//WAP creation of linked list

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*link;

};

int main()

{

struct node \*head=NULL;

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

head->data=45;

head->link=NULL;

cout<<head->data<<endl;

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

current->data=46;

current->link=NULL;

head->link=current;

cout<<current->data<<endl;

struct node \*current2=(struct node \*)malloc(sizeof(struct node));

current2->data=47;

current2->link=NULL;

current->link=current2;

cout<<current2->data;

return 0;

}

2.

#include<stdio.h>

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*link;

};

int main()

{

struct node \*head=NULL;

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

head->data=45;

head->link=NULL;

cout<<head->data<<endl;

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

current->data=46;

current->link=NULL;

head->link=current;

cout<<current->data<<endl;

//3rd node

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

current->data=467;

current->link=NULL;

cout<<current->data;

head->link->link=current;

return 0;

}

2) using new keyword same above program

#include<iostream>

using namespace std;

struct Node {

int data;

Node\* link;

};

int main() {

Node\* head = new Node;

head->data = 45;

head->link = nullptr;

cout << head->data << endl;

Node\* current = new Node;

current->data = 46;

current->link = nullptr;

head->link = current;

cout << current->data << endl;

// 3rd node

current = new Node;

current->data = 467;

current->link = nullptr;

cout << current->data << endl;

head->link->link = current;

return 0;

}

4) using class

#include<iostream>

using namespace std;

class Node {

public:

int data;

Node\* link;

Node(int value)

{

data=value;

link=nullptr;

}

};

int main() {

Node\* head = new Node(45);

cout << head->data << endl;

Node\* current = new Node(46);

head->link = current;

cout << current->data << endl;

// 3rd node

current = new Node(467);

cout << current->data << endl;

head->link->link = current;

return 0;

}

3) TRAVERSING OF SINGLY LINKED LIST

//#include<stdio.h>

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*link;

};

void count\_of\_nodes(struct node \*head)

{

int count=0;

if(head==NULL)

{

cout<<"linked list empty";

}

struct node \*ptr=NULL;

ptr=head;

while(ptr!=NULL)

{

count++;

ptr=ptr->link;

}

cout<<"total nodes are : "<<count;

}

int main()

{

// creation and insertion

struct node \*head=NULL;

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

head->data=45;

head->link=NULL;

cout<<head->data<<endl;

//node 2

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

current->data=46;

current->link=NULL;

head->link=current;

cout<<current->data<<endl;

//3rd node

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

current->data=467;

current->link=NULL;

cout<<current->data;

head->link->link=current;

// TRAVERSING

count\_of\_nodes(head);

return 0;

}

4)printing data

//#include<stdio.h>

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*link;

};

void count\_of\_nodes(struct node \*head)

{

int count=0;

if(head==NULL)

{

cout<<"linked list empty";

}

struct node \*ptr=NULL;

ptr=head;

while(ptr!=NULL)

{

cout<<"data is : "<<ptr->data<<endl;

count++;

ptr=ptr->link;

}

cout<<"total nodes are : "<<count;

}

int main()

{

// creation and insertion

struct node \*head=NULL;

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

head->data=45;

head->link=NULL;

cout<<head->data<<endl;

//node 2

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

current->data=46;

current->link=NULL;

head->link=current;

cout<<current->data<<endl;

//3rd node

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

current->data=467;

current->link=NULL;

cout<<current->data;

head->link->link=current;

// TRAVERSING

count\_of\_nodes(head);

return 0;

}

5) insert at end

//#include<stdio.h>

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*link;

};

void count\_of\_nodes(struct node \*head)

{

int count=0;

if(head==NULL)

{

cout<<"linked list empty";

}

struct node \*ptr=NULL;

ptr=head;

while(ptr!=NULL)

{

cout<<"data is : "<<ptr->data<<endl;

count++;

ptr=ptr->link;

}

cout<<"total nodes are : "<<count;

}

void insert\_end(struct node \*head,int data)

{

/\*if(head==NULL)

{

cout<<"linked list empty";

}\*/

struct node \*ptr, \*temp;

ptr=head;

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

temp->data=data;

temp->link=NULL;

while(ptr->link!=NULL)

{

ptr=ptr->link;

}

ptr->link=temp;

}

int main()

{

// creation and insertion

struct node \*head=NULL;

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

head->data=45;

head->link=NULL;

cout<<head->data<<endl;

//node 2

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

current->data=46;

current->link=NULL;

head->link=current;

cout<<current->data<<endl;

//3rd node

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

current->data=467;

current->link=NULL;

cout<<current->data;

head->link->link=current;

// TRAVERSING

count\_of\_nodes(head);

//insert data at end

insert\_end(head,67);

//after inserting new node traverse all data

cout<<"after inserting traversing all data"<<endl;

count\_of\_nodes(head);

return 0;

}

//insert at end using new operator

#include <iostream>

using namespace std;

struct node {

int data;

struct node\* link;

};

void count\_of\_nodes(struct node\* head) {

int count = 0;

if (head == NULL) {

cout << "linked list empty";

}

struct node\* ptr = NULL;

ptr = head;

while (ptr != NULL) {

cout << "data is : " << ptr->data << endl;

count++;

ptr = ptr->link;

}

cout << "total nodes are : " << count<<endl;

}

void insert\_end(struct node\* head, int data) {

/\*if (head == NULL) {

cout << "linked list empty";

}\*/

struct node\* ptr, \* temp;

ptr = head;

temp = new node; // Using new operator to allocate memory

temp->data = data;

temp->link = NULL;

while (ptr->link != NULL) {

ptr = ptr->link;

}

ptr->link = temp;

}

int main() {

struct node\* head = new node; // Using new operator to allocate memory

head->data = 45;

head->link = NULL;

cout << head->data << endl;

// node 2

struct node\* current = new node; // Using new operator to allocate memory

current->data = 46;

current->link = NULL;

head->link = current;

cout << current->data << endl;

// 3rd node

current = new node; // Using new operator to allocate memory

current->data = 467;

current->link = NULL;

cout << current->data<<endl;

head->link->link = current;

// TRAVERSING

count\_of\_nodes(head);

// insert data at the end

insert\_end(head, 67);

// after inserting new node traverse all data

cout << "after inserting traversing all data" << endl;

count\_of\_nodes(head);

return 0;

}

6) insert at beg

//#include<stdio.h>

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*link;

};

void count\_of\_nodes(struct node \*head)

{

int count=0;

if(head==NULL)

{

cout<<"linked list empty";

}

struct node \*ptr=NULL;

ptr=head;

while(ptr!=NULL)

{

cout<<"data is : "<<ptr->data<<endl;

count++;

ptr=ptr->link;

}

cout<<"total nodes are : "<<count;

}

struct node\* insert\_beg(struct node \*head,int data)

{

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

ptr->data=data;

ptr->link=head;

head=ptr;

return head;

}

int main()

{

// creation and insertion

struct node \*head=NULL;

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

head->data=45;

head->link=NULL;

cout<<head->data<<endl;

//node 2

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

current->data=46;

current->link=NULL;

head->link=current;

cout<<current->data<<endl;

//3rd node

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

current->data=467;

current->link=NULL;

cout<<current->data;

head->link->link=current;

// TRAVERSING

count\_of\_nodes(head);

//insert at beg

head=insert\_beg(head,10);

cout<<head->data;

// TRAVERSING again after inserting node at beg

cout<<"total nodes after insert node at beg : "<<endl;

count\_of\_nodes(head);

return 0;

}

7))) node deletion at last

//#include<stdio.h>

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*link;

};

void count\_of\_nodes(struct node \*head)

{

int count=0;

if(head==NULL)

{

cout<<"linked list empty";

}

struct node \*ptr=NULL;

ptr=head;

while(ptr!=NULL)

{

cout<<"data is : "<<ptr->data<<endl;

count++;

ptr=ptr->link;

}

cout<<"total nodes are : "<<count;

}

void delete\_end(struct node \*head)

{

if(head==NULL)

{

cout<<"list already empty";

}

else

{

struct node\* temp1=head;

struct node\* temp2=head;

while(temp1->link!=NULL)

{

temp2=temp1;

temp1=temp1->link;

}

temp2->link=NULL;

}

}

int main()

{

// creation and insertion

struct node \*head=NULL;

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

head->data=45;

head->link=NULL;

cout<<head->data<<endl;

//node 2

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

current->data=46;

current->link=NULL;

head->link=current;

cout<<current->data<<endl;

//3rd node

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

current->data=467;

current->link=NULL;

cout<<current->data;

head->link->link=current;

// TRAVERSING

count\_of\_nodes(head);

//delete at last

delete\_end(head);

//travere gain

cout<<"after deleting last elemnt "<<endl;

count\_of\_nodes(head);

return 0;

}

8) delet from beg

//#include<stdio.h>

#include<iostream>

#include<stdlib.h>

using namespace std;

struct node{

int data;

struct node \*link;

};

void count\_of\_nodes(struct node \*head)

{

int count=0;

if(head==NULL)

{

cout<<"linked list empty";

}

struct node \*ptr=NULL;

ptr=head;

while(ptr!=NULL)

{

cout<<"data is : "<<ptr->data<<endl;

count++;

ptr=ptr->link;

}

cout<<"total nodes are : "<<count;

}

struct node\* delete\_beg(struct node \*head)

{

if(head==NULL)

{

cout<<"list already empty";

}

else

{

head=head->link;

}

return head;

}

int main()

{

// creation and insertion

struct node \*head=NULL;

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

head->data=45;

head->link=NULL;

cout<<head->data<<endl;

//node 2

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

current->data=46;

current->link=NULL;

head->link=current;

cout<<current->data<<endl;

//3rd node

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

current->data=467;

current->link=NULL;

cout<<current->data;

head->link->link=current;

// TRAVERSING

count\_of\_nodes(head);

//delete at last

head=delete\_beg(head);

//travere gain

cout<<"after deleting beg elemnt "<<endl;

count\_of\_nodes(head);

return 0;

}

// COMPLETE CODE With PERFORMINMG ALL OPERATIONS

#include <iostream>

using namespace std;

struct node {

int data;

struct node\* link;

};

void count\_of\_nodes(struct node\* head) {

int count = 0;

if (head == NULL) {

cout << "linked list empty";

}

struct node\* ptr = NULL;

ptr = head;

while (ptr != NULL) {

cout << "data is : " << ptr->data << endl;

count++;

ptr = ptr->link;

}

cout << "total nodes are : " << count<<endl;

}

void insert\_end(struct node\* head, int data) {

/\*if (head == NULL) {

cout << "linked list empty";

}\*/

struct node\* ptr, \* temp;

ptr = head;

temp = new node; // Using new operator to allocate memory

temp->data = data;

temp->link = NULL;

while (ptr->link != NULL) {

ptr = ptr->link;

}

ptr->link = temp;

}

// insert at beg

struct node\* insert\_beg(struct node \*head,int data)

{

struct node \*ptr=new node;

ptr->data=data;

ptr->link=head;

head=ptr;

return head;

}

// delete element form end

void delete\_end(struct node \*head)

{

if(head==NULL)

{

cout<<"list already empty";

}

else

{

struct node\* temp1=head;

struct node\* temp2=head;

while(temp1->link!=NULL)

{

temp2=temp1;

temp1=temp1->link;

}

temp2->link=NULL;

}

}

// delete from beg

struct node\* delete\_beg(struct node \*head)

{

if(head==NULL)

{

cout<<"list already empty";

}

else

{

head=head->link;

/\*struct node\* temp = head;

head = head->link;

delete temp; \*/ // Free the memory of the deleted node

}

return head;

}

// ANOTHER method for deletion I added a new variable temp to hold

//the reference to the first node, then I updated head to point to the second node.

//Finally, I used the delete operator to free the

//memory occupied by the first node.

struct node\* delete\_beg(struct node\* head) {

if (head == NULL) {

cout << "List already empty";

} else {

struct node\* temp = head;

head = head->link;

delete temp; // Free the memory of the deleted node

}

return head;

}

int main() {

struct node\* head = new node; // Using new operator to allocate memory

head->data = 45;

head->link = NULL;

cout << head->data << endl;

// node 2

struct node\* current = new node; // Using new operator to allocate memory

current->data = 46;

current->link = NULL;

head->link = current;

cout << current->data << endl;

// 3rd node

current = new node; // Using new operator to allocate memory

current->data = 47;

current->link = NULL;

cout << current->data<<endl;

head->link->link = current;

// TRAVERSING

count\_of\_nodes(head);

// insert data at the end

insert\_end(head, 67);

// after inserting new node traverse all data

cout << "after inserting data at END traversing all data" << endl;

count\_of\_nodes(head);

head=insert\_beg(head, 76);

cout << "after inserting data at beg opertaion traversing all data" << endl;

count\_of\_nodes(head);

// deletion at end

delete\_end(head);

cout << "delete data from end end" << endl;

count\_of\_nodes(head);

return 0;

}

// INSERTION AT SPECIFIC POSITION

#include<iostream>

using namespace std;

struct Node {

int data;

Node\* link;

};

// Function to insert a new node after a given node

void insertAfter(Node\* prevNode, int newData) {

if (prevNode == nullptr) {

cout << "The previous node cannot be null." << endl;

return;

}

Node\* newNode = new Node;

newNode->data = newData;

newNode->link = prevNode->link;

prevNode->link = newNode;

}

// Function to insert a new node at a given position

void insertAtPosition(Node\* &head, int position, int newData) {

if (position < 0) {

cout << "Invalid position." << endl;

return;

}

Node\* newNode = new Node;

newNode->data = newData;

newNode->link = nullptr;

if (position == 0) {

newNode->link = head;

head = newNode;

return;

}

Node\* temp = head;

int i = 0;

while (temp != nullptr && i < position - 1) {

temp = temp->link;

i++;

}

if (temp == nullptr) {

cout << "Position out of bounds." << endl;

return;

}

newNode->link = temp->link;

temp->link = newNode;

}

int main() {

Node\* head = new Node;

head->data = 45;

head->link = nullptr;

cout << head->data << endl;

Node\* current = new Node;

current->data = 46;

current->link = nullptr;

head->link = current;

cout << current->data << endl;

// Insertion after current (2nd node)

insertAfter(current, 50);

// Insertion at position 2

insertAtPosition(head, 2, 60);

// Traversing and printing the list

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->link;

}

cout << endl;

return 0;

}