Ain Shams University
Faculty of Engineering
Computer and Systems Engineering Department

Data structure and algorithms

Name: يوسف محمود علي

Sec: 3

Insert at index ():

Code:

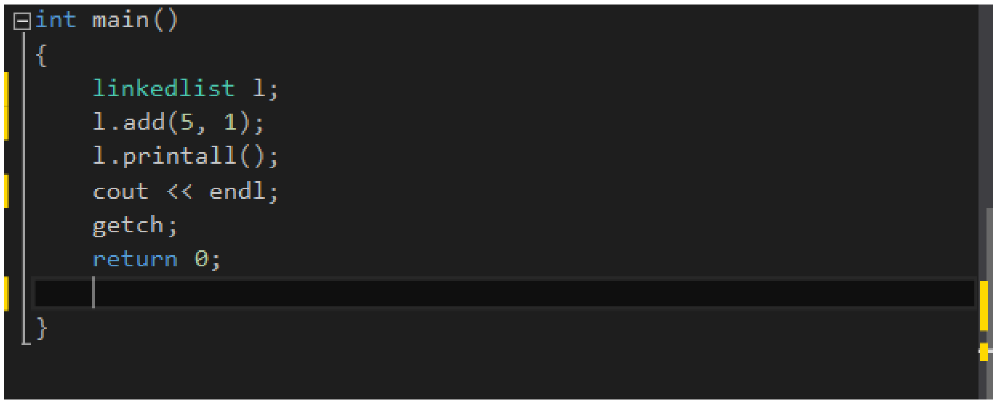
```
void linkedlist::add(int item, int pos)
{
	if (pos == 1)
		addfront(item);
	else{
		node *temp1 = new node(item);
		node *temp = head;
		if (head == nullptr)
		{
			addfront(item);
		}
		else
		{
			for (int i = 0; i < pos - 2; i++)
			{
				temp = temp->next;
				if (temp->next == nullptr)
				{
					addend(item);
					return;
				}

			}
			temp1->next = temp->next;
			temp->next = temp1;
		}

	}
}
```

Explanation:

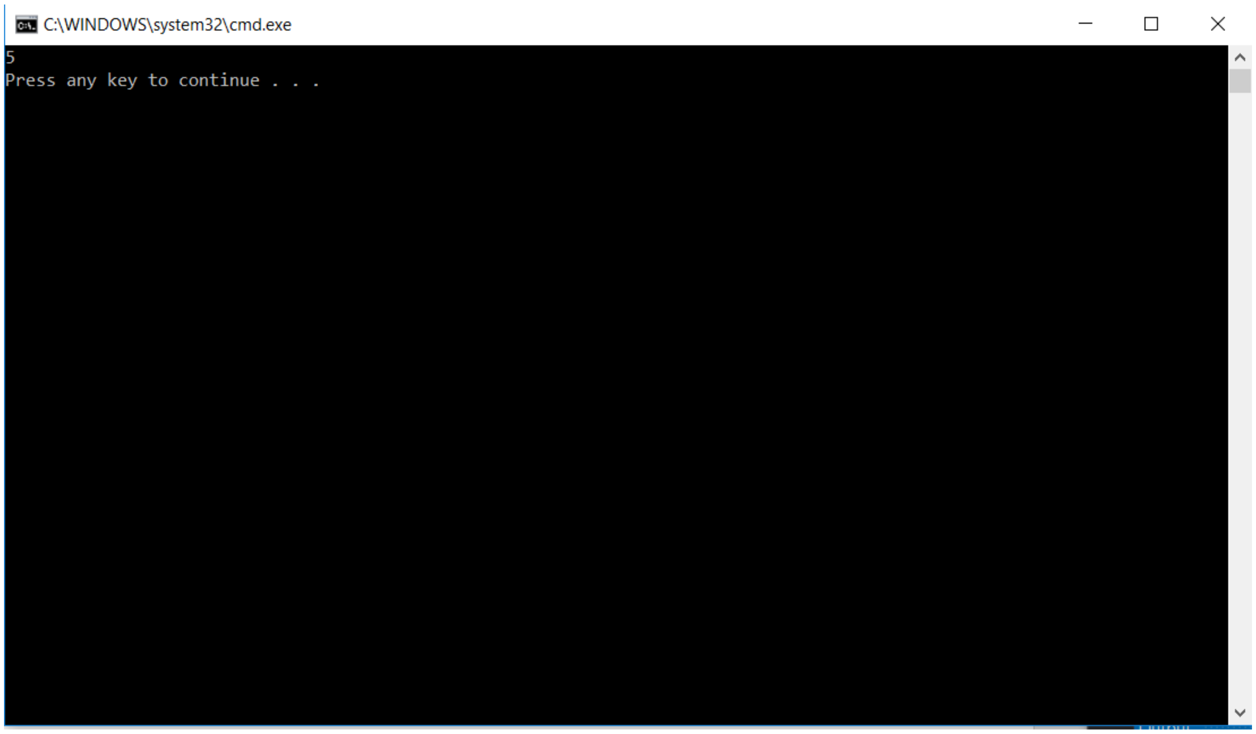
First we need to make sure that the list isn't empty which I do in the add front function then check if the index is the front then I made a for loop to get a temp pointer to point to the needed index if the temp reaches the end before the needed index then we call the addend function else we insert the new node and set the pointers.

Testing: Special case:

Insert in empty list:

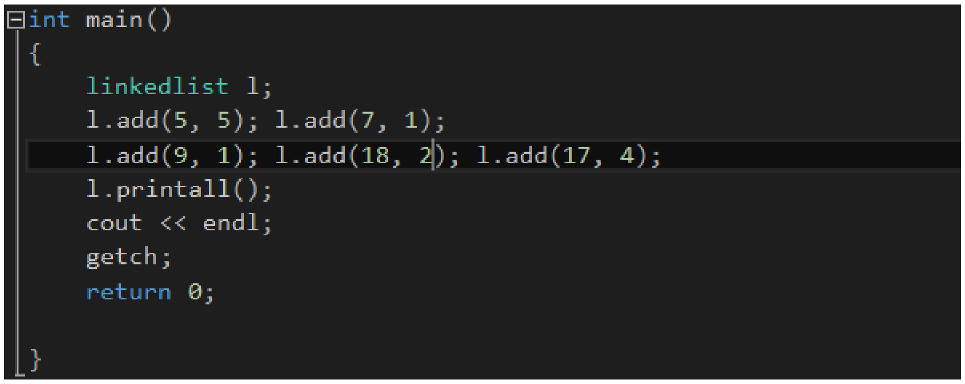


```
int main()
{
    linkedlist l;
    l.add(5, 1);
    l.printall();
    cout << endl;
    getch;
    return 0;

}
```

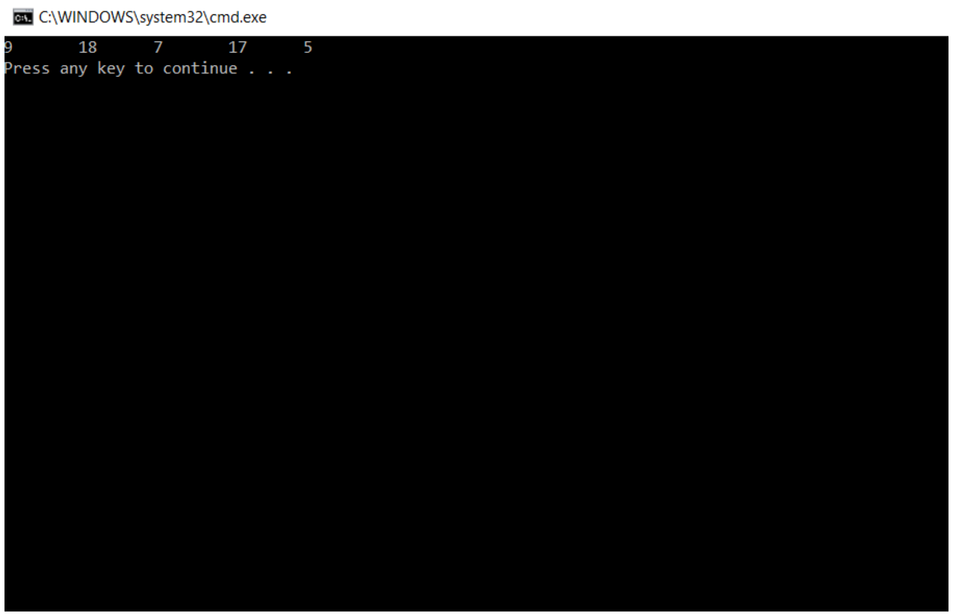


```
5
Press any key to continue . . .
```

General case:



```
int main()
{
    linkedlist l;
    l.add(5, 5); l.add(7, 1);
    l.add(9, 1); l.add(18, 2); l.add(17, 4);
    l.printall();
    cout << endl;
    getch;
    return 0;

}
```



```
C:\WINDOWS\system32\cmd.exe
9       18      7       17      5
Press any key to continue . . .
```

Is empty:

Code:

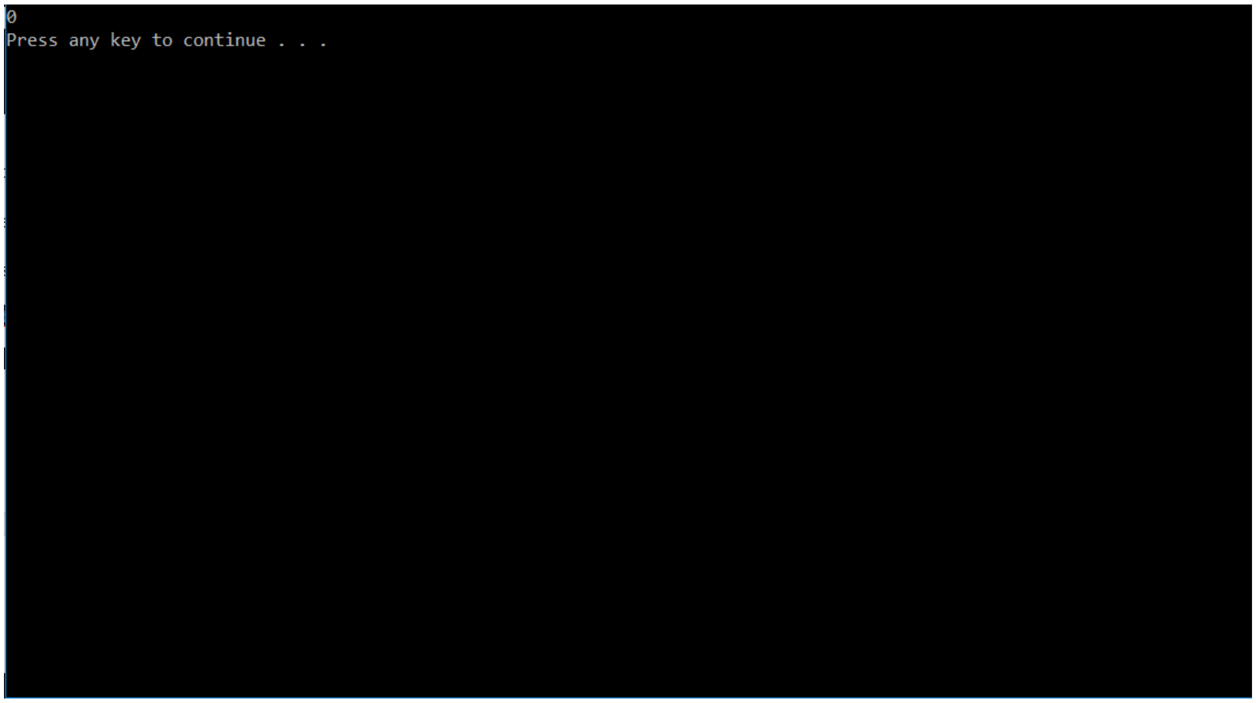
```
bool linkedlist::isempty()
{
	if (head == nullptr)
		return true;
	else
		return false;
}
```

Explanation:

We just need to check if the head pointer is null pointer

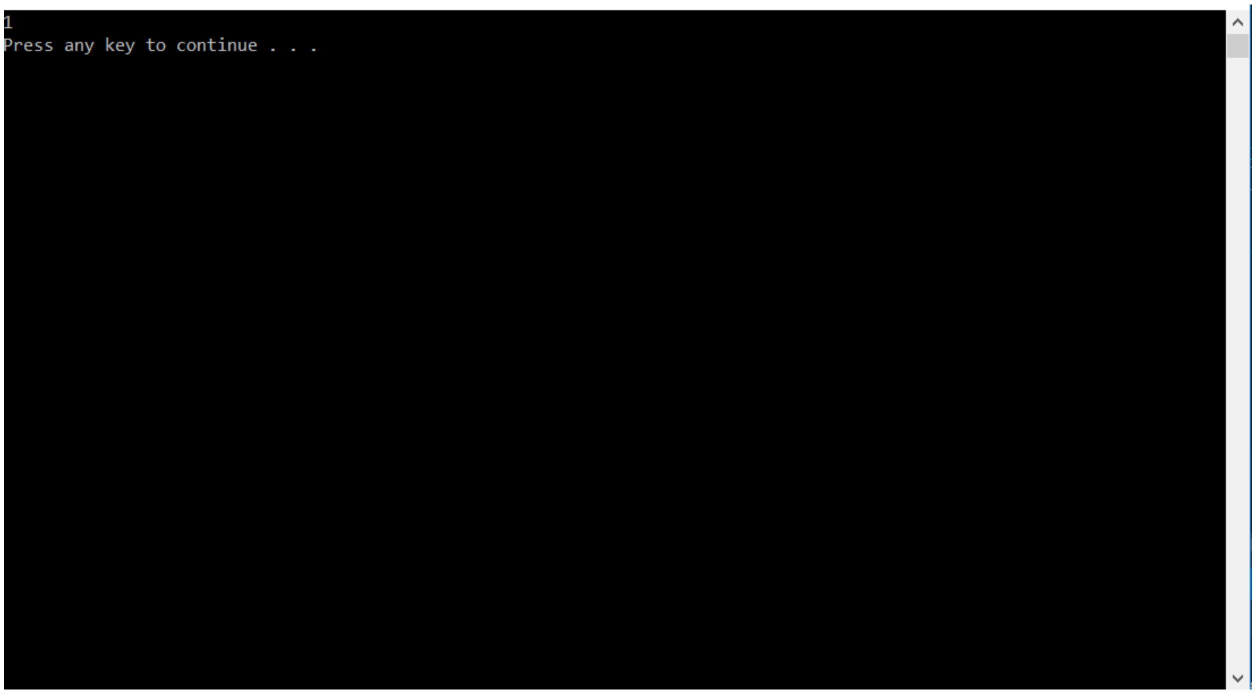
Testing:

```
int main()
{
	linkedlist l;
	l.add(5, 5); l.add(7, 1);
	l.add(9, 1); l.add(18, 2); l.add(17, 4);
	bool is = l.isempty();
	cout << is << endl;
	return 0;

}
```



```
0
Press any key to continue . . .
```

```cpp
int main()
{
	linkedlist l;
	bool is = l.isempty();
	cout << is << endl;
	return 0;

}
```



```
1
Press any key to continue . . .
```

Count:

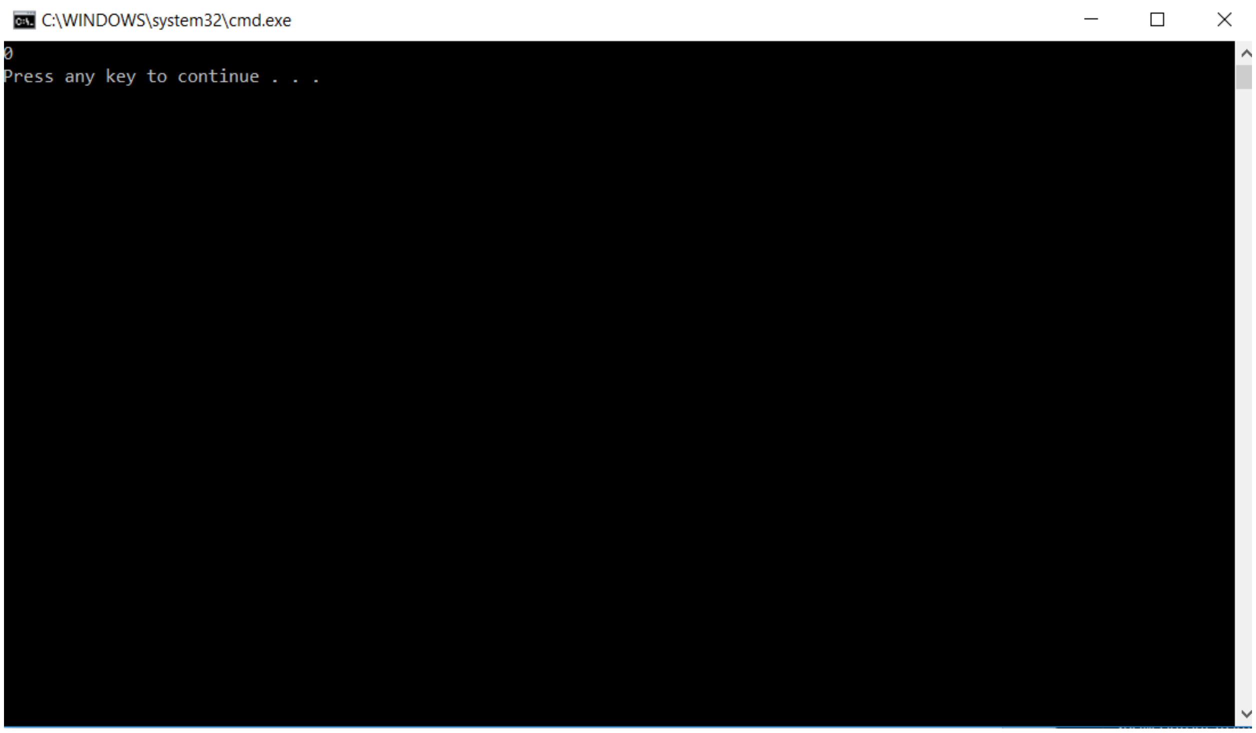
Code:

```
int linkedlist::count()
{
	int count = 0;
	node *p = head;
	if (p == nullptr)
		return 0;
	while (p->next != nullptr)
	{
		p = p->next;
		count++;
	}
	return count+1;
}
```

Explanation: First we need to check if the list is empty then we make a loop to increment a pointer and increment the count while every iteration the pointer isn't null.

Testing: Special case: The list is empty

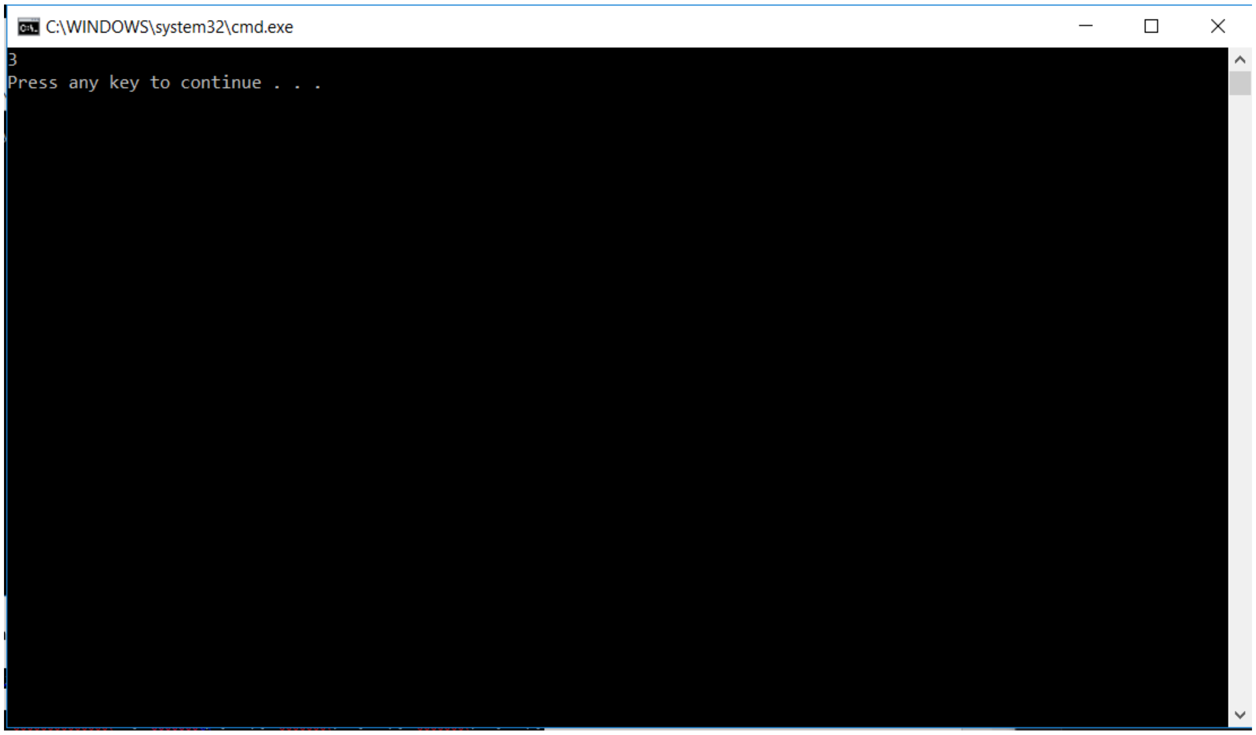
```
int main()
{
	linkedlist l;
	int c = l.count();
	cout << c << endl;
	return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
0
Press any key to continue . . .
```

General case:

```
int main()
{
	linkedlist l; l.add(7, 1); l.add(18, 5); l.add(17, 4);
	int c = l.count();
	cout << c << endl;
	return 0;
}
```



```
3
Press any key to continue . . .
```

Reverse elements:

Code:

```
linkedlist linkedlist:: rev(linkedlist l, int c)
{
	linkedlist rev;
	node *p = head;
	if (p == nullptr)
		return rev;
	for (int j = 0; j < c; j++)
	{
		node *p = head;
		for (int i = 0; i < c - j-1; i++)
		{
			p = p->next;
		}

		rev.add(p->data, j + 1);
```

```
        }
        return rev;
}
```

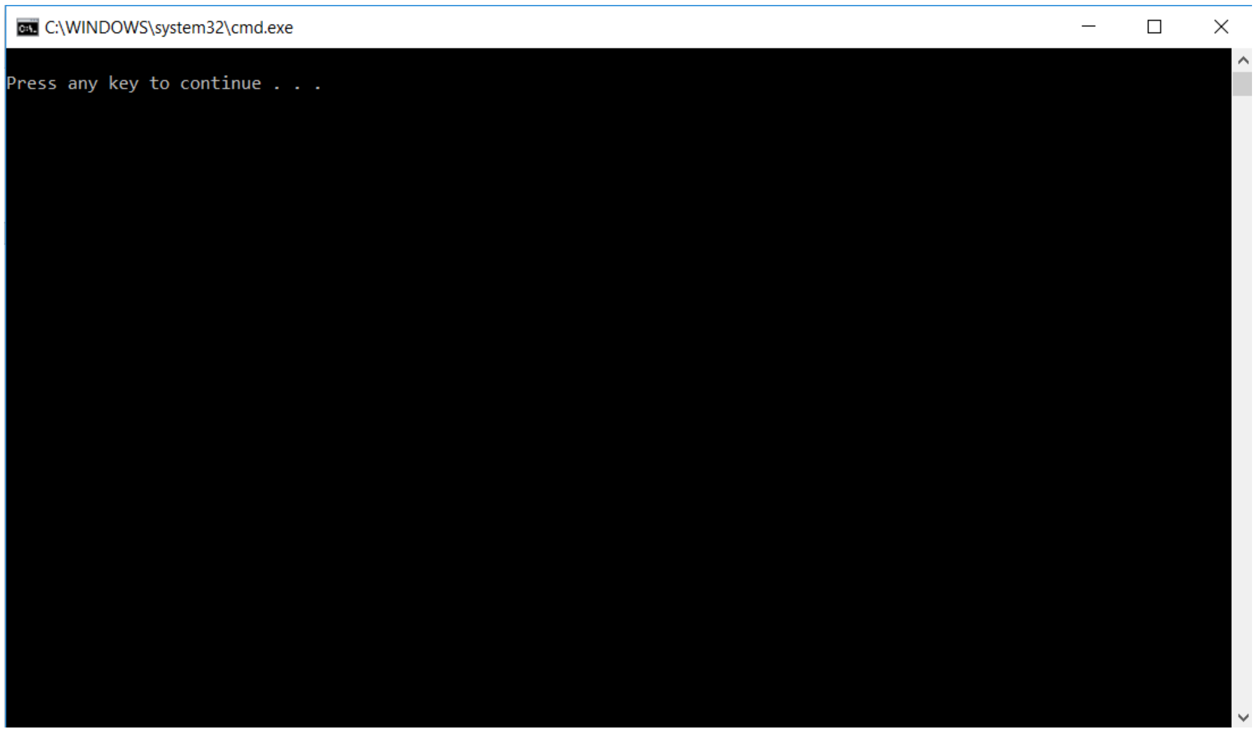
Explanation:

First we need to check if the list is empty and if it is, then the reversed list is empty then we create another list then we obtain the last element by a for loop, add to the reversed list then we repeat the whole process using a for loop.

Testing: Special case

Reversing an empty list

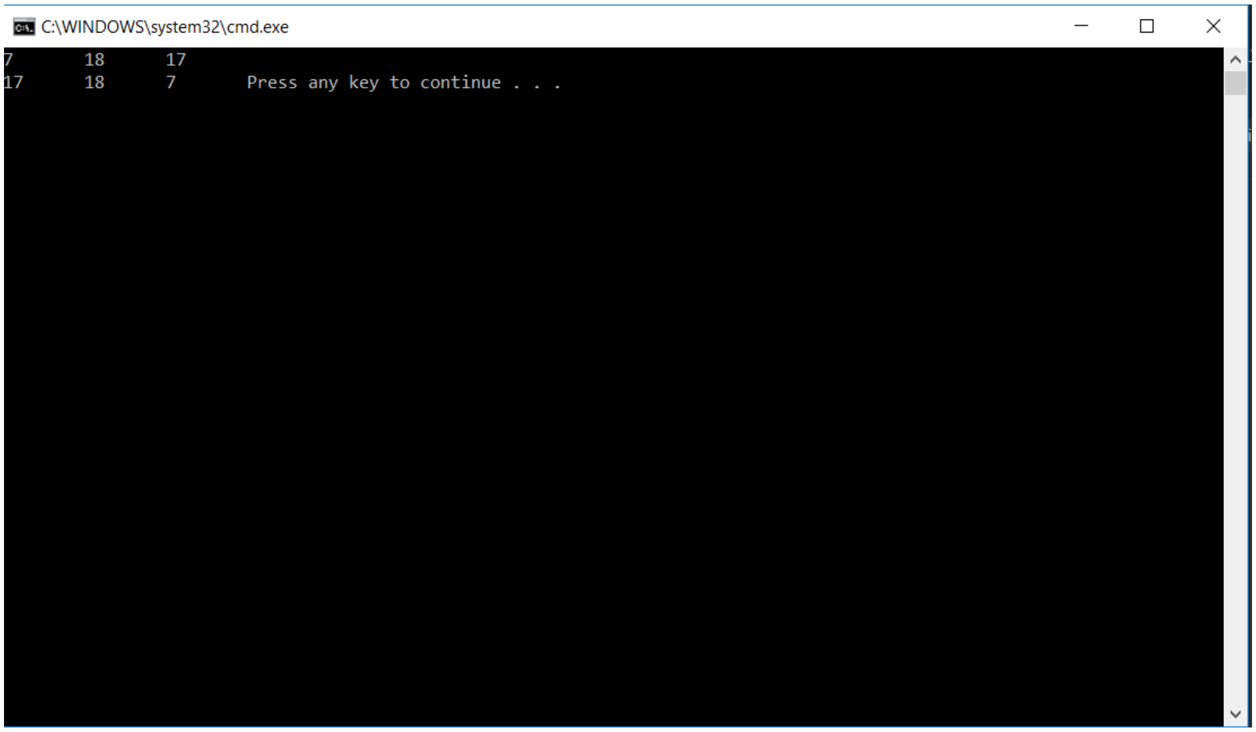
```
int main()
{
        linkedlist l, rev;
        int c = l.count();
        l.printall();
        cout << endl;
        rev = l.rev(l, c);
        rev.printall();
        return 0;
}
```



```
Press any key to continue . . .
```

General case:

```
int main()
{
        linkedlist l, rev;
        l.add(7, 1); l.add(18, 5); l.add(17, 4);
        int c = l.count();
        l.printall();
        cout << endl;
        rev = l.rev(l, c);
        rev.printall();
        return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
7       18      17
17      18      7       Press any key to continue . . .
```

Get the average of a list:

Code

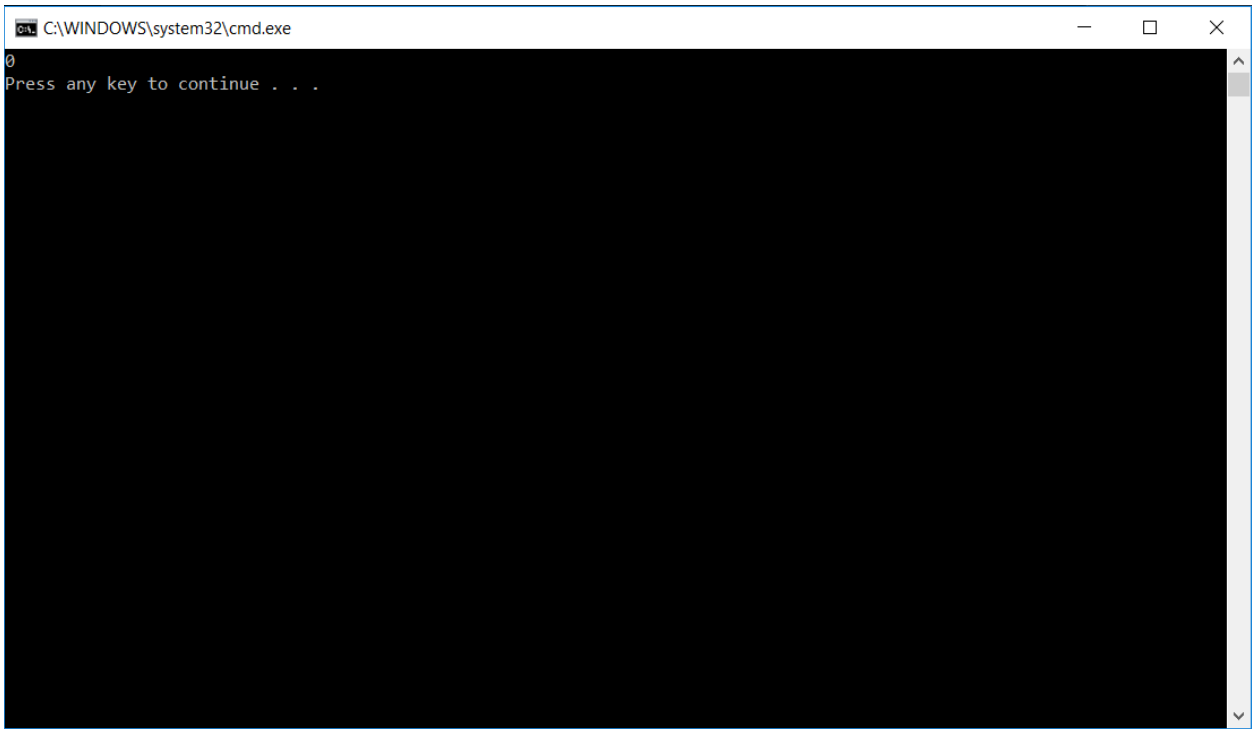
```
double linkedlist::av()
{
        double count = 0; double sum = 0;
        node *p = head;
        if (p == nullptr)
                return 0;
        sum = p->data;
        while (p->next != nullptr)
        {
                p = p->next;
                sum = sum + p->data;
                count++;
        }
        return sum/(count+1);
}
```

Explanation:

First we need to check if the list is empty, if it is, we just return zero then we need to obtain the sum of the list so we create a sum variable then add every element to it using loop then we use the count function that we implemented before then we divide the sum over the count to obtain average

Testing: empty list

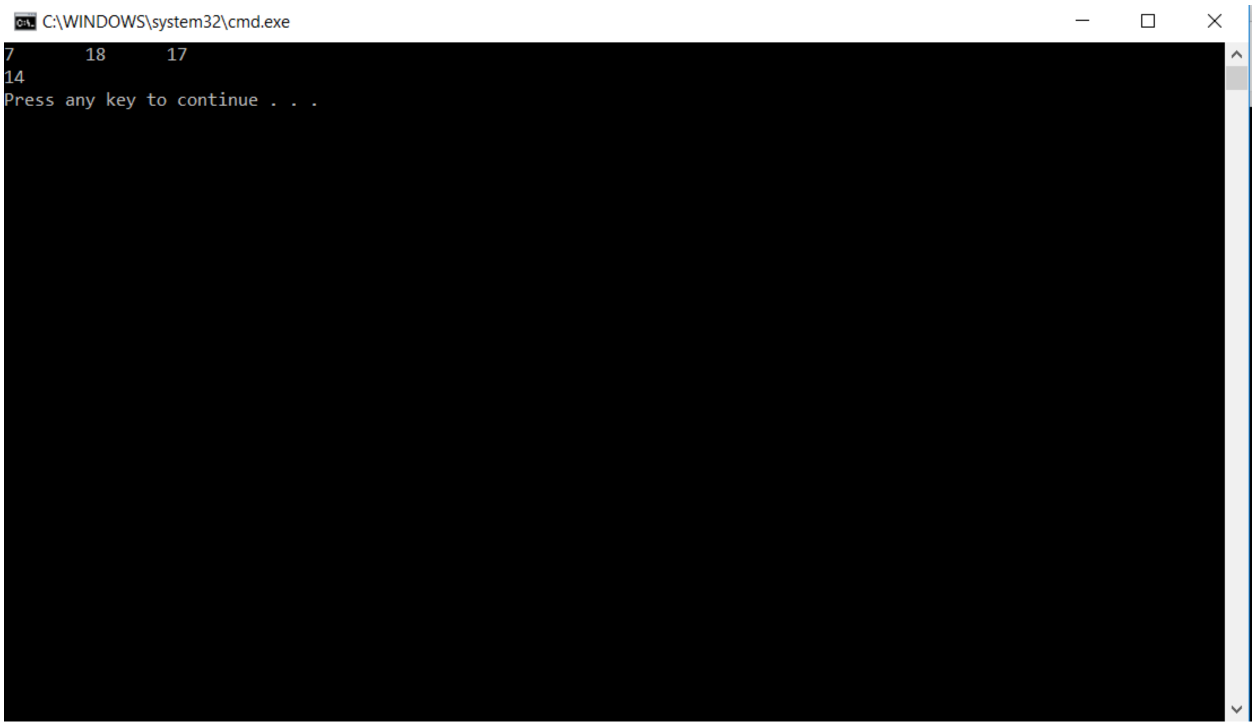
```
 int main()
{
	linkedlist l;
	double av = l.av();
	cout <<av<< endl;
	return 0;
}
```


C:\WINDOWS\system32\cmd.exe

```
0
Press any key to continue . . .
```

General case

```
int main()
{
	linkedlist l, rev;
	l.add(7, 1); l.add(18, 5); l.add(17, 4);
	double av = l.av();
	l.printall();
	cout << endl<< av<<endl;
	return 0;
}
```



```
7       18      17
14
Press any key to continue . . .
```

The polynomials problem:

Construct a polynomial linked list from an array values:

```
class poly
{
private:
	linkedlist coef;
public:
	poly()
	{
		linkedlist coef;
	}
	poly(double arr[], int size)
	{
		for (int i = 0; i < size; i++)
		{
			coef.addend(arr[i]);
		}
	}
	double evaluate(double x);
	node * gethead() { return coef.gethead(); }
	void printpoly(){ coef.printall(); }
	int countpoly(){ return coef.count(); }
	poly differ();
};
```

Explanation:

We construct a class a have a private data member of type list and a constructor which take the array the elements and inserts it into the list

Multiply two polynomials together and the result will be a new polynomial:

Code:

```
poly multi(poly poly1, poly poly2,int size1,int size2)
{
        int deg = (size1 - 1) + (size2 - 1)+1;
        double *arr;
        arr = new double[deg + 1];
        node *p1 = poly1.gethead(); node *p2 = poly2.gethead();
        if (p1 == nullptr || p2 == nullptr)
        {
                arr[0] = 0; deg = 1; poly result(arr, deg); return result;
        }
        double mul;
        for (int j = 0; j< deg; j++)
        {
                arr[j] = 0;
        }
        int i = 0; int j = 0;
        while (p1 != nullptr)
        {
                mul = p1->getData();
                while (p2 != nullptr)
                {
                        arr[i + j] = arr[i + j]+ (mul * p2->getData());
                        p2 = p2->getNext();
                        i++;
                }
                        p2 = poly2.gethead();
                        p1 = p1->getNext();
                        i = 0; j++;
        }
        poly result(arr, deg); return result;
}
```

Explanation:

We first to need if any of the poly is zero then the result will be zero

The idea is that if we take every coef in one poly and multi with the other poly the result will be a set of polys then we need to arrange them as the example follows:

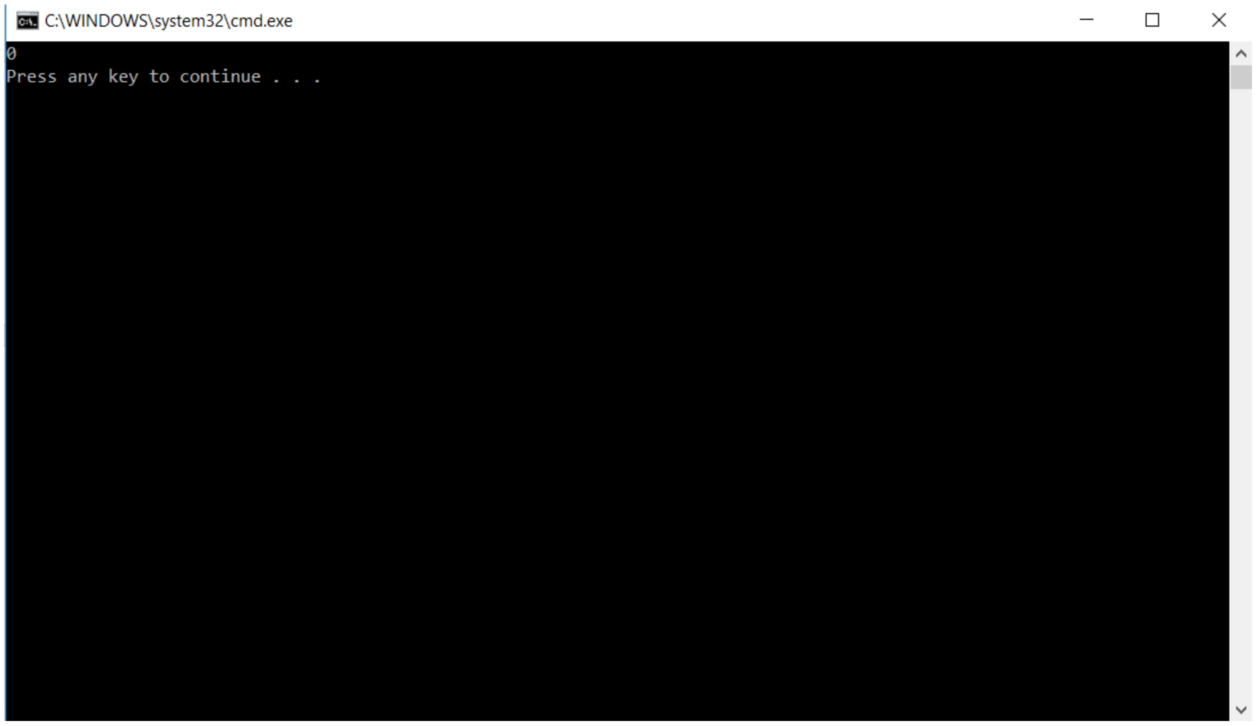
```
4   3   2

     3   2   1

          2   1  0
```
where the numbers is the powers of x that will make the summation "the result poly" much easier

Testing

Special case

One of the polys is zero ”empty list”:

```
int main()
{
	double arr[3] = { 2, 2,2 };
	poly poly1(arr, 3); poly poly2;
	poly result = multi(poly1, poly2, 3, 3);
	result.printpoly();
	return 0;
}
```

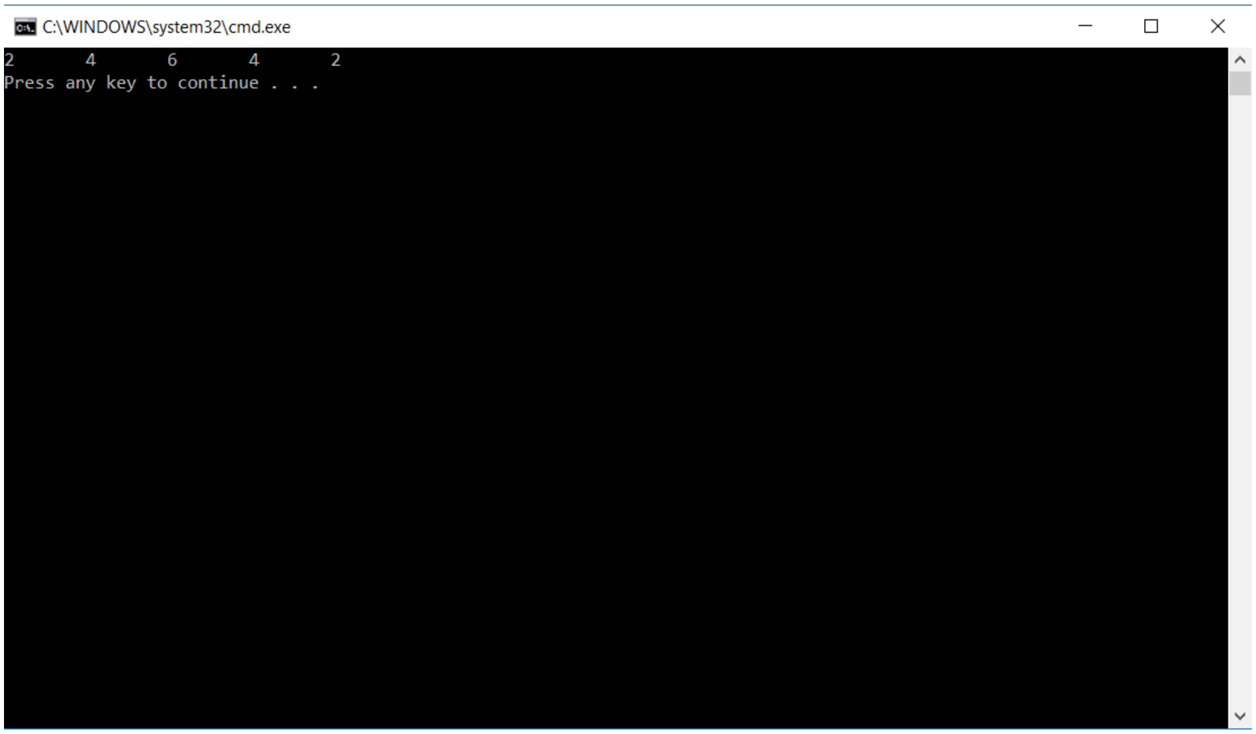


C:\WINDOWS\system32\cmd.exe

```
0
Press any key to continue . . .
```

General case:

```
int main()
{
	double arr1[3] = { 1, 1, 1 };
	double arr[3] = { 2, 2,2 };
	poly poly1(arr, 3); poly poly2(arr1,3);
	poly result = multi(poly1, poly2, 3, 3);
	result.printpoly();
	cout << endl;
	return 0;
}
```



```
2       4       6       4       2
Press any key to continue . . .
```

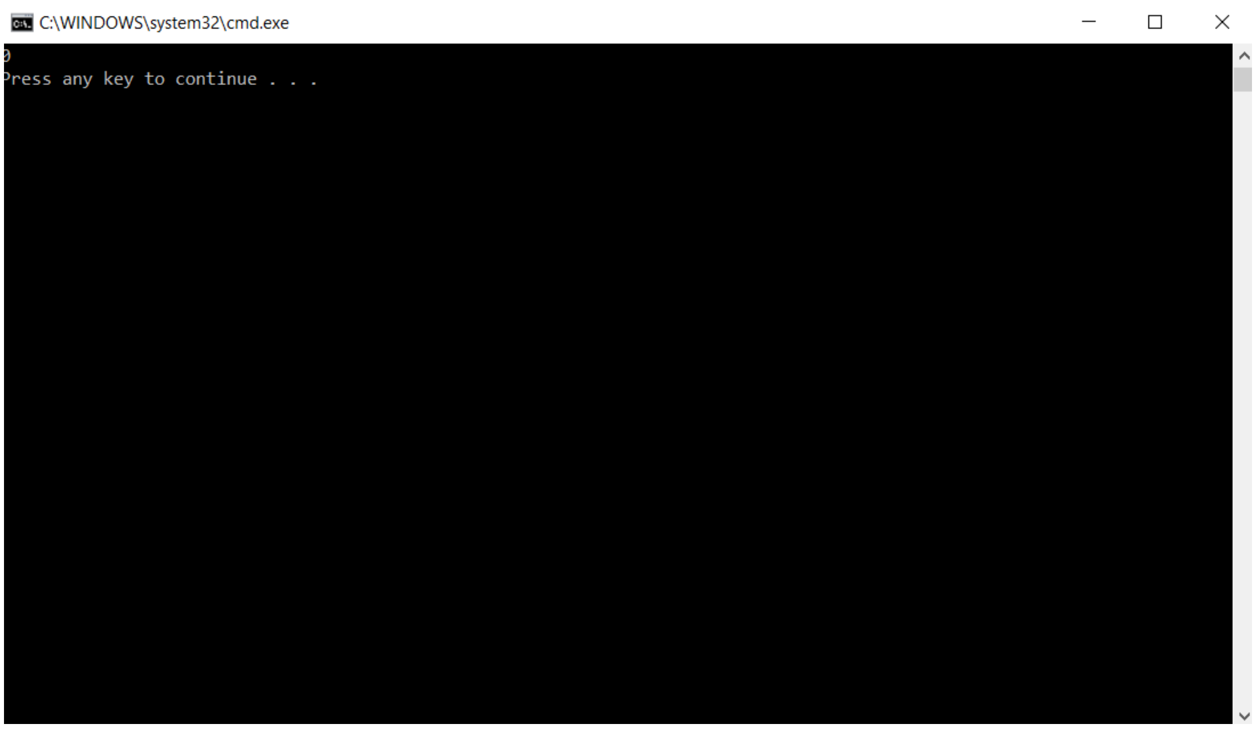
Evaluate a polynomial with a value x

Code:

```
double poly::evaluate(double x)
{
        double sum = 0; node *p = coef.gethead(); int deg = coef.count()-1;
        if (p == nullptr)
        {
                return sum;
        }
        while (p != nullptr)
        {
                sum = sum + (p->getData()*(pow(x,deg)));
                deg--;
                p = p->getNext();
        }
        return sum;
}
```

Testing: Empty list:

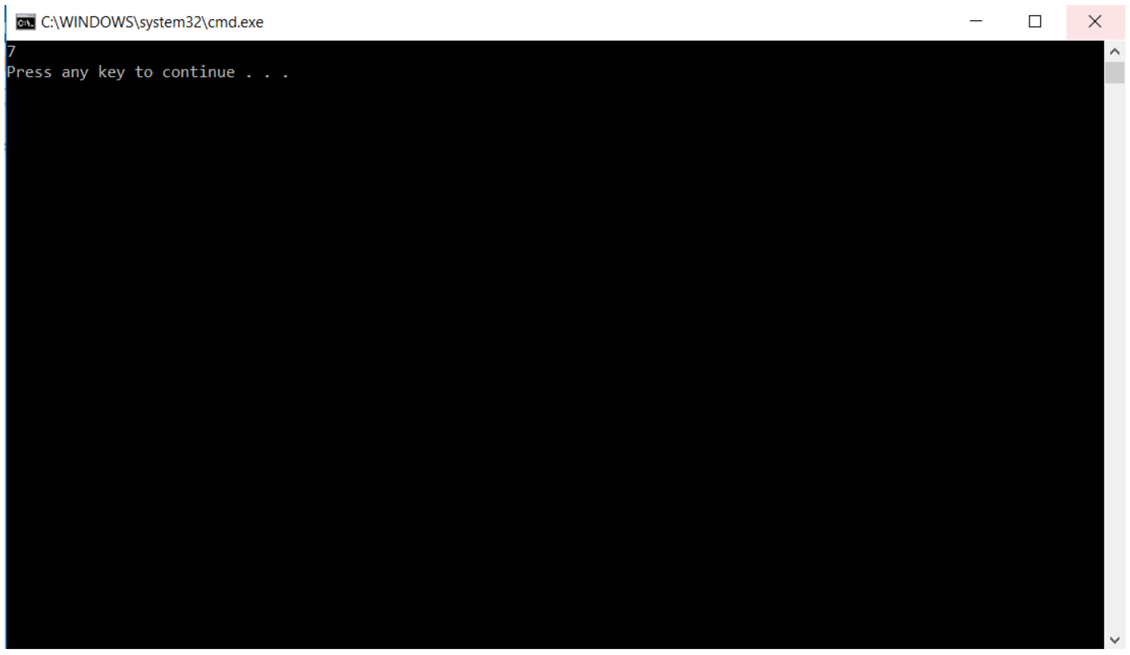
```
int main()
{
        poly poly1; double ev = poly1.evaluate(2);
        cout << ev<<endl;
        return 0;
}
```



```
0
Press any key to continue . . .
```

General case

```
int main()
{
	double arr[3] = { 1, 1, 1 };
	poly poly1(arr, 3); double ev = poly1.evaluate(2);
	cout << ev<<endl;
	return 0;
}
```



```
7
Press any key to continue . . .
```

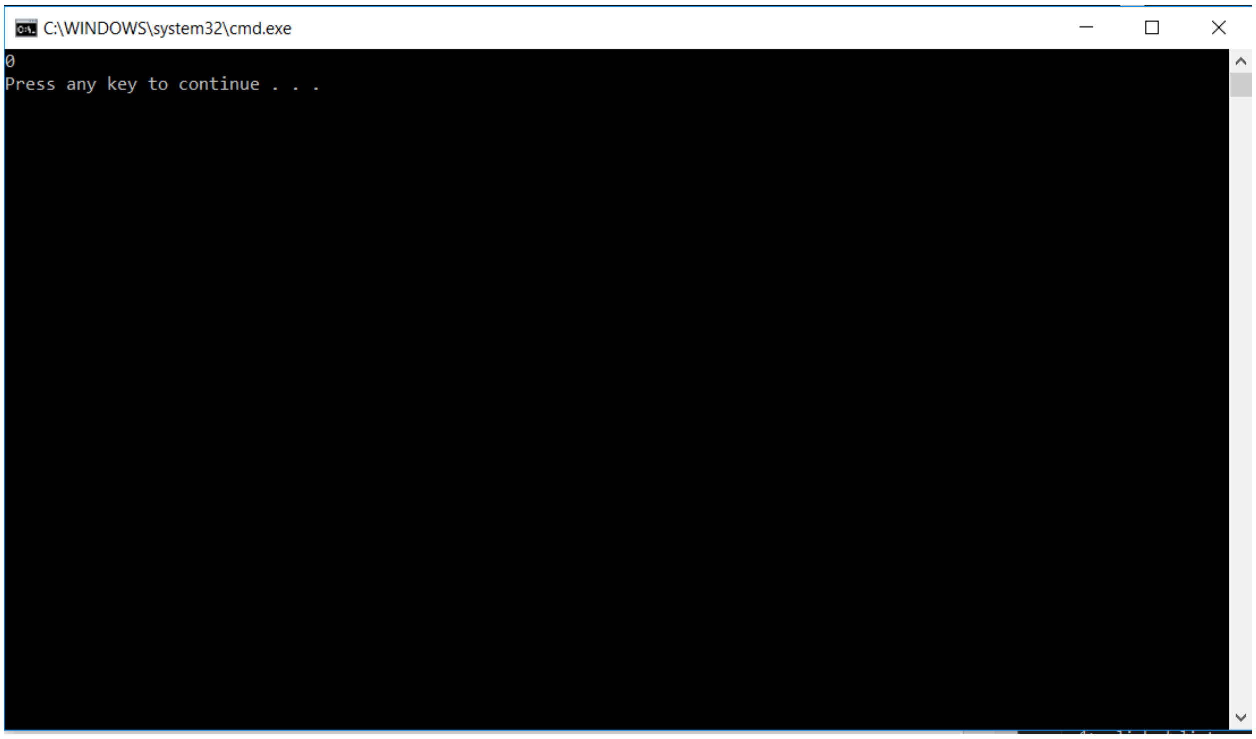
Differentiate the polynomial with respect to x i.e. dP(x)/dx

```
poly poly::differ()
{
	node *p = coef.gethead(); int deg = coef.count() - 1; double *arr;
	arr = new double[deg + 1]; int i = 0; int mul;
	if (p == nullptr)
	{
		arr[0] = 0; deg = 1; poly result(arr, deg); return result;
	}
	while (p != nullptr)
	{
		mul = p->getData();
		arr[i] = mul * (deg - i);
		p = p->getNext(); i++;
	}
	poly result(arr, deg); return result;
}
```

Testing:

Special case: Empty list:

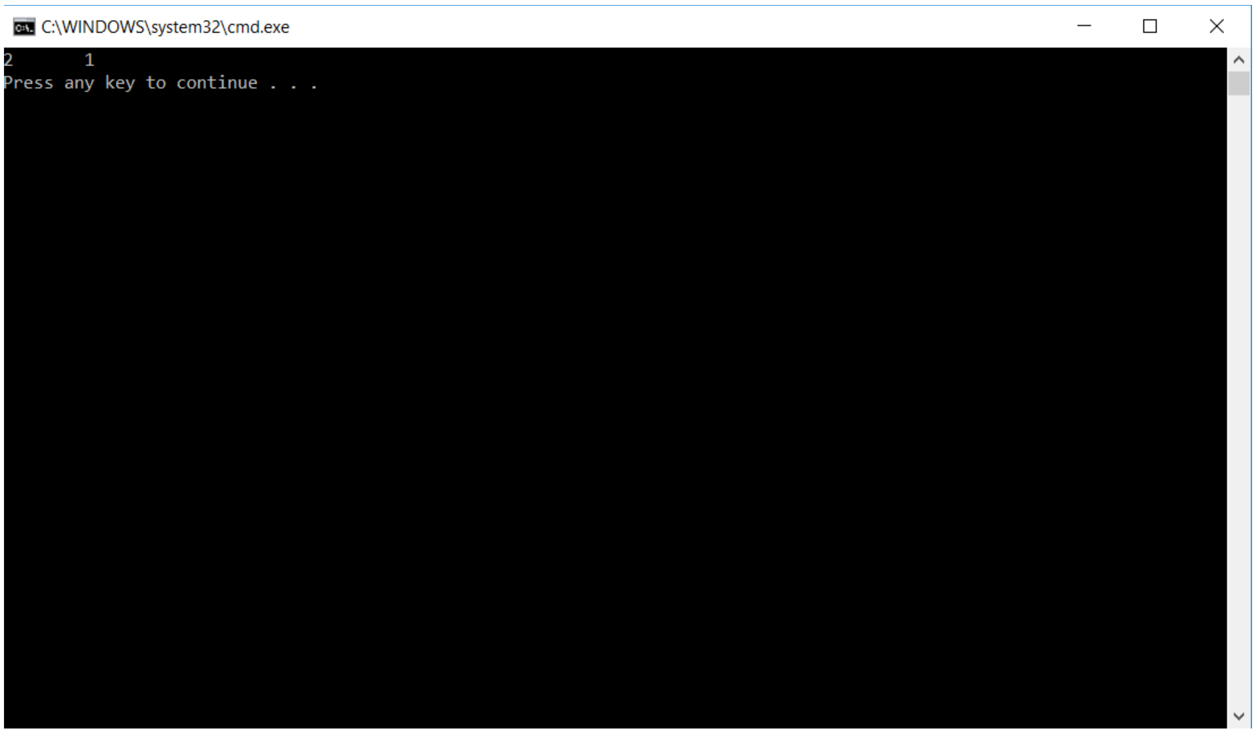
```
int main()
{
	poly poly1; poly result = poly1.differ();
	result.printpoly();
	cout <<endl;
	return 0;
}
```


C:\WINDOWS\system32\cmd.exe

```
0
Press any key to continue . . .
```

General case

```
int main()
{
	double arr[3] = { 1, 1, 1 };
	poly poly1(arr,3); poly result = poly1.differ();
	result.printpoly();
	cout <<endl;
	return 0;
}
```



```
2       1
Press any key to continue . . .
```

The complete code:

```
#include <iostream>
#include <cmath>
using namespace std;
class node
{
private:
	int data;
	node *next;
public:
	node()
	{
		data = 0;
		next = nullptr;
	}
	node(int d)
	{
		data = d;
		next = nullptr;
	}
	int getData() { return data; }
	void setData(int d) { data = d; }
	node * getNext() { return next; }
	void setNext(node * n) { next = n; }
	friend class linkedlist;
};
class linkedlist
{
private:
	node * head;
public:
	linkedlist()
	{
		head = nullptr;
	}
	linkedlist(int d)
	{
		head = new node(d);
	}
	void printall();
	void addfront(int item);
	void addend(int item);
	void add(int item, int pos);
	int count();
	double av();
	bool isempty();
	linkedlist rev(linkedlist l, int c);
	node * gethead() { return head; }
};
void linkedlist::printall()
{
	node *p = head;
	while (p != NULL)
	{
		cout << p->data << "\t";
		 p=p->next;
	}
}
void linkedlist::addfront(int item)
{
	node *temp = new node(item);

	if (head == nullptr)
	{
		head = temp;
	}
	else
	{
		temp->next = head;
		head = temp;
	}
}
void linkedlist::addend(int item)
{
	if (head == nullptr)
	{
		addfront(item);
```

```
        }
        else
        {
                node *p=head;
                while (p->next != nullptr)
                {
                        p=p->next;
                }
                node *n = new node(item);
                p->next = n;
        }
}
void linkedlist::add(int item, int pos)
{
        if (pos == 1)
                addfront(item);
        else{
                node *temp1 = new node(item);
                node *temp = head;
                if (head == nullptr)
                {
                        addfront(item);
                }
                else
                {
                        for (int i = 0; i < pos - 2; i++)
                        {
                                temp = temp->next;
                                if (temp == nullptr)
                                {
                                        addend(item);
                                        return;
                                }

                        }
                        temp1->next = temp->next;
                        temp->next = temp1;
                }

        }
}
int linkedlist::count()
{
        int count = 0;
        node *p = head;
        if (p == nullptr)
                return 0;
        while (p->next != nullptr)
        {
                p = p->next;
                count++;
        }
        return count+1;
}
double linkedlist::av()
{
        double count = 0; double sum = 0;
        node *p = head;
```

```
	if (p == nullptr)
		return 0;
	sum = p->data;
	while (p->next != nullptr)
	{
		p = p->next;
		sum = sum + p->data;
		count++;
	}
	return sum/(count+1);
}
bool linkedlist::isempty()
{
	if (head == nullptr)
		return true;
	else
		return false;
}
linkedlist linkedlist:: rev(linkedlist l, int c)
{
	linkedlist rev;
	node *p = head;
	if (p == nullptr)
		return rev;
	for (int j = 0; j < c; j++)
	{
		node *p = head;
		for (int i = 0; i < c - j-1; i++)
		{
			p = p->next;
		}
		rev.add(p->data, j + 1);
	}
	return rev;
}
class poly
{
private:
	linkedlist coef;
public:
	poly()
	{
		linkedlist coef;
	}
	poly(double arr[], int size)
	{
		for (int i = 0; i < size; i++)
		{
			coef.addend(arr[i]);
		}
	}
	double evaluate(double x);
	node * gethead() { return coef.gethead(); }
	void printpoly(){ coef.printall(); }
	int countpoly(){ return coef.count(); }
	poly differ();
};
poly poly::differ()
```

```
{
        node *p = coef.gethead(); int deg = coef.count() - 1; double *arr;
        arr = new double[deg + 1]; int i = 0; int mul;
        if (p == nullptr)
        {
                arr[0] = 0; deg = 1; poly result(arr, deg); return result;
        }
        while (p != nullptr)
        {
                mul = p->getData();
                arr[i] = mul * (deg - i);
                p = p->getNext(); i++;
        }
        poly result(arr, deg); return result;
}
double poly::evaluate(double x)
{
        double sum = 0; node *p = coef.gethead(); int deg = coef.count()-1;
        if (p == nullptr)
        {
                return sum;
        }
        while (p != nullptr)
        {
                sum = sum + (p->getData()*(pow(x,deg)));
                deg--;
                p = p->getNext();
        }
        return sum;
}
poly multi(poly poly1, poly poly2,int size1,int size2)
{
        int deg = (size1 - 1) + (size2 - 1)+1;
        double *arr;
        arr = new double[deg + 1];
        node *p1 = poly1.gethead(); node *p2 = poly2.gethead();
        if (p1 == nullptr || p2 == nullptr)
        {
                arr[0] = 0; deg = 1; poly result(arr, deg); return result;
        }
        double mul;
        for (int j = 0; j< deg; j++)
        {
                arr[j] = 0;
        }
        int i = 0; int j = 0;
        while (p1 != nullptr)
        {
                mul = p1->getData();
                while (p2 != nullptr)
                {
                        arr[i + j] = arr[i + j]+ (mul * p2->getData());
                        p2 = p2->getNext();
                        i++;
                }
                        p2 = poly2.gethead();
                        p1 = p1->getNext();
                        i = 0; j++;
        }
        poly result(arr, deg); return result;
}
int main()
{
        //your operations here
}
```