/\* C Program to Implement Singly Linked List & perform Create, Read, Update, Delete operations. \*/

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

// Declaring Structure for node …

typedef struct linked\_list

{

int data;

struct linked\_list \*link;

} node;

// Declaring structure Variables …

node \*first = NULL, \*current, \*temp, \*previous;

int op[4];

// Function containing Various Error Conditions …

void error(int x) {

if (x == 1) printf("LINKED LIST UNDERFLOW / NO ELEMENTS !\n");

if (x == 2) printf("LINKED LIST OVERFLOW / NOT ENOUGH MEMORY !\n");

if (x == 3) printf("LINKED LIST EMPTY !\n");

system("pause");

}

// Function to create a new Linked-List …

void create() {

int cycle = 1; char ch;

do {

if (cycle) first = NULL;

if (first == NULL) {

current = (node \*)malloc(sizeof(node));

if (current == NULL) error(2);

else {

cycle = 0;

printf("Enter Data for First Node: ");

scanf("%d", &current->data);

first = current;

}

} else {

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

if (temp == NULL) error(2);

else {

printf("Enter Data for Next Node: ");

scanf("%d", &temp->data);

current->link = temp;

current = temp;

}

}

printf("Creating Next Node …\nPress any key to confirm or .(Dot) to Quit: ");

ch = getch();

printf("%c\n", ch);

} while (ch != '.');

current->link = NULL;

temp = NULL;

free(temp);

printf("List created !\n");

system("pause");

}

// Function to insert a new element at First position in Linked-List …

void insert\_first() {

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

if (temp == NULL) error(2);

else {

printf("Enter Data for New First Node: ");

scanf("%d", &temp->data);

temp->link = first;

first = temp;

temp = NULL;

free(temp);

printf("Insertion Complete !\n");

system("pause");

}

}

// Function to insert a new element at Last position in Linked-List …

void insert\_last() {

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

if (temp == NULL) error(2);

else {

printf("Enter Data for New Last Node: ");

scanf("%d", &temp->data);

current = first;

while (current->link != NULL)

current = current->link;

current->link = temp;

temp->link = NULL;

current = temp;

temp = NULL;

free(temp);

printf("Insertion Complete !\n");

system("pause");

}

}

// Function to insert new element at user defined position in Linked-List …

void insert\_user() {

if (op[0] == 1) {

if (op[1] < 1) {

printf("Node value can't be Negative! Try Again :)\n");

system("pause");

} else if (op[1] > op[2] + 1) {

printf("Value must be Sequential ! Try Again :)\n");

system("pause");

} else if (op[1] == 1) {

insert\_first();

} else if (op[1] == op[2] + 1) {

insert\_last();

} else {

current = (node \*)malloc(sizeof(node));

if (current == NULL) error(2);

else {

int count = 2;

printf("Enter Data for New Node: ");

scanf("%d", &current->data);

temp = first;

while (count <= op[1]) {

count++;

previous = temp;

temp = temp->link;

}

previous->link = current;

current->link = temp;

printf("Insertion Complete !\n");

system("pause");

}

}

} else {

current = (node \*)malloc(sizeof(node));

if (current == NULL) error(2);

else {

int count = 1, not\_found = 0;

printf("Data will be inserted after the selected data info …\n");

printf("Enter Data for New Node: ");

scanf("%d", &current->data);

temp = first;

while (temp->data <= op[1]) {

previous = temp;

temp = temp->link;

count++;

if (count > op[2]) {

not\_found == 1;

break;

}

} if (not\_found == 1) {

printf("Element Not Found ! Try Again :)\n");

system("pause");

} else {

current->link = temp->link;

temp->link = current;

printf("Insertion Complete !\n");

system("pause");

}

}

}

}

// Function to insert new element in Sorted Linked-List …

void insert\_sorted() {

current = (node \*)malloc(sizeof(node));

if (current == NULL) error(2);

else {

int no\_op = 1;

printf("Treating data Ascendingly...\n");

printf("Enter Data for New Node: ");

scanf("%d", &current->data);

temp = first->link;

if (first->data > current->data) {

current->link = first;

first = current;

current = NULL;

} else if (current->data > op[3]) {

while (temp->link != NULL)

temp = temp->link;

temp->link = current;

current->link = NULL;

} else {

temp = first;

while (temp->data <= current->data) {

previous = temp;

temp = temp->link;

if (temp->data < previous->data) {

printf("Can't Input Data ! Array might not be Sorted !\n");

system("pause");

no\_op = 0;

break;

}

} if (no\_op) {

previous->link = current;

current->link = temp;

}

} if (no\_op) {

printf("Insertion Complete !\n");

system("pause");

}

}

}

// Function to delete the First element of Linked-List …

void delete\_first() {

temp = first;

first = first->link;

free(temp);

printf("Deletion Complete !\n");

system("pause");

}

// Function to delete the Last element of Linked-List …

void delete\_last() {

temp = first;

while (temp->link != NULL) {

current = temp;

temp = temp->link;

}

current->link = NULL;

free(temp);

printf("Deletion Complete !\n");

system("pause");

}

// Function to delete element from user defined position in Linked-List …

void delete\_user(){

if (op[0] == 1) {

if (op[1] < 1 || op[1] > op[2]) {

printf("Element Not Found ! Try Again :)\n");

system("pause");

} else if (op[1] == 1) {

delete\_first();

} else if (op[1] == op[2]) {

delete\_last();

} else {

int count = 2;

temp = first;

while (count <= op[1]) {

count++;

current = temp;

temp = temp->link;

}

current->link = temp->link;

free(temp);

printf("Deletion Complete !\n");

system("pause");

}

} else {

int count = 1, no\_op = 0;

if (op[1] == first->data)

delete\_first();

else {

temp = first;

while (count <= op[2]) {

if (temp->data == op[1]) break;

current = temp;

temp = temp->link;

count++;

if (count > op[2]) {

no\_op = 1;

break;

}

} if (no\_op) {

printf("Element Not Found ! Try Again :)\n");

system("pause");

} else {

current->link = temp->link;

free(temp);

printf("Deletion Complete !\n");

system("pause");

}

}

}

}

// Function to traverse Doubly-Linked-List …

void lookup() {

int i = 0;

temp = first;

while (temp != NULL) {

op[3] = temp->data;

printf("Information stored in Node %d: %d\n", ++i, op[3]);

temp = temp->link;

}

op[2] = i;

free(temp);

system("pause");

}

// Function to ask user to select the Node & Data to perform operation over …

void ask\_user() {

do {

printf("1: Using Node Info.\n");

printf("2: Using Data Info.\n");

printf("Select your Option: ");

scanf("%d", &op[0]);

switch (op[0]) {

case 1:

printf("Enter Node info : ");

scanf("%d", &op[1]);

break;

case 2:

printf("Enter Data info: ");

scanf("%d", &op[1]);

break;

default:

system("cls");

printf("Wrong Selection !!! Select Again !!!\n");

system("pause");

system("cls");

lookup(); break;

}

} while (op[0] > 2 || op[2] < 1);

}

// Main Function which initiates all other functions …

int main(int argc, char const \*argv[]) {

int sel, exit = 1;

do {

system("cls");

printf("SINGLY LINKED LIST DEMONSTRATION ~\n");

printf("1: Create a new Linked List.\n");

printf("2: Insert new node at First Position.\n");

printf("3: Insert new node at End of Linked List.\n");

printf("4: Insert new node at Desired Position.\n");

printf("5: Insert new node in Sorted Linked List.\n");

printf("6: Delete First Node.\n");

printf("7: Delete Last Node.\n");

printf("8: Delete Desired Node.\n");

printf("9: Lookup Linked List as in Current Position.\n");

printf("10: EXIT MENU.\n");

printf("What you wanna do ? Select your option & press Enter: ");

scanf("%d", &sel);

switch (sel) {

case 1:

system("cls");

create();

system("cls"); break;

case 2:

system("cls");

insert\_first();

system("cls"); break;

case 3:

system("cls");

insert\_last();

system("cls"); break;

case 4:

system("cls");

lookup(); ask\_user(); insert\_user();

system("cls"); break;

case 5:

system("cls");

insert\_sorted();

system("cls"); break;

case 6:

system("cls");

if (first == NULL) error(1);

else delete\_first();

system("cls"); break;

case 7:

system("cls");

if (first == NULL) error(1);

else delete\_last();

system("cls"); break;

case 8:

system("cls");

if (first == NULL) error(1);

else {

lookup();

ask\_user();

delete\_user();

}

system("cls"); break;

case 9:

system("cls");

if (first == NULL) error(3);

else lookup();

system("cls"); break;

case 10:

system("pause");

exit = 0; break;

default:

system("cls");

printf("Wrong Selection !!! Select Again !!!\n\n");

system("pause"); break;

}

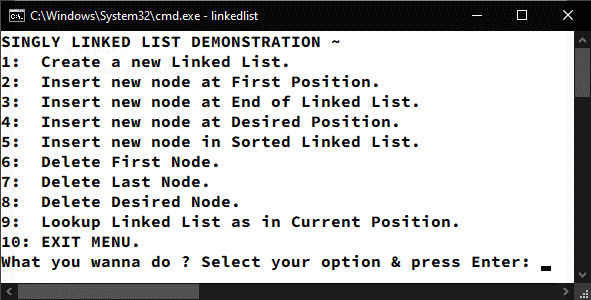
} while (exit);

return 0;

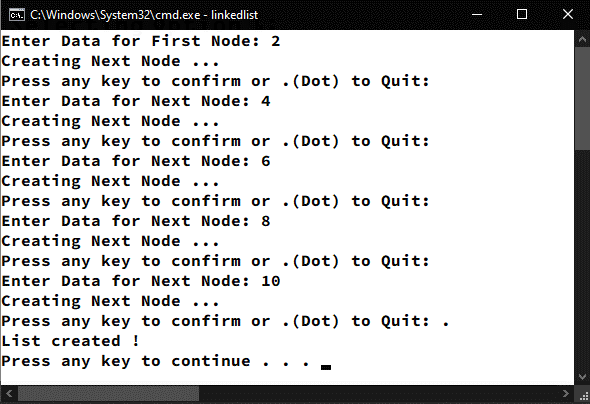
}

**Output:**

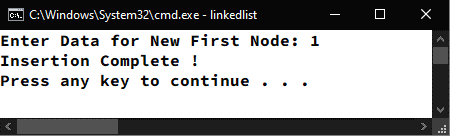
**Main Menu:**



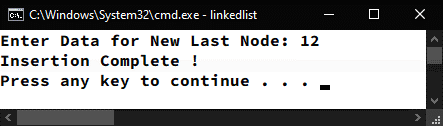
**Selecting Option 1:**



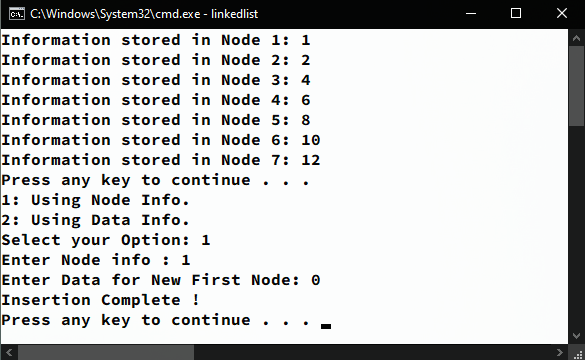
**Selecting Option 2:**

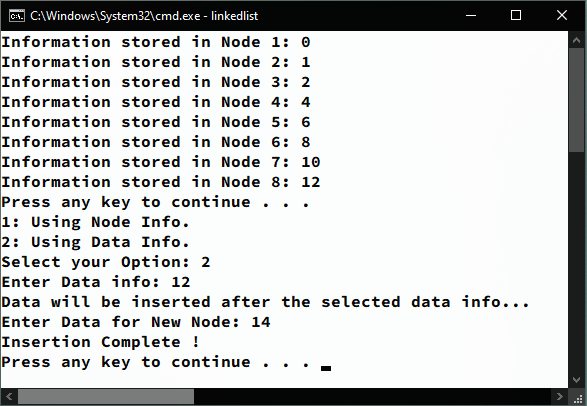


**Selecting Option 3:**

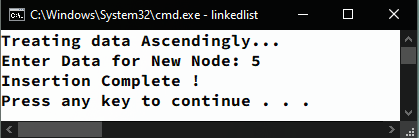


**Selecting Option 4:**

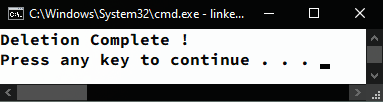




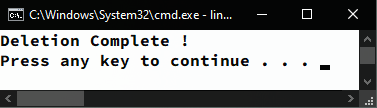
**Selecting Option 5:**



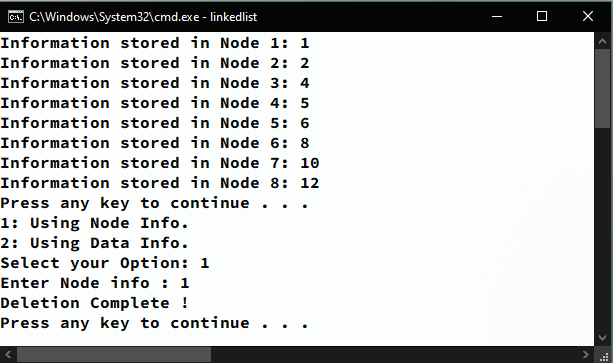
**Selecting Option 6:**

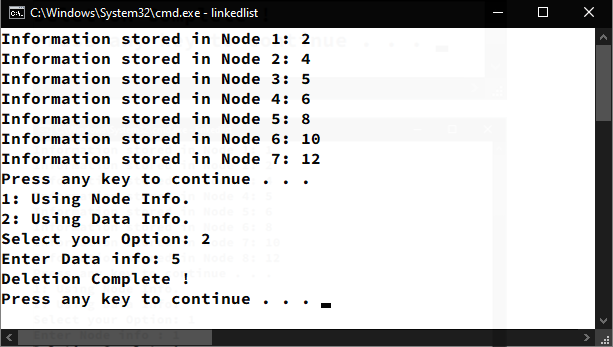


**Selecting Option 7:**

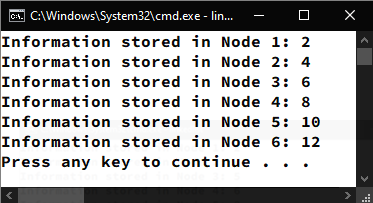


**Selecting Option 8:**

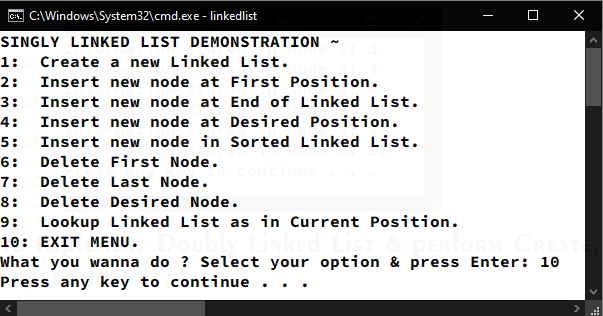




**Selecting Option 9:**



**Selecting Option 10:**



/\* C Program to Implement Doubly Linked List & perform Create, Read, Update, Delete operations. \*/

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

// Declaring Structure for node …

typedef struct linked\_list{

int data;

struct linked\_list \*next;

struct linked\_list \*previous;

} node;

// Declaring structure Variables …

node \*first = NULL, \*current, \*temp, \*tail;

int op[4];

// Function containing Various Error Conditions …

void error(int x) {

if (x == 1) **printf**("LINKED LIST UNDERFLOW / NO ELEMENTS !\n");

if (x == 2) **printf**("LINKED LIST OVERFLOW / NOT ENOUGH MEMORY !\n");

if (x == 3) **printf**("LINKED LIST EMPTY !\n");

system("pause");

}

// Function to create a new Doubly-Linked-List …

void create(){

int cycle = 1; char ch;

do {

if (cycle) first = NULL;

if (first == NULL) {

current = (node \*)**malloc**(sizeof(node));

if (current == NULL) error(2);

else {

cycle = 0;

**printf**("Enter Data for First Node: ");

**scanf**("%d", &current->data);

first = current;

first->previous = NULL;

}

} else {

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

if (temp == NULL) error(2);

else {

**printf**("Enter Data for Next Node: ");

**scanf**("%d", &temp->data);

temp->previous = current;

current->next = temp;

current = temp;

}

}

**printf**("Creating Next Node …\nPress any key to confirm or .(Dot) to Quit: ");

ch = getch();

**printf**("%c\n", ch);

} **while** (ch != '.');

current->next = NULL;

temp = NULL;

free(temp);

**printf**("List created !\n");

system("pause");

}

// Function to insert a new element at First position in Linked-List …

void insert\_first() {

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

if (temp == NULL) error(2);

else {

**printf**("Enter Data for New First Node: ");

**scanf**("%d", &temp->data);

temp->previous = NULL;

temp->next = first;

first = temp;

temp = NULL;

free(temp);

**printf**("Insertion Complete !\n");

system("pause");

}

}

// Function to insert a new element at Last position in Linked-List …

void insert\_last(){

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

if (temp == NULL) error(2);

else {

**printf**("Enter Data for New Last Node: ");

**scanf**("%d", &temp->data);

current = first;

**while** (current->next != NULL)

current = current->next;

current->next = temp;

temp->previous = current;

temp->next = NULL;

current = temp;

temp = NULL;

free(temp);

**printf**("Insertion Complete !\n");

system("pause");

}

}

// Function to insert new element at user defined position in Linked-List …

void insert\_user() {

if (op[0] == 1) {

if (op[1] < 1){

**printf**("Node value can't be Negative! Try Again :)\n");

system("pause");

} else if (op[1] > op[2] + 1) {

**printf**("Value must be Sequential ! Try Again :)\n");

system("pause");

} else if (op[1] == 1) {

insert\_first();

} else if (op[1] == op[2] + 1) {

insert\_last();

} else {

current = (node \*)**malloc**(sizeof(node));

if (current == NULL) error(2);

else {

int count = 2;

**printf**("Enter Data for New Node: ");

**scanf**("%d", &current->data);

temp = first;

**while** (count <= op[1]) {

count++;

tail = temp;

temp = temp->next;

}

tail->next = current;

current->next = temp;

current->previous = tail;

temp->previous = current;

**printf**("Insertion Complete !\n");

system("pause");

}

}

} else {

current = (node \*)**malloc**(sizeof(node));

if (current == NULL) error(2);

else {

int count = 1, not\_found = 0;

**printf**("Data will be inserted after the selected data info …\n");

**printf**("Enter Data for New Node: ");

**scanf**("%d", &current->data);

temp = first;

**while** (temp->data != op[1]) {

temp = temp->next;

count++;

if (count > op[2]) {

not\_found = 1;

**break**;

} if (not\_found == 1) {

**printf**("Element Not Found ! Try Again :)\n");

system("pause");

} else {

temp->next->previous = current;

current->next = temp->next;

current->previous = temp;

temp->next = current;

**printf**("Insertion Complete !\n");

system("pause");

}

}

}

}

// Function to insert new element in Sorted Linked-List …

void insert\_sorted() {

current = (node \*)**malloc**(sizeof(node));

if (current == NULL) error(2);

else {

int no\_op = 1;

**printf**("Treating data Ascendingly...\n");

**printf**("Enter Data for New Node: ");

**scanf**("%d", &current->data);

temp = first->next;

if (first->data > current->data) {

current->next = first;

first->previous = current;

current->previous = NULL;

first = current;

} else if (current->data > op[3]) {

**while** (temp->next != NULL)

temp = temp->next;

current->previous = temp;

temp->next = current;

current->next = NULL;

} else {

**while** (temp->data <= current->data) {

tail = temp;

temp = temp->next;

if (temp->data < tail->data) {

**printf**("Can't Input Data ! Array might not be Sorted !\n");

system("pause");

no\_op = 0;

**break**;

}

} if (no\_op) {

tail->next = current;

current->next = temp;

current->previous = tail;

temp->previous = current;

}

} if (no\_op) {

**printf**("Insertion Complete !\n");

system("pause");

}

}

}

// Function to delete the First element of Linked-List …

void delete\_first() {

temp = first;

first = first->next;

first->previous = NULL;

free(temp);

**printf**("Deletion Complete !\n");

system("pause");

}

// Function to delete the Last element of Linked-List …

void delete\_last() {

temp = first;

**while** (temp->next != NULL) {

current = temp;

temp = temp->next;

}

current->next = NULL;

free(temp);

**printf**("Deletion Complete !\n");

system("pause");

}

// Function to delete element from user defined position in Linked-List …

void delete\_user() {

if (op[0] == 1) {

if (op[1] < 1 || op[1] > op[2]) {

**printf**("Element Not Found ! Try Again :)\n");

system("pause");

} else if (op[1] == 1) {

delete\_first();

} else if (op[1] == op[2]) {

delete\_last();

} else {

int count = 2;

temp = first;

**while** (count <= op[1]) {

count++;

current = temp;

temp = temp->next;

}

current->next = temp->next;

free(temp);

**printf**("Deletion Complete !\n");

system("pause");

}

} else {

int count = 1, no\_op = 0;

if (op[1] == first->data) delete\_first();

else {

temp = first;

**while** (count <= op[2]) {

if (temp->data == op[1])

**break**;

current = temp;

temp = temp->next;

count++;

if (count > op[2]) {

no\_op = 1;

**break**;

}

} if (no\_op) {

**printf**("Element Not Found ! Try Again :)\n");

system("pause");

} else {

current->next = temp->next;

free(temp);

**printf**("Deletion Complete !\n");

system("pause");

}

}

}

}

// Function to traverse Doubly-Linked-List …

void lookup() {

int i = 0;

temp = first;

**while** (temp != NULL) {

op[3] = temp->data;

**printf**("Information stored in Node %d: %d\n", ++i, op[3]);

temp = temp->next;

}

op[2] = i;

system("pause");

}

// Function to reverse-traverse Doubly-Linked-List …

void lookup\_d() {

int i = 0;

temp = first;

**while** (temp != NULL) {

current = temp;

temp = temp->next;

}

**while** (current != NULL) {

**printf**("Information stored in Node %d: %d\n", ++i, current->data);

current = current->previous;

}

system("pause");

}

// Function to ask user to select the Node & Data to perform operation over …

void ask\_user() {

do {

**printf**("1: Using Node Info.\n");

**printf**("2: Using Data Info.\n");

**printf**("Select your Option: ");

**scanf**("%d", &op[0]);

**switch** (op[0]) {

**case** 1:

**printf**("Enter Node info : ");

**scanf**("%d", &op[1]);

**break**;

**case** 2:

**printf**("Enter Data info: ");

**scanf**("%d", &op[1]);

**break**;

default:

system("cls");

**printf**("Wrong Selection !!! Select Again !!!\n");

system("pause");

system("cls");

lookup(); **break**;

}

} **while** (op[0] > 2 || op[2] < 1);

}

// Main Function which initiates all other functions …

int main(int argc, char const \*argv[]) {

int sel, exit = 1;

do {

system("cls");

**printf**("DOUBLY LINKED LIST DEMONSTRATION ~\n");

**printf**("1: Create a new Linked List.\n");

**printf**("2: Insert new node at First Position.\n");

**printf**("3: Insert new node at End of Linked List.\n");

**printf**("4: Insert new node at Desired Position.\n");

**printf**("5: Insert new node in Sorted Linked List.\n");

**printf**("6: Delete First Node.\n");

**printf**("7: Delete Last Node.\n");

**printf**("8: Delete Desired Node.\n");

**printf**("9: Lookup Linked List Ascendigly.\n");

**printf**("10: Lookup Linked List Descendingly.\n");

**printf**("11: EXIT MENU.\n");

**printf**("What you wanna do ? Select your option & press Enter: ");

**scanf**("%d", &sel);

**switch** (sel) {

**case** 1:

system("cls");

create();

system("cls"); **break**;

**case** 2:

system("cls");

insert\_first();

system("cls"); **break**;

**case** 3:

system("cls");

insert\_last();

system("cls"); **break;**

**case** 4:

system("cls");

lookup(); ask\_user(); insert\_user();

system("cls"); **break**;

**case** 5:

system("cls");

insert\_sorted();

system("cls"); **break**;

**case** 6:

system("cls");

if (first == NULL) error(1);

else delete\_first();

system("cls"); **break**;

**case** 7:

system("cls");

if (first == NULL) error(1);

else delete\_last();

system("cls"); **break**;

**case** 8:

system("cls");

if (first == NULL) error(1);

else {

lookup();

ask\_user();

delete\_user();

}

system("cls"); **break**;

**case** 9:

system("cls");

if (first == NULL) error(3);

else lookup();

system("cls"); **break**;

**case** 10:

system("cls");

if (first == NULL) error(3);

else lookup\_d();

system("cls");

**break**;

**case** 11:

system("pause");

exit = 0;

**break**;

default:

system("cls");

**printf**("Wrong Selection !!! Select Again !!!\n\n");

system("pause"); **break**;

}

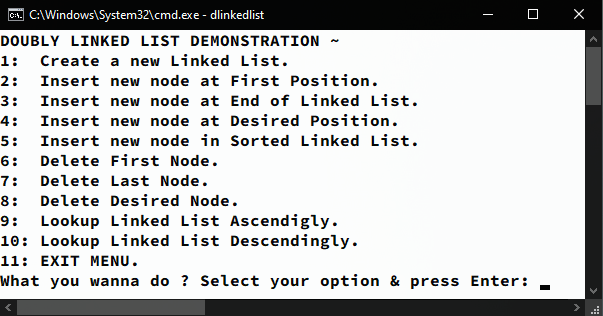
} **while** (exit);

return 0;

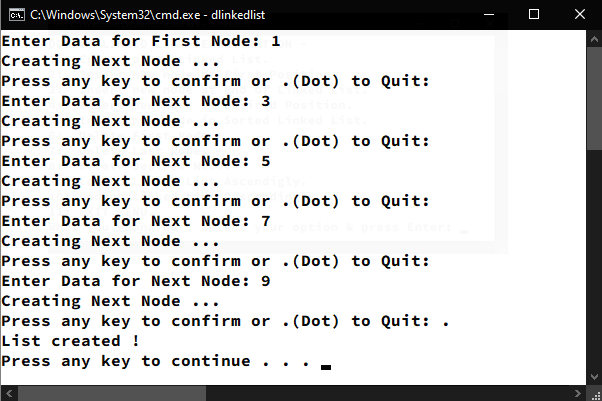
}

**Output:**

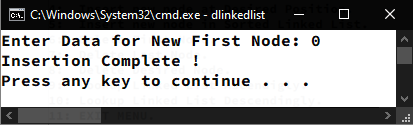
**Main Menu:**



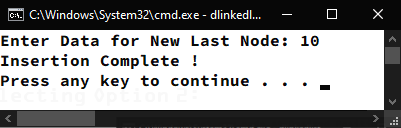
**Selecting Option 1:**



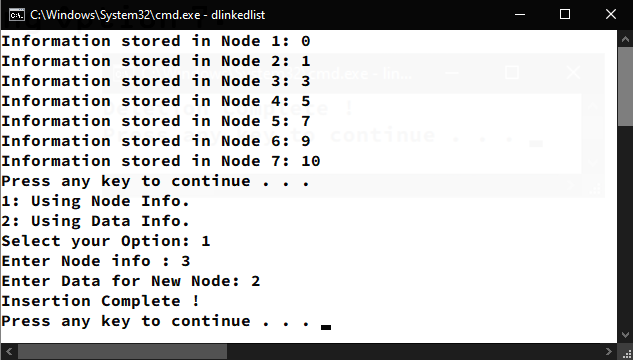
**Selecting Option 2:**

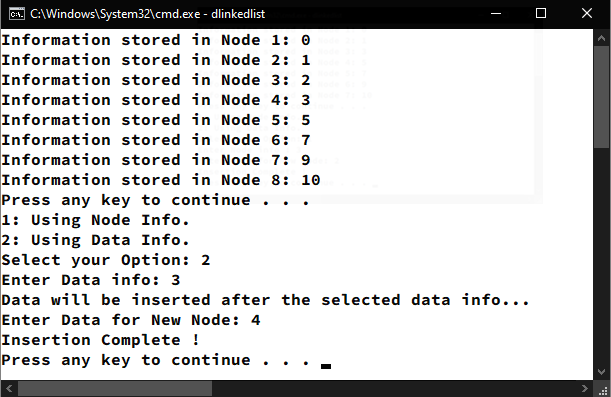


**Selecting Option 3:**

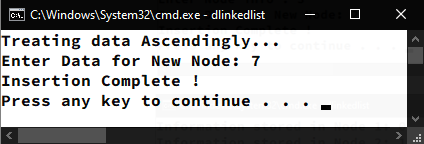


**Selecting Option 4:**

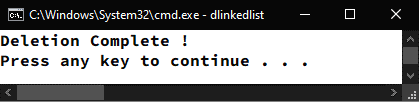




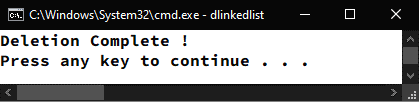
**Selecting Option 5:**



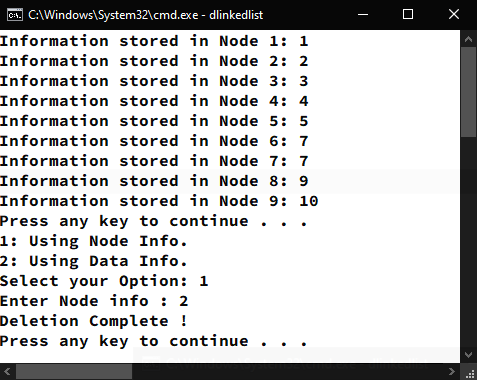
**Selecting Option 6:**

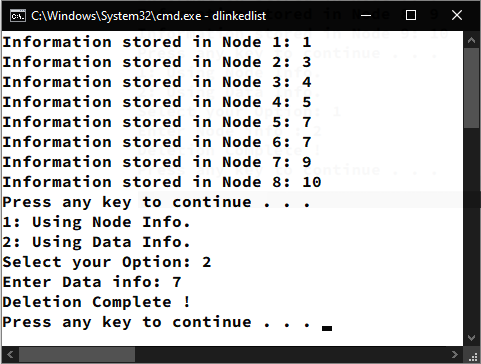


**Selecting Option 7:**

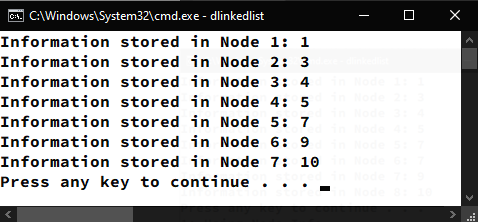


**Selecting Option 8:**

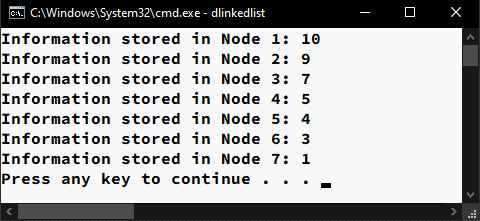




**Selecting Option 9:**



**Selecting Option 10:**



/\* Program to Implement Stack using Array & perform Create, Read, Update, Delete operations. \*/

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#define max 100

int top = 0;

int stack[max];

// Function to create New Stack …

void create() {

system("cls");

char ch;

do {

top++;

printf("Enter element %d of Stack: ", top);

scanf("%d", &stack[top]);

printf("Wanna Enter more ? Y/N\n");

ch = getch();

} while (ch != 'n' && ch != 'N');

printf("STACK CREATED !\n");

}

// Function to traverse Stack …

void traverse() {

system("cls");

for (int i = top; i > 0; i--) {

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

}

}

// Function to insert new element onto Stack …

void push() {

system("cls");

if (top == max) printf("\nSTACK OVERFLOW !!! HALT !!!");

else {

top++;

printf("Enter element to be Inserted at TOP: ");

scanf("%d", &stack[top]);

printf("ELEMENT INSERTED !\n");

}

}

// Function to remove element from Stack …

void pop() {

system("cls");

if (top == 0) printf("STACK UNDERFLOW !!! HALT !!!\n");

else {

stack[top] = '\0';

top = top - 1;

printf("ELEMENT DELETED !\n");

}

}

// Main Function which initiates all other functions …

int main(int argc, char const \*argv[]) {

int sel, exit = 1;

do {

system("cls");

printf("STACK USING ARRAY REPRESENTATION ~\n");

printf("1: Create Stack.\n");

printf("2: Read Stack.\n");

printf("3: Add Element.\n");

printf("4: Remove Element.\n");

printf("5: EXIT MENU.\n");

printf("Enter your Choice: ");

scanf("%d", &sel);

switch (sel) {

case 1:

create();

system("pause"); break;

case 2:

traverse();

system("pause"); break;

case 3:

push();

system("pause"); break;

case 4:

pop();

system("pause"); break;

case 5:

system("pause");

exit = 0; break;

default:

printf("WRONG SELECTION ! SELECT AGAIN !!!");

system("pause"); break;

}

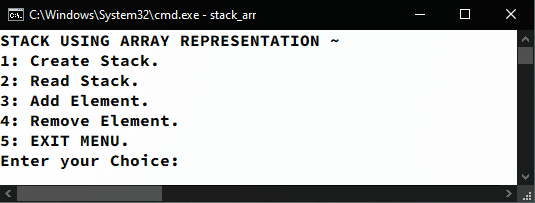
} while (exit);

return 0;

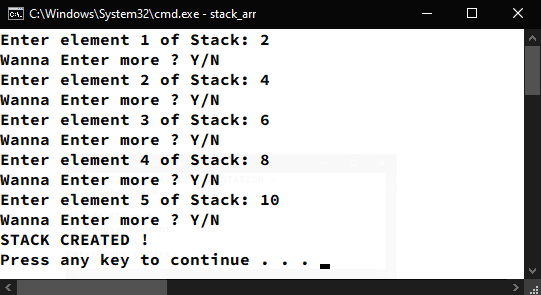
}

**Output:**

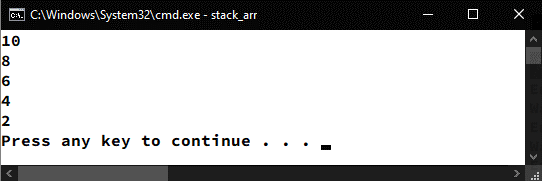
**Main Menu:**



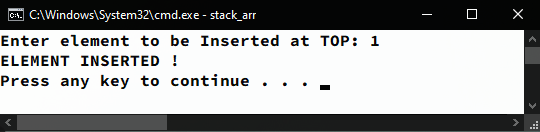
**Selecting Option 1:**



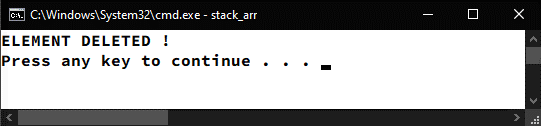
**Selecting Option 2:**



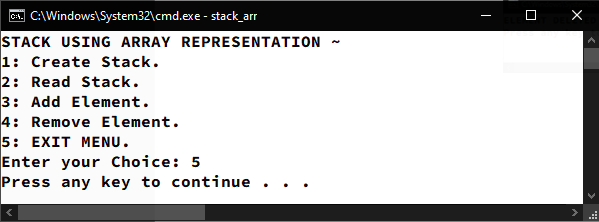
**Selecting Option 3:**



**Selecting Option 4:**



**Selecting Option 5:**



/\* Program to Implement Stack using Linked List & perform Create, Read, Update, Delete operations. \*/

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

// Declaring Structure for node …

typedef struct stack {

int data;

struct stack \*link;

} node;

// Declaring Structure Variable …

node \*top = NULL, \*current, \*temp;

// Function to Create New Stack …

void create() {

system("cls");

int cycle = 1; char ch;

do {

if (cycle) top = NULL;

if (top == NULL) {

cycle = 0;

current = (node \*)malloc(sizeof(node));

printf("Enter Data for Base: ");

scanf("%d", &current->data);

top = current;

top->link = NULL;

} else {

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

if (temp == NULL)

print("NOT ENOUGH MEMORY TO RECEIVE MORE DATA !\n");

else {

printf("Enter Next Data: ");

scanf("%d", &temp->data);

temp->link = current;

current = temp;

top = current;

}

}

printf("Creating Next Node …\nPress any key to confirm or .(Dot) to Quit: ");

ch = getch();

printf("%c\n", ch);

} while (ch != '.');

}

// Function to traverse Stack …

void traverse() {

system("cls");

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

temp = top;

int i = 0;

while (temp != NULL) {

printf("Data at Position %d: %d\n", ++i, temp->data);

temp = temp->link;

}

free(temp);

}

// Function to insert new element onto Stack …

void push() {

system("cls");

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

if (temp == NULL) printf("STACK OVERFLOW !\n");

else {

printf("Enter data for New TOP: ");

scanf("%d", &temp->data);

temp->link = top;

top = temp;

printf("ELEMENT INSERTED !\n");

}

}

// Function to remove element from Stack …

void pop() {

system("cls");

if (top == NULL) printf("STACK UNDERFLOW !\n");

else {

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

temp = top;

top = top->link;

free(temp);

printf("ELEMENT DELETED !\n");

}

}

// Main Function which initiates all other functions …

int main(int argc, char const \*argv[]) {

int sel, exit = 1;

do {

system("cls");

printf("STACK USING LINKED LIST REPRESENTATION ~\n");

printf("1: Create Stack.\n");

printf("2: Read Stack.\n");

printf("3: Add Element.\n");

printf("4: Remove Element.\n");

printf("5: EXIT MENU.\n");

printf("Enter your Choice: ");

scanf("%d", &sel);

switch (sel) {

case 1:

create();

system("pause"); break;

case 2:

traverse();

system("pause"); break;

case 3:

push();

system("pause"); break;

case 4:

pop();

system("pause"); break;

case 5:

system("pause");

exit = 0; break;

default:

printf("WRONG SELECTION ! SELECT AGAIN !!!");

system("pause"); break;

}

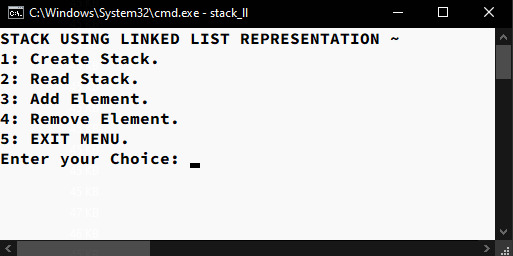
} while (exit);

return 0;

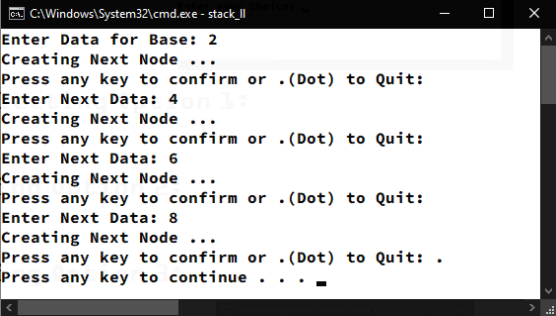
}

**Output:**

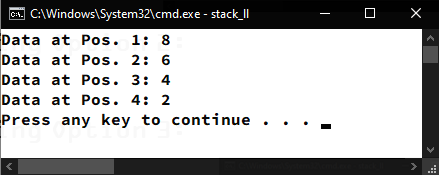
**Main Menu:**



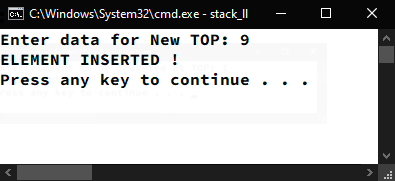
**Selecting Option 1:**



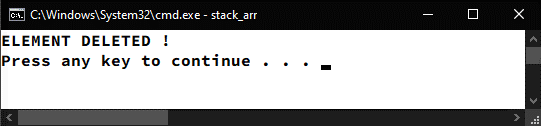
**Selecting Option 2:**



**Selecting Option 3:**



**Selecting Option 4:**



**Selecting Option 5:**

