

4. Call subset\_or\_not() routine that takes the two head and two tail pointers.

5. If the length of first subset>length of second subset, it is an invalid situation.

6.If condition in step 5 is not true, then go on comparing till the first node in list 1 is not equal to any of the nodes in list 2.

7. If in step 6, equal node in list 2 is found, control flows out of the loop.

8. If number of elements left- from node in list 2 which is equal to first node in list 1, to the end of list 2 is >= number of total nodes in list 1, then go on comparing

the following elements of both the lists until a pair of unequal elements is found or the end of list 1 is reached.

9. If the end of list 1 is reached while following step 8, then list 1 is present in list 2.

10. If an unequal pair is found, then check if it happend at the end of the first node. If it happened at the end of the first node, then list 1 is not present in list 2.

11. If an unequal pair is found and we're not at the end of list 1, then recursively call the subset\_or\_not function.\*/

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

}\*head,\*tail,\*nw;

int flag=1;

void creation(int n)

{

int i,item;

puts("Enter the items: ");

for(i=0;i<n;i++)

{

scanf("%d",&item);

nw=(struct node\*)malloc(sizeof(struct node));

nw->data=item;

if(flag==1)

{

head=nw;

tail=nw;

flag=0;

}

else

{

tail->next=nw;

tail=nw;

tail->next=NULL;

}

}

}

int length(struct node\* head,struct node\* tail)

{

struct node\* temp;

int count=0;

for(temp=head;temp->next!=NULL;temp=temp->next)

count++;

return(++count);

}

void subset\_or\_not(struct node \*head1,struct node \*tail1,struct node \*head2,struct node \*tail2)

{

struct node \*temp1,\*temp2;

int length1;// I will be using length of list one more than once, so once length of list one is found and is smaller than or equal to list 2, I shall store it in length1.

length1=length(head1,tail1);

if(length1>length(head2,tail2))

{

puts("First list can't be longer than the second!");

return;

}

else

{

for(temp1=head1,temp2=head2;temp1->data!=temp2->data;temp2=temp2->next)

{

if(temp2->next!=NULL)

continue;

else

{

puts("Not even a single node matches, list 1 not present in list 2.");

return;

}

}

if(length1<=length(temp2,tail2))//if length of list 1 is smaller than the length of list two from the matching node to its tail, then only list 1 subset of list2

{

for(;temp1->data==temp2->data;temp1=temp1->next,temp2=temp2->next)

{

if(temp1->next==NULL)

{

puts("List 1 is present in List 2.");

exit(0);//this line ensures if list 1 is present in list 2, the control will flow out of the loop, but will exit from inside itself.

}

continue;

}

if(temp1->next==NULL)//this cond is out of the loop, that means the last node in list 1 didn't match with some node in list 2.

{

puts("List 1 is not present in List 2.");

exit(0);

}

subset\_or\_not(head1,tail1,temp2,tail2);//again call the subset\_or\_not() fn recursively, but now comparison will begin where it ended previously.

}

else

puts("List 1 is not contained in List 2.");

}

}

int main()

{

struct node \*head1,\*tail1,\*head2,\*tail2;

int total\_1,total\_2;

puts("Enter the number of nodes in first list: ");

scanf("%d",&total\_1);

creation(total\_1);

head1=head;

tail1=tail;

flag=1;

puts("Enter the number of nodes in second list: ");

scanf("%d",&total\_2);

creation(total\_2);

head2=head;

tail2=tail;

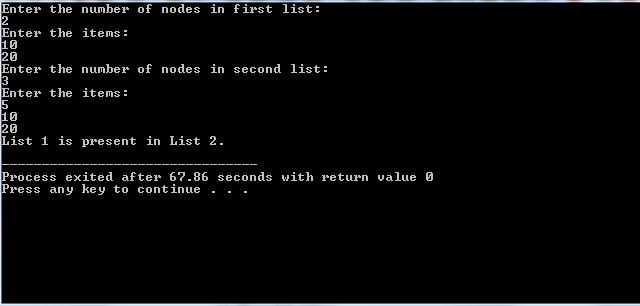
flag=1;

subset\_or\_not(head1,tail1,head2,tail2);

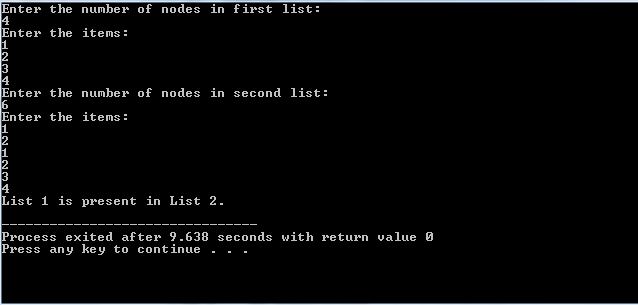
return 0;

}

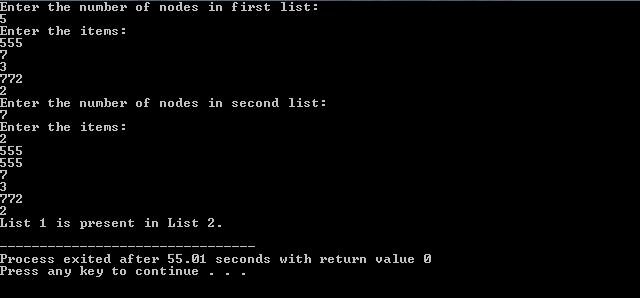
**TEST CASE 1:**



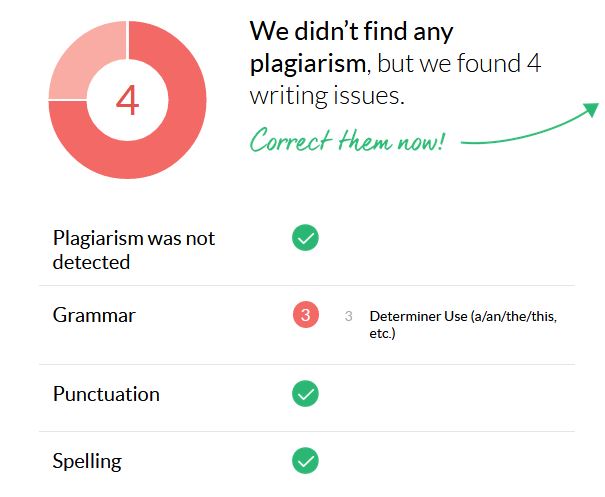
**TEST CASE 2:**



**TEST CASE 3:**

****

**PLAGIARISM CHECK:**

****