#ssinclude <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the next node

}\*stnode;

void createNodeList(int n); // function to create the list

void displayList(); // function to display the list

int main()

{

int n;

printf("\n\n Linked List : To create and display Singly Linked List :\n");

printf("-------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode->num = num;

stnode->nextptr = NULL; // links the address field to NULL

tmp = stnode;

// Creating n nodes and adding to linked list

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL)

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" List is empty.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

1…

3 ans

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void reverseDispList(); //function to convert the list in reverse

void displayList(); //function to display the list

int main()

{

int n;

printf("\n\n Linked List : Create a singly linked list and print it in reverse order :\n");

printf("------------------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

reverseDispList();

printf("\n The list in reverse are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

void reverseDispList()

{

struct node \*prevNode, \*curNode;

if(stnode != NULL)

{

prevNode = stnode;

curNode = stnode->nextptr;

stnode = stnode->nextptr;

prevNode->nextptr = NULL; //convert the first node as last

while(stnode != NULL)

{

stnode = stnode->nextptr;

curNode->nextptr = prevNode;

prevNode = curNode;

curNode = stnode;

}

stnode = prevNode; //convert the last node as head

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

2ans

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

int NodeCount(); //function to count the nodes

void displayList(); //function to display the list

int main()

{

int n,totalNode;

printf("\n\n Linked List : Create a singly linked list and count the number of nodes :\n");

printf("------------------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

totalNode = NodeCount();

printf("\n Total number of nodes = %d\n", totalNode);

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

int NodeCount()

{

int ctr = 0;

struct node \*tmp;

tmp = stnode;

while(tmp != NULL)

{

ctr++;

tmp = tmp->nextptr;

}

return ctr;

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

//4

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void FirstNodeDeletion(); //function to delete the first node

void displayList(); //function to display the list

int main()

{

int n,num,pos;

printf("\n\n Linked List : Delete first node of Singly Linked List :\n");

printf("------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

FirstNodeDeletion();

printf("\n Data, after deletion of first node : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

void FirstNodeDeletion()

{

struct node \*toDelptr;

if(stnode == NULL)

{

printf(" There are no node in the list.");

}

else

{

toDelptr = stnode;

stnode = stnode->nextptr;

printf("\n Data of node 1 which is being deleted is : %d\n", toDelptr->num);

free(toDelptr); // Clears the memory occupied by first node

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

//5

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void MiddleNodeDeletion(int pos); //function to delete a node from middle

void displayList(); //function to display the list

int main()

{

int n,num,pos;

printf("\n\n Linked List : Delete a node from the middle of Singly Linked List. :\n");

printf("-------------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

printf("\n Input the position of node to delete : ");

scanf("%d", &pos);

if(pos<=1 || pos>=n)

{

printf("\n Deletion can not be possible from that position.\n ");

}

if(pos>1 && pos<n)

{

printf("\n Deletion completed successfully.\n ");

MiddleNodeDeletion(pos);

}

printf("\n The new list are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

void MiddleNodeDeletion(int pos)

{

int i;

struct node \*toDelMid, \*preNode;

if(stnode == NULL)

{

printf(" There are no nodes in the List.");

}

else

{

toDelMid = stnode;

preNode = stnode;

for(i=2; i<=pos; i++)

{

preNode = toDelMid;

toDelMid = toDelMid->nextptr;

if(toDelMid == NULL)

break;

}

if(toDelMid != NULL)

{

if(toDelMid == stnode)

stnode = stnode->nextptr;

preNode->nextptr = toDelMid->nextptr;

toDelMid->nextptr = NULL;

free(toDelMid);

}

else

{

printf(" Deletion can not be possible from that position.");

}

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

//6

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void LastNodeDeletion(); //function to delete the last nodes

void displayList(); //function to display the list

int main()

{

int n,num,pos;

printf("\n\n Linked List : Delete the last node of Singly Linked List :\n");

printf("---------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

LastNodeDeletion();

printf("\n The new list after deletion the last node are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

// Deletes the last node of the linked list

void LastNodeDeletion()

{

struct node \*toDelLast, \*preNode;

if(stnode == NULL)

{

printf(" There is no element in the list.");

}

else

{

toDelLast = stnode;

preNode = stnode;

/\* Traverse to the last node of the list\*/

while(toDelLast->nextptr != NULL)

{

preNode = toDelLast;

toDelLast = toDelLast->nextptr;

}

if(toDelLast == stnode)

{

stnode = NULL;

}

else

{

/\* Disconnects the link of second last node with last node \*/

preNode->nextptr = NULL;

}

/\* Delete the last node \*/

free(toDelLast);

}

}

// function to display the entire list

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the empty list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

//7

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num;

struct node \*next;

};

int create(struct node \*\*);

int palin\_check (struct node \*, int);

void release(struct node \*\*);

int main()

{

struct node \*p = NULL;

int result, count;

printf("Enter data into the list\n");

count = create(&p);

result = palin\_check(p, count);

if (result == 1)

{

printf("The linked list is a palindrome.\n");

}

else

{

printf("The linked list is not a palindrome.\n");

}

release (&p);

return 0;

}

int palin\_check (struct node \*p, int count)

{

int i = 0, j;

struct node \*front, \*rear;

while (i != count / 2)

{

front = rear = p;

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

{

front = front->next;

}

for (j = 0; j < count - (i + 1); j++)

{

rear = rear->next;

}

if (front->num != rear->num)

{

return 0;

}

else

{

i++;

}

}

return 1;

}

int create (struct node \*\*head)

{

int c, ch, count = 0;

struct node \*temp;

do

{

printf("Enter number: ");

scanf("%d", &c);

count++;

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

temp->num = c;

temp->next = \*head;

\*head = temp;

printf("Do you wish to continue [1/0]: ");

scanf("%d", &ch);

}while (ch != 0);

printf("\n");

return count;

}

void release (struct node \*\*head)

{

struct node \*temp = \*head;

while ((\*head) != NULL)

{

(\*head) = (\*head)->next;

free(temp);

temp = \*head;

}

}

//8

1. #include <stdio.h>
3. //Represent a node of the singly linked list
4. **struct** node{
5. **int** data;
6. **struct** node \*next;
7. };
9. //Represent the head and tail of the singly linked list
10. **struct** node \*head, \*tail = NULL;
12. //addNode() will add a new node to the list
13. **void** addNode(**int** data) {
14. //Create a new node
15. **struct** node \*newNode = (**struct** node\*)malloc(**sizeof**(**struct** node));
16. newNode->data = data;
17. newNode->next = NULL;
19. //Checks if the list is empty
20. **if**(head == NULL) {
21. //If list is empty, both head and tail will point to new node
22. head = newNode;
23. tail = newNode;
24. }
25. **else** {
26. //newNode will be added after tail such that tail's next will point to newNode
27. tail->next = newNode;
28. //newNode will become new tail of the list
29. tail = newNode;
30. }
31. }
33. //minNode() will find out the minimum value node in the list
34. **void** minNode() {
35. **struct** node \*current = head;
36. **int** min;
38. **if**(head == NULL) {
39. printf("List is empty \n");
40. }
41. **else** {
42. //Initializing min with head node data
43. min = head->data;
45. **while**(current != NULL){
46. //If current node's data is smaller than min
47. //Then, replace value of min with current node's data
48. **if**(min > current->data) {
49. min = current->data;
50. }
51. current= current->next;
52. }
53. printf("Minimum value node in the list: %d\n", min);
54. }
55. }
57. //maxNode() will find out the maximum value node in the list
58. **void** maxNode() {
59. **struct** node \*current = head;
60. **int** max;
62. **if**(head == NULL) {
63. printf("List is empty \n");
64. }
65. **else** {
66. //Initializing max with head node data
67. max = head->data;
69. **while**(current != NULL){
70. //If current node's data is greater than max
71. //Then, replace value of max with current node's data
72. **if**(max < current->data) {
73. max = current->data;
74. }
75. current = current->next;
76. }
77. printf("Maximum value node in the list: %d\n", max);
78. }
79. }
81. **int** main()
82. {
83. //Adds data to the list
84. addNode(5);
85. addNode(8);
86. addNode(1);
87. addNode(6);
89. //Display the minimum value node in the list
90. minNode();
92. //Display the maximum value node in the list
93. maxNode();
95. **return** 0;
96. }

//9

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void insertNodeAtMiddle(int num, int pos); //function to insert node at the middle

void displayList(); //function to display the list

int main()

{

int n,num,pos;

printf("\n\n Linked List : Insert a new node at the middle of the Linked List :\n");

printf("-----------------------------------------------------------------------\n");

printf(" Input the number of nodes (3 or more) : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

printf("\n Input data to insert in the middle of the list : ");

scanf("%d", &num);

printf(" Input the position to insert new node : " );

scanf("%d", &pos);

if(pos<=1 || pos>=n)

{

printf("\n Insertion can not be possible in that position.\n ");

}

if(pos>1 && pos<n)

{

insertNodeAtMiddle(num, pos);

printf("\n Insertion completed successfully.\n ");

}

printf("\n The new list are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

void insertNodeAtMiddle(int num, int pos)

{

int i;

struct node \*fnNode, \*tmp;

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

if(fnNode == NULL)

{

printf(" Memory can not be allocated.");

}

else

{

fnNode->num = num; //Links the data part

fnNode->nextptr = NULL;

tmp = stnode;

for(i=2; i<=pos-1; i++)

{

tmp = tmp->nextptr;

if(tmp == NULL)

break;

}

if(tmp != NULL)

{

fnNode->nextptr = tmp->nextptr; //Links the address part of new node

tmp->nextptr = fnNode;

}

else

{

printf(" Insert is not possible to the given position.\n");

}

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the empty list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

//10

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void NodeInsertatBegin(int num); //function to insert node at the beginning

void displayList(); //function to display the list

int main()

{

int n,num;

printf("\n\n Linked List : Insert a new node at the beginning of a Singly Linked List:\n");

printf("------------------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

printf("\n Input data to insert at the beginning of the list : ");

scanf("%d", &num);

NodeInsertatBegin(num);

printf("\n Data after inserted in the list are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

void NodeInsertatBegin(int num)

{

struct node \*fnNode;

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

if(fnNode == NULL)

{

printf(" Memory can not be allocated.");

}

else

{

fnNode->num = num; //Links the data part

fnNode->nextptr = stnode; //Links the address part

stnode = fnNode; //Makes stnode as first node

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

//11

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num; //Data of the node

struct node \*nextptr; //Address of the node

}\*stnode;

void createNodeList(int n); //function to create the list

void NodeInsertatEnd(int num); //function to insert node at the end

void displayList(); //function to display the list

int main()

{

int n,num;

printf("\n\n Linked List : Insert a new node at the end of a Singly Linked List :\n");

printf("-------------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

createNodeList(n);

printf("\n Data entered in the list are : \n");

displayList();

printf("\n Input data to insert at the end of the list : ");

scanf("%d", &num);

NodeInsertatEnd(num);

printf("\n Data, after inserted in the list are : \n");

displayList();

return 0;

}

void createNodeList(int n)

{

struct node \*fnNode, \*tmp;

int num, i;

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

if(stnode == NULL) //check whether the stnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

}

else

{

// reads data for the node through keyboard

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode-> num = num;

stnode-> nextptr = NULL; //Links the address field to NULL

tmp = stnode;

//Creates n nodes and adds to linked list

for(i=2; i<=n; i++)

{

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

if(fnNode == NULL) //check whether the fnnode is NULL and if so no memory allocation

{

printf(" Memory can not be allocated.");

break;

}

else

{

printf(" Input data for node %d : ", i);

scanf(" %d", &num);

fnNode->num = num; // links the num field of fnNode with num

fnNode->nextptr = NULL; // links the address field of fnNode with NULL

tmp->nextptr = fnNode; // links previous node i.e. tmp to the fnNode

tmp = tmp->nextptr;

}

}

}

}

void NodeInsertatEnd(int num)

{

struct node \*fnNode, \*tmp;

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

if(fnNode == NULL)

{

printf(" Memory can not be allocated.");

}

else

{

fnNode->num = num; //Links the data part

fnNode->nextptr = NULL;

tmp = stnode;

while(tmp->nextptr != NULL)

tmp = tmp->nextptr;

tmp->nextptr = fnNode; //Links the address part

}

}

void displayList()

{

struct node \*tmp;

if(stnode == NULL)

{

printf(" No data found in the empty list.");

}

else

{

tmp = stnode;

while(tmp != NULL)

{

printf(" Data = %d\n", tmp->num); // prints the data of current node

tmp = tmp->nextptr; // advances the position of current node

}

}

}

//12

#include <stdio.h>

#include <stdlib.h>

struct node {

int num;

struct node \* preptr;

struct node \* nextptr;

}\*stnode, \*ennode;

void DlListcreation(int n);

void displayDlList();

int main()

{

int n;

stnode = NULL;

ennode = NULL;

printf("\n\n Doubly Linked List : Create and display a doubly linked list :\n");

printf("-------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

DlListcreation(n);

displayDlList();

return 0;

}

void DlListcreation(int n)

{

int i, num;

struct node \*fnNode;

if(n >= 1)

{

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

if(stnode != NULL)

{

printf(" Input data for node 1 : "); // assigning data in the first node

scanf("%d", &num);

stnode->num = num;

stnode->preptr = NULL;

stnode->nextptr = NULL;

ennode = stnode;

// putting data for rest of the nodes

for(i=2; i<=n; i++)

{

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

if(fnNode != NULL)

{

printf(" Input data for node %d : ", i);

scanf("%d", &num);

fnNode->num = num;

fnNode->preptr = ennode; // new node is linking with the previous node

fnNode->nextptr = NULL;

ennode->nextptr = fnNode; // previous node is linking with the new node

ennode = fnNode; // assign new node as last node

}

else

{

printf(" Memory can not be allocated.");

break;

}

}

}

else

{

printf(" Memory can not be allocated.");

}

}

}

void displayDlList()

{

struct node \* tmp;

int n = 1;

if(stnode == NULL)

{

printf(" No data found in the List yet.");

}

else

{

tmp = stnode;

printf("\n\n Data entered on the list are :\n");

while(tmp != NULL)

{

printf(" node %d : %d\n", n, tmp->num);

n++;

tmp = tmp->nextptr; // current pointer moves to the next node

}

}

}

//13

#include <stdio.h>

#include <stdlib.h>

struct node {

int num;

struct node \* nextptr;

}\*stnode;

struct node \*tail,\*p,\*q,\*store;

void ClListcreation(int n);

void ClListDeleteFirstNode();

void displayClList(int a);

int main()

{

int n,num1,a,insPlc;

stnode = NULL;

printf("\n\n Circular Linked List : Delete node from the beginning of a circular linked list :\n");

printf("--------------------------------------------------------------------------------------\n");

printf(" Input the number of nodes : ");

scanf("%d", &n);

ClListcreation(n);

a=1;

displayClList(a);

ClListDeleteFirstNode();

a=2;

displayClList(a);

return 0;

}

void ClListcreation(int n)

{

int i, num;

struct node \*preptr, \*newnode;

if(n >= 1)

{

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

printf(" Input data for node 1 : ");

scanf("%d", &num);

stnode->num = num;

stnode->nextptr = NULL;

preptr = stnode;

for(i=2; i<=n; i++)

{

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

printf(" Input data for node %d : ", i);

scanf("%d", &num);

newnode->num = num;

newnode->nextptr = NULL; // next address of new node set as NULL

preptr->nextptr = newnode; // previous node is linking with new node

preptr = newnode; // previous node is advanced

}

preptr->nextptr = stnode; //last node is linking with first node

}

}

void ClListDeleteFirstNode()

{

p=stnode;

while(p->nextptr!=stnode)

{

p=p->nextptr;

}

store=stnode;

stnode=stnode->nextptr;

printf("\n The deleted node is -> %d",store->num);

p->nextptr=stnode;

free (store);

}

void displayClList(int m)

{

struct node \*tmp;

int n = 1;

if(stnode == NULL)

{

printf(" No data found in the List yet.");

}

else

{

tmp = stnode;

if (m==1)

{

printf("\n Data entered in the list are :\n");

}

else

{

printf("\n After deletion the new list are :\n");

}

do {

printf(" Data %d = %d\n", n, tmp->num);

tmp = tmp->nextptr;

n++;

}while(tmp != stnode);

}

}

//14

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

int STACK[MAX],TOP;

/\* display stack element\*/

void display(int []);

/\* push (insert) item into stack\*/

void PUSH(int [],int);

/\* pop (remove) item from stack\*/

void POP (int []);

void main()

{

int ITEM=0;

int choice=0;

TOP=-1;

while(1)

{

/\*clrscr();\*/

printf("Enter Choice (1: display, 2: insert (PUSH), 3: remove(POP)), 4: Exit..:");

scanf("%d",&choice);

switch(choice)

{

case 1:

display(STACK);

break;

case 2:

printf("Enter Item to be insert :");

scanf("%d",&ITEM);

PUSH(STACK,ITEM);

break;

case 3:

POP(STACK);

break;

case 4:

exit(0);

default:

printf("\nInvalid choice.");

break;

}

getch();

}// end of while(1)

}

/\* function : display(),

to display stack elements.

\*/

void display(int stack[])

{

int i=0;

if(TOP==-1)

{

printf("Stack is Empty .\n");

return;

}

printf("%d <-- TOP ",stack[TOP]);

for(i=TOP-1;i >=0;i--)

{

printf("\n%d",stack[i]);

}

printf("\n\n");

}

/\* function : PUSH(),

to push an item into stack.

\*/

void PUSH(int stack[],int item)

{

if(TOP==MAX-1)

{

printf("\nSTACK is FULL CAN't ADD ITEM\n");

return;

}

TOP++;

stack[TOP]=item;

}

/\* function : POP(),

to pop an item from stack.

\*/

void POP(int stack[])

{

int deletedItem;

if(TOP==-1)

{

printf("STACK is EMPTY.\n");

return;

}

deletedItem=stack[TOP];

TOP--;

printf("%d deleted successfully\n",deletedItem);

return;

}

//15

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head = NULL;

void push(int val)

{

//create new node

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

newNode->data = val;

//make the new node points to the head node

newNode->next = head;

//make the new node as head node

//so that head will always point the last inserted data

head = newNode;

}

void pop()

{

//temp is used to free the head node

struct node \*temp;

if(head == NULL)

printf("Stack is Empty\n");

else

{

printf("Poped element = %d\n", head->data);

//backup the head node

temp = head;

//make the head node points to the next node.

//logically removing the node

head = head->next;

//free the poped element's memory

free(temp);

}

}

//print the linked list

void printList()

{

struct node \*temp = head;

//iterate the entire linked list and print the data

while(temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

}

int main()

{

push(10);

push(20);

push(30);

printf("Linked List\n");

printList();

pop();

printf("After the pop, the new linked list\n");

printList();

pop();

printf("After the pop, the new linked list\n");

printList();

return 0;

}

//16

#include<stdio.h>

#include<stdlib.h>

# define N 20 //defining the size of queue

int s[N],top = -1;

int pop()//function to remove an element from stack

{

return s[top--];

}

void push(int x)//function to insert an element into stack

{

if(top == N-1)

printf("Stack is Full");

else

{

top++;

s[top] = x;

}

}

void enqueue(int x)//function to insert element in the queue using recursive stack call

{

push(x);

}

void print()//function to print elements of a queue

{

int i;

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

printf("\n%d",s[i]);

}

int dequeue()//function to dequeue element from a queue using recursive stack call

{

int data,res;

if(top == -1)

printf("Queue is Empty");

else if(top == 0)

return pop();

data = pop();

res = dequeue();

push(data);

return res;

}

int main()

{

int opt,n,i,data,t;

printf("Enter Your Choice:-");

do{

printf("\n\n1 for Insert the Data in Queue\n2 for show the Data in Queue \n3 for Delete the data from the Queue\n0 for Exit\n");

scanf("%d",&opt);

switch(opt){

case 1:

printf("\nEnter the number of elements:");

scanf("%d",&n);

printf("\nEnter your data\n");

i=0;

while(i<n){

scanf("%d",&data);

enqueue(data);

i++;

}

break;

case 2:

print();

break;

case 3:

t = dequeue();

printf("Dequeued element:%d",t);

break;

case 0:

break;

default:

printf("\nIncorrect Choice");

}

}while(opt!=0);

}

//17

#include<stdio.h>

#include<stdlib.h> /\* for exit() \*/

#include<ctype.h> /\* for isdigit(char ) \*/

#include<string.h>

#define SIZE 100

/\* declared here as global variable because stack[]

\* is used by more than one fucntions \*/

**char** stack[SIZE];

**int** top = -1;

/\* define push operation \*/

**void** push(**char** item)

{

**if**(top >= SIZE-1)

{

printf("\nStack Overflow.");

}

**else**

{

top = top+1;

stack[top] = item;

}

}

/\* define pop operation \*/

**char** pop()

{

**char** item ;

**if**(top <0)

{

printf("stack under flow: invalid infix expression");

getchar();

/\* underflow may occur for invalid expression \*/

/\* where ( and ) are not matched \*/

exit(1);

}

**else**

{

item = stack[top];

top = top-1;

**return**(item);

}

}

/\* define function that is used to determine whether any symbol is operator or not

(that is symbol is operand)

\* this fucntion returns 1 if symbol is opreator else return 0 \*/

**int** is\_operator(**char** symbol)

{

**if**(symbol == '^' || symbol == '\*' || symbol == '/' || symbol == '+' || symbol =='-')

{

**return** 1;

}

**else**

{

**return** 0;

}

}

/\* define fucntion that is used to assign precendence to operator.

\* Here ^ denotes exponent operator.

\* In this fucntion we assume that higher integer value

\* means higher precendence \*/

**int** precedence(**char** symbol)

{

**if**(symbol == '^')/\* exponent operator, highest precedence\*/

{

**return**(3);

}

**else** **if**(symbol == '\*' || symbol == '/')

{

**return**(2);

}

**else** **if**(symbol == '+' || symbol == '-') /\* lowest precedence \*/

{

**return**(1);

}

**else**

{

**return**(0);

}

}

**void** InfixToPostfix(**char** infix\_exp[], **char** postfix\_exp[])

{

**int** i, j;

**char** item;

**char** x;

push('('); /\* push '(' onto stack \*/

strcat(infix\_exp,")"); /\* add ')' to infix expression \*/

i=0;

j=0;

item=infix\_exp[i]; /\* initialize before loop\*/

**while**(item != '\0') /\* run loop till end of infix expression \*/

{

**if**(item == '(')

{

push(item);

}

**else** **if**( isdigit(item) || isalpha(item))

{

postfix\_exp[j] = item; /\* add operand symbol to postfix expr \*/

j++;

}

**else** **if**(is\_operator(item) == 1) /\* means symbol is operator \*/

{

x=pop();

**while**(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix\_exp[j] = x; /\* so pop all higher precendence operator and \*/

j++;

x = pop(); /\* add them to postfix expresion \*/

}

push(x);

/\* because just above while loop will terminate we have

oppped one extra item

for which condition fails and loop terminates, so that one\*/

push(item); /\* push current oprerator symbol onto stack \*/

}

**else** **if**(item == ')') /\* if current symbol is ')' then \*/

{

x = pop(); /\* pop and keep popping until \*/

**while**(x != '(') /\* '(' encounterd \*/

{

postfix\_exp[j] = x;

j++;

x = pop();

}

}

**else**

{ /\* if current symbol is neither operand not '(' nor ')' and nor

operator \*/

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

i++;

item = infix\_exp[i]; /\* go to next symbol of infix expression \*/

} /\* while loop ends here \*/

**if**(top>0)

{

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

**if**(top>0)

{

printf("\nInvalid infix Expression.\n"); /\* the it is illegeal symbol \*/

getchar();

exit(1);

}

postfix\_exp[j] = '\0'; /\* add sentinel else puts() fucntion \*/

/\* will print entire postfix[] array upto SIZE \*/

}

/\* main function begins \*/

**int** main()

{

**char** infix[SIZE], postfix[SIZE]; /\* declare infix string and postfix string \*/

/\* why we asked the user to enter infix expression

\* in parentheses ( )

\* What changes are required in porgram to

\* get rid of this restriction since it is not

\* in algorithm

\* \*/

printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");

printf("\nEnter Infix expression : ");

gets(infix);

InfixToPostfix(infix,postfix); /\* call to convert \*/

printf("Postfix Expression: ");

puts(postfix); /\* print postfix expression \*/

**return** 0;

}

//18

#include<stdio.h>

#include <string.h>

#include <ctype.h>

char operand\_stack[50][80], operator\_stack[50];

int top\_operator = -1, top\_operand = -1;

int push\_operator(char operator)

{

operator\_stack[++top\_operator] = operator;

}

int push\_operand(char \*opnd)

{

strcpy(operand\_stack[++top\_operand], opnd);

}

char pop\_operator()

{

return(operator\_stack[top\_operator--]);

}

char \*pop\_operand()

{

return(operand\_stack[top\_operand--]);

}

int empty(int t)

{

if(t == 0)

{

return 1;

}

else

{

return 0;

}

}

int main()

{

char prefix\_expression[50], ch, temporary\_string[50], operand\_a[50], operand\_b[50], operator[2];

int count = 0, k = 0, operand\_count = 0;

printf("\nEnter a Prefix Expression:\t");

scanf("%s", prefix\_expression);

while((ch = prefix\_expression[count++]) != '\0')

{

if(isalnum(ch))

{

temporary\_string[0] = ch;

temporary\_string[1]='\0';

push\_operand(temporary\_string);

operand\_count++;

if(operand\_count >= 2)

{

strcpy(operand\_b, pop\_operand());

strcpy(operand\_a, pop\_operand());

strcpy(temporary\_string, "(");

strcat(temporary\_string, operand\_a);

ch = pop\_operator();

operator[0] = ch;

operator[1] = '\0';

strcat(temporary\_string, operator);

strcat(temporary\_string, operand\_b);

strcat(temporary\_string, ")");

push\_operand(temporary\_string);

operand\_count = operand\_count - 1;

}

}

else

{

push\_operator(ch);

if(operand\_count == 1)

{

operand\_count = 0;

}

}

}

if(!empty(top\_operand))

{

strcpy(operand\_b, pop\_operand());

strcpy(operand\_a, pop\_operand());

strcpy(temporary\_string, "(");

strcat(temporary\_string, operand\_a);

ch = pop\_operator();

operator[0] = ch;

operator[1] = '\0';

strcat(temporary\_string, operator);

strcat(temporary\_string, operand\_b);

strcat(temporary\_string, ")");

push\_operand(temporary\_string);

}

printf("\nInfix Expression:\t %s\n", operand\_stack[top\_operand]);

return 0;

}

//19

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

# define MAX 20

char str[MAX],stack[MAX];

int top=-1;

void push(char c)

{

stack[++top]=c;

}

char pop()

{

return stack[top--];

}

void pre\_post()

{

int n,i,j=0; char c[20];

char a,b,op;

printf("Enter the prefix expression\n");

gets(str);

n=strlen(str);

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

stack[i]='\0';

printf("Postfix expression is:\t");

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

{

if(str[i]=='+'||str[i]=='-'||str[i]=='\*'||str[i]=='/')

{

push(str[i]);

}

else

{ c[j++]=str[i];

while((top!=-1)&&(stack[top]=='@'))

{

a=pop(); c[j++]=pop();

}

push('@');

}

}

c[j]='\0';

printf("%s",c);

}

main()

{

pre\_post();

}

//20

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#define MAX 20

char str[MAX], stack[MAX];

int top = -1;

char pop()

{

return stack[top--];

}

void push(char ch)

{

stack[++top] = ch;

}

void postfix\_to\_prefix(char expression[])

{

int count, length;

length = strlen(expression);

printf("\nPrefix Expression:\t");

for(count = length - 1; count >= 0; count--)

{

printf("%c", expression[count]);

}

}

int main()

{

char postfix\_expression[35];

printf("\nEnter Postfix Expression:\t");

scanf("%s", postfix\_expression);

postfix\_to\_prefix(postfix\_expression);

printf("\n");

return 0;

}

//21

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

# define MAX 20

char str[MAX], stack[MAX];

int top = -1;

char pop()

{

return stack[top--];

}

void push(char ch)

{

stack[++top] = ch;

}

void postfix\_to\_infix(char expression[])

{

int count, length;

char element, operator;

length = strlen(expression);

for(count = 0; count < MAX; count++)

{

stack[count] = 0;

}

printf("\nInfix Expression:\t");

printf("%c", expression[0]);

for(count = 1; count < length; count++)

{

if(expression[count] == '-' || expression[count] == '/' || expression[count] == '\*'|| expression[count] == '+')

{

element = pop();

operator = expression[count];

printf(" %c %c", operator, element);

}

else

{

push(expression[count]);

}

}

printf("%c", expression[top--]);

}

int main()

{

char postfix\_expression[50];

printf("\nEnter Postfix Expression:\t");

scanf("%s", postfix\_expression);

postfix\_to\_infix(postfix\_expression);

printf("\n");

return 0;

}

//22

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// A structure to represent a stack

struct Stack {

int top;

int maxSize;

// we are storing string in integer array, this will not give error

// as values will be stored in ASCII and returned in ASCII thus, returned as string again

int\* array;

};

struct Stack\* create(int max)

{

struct Stack\* stack = (struct Stack\*)malloc(sizeof(struct Stack));

stack->maxSize = max;

stack->top = -1;

stack->array = (int\*)malloc(stack->maxSize \* sizeof(int));

return stack;

}

// Checking with this function is stack is full or not

// Will return true is stack is full else false

//Stack is full when top is equal to the last index

int isFull(struct Stack\* stack)

{

if(stack->top == stack->maxSize - 1){

printf("Will not be able to push maxSize reached\n");

}

// Since array starts from 0, and maxSize starts from 1

return stack->top == stack->maxSize - 1;

}

// By definition the Stack is empty when top is equal to -1

// Will return true if top is -1

int isEmpty(struct Stack\* stack)

{

return stack->top == -1;

}

// Push function here, inserts value in stack and increments stack top by 1

void push(struct Stack\* stack, int item)

{

if (isFull(stack))

return;

stack->array[++stack->top] = item;

}

// Function to remove an item from stack. It decreases top by 1

int pop(struct Stack\* stack)

{

if (isEmpty(stack))

return INT\_MIN;

return stack->array[stack->top--];

}

// Function to return the top from stack without removing it

int peek(struct Stack\* stack)

{

if (isEmpty(stack))

return INT\_MIN;

return stack->array[stack->top];

}

// A utility function to check if the given character is operand

int checkIfOperand(char ch)

{

return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');

}

// Fucntion to compare precedence

// If we return larger value means higher precedence

int precedence(char ch)

{

switch (ch)

{

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

}

return -1;

}

// The driver function for infix to postfix conversion

int getPostfix(char\* expression)

{

int i, j;

// Stack size should be equal to expression size for safety

struct Stack\* stack = create(strlen(expression));

if(!stack) // just checking is stack was created or not

return -1 ;

for (i = 0, j = -1; expression[i]; ++i)

{

// Here we are checking is the character we scanned is operand or not

// and this adding to to output.

if (checkIfOperand(expression[i]))

expression[++j] = expression[i];

// Here, if we scan character ‘(‘, we need push it to the stack.

else if (expression[i] == '(')

push(stack, expression[i]);

// Here, if we scan character is an ‘)’, we need to pop and print from the stack

// do this until an ‘(‘ is encountered in the stack.

else if (expression[i] == ')')

{

while (!isEmpty(stack) && peek(stack) != '(')

expression[++j] = pop(stack);

if (!isEmpty(stack) && peek(stack) != '(')

return -1; // invalid expression

else

pop(stack);

}

else // if an opertor

{

while (!isEmpty(stack) && precedence(expression[i]) <= precedence(peek(stack)))

expression[++j] = pop(stack);

push(stack, expression[i]);

}

}

// Once all inital expression characters are traversed

// adding all left elements from stack to exp

while (!isEmpty(stack))

expression[++j] = pop(stack);

expression[++j] = '\0';

}

void reverse(char \*exp){

int size = strlen(exp);

int j = size, i=0;

char temp[size];

temp[j--]='\0';

while(exp[i]!='\0')

{

temp[j] = exp[i];

j--;

i++;

}

strcpy(exp,temp);

}

void brackets(char\* exp){

int i = 0;

while(exp[i]!='\0')

{

if(exp[i]=='(')

exp[i]=')';

else if(exp[i]==')')

exp[i]='(';

i++;

}

}

void InfixtoPrefix(char \*exp){

int size = strlen(exp);

// reverse string

reverse(exp);

//change brackets

brackets(exp);

//get postfix

getPostfix(exp);

// reverse string again

reverse(exp);

}

int main()

{

printf("The infix is: ");

char expression[] = "((a/b)+c)-(d+(e\*f))";

printf("%s\n",expression);

InfixtoPrefix(expression);

printf("The prefix is: ");

printf("%s\n",expression);

return 0;

}

//23

#include<stdio.h>

int stack[20];

int top = -1;

void push(int x)

{

stack[++top] = x;

}

int pop()

{

return stack[top--];

}

int main()

{

char exp[20];

char \*e;

int n1,n2,n3,num;

printf("Enter the expression :: ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isdigit(\*e))

{

num = \*e - 48;

push(num);

}

else

{

n1 = pop();

n2 = pop();

switch(\*e)

{

case '+':

{

n3 = n1 + n2;

break;

}

case '-':

{

n3 = n2 - n1;

break;

}

case '\*':

{

n3 = n1 \* n2;

break;

}

case '/':

{

n3 = n2 / n1;

break;

}

}

push(n3);

}

e++;

}

printf("\nThe result of expression %s = %d\n\n",exp,pop());

return 0;

}

//24// C program to sort a stack using recursion

#include <stdio.h>

#include <stdlib.h>

// Stack is represented using linked list

struct stack {

int data;

struct stack\* next;

};

// Utility function to initialize stack

void initStack(struct stack\*\* s) { \*s = NULL; }

// Utility function to chcek if stack is empty

int isEmpty(struct stack\* s)

{

if (s == NULL)

return 1;

return 0;

}

// Utility function to push an item to stack

void push(struct stack\*\* s, int x)

{

struct stack\* p = (struct stack\*)malloc(sizeof(\*p));

if (p == NULL) {

fprintf(stderr, "Memory allocation failed.\n");

return;

}

p->data = x;

p->next = \*s;

\*s = p;

}

// Utility function to remove an item from stack

int pop(struct stack\*\* s)

{

int x;

struct stack\* temp;

x = (\*s)->data;

temp = \*s;

(\*s) = (\*s)->next;

free(temp);

return x;

}

// Function to find top item

int top(struct stack\* s) { return (s->data); }

// Recursive function to insert an item x in sorted way

void sortedInsert(struct stack\*\* s, int x)

{

// Base case: Either stack is empty or newly inserted

// item is greater than top (more than all existing)

if (isEmpty(\*s) || x > top(\*s)) {

push(s, x);

return;

}

// If top is greater, remove the top item and recur

int temp = pop(s);

sortedInsert(s, x);

// Put back the top item removed earlier

push(s, temp);

}

// Function to sort stack

void sortStack(struct stack\*\* s)

{

// If stack is not empty

if (!isEmpty(\*s)) {

// Remove the top item

int x = pop(s);

// Sort remaining stack

sortStack(s);

// Push the top item back in sorted stack

sortedInsert(s, x);

}

}

// Utility function to print contents of stack

void printStack(struct stack\* s)

{

while (s) {

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

s = s->next;

}

printf("\n");

}

// Driver code

int main(void)

{

struct stack\* top;

initStack(&top);

push(&top, 30);

push(&top, -5);

push(&top, 18);

push(&top, 14);

push(&top, -3);

printf("Stack elements before sorting:\n");

printStack(top);

sortStack(&top);

printf("\n\n");

printf("Stack elements after sorting:\n");

printStack(top);

return 0;

}

//25

#include<stdio.h>

#define n 5

int main()

{

int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;

printf("Queue using Array");

printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");

while(ch)

{

printf("\nEnter the Choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

if(rear==x)

printf("\n Queue is Full");

else

{

printf("\n Enter no %d:",j++);

scanf("%d",&queue[rear++]);

}

break;

case 2:

if(front==rear)

{

printf("\n Queue is empty");

}

else

{

printf("\n Deleted Element is %d",queue[front++]);

x++;

}

break;

case 3:

printf("\nQueue Elements are:\n ");

if(front==rear)

printf("\n Queue is Empty");

else

{

for(i=front; i<rear; i++)

{

printf("%d",queue[i]);

printf("\n");

}

break;

case 4:

exit(0);

default:

printf("Wrong Choice: please see the options");

}

}

}

return 0;

}

//26

/\*

\* C Program to Implement Queue Data Structure using Linked List

\*/

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*front,\*rear,\*temp,\*front1;

int frontelement();

void enq(int data);

void deq();

void empty();

void display();

void create();

void queuesize();

int count = 0;

void main()

{

int no, ch, e;

printf("\n 1 - Enque");

printf("\n 2 - Deque");

printf("\n 3 - Front element");

printf("\n 4 - Empty");

printf("\n 5 - Exit");

printf("\n 6 - Display");

printf("\n 7 - Queue size");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

enq(no);

break;

case 2:

deq();

break;

case 3:

e = frontelement();

if (e != 0)

printf("Front element : %d", e);

else

printf("\n No front element in Queue as queue is empty");

break;

case 4:

empty();

break;

case 5:

exit(0);

case 6:

display();

break;

case 7:

queuesize();

break;

default:

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

/\* Create an empty queue \*/

void create()

{

front = rear = NULL;

}

/\* Returns queue size \*/

void queuesize()

{

printf("\n Queue size : %d", count);

}

/\* Enqueing the queue \*/

void enq(int data)

{

if (rear == NULL)

{

rear = (struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = NULL;

rear->info = data;

front = rear;

}

else

{

temp=(struct node \*)malloc(1\*sizeof(struct node));

rear->ptr = temp;

temp->info = data;

temp->ptr = NULL;

rear = temp;

}

count++;

}

/\* Displaying the queue elements \*/

void display()

{

front1 = front;

if ((front1 == NULL) && (rear == NULL))

{

printf("Queue is empty");

return;

}

while (front1 != rear)

{

printf("%d ", front1->info);

front1 = front1->ptr;

}

if (front1 == rear)

printf("%d", front1->info);

}

/\* Dequeing the queue \*/

void deq()

{

front1 = front;

if (front1 == NULL)

{

printf("\n Error: Trying to display elements from empty queue");

return;

}

else

if (front1->ptr != NULL)

{

front1 = front1->ptr;

printf("\n Dequed value : %d", front->info);

free(front);

front = front1;

}

else

{

printf("\n Dequed value : %d", front->info);

free(front);

front = NULL;

rear = NULL;

}

count--;

}

/\* Returns the front element of queue \*/

int frontelement()

{

if ((front != NULL) && (rear != NULL))

return(front->info);

else

return 0;

}

/\* Display if queue is empty or not \*/

void empty()

{

if ((front == NULL) && (rear == NULL))

printf("\n Queue empty");

else

printf("Queue not empty");

}

// 26

//27

/\*

\* C program to input N numbers and store them in an array.

\* Do a linear search for a given key and report success

\* or failure.

\*/

#include <stdio.h>

void main()

{ int num;

int i, keynum, found = 0;

printf("Enter the number of elements ");

scanf("%d", &num);

int array[num];

printf("Enter the elements one by one \n");

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

{

scanf("%d", &array[i]);

}

printf("Enter the element to be searched ");

scanf("%d", &keynum);

/\* Linear search begins \*/

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

{

if (keynum == array[i] )

{

found = 1;

break;

}

}

if (found == 1)

printf("Element is present in the array at position %d",i+1);

else

printf("Element is not present in the array\n");

}

//28

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

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

scanf("%d", &array[c]);

printf("Enter value to find\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

if (first > last)

printf("Not found! %d isn't present in the list.\n", search);

return 0;

}

//29

/\* Implementing Bubble sort in a C Program

\* Written by: Chaitanya.

\*/

#include<stdio.h>

int main(){

int count, temp, i, j, number[30];

printf("How many numbers are u going to enter?: ");

scanf("%d",&count);

printf("Enter %d numbers: ",count);

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

scanf("%d",&number[i]);

/\* This is the main logic of bubble sort algorithm

\*/

for(i=count-2;i>=0;i--){

for(j=0;j<=i;j++){

if(number[j]>number[j+1]){

temp=number[j];

number[j]=number[j+1];

number[j+1]=temp;

}

}

}

printf("Sorted elements: ");

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

printf(" %d",number[i]);

return 0;

}

//30

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

#define SIZE 20

struct DataItem {

int data;

int key;

};

struct DataItem\* hashArray[SIZE];

struct DataItem\* dummyItem;

struct DataItem\* item;

int hashCode(int key) {

return key % SIZE;

}

struct DataItem \*search(int key) {

//get the hash

int hashIndex = hashCode(key);

//move in array until an empty

while(hashArray[hashIndex] != NULL) {

if(hashArray[hashIndex]->key == key)

return hashArray[hashIndex];

//go to next cell

++hashIndex;

//wrap around the table

hashIndex %= SIZE;

}

return NULL;

}

void insert(int key,int data) {

struct DataItem \*item = (struct DataItem\*) malloc(sizeof(struct DataItem));

item->data = data;

item->key = key;

//get the hash

int hashIndex = hashCode(key);

//move in array until an empty or deleted cell

while(hashArray[hashIndex] != NULL && hashArray[hashIndex]->key != -1) {

//go to next cell

++hashIndex;

//wrap around the table

hashIndex %= SIZE;

}

hashArray[hashIndex] = item;

}

struct DataItem\* delete(struct DataItem\* item) {

int key = item->key;

//get the hash

int hashIndex = hashCode(key);

//move in array until an empty

while(hashArray[hashIndex] != NULL) {

if(hashArray[hashIndex]->key == key) {

struct DataItem\* temp = hashArray[hashIndex];

//assign a dummy item at deleted position

hashArray[hashIndex] = dummyItem;

return temp;

}

//go to next cell

++hashIndex;

//wrap around the table

hashIndex %= SIZE;

}

return NULL;

}

void display() {

int i = 0;

for(i = 0; i<SIZE; i++) {

if(hashArray[i] != NULL)

printf(" (%d,%d)",hashArray[i]->key,hashArray[i]->data);

else

printf(" ~~ ");

}

printf("\n");

}

int main() {

dummyItem = (struct DataItem\*) malloc(sizeof(struct DataItem));

dummyItem->data = -1;

dummyItem->key = -1;

insert(1, 20);

insert(2, 70);

insert(42, 80);

insert(4, 25);

insert(12, 44);

insert(14, 32);

insert(17, 11);

insert(13, 78);

insert(37, 97);

display();

item = search(37);

if(item != NULL) {

printf("Element found: %d\n", item->data);

} else {

printf("Element not found\n");

}

delete(item);

item = search(37);

if(item != NULL) {

printf("Element found: %d\n", item->data);

} else {

printf("Element not found\n");

}

}

//31

/\*

\* C Program to Construct a Binary Search Tree and perform deletion, inorder traversal on it

\*/

#include <stdio.h>

#include <stdlib.h>

struct btnode

{

int value;

struct btnode \*l;

struct btnode \*r;

}\*root = NULL, \*temp = NULL, \*t2, \*t1;

void delete1();

void insert();

void delete();

void inorder(struct btnode \*t);

void create();

void search(struct btnode \*t);

void preorder(struct btnode \*t);

void postorder(struct btnode \*t);

void search1(struct btnode \*t,int data);

int smallest(struct btnode \*t);

int largest(struct btnode \*t);

int flag = 1;

void main()

{

int ch;

printf("\nOPERATIONS ---");

printf("\n1 - Insert an element into tree\n");

printf("2 - Delete an element from the tree\n");

printf("3 - Inorder Traversal\n");

printf("4 - Preorder Traversal\n");

printf("5 - Postorder Traversal\n");

printf("6 - Exit\n");

while(1)

{

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

inorder(root);

break;

case 4:

preorder(root);

break;

case 5:

postorder(root);

break;

case 6:

exit(0);

default :

printf("Wrong choice, Please enter correct choice ");

break;

}

}

}

/\* To insert a node in the tree \*/

void insert()

{

create();

if (root == NULL)

root = temp;

else

search(root);

}

/\* To create a node \*/

void create()

{

int data;

printf("Enter data of node to be inserted : ");

scanf("%d", &data);

temp = (struct btnode \*)malloc(1\*sizeof(struct btnode));

temp->value = data;

temp->l = temp->r = NULL;

}

/\* Function to search the appropriate position to insert the new node \*/

void search(struct btnode \*t)

{

if ((temp->value > t->value) && (t->r != NULL)) /\* value more than root node value insert at right \*/

search(t->r);

else if ((temp->value > t->value) && (t->r == NULL))

t->r = temp;

else if ((temp->value < t->value) && (t->l != NULL)) /\* value less than root node value insert at left \*/

search(t->l);

else if ((temp->value < t->value) && (t->l == NULL))

t->l = temp;

}

/\* recursive function to perform inorder traversal of tree \*/

void inorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display");

return;

}

if (t->l != NULL)

inorder(t->l);

printf("%d -> ", t->value);

if (t->r != NULL)

inorder(t->r);

}

/\* To check for the deleted node \*/

void delete()

{

int data;

if (root == NULL)

{

printf("No elements in a tree to delete");

return;

}

printf("Enter the data to be deleted : ");

scanf("%d", &data);

t1 = root;

t2 = root;

search1(root, data);

}

/\* To find the preorder traversal \*/

void preorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display");

return;

}

printf("%d -> ", t->value);

if (t->l != NULL)

preorder(t->l);

if (t->r != NULL)

preorder(t->r);

}

/\* To find the postorder traversal \*/

void postorder(struct btnode \*t)

{

if (root == NULL)

{

printf("No elements in a tree to display ");

return;

}

if (t->l != NULL)

postorder(t->l);

if (t->r != NULL)

postorder(t->r);

printf("%d -> ", t->value);

}

/\* Search for the appropriate position to insert the new node \*/

void search1(struct btnode \*t, int data)

{

if ((data>t->value))

{

t1 = t;

search1(t->r, data);

}

else if ((data < t->value))

{

t1 = t;

search1(t->l, data);

}

else if ((data==t->value))

{

delete1(t);

}

}

/\* To delete a node \*/

void delete1(struct btnode \*t)

{

int k;

/\* To delete leaf node \*/

if ((t->l == NULL) && (t->r == NULL))

{

if (t1->l == t)

{

t1->l = NULL;

}

else

{

t1->r = NULL;

}

t = NULL;

free(t);

return;

}

/\* To delete node having one left hand child \*/

else if ((t->r == NULL))

{

if (t1 == t)

{

root = t->l;

t1 = root;

}

else if (t1->l == t)

{

t1->l = t->l;

}

else

{

t1->r = t->l;

}

t = NULL;

free(t);

return;

}

/\* To delete node having right hand child \*/

else if (t->l == NULL)

{

if (t1 == t)

{

root = t->r;

t1 = root;

}

else if (t1->r == t)

t1->r = t->r;

else

t1->l = t->r;

t == NULL;

free(t);

return;

}

/\* To delete node having two child \*/

else if ((t->l != NULL) && (t->r != NULL))

{

t2 = root;

if (t->r != NULL)

{

k = smallest(t->r);

flag = 1;

}

else

{

k =largest(t->l);

flag = 2;

}

search1(root, k);

t->value = k;

}

}

/\* To find the smallest element in the right sub tree \*/

int smallest(struct btnode \*t)

{

t2 = t;

if (t->l != NULL)

{

t2 = t;

return(smallest(t->l));

}

else

return (t->value);

}

/\* To find the largest element in the left sub tree \*/

int largest(struct btnode \*t)

{

if (t->r != NULL)

{

t2 = t;

return(largest(t->r));

}

else

return(t->value);

}