|  |  |
| --- | --- |
| #include<iostream>  using namespace std;  class List;  class Node  {  friend class List;  private:  int data;  Node \* next;  Node \* prev;  public:  Node()  {  next = NULL;  prev = NULL;  }  Node(int x)  {  data = x;  next = NULL;  prev = NULL;  }  };  class List // دو طرفه معمولی  {  public:  Node \* First;  Node \* Last;  List()  {  First = NULL;  Last = NULL;  }  bool InsAtFirst(int x);  bool InsAtLast(int x);  bool InsNextP(Node \* p, int x);  bool InsPerP(Node \* p, int x);  bool DeleteFromLast();  bool DeleteFromFirst();  bool DeleteFromP(Node \* p);  bool DeleteFromNextP(Node \* p);  bool DeleteFromPerP(Node \* p);  void PrintFromFirst();  void PrintFromLast();  void MakeNull();  Node \* Find(int x);  };  bool List::DeleteFromFirst()  {  if (First == NULL)  return false;  if (First == Last)  {  delete First;  First = NULL;  Last = NULL;  return true;  }  First = First->next;  First->prev = NULL;  return true;  }  bool List::DeleteFromP(Node \* p)  {  if (First == NULL)  return false;  if (p == First)  {  First = p->next;  delete p;  return true;  }  if (p == Last)  {  Last->prev->next = NULL;  return true;  }  Node \* help = First->next;  while (help != p)  help = help->next;  help->next->prev = help->prev;  help->prev->next = help->next;  delete help;  return true;  }  bool List::DeleteFromNextP(Node \* p)  {  if (p->next == NULL)  return false;  Node \* help;  help = p->next->next;  delete p->next;  p->next = help;  help->prev = p;  return true;  }  bool List::DeleteFromPerP(Node \* p)  {  if (p->prev == NULL)  return false;  Node \* help;  if (p->prev == First)  {  First = First->next;  First->prev = NULL;  return true;  }  help = p->prev->prev;  delete p->prev;  p->prev = help;  help->next = p;  return true;  }  void List::PrintFromFirst()  {  if (First == NULL)  return;  Node \* help = First;  while (help != NULL)  {  cout << help->data << "\t";  help = help->next;  }  } | bool List::InsAtLast(int x)  {  Node \* temp = new Node(x);  if (First == NULL && Last == NULL)  First = temp;  Last->next = temp;  temp->prev = Last;  temp->next = NULL;  Last = temp;  return true;  }  bool List::InsNextP(Node \* p, int x)  {  Node \* temp = new Node(x);  if (p->next == NULL)  Last = temp;  else  temp->next->prev = temp;  temp->prev = p;  temp->next = p->next;  p->next = temp;  return true;  }  bool List::InsPerP(Node \* p, int x)  {  Node \* temp = new Node(x);  temp->next = p;  temp->prev = p->prev;  if (p->prev == NULL)  Last = temp;  else  temp->next->prev = p;  p->prev = temp->prev;  temp->prev->next = temp;  temp->next->prev = temp;  return true;  }  bool List::InsAtFirst(int x)  {  Node \* temp = new Node(x);  if (First != NULL)  First->prev = temp;  if (Last == NULL)  Last = temp;  temp->prev = NULL;  temp->next = First;  First = temp;  return true;  }  bool List::DeleteFromLast()  {  if (First == NULL) return false;  if (First == Last)  {  delete First;  First = NULL;  Last = NULL;  return true;  }  Last->prev->next = NULL;  delete Last;  Last = Last->prev;  return true;  }  void List::PrintFromLast()  {  if (Last == NULL)  return;  Node \* help = Last;  while (help != NULL)  {  cout << help->data << "\t";  help = help->prev;  }  }  void List::MakeNull()  {  Node \* help;  if (!First)  return;  help = First->next;  while (help != NULL)  {  delete help->prev;  help->prev = NULL;  help = help->next;  }  delete Last;  Last = NULL;  return;  }  Node \* List::Find(int x)  {  Node \* help = First;  while (help != NULL)  {  if (help->data == x)  return help;  help = help->next;  }  return NULL;  } |

|  |  |
| --- | --- |
| #include<iostream>  using namespace std;  class Node  {  public:  int data;  Node \* next;  Node()  {  next = NULL;  }  Node(int x)  {  data = x;  next = NULL;  }  };  class List // حلقوي خطي  {  public:  Node \* First;  Node \* Last;  List()  {  First = NULL;  Last = NULL;  }  bool InsAtFirst(int x);  bool InsAtLast(int x);  bool InsNextP(Node \* p, int x);  bool InsPerP(Node \* p, int x);  bool DeleteFromFirst();  bool DeleteFromLast();  bool DeleteFromP(Node \* p);  bool DeleteFromPerP(Node \* p);  bool DeleteFromNextP(Node \* p);  void PrintFromFirst();  void PrintFromLast();  void f\_PrintFromLast(Node \* temp);  void MakeNull();  Node \* Find(int x);  };  bool List::InsAtFirst(int x)  {  Node \* temp = new Node(x);  if (!temp)  return false;  temp->next = First;  First = temp;  Last->next = First;  return true;  }  bool List::InsAtLast(int x)  {  Node \* temp = new Node(x);  if (!temp) return false;  if (First == NULL && Last == NULL)  First = temp;  Last->next = temp;  temp->next = NULL;  Last = temp;  Last->next = First;  return true;  }  bool List::DeleteFromP(Node \* p)  {  Node \* help = First->next;  if (!First)  return false;  if (p == First)  {  First = p->next;  delete p;  return true;  }  if (p == Last)  {  while (help->next != p)  help = help->next;  Last = help;  Last->next = First;  delete help;  return true;  }  while (help != p)  help = help->next;  help->next = p->next;  delete p;  return true;  }  void List::PrintFromFirst()  {  if (First == NULL)  return;  Node \* help = First;  while (help != NULL)  {  cout << help->data << "\t";  help = help->next;  }  }  void List::PrintFromLast()  {  if (!First)  return;  f\_PrintFromLast(First);  }  void List::f\_PrintFromLast(Node \* temp)  {  if (temp->next)  f\_PrintFromLast(temp->next);  cout << temp->data << "\t";  } | bool List::InsNextP(Node \* p, int x)  {  Node \* temp = new Node(x);  if (p->next == NULL)  Last = temp;  temp->next = p->next;  p->next = temp;  temp->next = p;  return true;  }  bool List::InsPerP(Node \* p, int x)  {  Node \* temp = new Node(x);  if (p == First)  {  temp->next = First;  Last->next = First;  First = temp;  return true;  }  if (p == Last)  {  temp->next = Last;  return true;  }  temp->next = p;  return true;  }  bool List::DeleteFromFirst()  {  if (First == Last)  {  delete First;  First = NULL;  Last = NULL;  return true;  }  Node \* help = First->next;  Last->next = help;  delete help;  First = Last->next;  return true;  }  bool List::DeleteFromLast()  {  if (!First)  return false;  if (First == Last)  {  delete First;  First = NULL;  Last = NULL;  return true;  }  Node \* temp = First;  while (!temp->next->next)  {  temp = temp->next;  }  delete temp->next;  temp->next = NULL;  Last = temp;  return true;  }  void List::MakeNull()  {  if (!Last)  return;  Node \* temp;  while (First)  {  temp = First;  First = temp->next;  delete temp;  }  First = NULL;  Last = NULL;  return;  }  Node \* List::Find(int x)  {  Node \* help = First;  while (help != NULL)  {  if (help->data == x)  return help;  help = help->next;  }  return NULL;  } |

|  |  |
| --- | --- |
| #include<iostream>  using namespace std;  class Node  {  public:  int data;  Node \* next;  Node \* prev;  Node()  {  next = NULL;  prev = NULL;  }  Node(int x)  {  data = x;  next = NULL;  prev = NULL;  }  };  class List // حلقوي دو طرفه  {  public:  Node \* First;  Node \* Last;  List()  {  First = NULL;  Last = NULL;  }  bool InsAtFirst(int x);  bool InsAtLast(int x);  bool InsNextP(Node \* p, int x);  bool InsPerP(Node \* p, int x);  bool DeleteFromFirst();  bool DeleteFromLast();  bool DeleteFromP(Node \* p);  bool DeleteFromPerP(Node \* p);  bool DeleteFromNextP(Node \* p);  void PrintFromFirst();  void PrintFromLast();  void MakeNull();  Node \* Find(int x);  };  bool List::InsAtFirst(int x)  {  Node \* temp = new Node(x);  if (First != NULL)  First->prev = temp;  if (Last == NULL)  Last = temp;  temp->prev = NULL;  temp->next = First;  First = temp;  Last->next = First->prev;  return true;  }  bool List::DeleteFromLast()  {  if (First == Last)  {  delete First;  First = NULL;  Last = NULL;  return true;  }  Last->prev->next = NULL;  delete Last;  Last = Last->prev;  First->prev = Last->next;  return true;  }  bool List::DeleteFromP(Node \* p)  {  if (First == NULL)  return false;  if (p == First)  {  First = p->next;  delete p;  return true;  }  if (p == Last)  {  Last->prev->next = NULL;  return true;  }  Node \* help = First->next;  while (help != p)  help = help->next;  help->next->prev = help->prev;  help->prev->next = help->next;  delete help;  First->next = Last->prev;  return true;  }  bool List::DeleteFromPerP(Node \* p)  {  if (p->prev == NULL)  return false;  if (p->prev == First)  {  First = First->next;  First->prev = NULL;  return true;  }  Node \* help = p->prev->prev;  delete p->prev;  p->prev = help;  help->next = p;  p->prev = help->next;  return true;  } | bool List::InsAtLast(int x)  {  Node \* temp = new Node(x);  if (First == NULL && Last == NULL)  First = temp;  Last->next = temp;  temp->prev = Last;  temp->next = NULL;  Last = temp;  First->prev = Last->next;  return true;  }  bool List::InsNextP(Node \* p, int x)  {  Node \* temp = new Node(x);  if (p->next == NULL)  Last = temp;  else  temp->next->prev = temp;  temp->prev = p;  temp->next = p->next;  p->next = temp;  p->prev = temp->next;  return true;  }  bool List::InsPerP(Node \* p, int x)  {  Node \* temp = new Node(x);  if (p == First)  {  temp->next = First;  temp->prev = Last;  First = temp;  return true;  }  if (p == Last)  {  temp->next = Last;  Last->prev = temp;  return true;  }  temp->prev = p->prev;  temp->next = p;  p->prev = temp->next;  return true;  }  bool List::DeleteFromFirst()  {  if (First == Last)  {  delete First;  First = NULL;  Last = NULL;  return true;  }  First = First->next;  First->prev = NULL;  First = Last->prev;  return true;  }  bool List::DeleteFromNextP(Node \* p)  {  if (p->next == NULL)  return false;  Node \* help = p->next->next;  delete p->next;  p->next = help;  help->prev = p;  p->prev = help->next;  return true;  }  void List::PrintFromFirst()  {  if (First == NULL)  return;  Node \* help = First;  while (help != NULL)  {  cout << help->data << "\t";  help = help->next;  }  }  void List::PrintFromLast()  {  if (Last == NULL)  return;  Node \* help = Last;  while (help != NULL)  {  cout << help->data << "\t";  help = help->prev;  }  }  void List::MakeNull()  {  if (!Last)  return;  Node \* help = Last->prev;  while (help)  {  delete help->next;  help->next = NULL;  help = help->prev;  }  delete First;  First = NULL;  return;  }  Node \* List::Find(int x)  {  Node \* help = First;  while (help != NULL)  {  if (help->data == x)  return help;  help = help->next;  }  return NULL;  } |