**Question#1**

1. **O(log n)**
2. **O(log n)**
3. **O(N^2)**
4. **O(N^3)**
5. **O(N^2)**
6. **O(N\*sqrt(N))**
7. **O(N)**
8. **O(N^2)**
9. **O(N^2)**
10. **O(N)**

**Question#2**

**Code:**

#include<iostream>

using namespace std;

struct node {

int data;

node\* next;

node() :data(0), next(nullptr){};

};

class BaseSingly {

protected:

int count;

node\* head;

node\* current;

public:

BaseSingly() {

count = 0;

head = nullptr;

current = nullptr;

}

virtual void insert\_at\_front(int val) = 0;

virtual void insert\_at\_end(int val) = 0;

virtual void insert\_at\_middle(int index, int val) = 0;

virtual void delete\_at\_front() = 0;

virtual void delete\_at\_end() = 0;

virtual void delete\_at\_index(int index) = 0;

virtual void print() = 0;

virtual void search(int val) = 0;

};

class singlylist : public BaseSingly {

public:

singlylist():BaseSingly() {}

~singlylist(){

for (int i = 1; i <= count; i++) {

delete\_at\_end();

}

}

void insert\_at\_front(int val) {

if (head == nullptr) {

head = new node;

current = head;

head->data = val;

count++;

}

else {

node\* temp = new node;

temp->data = val;

temp->next = head;

head = temp;

count++;

}

}

void insert\_at\_end(int val) {

if (head == nullptr) {

insert\_at\_front(val);

}

else {

current = head;

for (int i = 1; i < count; i++) {

current = current->next;

}

node\* temp = new node;

temp->data = val;

current->next = temp;

count++;

}

}

void insert\_at\_middle(int index, int val) {

if (index< 0 || index>count) {

cout << "Invalid Index" << endl;

}

else {

current = head;

for (int i = 1; i < index; i++) {

current = current->next;

}

for (int i = 1; i <= ((count - index) / 2); i++) {

current = current->next;

}

node\* temp = new node;

temp->data = val;

temp->next = current->next;

current->next = temp;

count++;

}

}

void delete\_at\_front() {

current = head;

head = current->next;

delete current;

current = head;

count--;

}

void delete\_at\_end() {

current = head;

for (int i = 1; i <= count; i++) {

current = current->next;

}

delete current;

count--;

current = head;

for (int i = 1; i <= count; i++) {

current = current->next;

}

current->next = nullptr;

}

void delete\_at\_index(int index) {

current = head;

if (index == 1) {

delete\_at\_front();

}

else if (index == count) {

delete\_at\_end();

}

else if (index > 1 && index < count) {

current = head->next;

for (int i = 2; i < index - 1; i++) {

current = current->next;

}

node\* temp = new node;

temp = current->next;

current->next = current->next->next;

delete temp;

count--;

}

else {

cout << "Invalid index" << endl;

}

}

virtual void search(int val) {

current = head;

bool found = false;

for (int i = 1; i < count; i++) {

if (current->data == val) {

cout << "Found at Index : "<< i << endl;

found = true;

break;

}

current = current->next;

}

if (found == false) {

cout << "Not Found" << endl;

}

}

void print() {

current = head;

for (int i = 1; i <= count; i++) {

cout << "[" << current->data << "]";

current = current->next;

}

cout << endl;

}

};

int main() {

singlylist obj;

cout << "Insert at End" << endl;

for (int i = 1; i <= 5; i++) {

obj.insert\_at\_end(i);

}

obj.print();

obj.print();

cout << "Insert at Front" << endl;

obj.insert\_at\_front(1);

obj.insert\_at\_front(2);

obj.print();

cout << "Insert at Middle" << endl;

obj.insert\_at\_middle(3, 69);

obj.print();

obj.search(69);

cout << "Delete at Front" << endl;

obj.delete\_at\_front();

obj.print();

cout << "Delete at End" << endl;

obj.delete\_at\_end();

obj.print();

cout << "Delete at Index" << endl;

obj.delete\_at\_index(3);

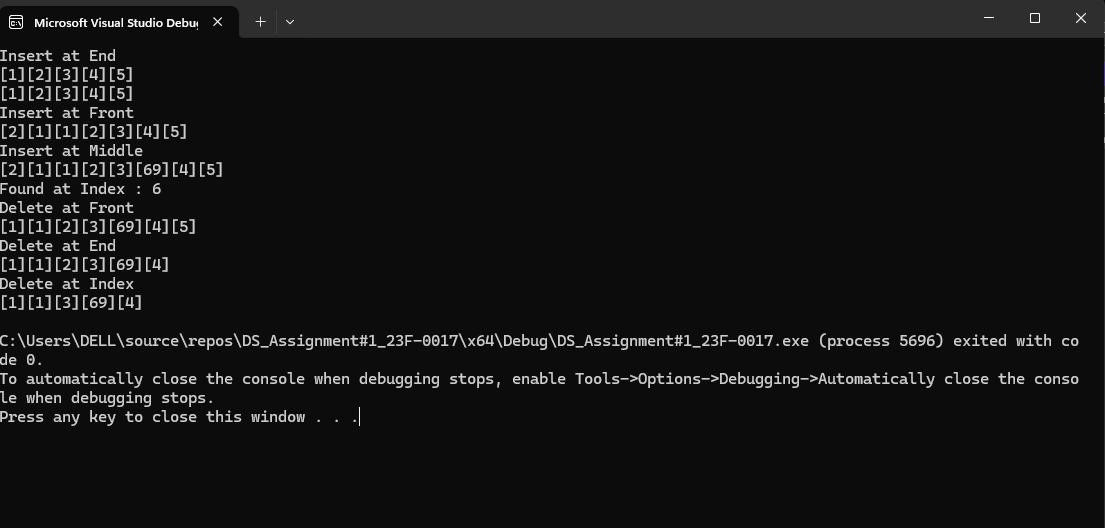
obj.print();

obj.~singlylist();

return 0;

}

**Output:**

****

**Question#3**

**Code:**

#include<iostream>

using namespace std;

struct node {

int data;

node\* next;

node\* prev;

node() :next(nullptr),prev(nullptr),data(0){}

};

class BaseDoubly {

protected:

node\* head;

node\* current;

node\* tail;

int count;

public:

BaseDoubly() {

head = nullptr;

tail = nullptr;

count = 0;

}

virtual void insert\_at\_front(int val) = 0;

virtual void insert\_at\_end(int val) = 0;

virtual void insert\_at\_middle(int index, int val) = 0;

virtual void delete\_at\_front() = 0;

virtual void delete\_at\_back() = 0;

virtual void delete\_at\_index(int index) = 0;;

virtual void print() = 0;

virtual void search(int val) = 0;

};

class doublyList :public BaseDoubly{

private:

node\* head;

node\* current;

node\* tail;

int count;

public:

doublyList():BaseDoubly() {}

~doublyList() {

for (int i = 1; i <= count; i++) {

delete\_at\_back();

}

}

void insert\_at\_front(int val){

if (head == nullptr) {

head = new node;

head->next = nullptr;

head->prev = nullptr;

head->data = val;

current = head;

count++;

}

else {

node\* temp = new node;

temp->data = val;

temp->next = head;

head->prev = temp;

tail = current;

head = temp;

count++;

}

}

void insert\_at\_end(int val) {

if (head == nullptr) {

insert\_at\_front(val);

}

else {

node\* temp = new node;

temp->data = val;

current = head;

for (int i = 1; i < count; i++) {

current = current->next;

}

current->next = temp;

temp->prev = current;

tail = current->next;

count++;

}

}

void insert\_at\_middle(int index, int val) {

if (index< 0 || index>count) {

cout << "Invalid Index" << endl;

}

else {

current = head;

for (int i = 1; i < index; i++) {

current = current->next;

}

for (int i = 1; i <= ((count - index) / 2); i++) {

current = current->next;

}

node\* temp = new node;

temp->data = val;

temp->next = current->next;

current->next->prev = temp;

temp->prev = current;

current->next = temp;

count++;

}

}

void delete\_at\_front() {

current = head;

head = current->next;

head->prev = nullptr;

delete current;

current = head;

count--;

}

void delete\_at\_back() {

current = tail;

tail = current->prev;

tail->next = nullptr;

delete current;

current = head;

count--;

}

void delete\_at\_index(int index) {

if (index == 1) {

delete\_at\_front();

}

else if (index == count) {

delete\_at\_back();

}

else if (index > 1 && index < count) {

current = head->next;

for (int i = 2; i < index; i++) {

current = current->next;

}

current->prev->next = current->next;

current->next->prev = current->prev;

delete current;

count--;

}

else {

cout << "Invalid Index" << endl;

}

}

void reverseDoubly() {

current = head;

node\* tempPrev = tail;

for (int i = 1; i <= count / 2; i++) {

int temp = current->data;

current->data = tempPrev->data;

tempPrev->data = temp;

current = current->next;

tempPrev = tempPrev->prev;

}

}

void middleDoubly() {

current = head;

node\* temp = tail;

for (int i = 1; i <= count/2; i++) {

current = current->next;

temp = temp->prev;

}

cout << "Middle Element : " << current->data << endl;

cout << "Middle Element : " << temp->data << endl;

}

bool palindromeDoubly() {

current = head;

node\* temp = tail;

for (int i = 1; i <= count / 2; i++) {

current = current->next;

temp = temp->prev;

if (current->data != temp->data) {

return false;

}

}

return true;

}

void search(int val) {

current = head;

bool found = false;

for (int i = 1; i <= count; i++) {

if (current->data == val) {

cout << "Found at Index : " << i << endl;

found = true;

break;

}

current = current->next;

}

if (found == false) {

cout << "Not Found" << endl;

}

}

void print() {

current = head;

for (int i = 1; i <= count; i++) {

cout << "[" << current->data << "]";

current = current->next;

}

cout << endl;

}

};

int main() {

doublyList obj;

cout << "Insert at Front" << endl;

for (int i = 5; i > 0; i--) {

obj.insert\_at\_front(i);

}

obj.print();

cout << "Insert at End" << endl;

for (int i = 1; i < 6; i++) {

obj.insert\_at\_end(i);

}

obj.print();

cout << "Delete at Back" << endl;

obj.delete\_at\_back();

obj.print();

cout << "Delete at Front" << endl;

obj.delete\_at\_front();

obj.print();

cout << "Delete at Index" << endl;

obj.delete\_at\_index(2);

obj.print();

cout << "Insert at Middle" << endl;

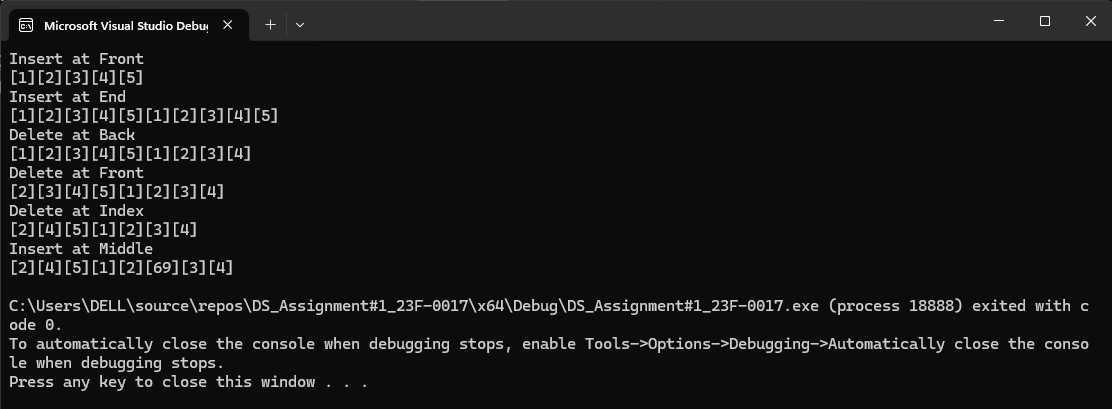
obj.insert\_at\_middle(3, 69);

obj.print();

return 0;

}

**Output:**

****

**Question#4**

**Code:**

#include<iostream>

using namespace std;

struct node {

int data;

node\* next;

node() :data(0), next(nullptr){};

};

class BaseSingly {

protected:

int count;

node\* head;

node\* current;

public:

BaseSingly() {

count = 0;

head = nullptr;

current = nullptr;

}

virtual void insert\_at\_front(int val) = 0;

virtual void insert\_at\_end(int val) = 0;

virtual void insert\_at\_middle(int index, int val) = 0;

virtual void delete\_at\_front() = 0;

virtual void delete\_at\_end() = 0;

virtual void delete\_at\_index(int index) = 0;

virtual void print() = 0;

virtual void search(int val) = 0;

virtual void reverse() = 0;

virtual void removeDuplicates() = 0;

};

class singlylist : public BaseSingly {

public:

singlylist():BaseSingly() {}

~singlylist(){

for (int i = 1; i <= count; i++) {

delete\_at\_end();

}

}

void insert\_at\_front(int val) {

if (head == nullptr) {

head = new node;

current = head;

head->data = val;

count++;

}

else {

node\* temp = new node;

temp->data = val;

temp->next = head;

head = temp;

count++;

}

}

void insert\_at\_end(int val) {

if (head == nullptr) {

insert\_at\_front(val);

}

else {

current = head;

for (int i = 1; i < count; i++) {

current = current->next;

}

node\* temp = new node;

temp->data = val;

current->next = temp;

count++;

}

}

void insert\_at\_middle(int index, int val) {

if (index< 0 || index>count) {

cout << "Invalid Index" << endl;

}

else {

current = head;

for (int i = 1; i < index; i++) {

current = current->next;

}

for (int i = 1; i <= ((count - index) / 2); i++) {

current = current->next;

}

node\* temp = new node;

temp->data = val;

temp->next = current->next;

current->next = temp;

count++;

}

}

void delete\_at\_front() {

current = head;

head = current->next;

delete current;

current = head;

count--;

}

void delete\_at\_end() {

current = head;

for (int i = 1; i <= count; i++) {

current = current->next;

}

delete current;

count--;

current = head;

for (int i = 1; i <= count; i++) {

current = current->next;

}

current->next = nullptr;

}

void delete\_at\_index(int index) {

current = head;

if (index == 1) {

delete\_at\_front();

}

else if (index == count) {

delete\_at\_end();

}

else if (index > 1 && index < count) {

current = head->next;

for (int i = 2; i < index - 1; i++) {

current = current->next;

}

node\* temp = current->next;

current->next = current->next->next;

delete temp;

count--;

}

else {

cout << "Invalid index" << endl;

}

}

virtual void search(int val) {

current = head;

bool found = false;

for (int i = 1; i < count; i++) {

if (current->data == val) {

cout << "Found at Index : "<< i << endl;

found = true;

break;

}

current = current->next;

}

if (found == false) {

cout << "Not Found" << endl;

}

}

void print() {

current = head;

for (int i = 1; i <= count; i++) {

cout << "[" << current->data << "]";

current = current->next;

}

cout << endl;

}

void removeDuplicates() {

current = head;

for (int i = 1; i <= count; i++) {

if (current->data == current->next->data) {

delete\_at\_index(i);

}

current = current->next;

}

}

void reverse() {

node\* prev = nullptr;

current = head;

node\* next = nullptr;

while (current != nullptr) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

}

};

int main() {

singlylist obj;

cout << "Insert at End" << endl;

for (int i = 1; i <= 5; i++) {

obj.insert\_at\_end(i);

}

obj.print();

cout << "Reverse" << endl;

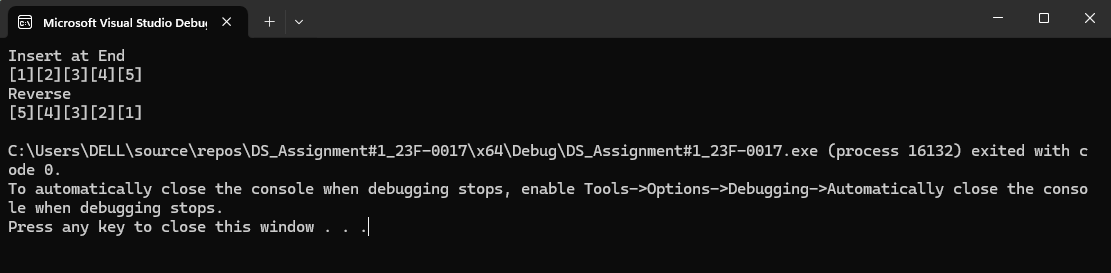
obj.reverse();

obj.print();

return 0;

}

**Output:**

****

**Question#5**

**Code:**

#include<iostream>

using namespace std;

struct node {

int data;

node\* next;

node\* prev;

node() :next(nullptr),prev(nullptr),data(0){}

};

class BaseDoubly {

protected:

node\* head;

node\* current;

node\* tail;

int count;

public:

BaseDoubly() {

head = nullptr;

tail = nullptr;

count = 0;

}

virtual void insert\_at\_front(int val) = 0;

virtual void insert\_at\_end(int val) = 0;

virtual void insert\_at\_middle(int index, int val) = 0;

virtual void delete\_at\_front() = 0;

virtual void delete\_at\_back() = 0;

virtual void delete\_at\_index(int index) = 0;;

virtual void print() = 0;

virtual void search(int val) = 0;

virtual void sort() = 0;

};

class doublyList :public BaseDoubly{

private:

node\* head;

node\* current;

node\* tail;

int count;

public:

doublyList():BaseDoubly() {}

~doublyList() {

for (int i = 1; i <= count; i++) {

delete\_at\_back();

}

}

void insert\_at\_front(int val){

if (head == nullptr) {

head = new node;

head->next = nullptr;

head->prev = nullptr;

head->data = val;

current = head;

count++;

}

else {

node\* temp = new node;

temp->data = val;

temp->next = head;

head->prev = temp;

tail = current;

head = temp;

count++;

}

}

void insert\_at\_end(int val) {

if (head == nullptr) {

insert\_at\_front(val);

}

else {

node\* temp = new node;

temp->data = val;

current = head;

for (int i = 1; i < count; i++) {

current = current->next;

}

current->next = temp;

temp->prev = current;

tail = current->next;

count++;

}

}

void insert\_at\_middle(int index, int val) {

if (index< 0 || index>count) {

cout << "Invalid Index" << endl;

}

else {

current = head;

for (int i = 1; i < index; i++) {

current = current->next;

}

for (int i = 1; i <= ((count - index) / 2); i++) {

current = current->next;

}

node\* temp = new node;

temp->data = val;

temp->next = current->next;

current->next->prev = temp;

temp->prev = current;

current->next = temp;

count++;

}

}

void delete\_at\_front() {

current = head;

head = current->next;

head->prev = nullptr;

delete current;

current = head;

count--;

}

void delete\_at\_back() {

current = tail;

tail = current->prev;

tail->next = nullptr;

delete current;

current = head;

count--;

}

void delete\_at\_index(int index) {

if (index == 1) {

delete\_at\_front();

}

else if (index == count) {

delete\_at\_back();

}

else if (index > 1 && index < count) {

current = head->next;

for (int i = 2; i < index; i++) {

current = current->next;

}

current->prev->next = current->next;

current->next->prev = current->prev;

delete current;

count--;

}

else {

cout << "Invalid Index" << endl;

}

}

void reverseDoubly() {

current = head;

node\* tempPrev = tail;

for (int i = 1; i <= count / 2; i++) {

int temp = current->data;

current->data = tempPrev->data;

tempPrev->data = temp;

current = current->next;

tempPrev = tempPrev->prev;

}

}

void middleDoubly() {

current = head;

node\* temp = tail;

for (int i = 1; i <= count/2; i++) {

current = current->next;

temp = temp->prev;

}

cout << "Middle Element : " << current->data << endl;

cout << "Middle Element : " << temp->data << endl;

}

bool palindromeDoubly() {

current = head;

node\* temp = tail;

for (int i = 1; i <= count / 2; i++) {

current = current->next;

temp = temp->prev;

if (current->data != temp->data) {

return false;

}

}

return true;

}

void search(int val) {

current = head;

bool found = false;

for (int i = 1; i <= count; i++) {

if (current->data == val) {

cout << "Found at Index : " << i << endl;

found = true;

break;

}

current = current->next;

}

if (found == false) {

cout << "Not Found" << endl;

}

}

void print() {

current = head;

for (int i = 1; i <= count; i++) {

cout << "[" << current->data << "]";

current = current->next;

}

cout << endl;

}

void sort() {

for (int i = 1; i <= count; i++) {

current = head;

for (int j = 1; j <= count - i; j++) {

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

int temp = current->data;

current->data = current->next->data;

current->next->data = temp;

}

current = current->next;

}

}

}

};

int main() {

doublyList obj;

cout << "Insert at Front" << endl;

for (int i = 1; i <= 5; i++) {

obj.insert\_at\_front(i);

}

obj.print();

cout << "Sort" << endl;

obj.sort();

obj.sort();

obj.print();

//cout << "Insert at End" << endl;

//for (int i = 1; i < 6; i++) {

// obj.insert\_at\_end(i);

//}

//obj.print();

//cout << "Delete at Back" << endl;

//obj.delete\_at\_back();

//obj.print();

//cout << "Delete at Front" << endl;

//obj.delete\_at\_front();

//obj.print();

//cout << "Delete at Index" << endl;

//obj.delete\_at\_index(2);

//obj.print();

//cout << "Insert at Middle" << endl;

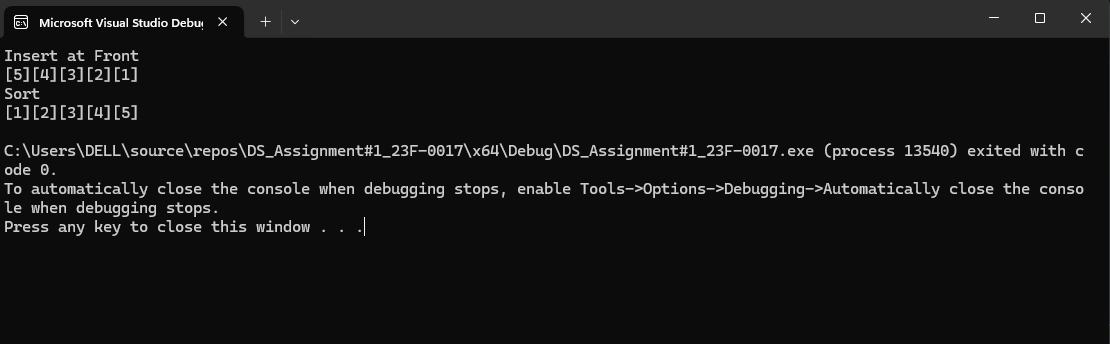
//obj.insert\_at\_middle(3, 69);

//obj.print();

return 0;

}

**Output:**

****

**Question#6**

**Code:**

#include<iostream>

using namespace std;

struct node {

int data;

node\* next;

node() :data(0), next(nullptr){};

};

class BaseSingly {

protected:

int count;

node\* head;

node\* current;

public:

BaseSingly() {

count = 0;

head = nullptr;

current = nullptr;

}

virtual void insert\_at\_front(int val) = 0;

virtual void insert\_at\_end(int val) = 0;

virtual void insert\_at\_middle(int index, int val) = 0;

virtual void delete\_at\_front() = 0;

virtual void delete\_at\_end() = 0;

virtual void delete\_at\_index(int index) = 0;

virtual void print() = 0;

virtual void search(int val) = 0;

virtual void removeDuplicates() = 0;

};

class singlylist : public BaseSingly {

public:

singlylist():BaseSingly() {}

~singlylist(){

for (int i = 1; i <= count; i++) {

delete\_at\_end();

}

}

void insert\_at\_front(int val) {

if (head == nullptr) {

head = new node;

current = head;

head->data = val;

count++;

}

else {

node\* temp = new node;

temp->data = val;

temp->next = head;

head = temp;

count++;

}

}

void insert\_at\_end(int val) {

if (head == nullptr) {

insert\_at\_front(val);

}

else {

current = head;

for (int i = 1; i < count; i++) {

current = current->next;

}

node\* temp = new node;

temp->data = val;

current->next = temp;

count++;

}

}

void insert\_at\_middle(int index, int val) {

if (index< 0 || index>count) {

cout << "Invalid Index" << endl;

}

else {

current = head;

for (int i = 1; i < index; i++) {

current = current->next;

}

for (int i = 1; i <= ((count - index) / 2); i++) {

current = current->next;

}

node\* temp = new node;

temp->data = val;

temp->next = current->next;

current->next = temp;

count++;

}

}

void delete\_at\_front() {

current = head;

head = current->next;

delete current;

current = head;

count--;

}

void delete\_at\_end() {

current = head;

for (int i = 1; i <= count; i++) {

current = current->next;

}

delete current;

count--;

current = head;

for (int i = 1; i <= count; i++) {

current = current->next;

}

current->next = nullptr;

}

void delete\_at\_index(int index) {

current = head;

if (index == 1) {

delete\_at\_front();

}

else if (index == count) {

delete\_at\_end();

}

else if (index > 1 && index < count) {

current = head->next;

for (int i = 2; i < index - 1; i++) {

current = current->next;

}

node\* temp = current->next;

current->next = current->next->next;

delete temp;

count--;

}

else {

cout << "Invalid index" << endl;

}

}

virtual void search(int val) {

current = head;

bool found = false;

for (int i = 1; i < count; i++) {

if (current->data == val) {

cout << "Found at Index : "<< i << endl;

found = true;

break;

}

current = current->next;

}

if (found == false) {

cout << "Not Found" << endl;

}

}

void print() {

current = head;

for (int i = 1; i <= count; i++) {

cout << "[" << current->data << "]";

current = current->next;

}

cout << endl;

}

void removeDuplicates() {

current = head;

for (int i = 1; i <= count; i++) {

if (current->data == current->next->data) {

delete\_at\_index(i);

}

current = current->next;

}

}

};

int main() {

singlylist obj;

cout << "Insert at End" << endl;

for (int i = 1; i <= 5; i++) {

obj.insert\_at\_end(i);

obj.insert\_at\_end(i);

}

obj.print();

cout << "Remove Duplicates" << endl;

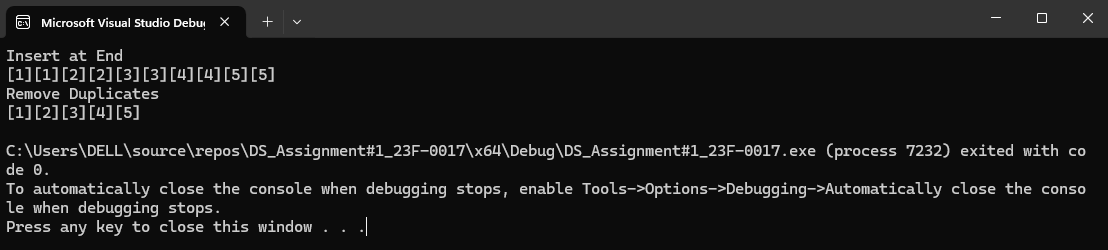
obj.removeDuplicates();

obj.print();

return 0;

}

**Output:**

****