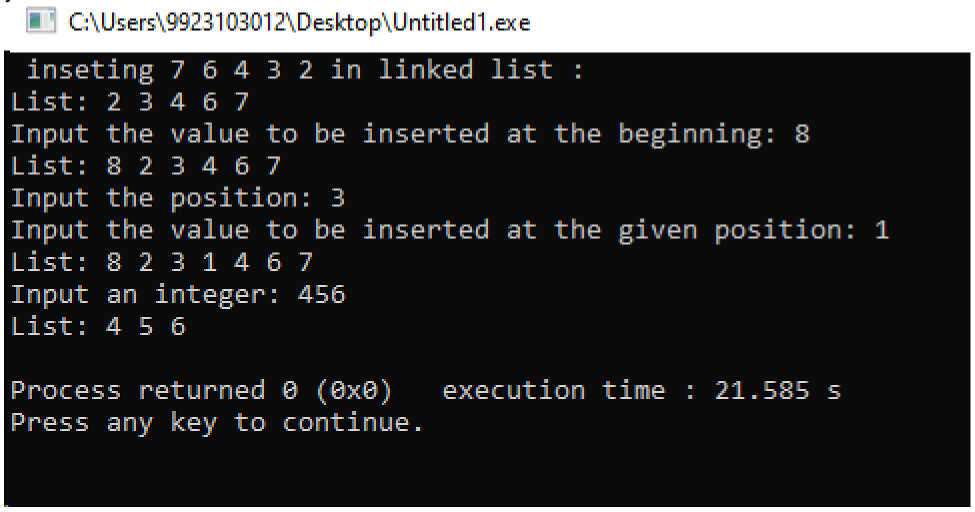
15B17CI371 – Data Structures
Lab ODD 2024
Week 1-LAB B
Practice Lab
[CO: C270.1]

1. **Given a linked list, write functions to a. Insert an element at the beginning of the linked list. b. Insert an element at a specific location in a linked list. c. Take input an integer number, split the number in its digits and stores the digits in a linked list structure.**

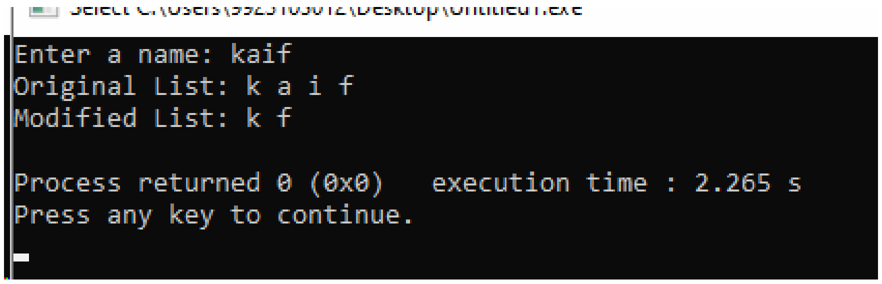
```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* next;

    Node(int val) : data(val), next(NULL) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(NULL) {}

    void prepend(int data) {
        Node* newNode = new Node(data);
        newNode->next = head;
        head = newNode;
    }

    void insert(int data, int pos) {
        Node* newNode = new Node(data);
        if (pos == 0) {
            newNode->next = head;
            head = newNode;
        } else {
            Node* temp = head;
            for (int i = 0; i < pos - 1 && temp != NULL; ++i) {
                temp = temp->next;
            }
            if (temp != NULL) {
                newNode->next = temp->next;
                temp->next = newNode;
            } else {
                cout << "Position out of bounds" << endl;
                delete newNode;
            }
        }
    }

    void print() const {
        cout << "List: ";
        Node* temp = head;
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    void split(int data) {
        if (data == 0) {
            prepend(0);
            return;
        }
        while (data > 0) {
            prepend(data % 10);
            data /= 10;
        }
    }
};

int main() {
    LinkedList list;
    cout << "Inserting 7 6 4 3 2 into the linked list:" << endl;
    list.prepend(7);
    list.prepend(6);
    list.prepend(4);
    list.prepend(3);
    list.prepend(2);
    list.print();

    int val, pos;
    cout << "Input the value to be inserted at the beginning: ";
    cin >> val;
    list.prepend(val);
    list.print();
    cout << "Input the position: ";
    cin >> pos;
    cout << "Input the value to be inserted at the given position: ";
    cin >> val;
    list.insert(val, pos);
    list.print();

    cout << "Input an integer: ";
    cin >> val;
    LinkedList list2;
    list2.split(val);
    list2.print();

    return 0;
}
```



```
C:\Users\9923103012\Desktop\Untitled1.exe
 inseting 7 6 4 3 2 in linked list :
List: 2 3 4 6 7
Input the value to be inserted at the beginning: 8
List: 8 2 3 4 6 7
Input the position: 3
Input the value to be inserted at the given position: 1
List: 8 2 3 1 4 6 7
Input an integer: 456
List: 4 5 6

Process returned 0 (0x0)   execution time : 21.585 s
Press any key to continue.
```

2.Write a program which reads a name and generates the link list of the characters in that name. Later it removes the vowels from the link list and displays the modified link list.

```
#include <iostream>

using namespace std;



class Node {

public:

    char data;
    Node* next;

    Node(char val) : data(val), next(NULL) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(NULL) {}

    void insertAtBeginning(char data) {
        Node* newNode = new Node(data);
        newNode->next = head;
        head = newNode;
    }

    void printList() const {
        cout << "List: ";
        Node* temp = head;
        while (temp != NULL) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;

    }


    bool isVowel(char x) const {

        return x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u' ||

               x == 'A' || x == 'E' || x == 'I' || x == 'O' || x == 'U';

    }


    void deleteVowels() {


        while (head != NULL && isVowel(head->data)) {

            Node* temp = head;

            head = head->next;

            delete temp;

        }




        Node* current = head;

        Node* prev = NULL;

        while (current != NULL) {

            if (isVowel(current->data)) {

                prev->next = current->next;

                delete current;

                current = prev->next;

            } else {
                prev = current;
                current = current->next;
            }
        }
    }

    void createListFromName(const string& name) {
        for (int i = name.length() - 1; i >= 0; i--) {
            insertAtBeginning(name[i]);
        }
    }
    };

    int main() {
    LinkedList list;
    string name;

    cout << "Enter a name: ";
    cin >> name;

    list.createListFromName(name);

    cout << "Original ";
    list.printList();
    list.deleteVowels();



    cout << "Modified ";

    list.printList();



    return 0;

}
```



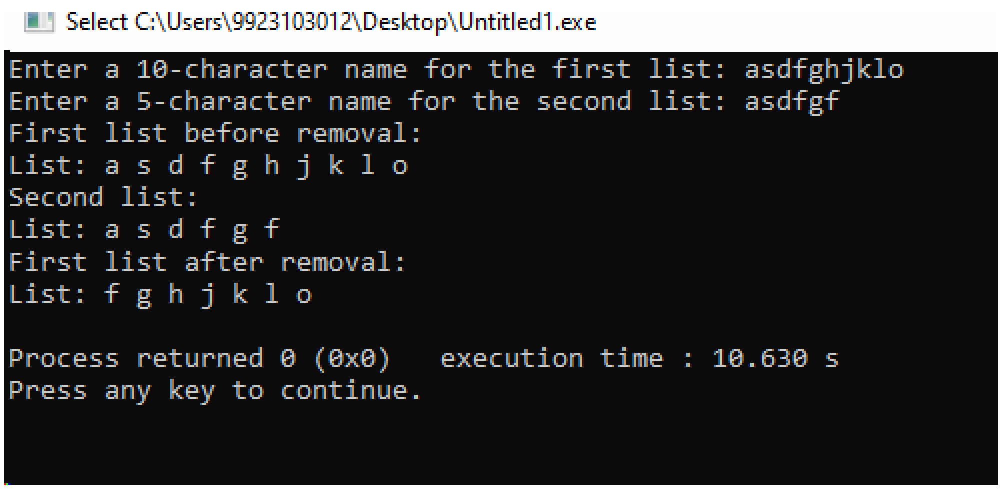
```
Enter a name: kaif
Original List: k a i f
Modified List: k f

Process returned 0 (0x0)   execution time : 2.265 s
Press any key to continue.
```

3. Create a link list of users supplied ten characters to store a name. Create a second link list of same type of user supplied five characters. Now using a function remove(), traverse first link list and if any three consecutive characters of second link list appears as consecutive characters of first link list, remove those from first link list.

```
#include <iostream>
using namespace std;

class Node {
public:
    char data;
    Node* next;

    Node(char val) : data(val), next(NULL) {}
};

class LL {
private:
    Node* head;

public:
    LL() : head(NULL) {}

    void addFront(char data) {
        Node* newNode = new Node(data);
        newNode->next = head;
        head = newNode;
    }
```

```
void print() const {
   cout << "List: ";
   Node* temp = head;
   while (temp != NULL) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}

bool isVowel(char x) const {
   return x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u' ||
        x == 'A' || x == 'E' || x == 'I' || x == 'O' || x == 'U';
}

void remPattern(const string& pat) {
   int len = pat.length();

   if (len == 0) return;

   Node* curr = head;
   Node* prev = NULL;

   while (curr != NULL) {
      Node* temp = curr;
      bool match = true;

      for (int i = 0; i < len; ++i) {
         if (temp == NULL || temp->data != pat[i]) {
            match = false;
            break;
         }
         temp = temp->next;
      }

      if (match) {
         if (prev == NULL) {
            head = curr->next;
         } else {
            prev->next = curr->next;
         }

         while (curr != temp) {
            Node* toDel = curr;
            curr = curr->next;
            delete toDel;
         }
         continue;
      }

      prev = curr;
      curr = curr->next;
   }
}

void fillList(const string& str) {
   for (int i = str.length() - 1; i >= 0; i--) {
      addFront(str[i]);
```

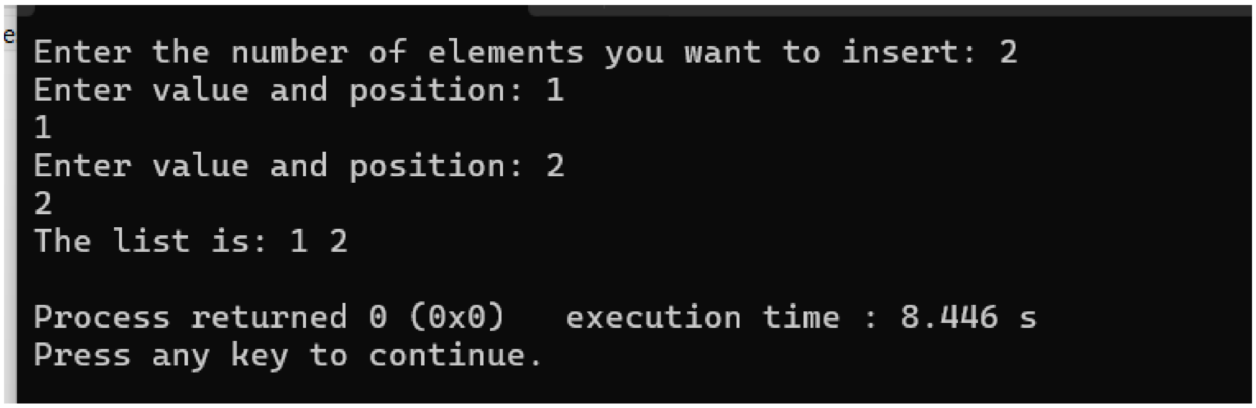
```
        }
    }
};

int main() {
    LL list1, list2;

    string name1, name2;

    cout << "Enter a 10-character name for the first list: ";
    cin >> name1;

    cout << "Enter a 5-character name for the second list: ";
    cin >> name2;

    list1.fillList(name1);
    list2.fillList(name2);

    cout << "First list before removal:" << endl;
    list1.print();

    cout << "Second list:" << endl;
    list2.print();

    for (int i = 0; i <= name2.length() - 3; ++i) {
        string sub = name2.substr(i, 3);
        list1.remPattern(sub);
    }

    cout << "First list after removal:" << endl;
    list1.print();

    return 0;
}
```


Select C:\Users\9923103012\Desktop\Untitled1.exe

```
Enter a 10-character name for the first list: asdfghjklo
Enter a 5-character name for the second list: asdfgf
First list before removal:
List: a s d f g h j k l o
Second list:
List: a s d f g f
First list after removal:
List: f g h j k l o

Process returned 0 (0x0)   execution time : 10.630 s
Press any key to continue.
```

4.Write a program to insert an element at specific location in doubly linked list.

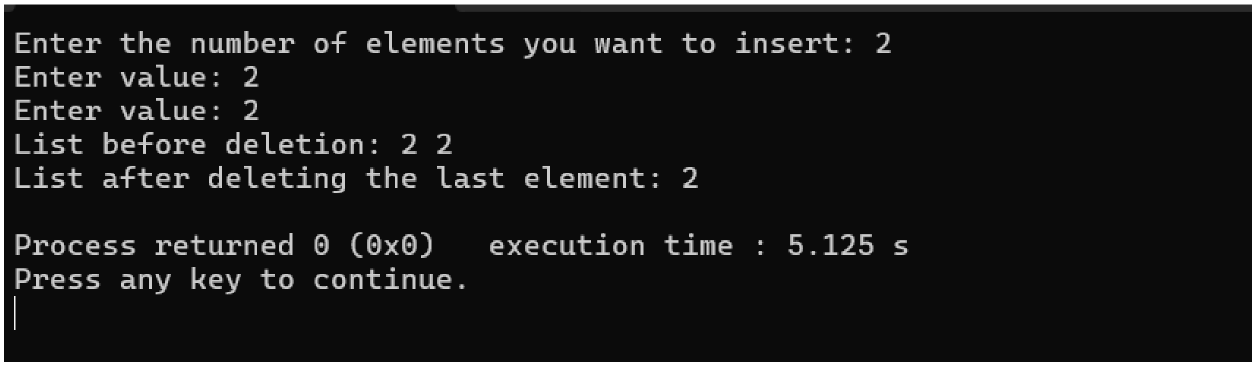
```
#include <iostream>

using namespace std;


struct Node {

    int data;

    Node* prev;

    Node* next;

};


Node* createNode(int val) {

    Node* n = new Node();

    n->data = val;

    n->prev = nullptr;

    n->next = nullptr;

    return n;

}


void insert(Node*& head, int val, int pos) {

    Node* n = createNode(val);

    if (pos == 1) {

        n->next = head;

        if (head != nullptr) {

            head->prev = n;

        }
        head = n;

        return;

    }


    Node* temp = head;

    for (int i = 1; temp != nullptr && i < pos - 1; ++i) {

        temp = temp->next;

    }


    if (temp == nullptr) {

        return;  // Position is out of bounds

    }


    n->next = temp->next;

    if (temp->next != nullptr) {

        temp->next->prev = n;

    }

    temp->next = n;

    n->prev = temp;

}


void display(Node* head) {

    Node* temp = head;

    while (temp != nullptr) {

        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    int n, val, pos;

    cout << "Enter the number of elements you want to insert: ";
    cin >> n;

    for (int i = 0; i < n; ++i) {
        cout << "Enter value and position: ";
        cin >> val >> pos;
        insert(head, val, pos);
    }

    cout << "The list is: ";
    display(head);

    return 0;
}
```


```
Enter the number of elements you want to insert: 2
Enter value and position: 1
1
Enter value and position: 2
2
The list is: 1 2

Process returned 0 (0x0)   execution time : 8.446 s
Press any key to continue.
```

5. Write a program to delete last element from the doubly linked list.

```
#include <iostream>

using namespace std;


struct Node {

   int data;

   Node* prev;

   Node* next;

};


Node* createNode(int val) {

   Node* n = new Node();

   n->data = val;

   n->prev = nullptr;

   n->next = nullptr;

   return n;

}
```

```
void insertEnd(Node*& head, int val) {
    Node* n = createNode(val);
    if (head == nullptr) {
        head = n;
        return;
    }

    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }

    temp->next = n;
    n->prev = temp;
}

void deleteLast(Node*& head) {
    if (head == nullptr) {
        return;  // List is empty
    }

    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        return;
    }

    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }

    temp->prev->next = nullptr;
    delete temp;
}

void display(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    int n, val;

    cout << "Enter the number of elements you want to insert: ";
    cin >> n;


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

        cout << "Enter value: ";

        cin >> val;

        insertEnd(head, val);

    }


    cout << "List before deletion: ";

    display(head);


    deleteLast(head);


    cout << "List after deleting the last element: ";

    display(head);


    return 0;

}
```



```
Enter the number of elements you want to insert: 2
Enter value: 2
Enter value: 2
List before deletion: 2 2
List after deleting the last element: 2

Process returned 0 (0x0)   execution time : 5.125 s
Press any key to continue.
```

6. Given a doubly linked list of any number of nodes, write a function

ExtremeSwap(), which will swap values of the node at extreme pairs. For e.g., if

the node values of a doubly linked list are:

1 2 3 4 5 6 7 8

After first call, values will be

8 2 3 4 5 6 7 1

After second call, values will be

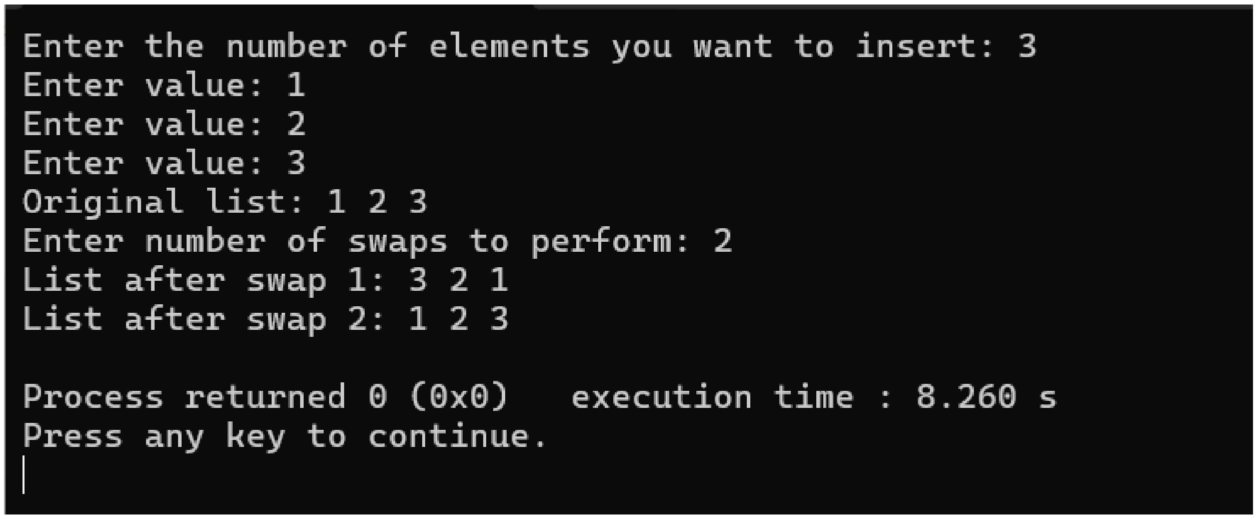
8 7 3 4 5 6 2 1

And finally, function will stop after fourth call, and the values will be

8 7 6 5 4 3 2 1

```
#include <iostream>

using namespace std;


struct Node {

   int data;

   Node* prev;

   Node* next;

};


Node* createNode(int val) {

   Node* n = new Node();

   n->data = val;

   n->prev = nullptr;

   n->next = nullptr;

   return n;
```

```
}


void insertEnd(Node*& head, int val) {
    Node* n = createNode(val);
    if (head == nullptr) {
        head = n;
        return;
    }

    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }

    temp->next = n;
    n->prev = temp;
}

void extremeSwap(Node*& head) {
    if (head == nullptr) return;

    Node* start = head;
    Node* end = head;
    while (end->next != nullptr) {
        end = end->next;
    }

    while (start != end && start->prev != end) {
        int temp = start->data;
        start->data = end->data;
        end->data = temp;

        start = start->next;
        end = end->prev;
    }
}

void display(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    int n, val;
    cout << "Enter the number of elements you want to insert: ";
    cin >> n;

    for (int i = 0; i < n; ++i) {
        cout << "Enter value: ";
        cin >> val;
        insertEnd(head, val);
    }

    cout << "Original list: ";
    display(head);

    int swaps;
    cout << "Enter number of swaps to perform: ";
    cin >> swaps;

    for (int i = 0; i < swaps; ++i) {
        extremeSwap(head);
        cout << "List after swap " << i + 1 << ": ";
        display(head);
    }

    return 0;
}
```



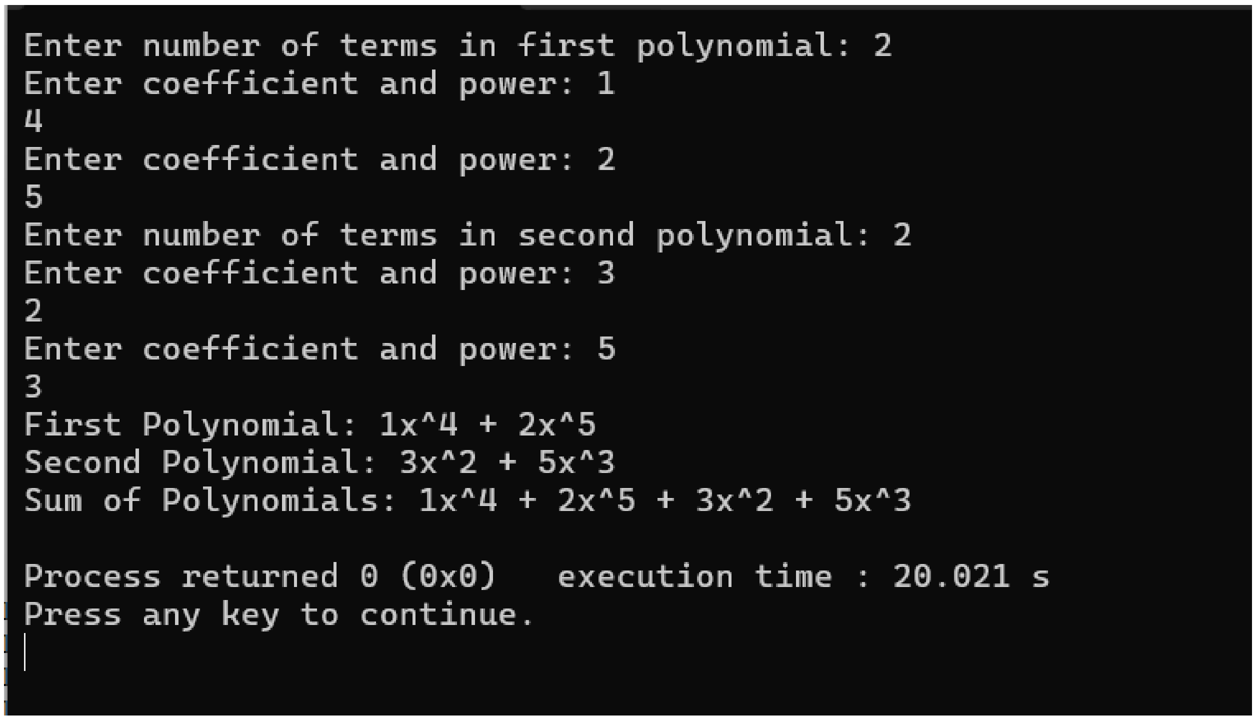
```
Enter the number of elements you want to insert: 3
Enter value: 1
Enter value: 2
Enter value: 3
Original list: 1 2 3
Enter number of swaps to perform: 2
List after swap 1: 3 2 1
List after swap 2: 1 2 3

Process returned 0 (0x0)   execution time : 8.260 s
Press any key to continue.
```

7. Write a program to implement addition of two polynomials. Each node must

contain the value of the coefficient as well as its power as data components.

```
#include <iostream>

using namespace std;


struct Node {

   int coeff;

   int power;

   Node* next;

};


Node* createNode(int c, int p) {

   Node* n = new Node();

   n->coeff = c;

   n->power = p;
```

```
    n->next = nullptr;

    return n;

}


void insertEnd(Node*& head, int c, int p) {

    Node* n = createNode(c, p);

    if (head == nullptr) {

        head = n;

        return;

    }


    Node* temp = head;

    while (temp->next != nullptr) {

        temp = temp->next;

    }


    temp->next = n;

}


Node* addPoly(Node* p1, Node* p2) {

    Node* result = nullptr;


    while (p1 != nullptr && p2 != nullptr) {

        if (p1->power > p2->power) {

            insertEnd(result, p1->coeff, p1->power);
            p1 = p1->next;
        } else if (p1->power < p2->power) {
            insertEnd(result, p2->coeff, p2->power);
            p2 = p2->next;
        } else {
            insertEnd(result, p1->coeff + p2->coeff, p1->power);
            p1 = p1->next;
            p2 = p2->next;
        }
    }

    while (p1 != nullptr) {
        insertEnd(result, p1->coeff, p1->power);
        p1 = p1->next;
    }

    while (p2 != nullptr) {
        insertEnd(result, p2->coeff, p2->power);
        p2 = p2->next;
    }

    return result;
}

void displayPoly(Node* head) {
    Node* temp = head;

    while (temp != nullptr) {

        cout << temp->coeff << "x^" << temp->power;

        temp = temp->next;

        if (temp != nullptr) cout << " + ";

    }

    cout << endl;

}


int main() {

    Node* poly1 = nullptr;

    Node* poly2 = nullptr;


    int n1, n2, coeff, power;


    cout << "Enter number of terms in first polynomial: ";

    cin >> n1;

    for (int i = 0; i < n1; ++i) {

        cout << "Enter coefficient and power: ";

        cin >> coeff >> power;

        insertEnd(poly1, coeff, power);

    }


    cout << "Enter number of terms in second polynomial: ";

    cin >> n2;
    for (int i = 0; i < n2; ++i) {

        cout << "Enter coefficient and power: ";

        cin >> coeff >> power;

        insertEnd(poly2, coeff, power);

    }


    cout << "First Polynomial: ";

    displayPoly(poly1);


    cout << "Second Polynomial: ";

    displayPoly(poly2);


    Node* sum = addPoly(poly1, poly2);


    cout << "Sum of Polynomials: ";

    displayPoly(sum);


    return 0;

}
```



```
Enter number of terms in first polynomial: 2
Enter coefficient and power: 1
4
Enter coefficient and power: 2
5
Enter number of terms in second polynomial: 2
Enter coefficient and power: 3
2
Enter coefficient and power: 5
3
First Polynomial: 1x^4 + 2x^5
Second Polynomial: 3x^2 + 5x^3
Sum of Polynomials: 1x^4 + 2x^5 + 3x^2 + 5x^3

Process returned 0 (0x0)   execution time : 20.021 s
Press any key to continue.
```

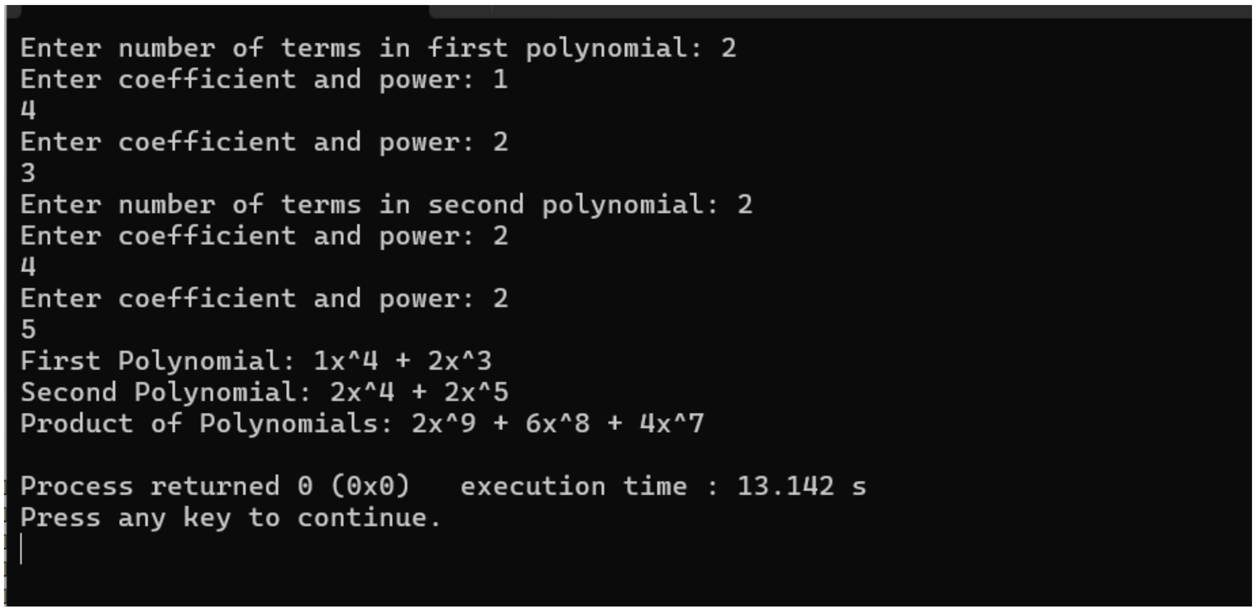
8. Write a program to implement multiplication of two polynomials. Each node must contain the value of the coefficient as well as its power as data components. Take care of law of exponent multiplication.

```
#include <iostream>

using namespace std;


struct Node {

   int coeff;

   int power;

   Node* next;

};


Node* createNode(int c, int p) {

   Node* n = new Node();
```

```
    n->coeff = c;

    n->power = p;

    n->next = nullptr;

    return n;

}


void insertEnd(Node*& head, int c, int p) {

    Node* n = createNode(c, p);

    if (head == nullptr) {

        head = n;

        return;

    }


    Node* temp = head;

    while (temp->next != nullptr) {

        temp = temp->next;

    }


    temp->next = n;

}


Node* multiplyPoly(Node* p1, Node* p2) {

    Node* result = nullptr;


    for (Node* i = p1; i != nullptr; i = i->next) {
```

```
    for (Node* j = p2; j != nullptr; j = j->next) {
        int c = i->coeff * j->coeff;
        int p = i->power + j->power;
        Node* temp = result;
        Node* prev = nullptr;
        while (temp != nullptr && temp->power > p) {
            prev = temp;
            temp = temp->next;
        }
        if (temp != nullptr && temp->power == p) {
            temp->coeff += c;
        } else {
            Node* newNode = createNode(c, p);
            if (prev == nullptr) {
                newNode->next = result;
                result = newNode;
            } else {
                newNode->next = prev->next;
                prev->next = newNode;
            }
        }
    }
}

return result;
```

```
}


void displayPoly(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->coeff << "x^" << temp->power;
        temp = temp->next;
        if (temp != nullptr) cout << " + ";
    }
    cout << endl;
}


int main() {
    Node* poly1 = nullptr;
    Node* poly2 = nullptr;


    int n1, n2, coeff, power;


    cout << "Enter number of terms in first polynomial: ";
    cin >> n1;
    for (int i = 0; i < n1; ++i) {
        cout << "Enter coefficient and power: ";
        cin >> coeff >> power;
        insertEnd(poly1, coeff, power);
    }
    cout << "Enter number of terms in second polynomial: ";
    cin >> n2;
    for (int i = 0; i < n2; ++i) {
        cout << "Enter coefficient and power: ";
        cin >> coeff >> power;
        insertEnd(poly2, coeff, power);
    }

    cout << "First Polynomial: ";
    displayPoly(poly1);

    cout << "Second Polynomial: ";
    displayPoly(poly2);

    Node* product = multiplyPoly(poly1, poly2);

    cout << "Product of Polynomials: ";
    displayPoly(product);

    return 0;
}
```


```
Enter number of terms in first polynomial: 2
Enter coefficient and power: 1
4
Enter coefficient and power: 2
3
Enter number of terms in second polynomial: 2
Enter coefficient and power: 2
4
Enter coefficient and power: 2
5
First Polynomial: 1x^4 + 2x^3
Second Polynomial: 2x^4 + 2x^5
Product of Polynomials: 2x^9 + 6x^8 + 4x^7

Process returned 0 (0x0)   execution time : 13.142 s
Press any key to continue.
```