|  |  |
| --- | --- |
| ASSIGNMENT 5 SOLUTIONS | Ishaan Mishra  1024030346 |

# Lab Assignment 5: Singly Linked List (C++ Solutions)

## Q1. Menu Driven Singly Linked List Operations

#include <iostream>  
using namespace std;  
  
struct Node {  
 int data;  
 Node\* next;  
};  
  
Node\* head = NULL;  
  
void insertBegin(int val) {  
 Node\* newNode = new Node();  
 newNode->data = val;  
 newNode->next = head;  
 head = newNode;  
}  
  
void insertEnd(int val) {  
 Node\* newNode = new Node();  
 newNode->data = val;  
 newNode->next = NULL;  
 if (head == NULL) head = newNode;  
 else {  
 Node\* temp = head;  
 while (temp->next != NULL) temp = temp->next;  
 temp->next = newNode;  
 }  
}  
  
void insertAfter(int key, int val) {  
 Node\* temp = head;  
 while (temp != NULL && temp->data != key) temp = temp->next;  
 if (temp == NULL) { cout << "Key not found\n"; return; }  
 Node\* newNode = new Node();  
 newNode->data = val;  
 newNode->next = temp->next;  
 temp->next = newNode;  
}  
  
void insertBefore(int key, int val) {  
 if (head == NULL) return;  
 if (head->data == key) { insertBegin(val); return; }  
 Node\* temp = head;  
 while (temp->next != NULL && temp->next->data != key) temp = temp->next;  
 if (temp->next == NULL) { cout << "Key not found\n"; return; }  
 Node\* newNode = new Node();  
 newNode->data = val;  
 newNode->next = temp->next;  
 temp->next = newNode;  
}  
  
void deleteBegin() {  
 if (head == NULL) return;  
 Node\* temp = head;  
 head = head->next;  
 delete temp;  
}  
  
void deleteEnd() {  
 if (head == NULL) return;  
 if (head->next == NULL) { delete head; head = NULL; return; }  
 Node\* temp = head;  
 while (temp->next->next != NULL) temp = temp->next;  
 delete temp->next;  
 temp->next = NULL;  
}  
  
void deleteKey(int key) {  
 if (head == NULL) return;  
 if (head->data == key) { deleteBegin(); return; }  
 Node\* temp = head;  
 while (temp->next != NULL && temp->next->data != key) temp = temp->next;  
 if (temp->next == NULL) { cout << "Key not found\n"; return; }  
 Node\* delNode = temp->next;  
 temp->next = delNode->next;  
 delete delNode;  
}  
  
void search(int key) {  
 Node\* temp = head;  
 int pos = 1;  
 while (temp != NULL) {  
 if (temp->data == key) { cout << "Found at position " << pos << "\n"; return; }  
 temp = temp->next; pos++;  
 }  
 cout << "Not found\n";  
}  
  
void display() {  
 Node\* temp = head;  
 cout << "List: ";  
 while (temp != NULL) {  
 cout << temp->data << " ";  
 temp = temp->next;  
 }  
 cout << endl;  
}  
  
int main() {  
 int ch, val, key;  
 while (1) {  
 cout << "1.InsertBegin 2.InsertEnd 3.InsertAfter 4.InsertBefore 5.DeleteBegin 6.DeleteEnd 7.DeleteKey 8.Search 9.Display 10.Exit\n";  
 cin >> ch;  
 switch (ch) {  
 case 1: cin >> val; insertBegin(val); break;  
 case 2: cin >> val; insertEnd(val); break;  
 case 3: cin >> key >> val; insertAfter(key,val); break;  
 case 4: cin >> key >> val; insertBefore(key,val); break;  
 case 5: deleteBegin(); break;  
 case 6: deleteEnd(); break;  
 case 7: cin >> key; deleteKey(key); break;  
 case 8: cin >> key; search(key); break;  
 case 9: display(); break;  
 case 10: return 0;  
 }  
 }  
}

## Q2. Count and Delete All Occurrences of a Key

#include <iostream>  
using namespace std;  
  
struct Node {  
 int data;  
 Node\* next;  
};  
Node\* head = NULL;  
  
void insertEnd(int val) {  
 Node\* newNode = new Node();  
 newNode->data = val;  
 newNode->next = NULL;  
 if (!head) head = newNode;  
 else {  
 Node\* temp = head;  
 while (temp->next) temp = temp->next;  
 temp->next = newNode;  
 }  
}  
  
void deleteKey(int key) {  
 while (head && head->data == key) {  
 Node\* temp = head;  
 head = head->next;  
 delete temp;  
 }  
 Node\* curr = head;  
 while (curr && curr->next) {  
 if (curr->next->data == key) {  
 Node\* temp = curr->next;  
 curr->next = temp->next;  
 delete temp;  
 } else curr = curr->next;  
 }  
}  
  
int countKey(int key) {  
 int count = 0;  
 Node\* temp = head;  
 while (temp) {  
 if (temp->data == key) count++;  
 temp = temp->next;  
 }  
 return count;  
}  
  
void display() {  
 Node\* temp = head;  
 while (temp) {  
 cout << temp->data << " ";  
 temp = temp->next;  
 }  
 cout << endl;  
}  
  
int main() {  
 insertEnd(1); insertEnd(2); insertEnd(1); insertEnd(2); insertEnd(1); insertEnd(3); insertEnd(1);  
 int key = 1;  
 cout << "Count: " << countKey(key) << endl;  
 deleteKey(key);  
 display();  
 return 0;  
}

## Q3. Find Middle of Linked List

#include <iostream>  
using namespace std;  
  
struct Node {  
 int data;  
 Node\* next;  
};  
Node\* head = NULL;  
  
void insertEnd(int val) {  
 Node\* newNode = new Node();  
 newNode->data = val;  
 newNode->next = NULL;  
 if (!head) head = newNode;  
 else {  
 Node\* temp = head;  
 while (temp->next) temp = temp->next;  
 temp->next = newNode;  
 }  
}  
  
void findMiddle() {  
 Node\* slow = head;  
 Node\* fast = head;  
 while (fast && fast->next) {  
 slow = slow->next;  
 fast = fast->next->next;  
 }  
 if (slow) cout << "Middle: " << slow->data << endl;  
}  
  
int main() {  
 for (int i=1;i<=5;i++) insertEnd(i);  
 findMiddle();  
 return 0;  
}

## Q4. Reverse Linked List

#include <iostream>  
using namespace std;  
  
struct Node {  
 int data;  
 Node\* next;  
};  
Node\* head = NULL;  
  
void insertEnd(int val) {  
 Node\* newNode = new Node();  
 newNode->data = val;  
 newNode->next = NULL;  
 if (!head) head = newNode;  
 else {  
 Node\* temp = head;  
 while (temp->next) temp = temp->next;  
 temp->next = newNode;  
 }  
}  
  
void reverse() {  
 Node\* prev = NULL;  
 Node\* curr = head;  
 while (curr) {  
 Node\* next = curr->next;  
 curr->next = prev;  
 prev = curr;  
 curr = next;  
 }  
 head = prev;  
}  
  
void display() {  
 Node\* temp = head;  
 while (temp) {  
 cout << temp->data << " ";  
 temp = temp->next;  
 }  
 cout << endl;  
}  
  
int main() {  
 for (int i=1;i<=4;i++) insertEnd(i);  
 reverse();  
 display();  
 return 0;  
}