物件導向程式設計

Linked List

Joseph Chuang-Chieh Lin
Dept. CSIE, Tamkang University

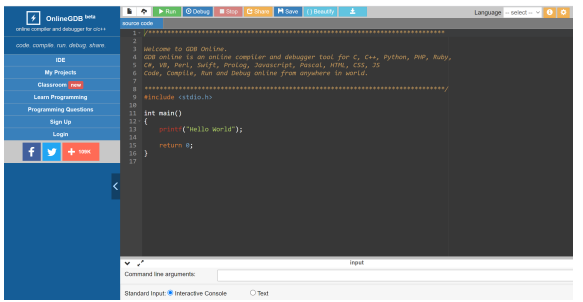
Platform

- Dev-C++

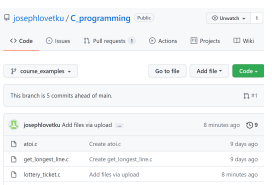
  Click here to download.

  Note: Please use this version otherwise you can't compile your programs/projects in Win10.

- OnlineGDB (https://www.onlinegdb.com/)



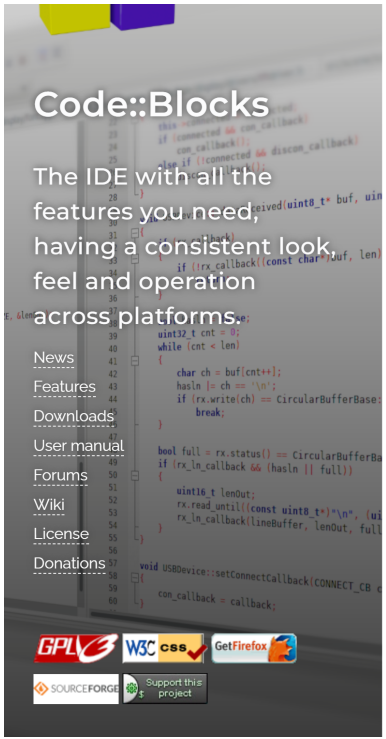
My GitHub page:
click the link here to visit.



- Other resources:
  - MIT OpenCourseWare - Introduction to C++ [link].
  - Learning C++ Programming [Programiz].
  - GeeksforGeeks [link]

Platform/IDE

- https://www.codeblocks.org/



Code::Blocks

Code::Blocks

The free C/C++ and Fortran IDE.

Code::Blocks is a free C/C++ and Fortran IDE built to meet the most demanding needs of its users. It is designed to be very extensible and fully configurable.

Built around a plugin framework, Code::Blocks can be extended with plugins. Any kind of functionality can be added by installing/coding a plugin. For instance, event compiling and debugging functionality is provided by plugins!

If you 're new here, you can read the **user manual** or visit the **Wiki** for documentation. And don't forget to visit and join our **forums** to find help or general discussion about Code::Blocks.

We hope you enjoy using Code::Blocks!

The Code::Blocks Team

Latest news

Migration successful

We are very happy to announce that the process of migrating to the new infrastructure has completed successfully!

Read more

Node: Inherited from BaseNode


```
class BaseNode {
public:
    string nodeType;
    BaseNode() = default;
    BaseNode(string s): nodeType(s) {}
    ~BaseNode() = default;
};
```

```
class Node : public BaseNode {
private:
    int data;
    Node *next;
public:
    Node() = default;
    Node(int a) ... {} // TO BE DONE
    Node(string s, int a) ... {} TO BE DONE
    // destructor TO BE DONE
    void printNode() {
        * print out the node's content *
    }
    friend class LinkedList;
};
```

Node: Inherited from BaseNode


```
int main() {
    BaseNode bn("TKU");
    Node n1("CSIE", 100);
    Node n2(200);
    n1.printNode();
    n2.printNode();
    return 0;
}
```

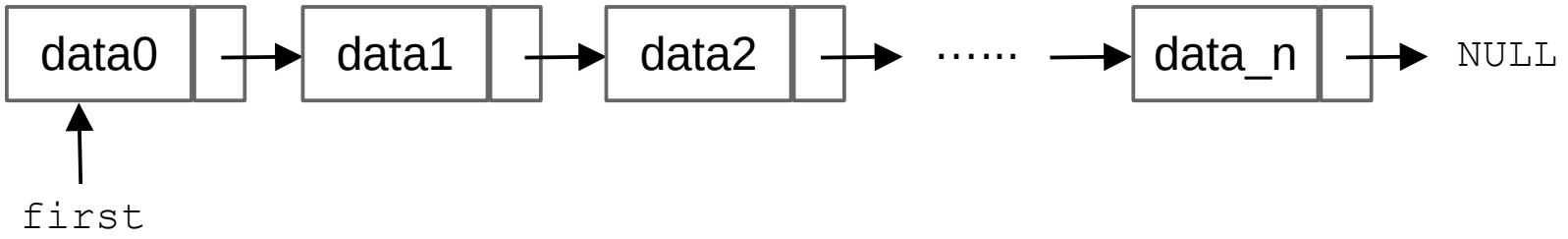
Output:

```
CSIE(100)
(200)
```

Class LinkedList

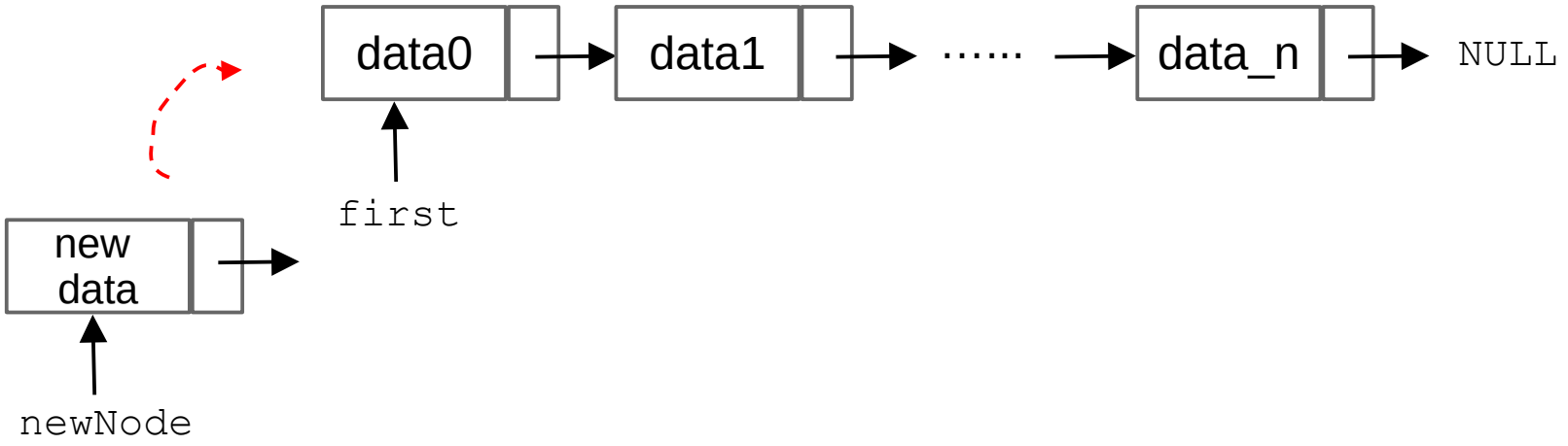
```
class LinkedList {
private:
    int size;              // size: the size of the Linked list
    Node *first;              // pointing to the first node of the list
public:
    LinkedList(): size(0), first(nullptr) {};
    void PrintList();             // TBD; print content of all nodes in the list
    void Push_front(int x);       // add a node at the front of the list
    void Push_front(string s, int x);  // TBD; add a node at the front of the list
    void Push_back(int x);        // add a note at the rear of the list
    void Push_back(string s, int x);   // TBD; add a note at the rear of the list
};
```

PrintList()



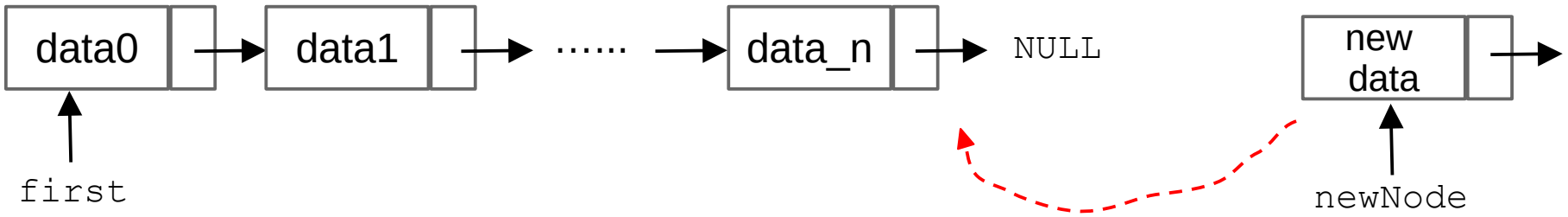

```
void LinkedList::PrintList() {
    // print the constent of the list
    if (first == nullptr) {          // no data in the list
        cout << "List is empty." << endl;
        return;
    }

    Node *current = first;
    while (current != nullptr) {         // Traversing the list and print the content
        /* print the current node' content */
        current = current->next;
    }
    cout << endl;
}
```

Push_front()




```
void LinkedList::Push_front(int x) {
    // // allocate new memory for the new node
    Node *newNode = new Node(x);    // Note: the parameterized constructor is used
    newNode->next = first;
    first = newNode;
    /* we also need to increase the size of the list*/
}
```

Push_back()




```
void LinkedList::Push_back(int x) {
    Node *newNode = new Node(x);

    if (first == nullptr) {   // the case that the list is empty
        first = newNode;
    } else { // get to the last position of the list
        Node *current = first;
        while (current->next != nullptr) { // Traversing the list...
            current = current->next;
        }
        current->next = newNode;
    }
    /* we also need to increase the size of the list*/
}
```

Main function and Sample Input & Output

```
int main() {
    LinkedList list;
    list.PrintList();     // so far the list is empty
    list.Push_back(5);    // list: 5
    list.Push_back(3);    // list: 5 3
    list.Push_front(9);   // list: 9 5 3
    list.PrintList();     // showing:  9 5 3
    return 0;
}
```

```
List is empty.
(9)(5)(3)
```

Inheriting from another `LinkedList`

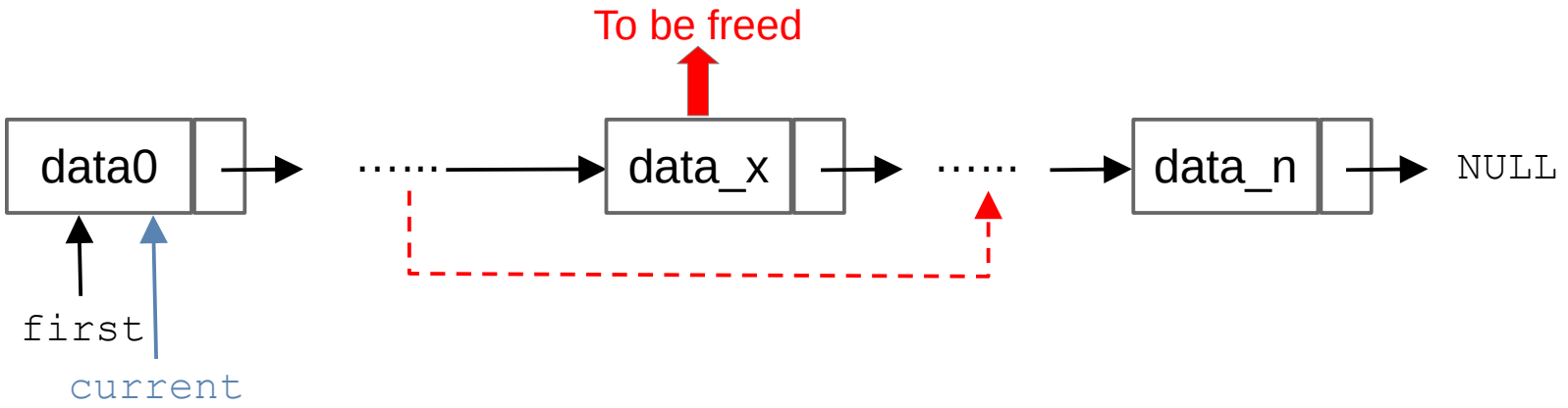
```
class Node : public BaseNode {
private:
    int data;
    Node *next;
public:
    Node() = default;
    Node(int a):
        BaseNode(""), data(a), next(nullptr) { };
    Node(string s, int a):
        BaseNode(s), data(a), next(nullptr) {};
    ~Node() = default;
    void printNode() {
        cout << nodeType << "(" << data << ")";
    }
    friend class SimpleList;
    friend class LinkedList;
};
```

```
class SimpleList {
protected:
    int size;
    Node *first;
public:
    SimpleList():
        size(0), first(nullptr) {};
    ~SimpleList() = default;
    void PrintList();
    void Push_front(int x);
    void Push_front(string s, int x);
};
```

Class LinkedList

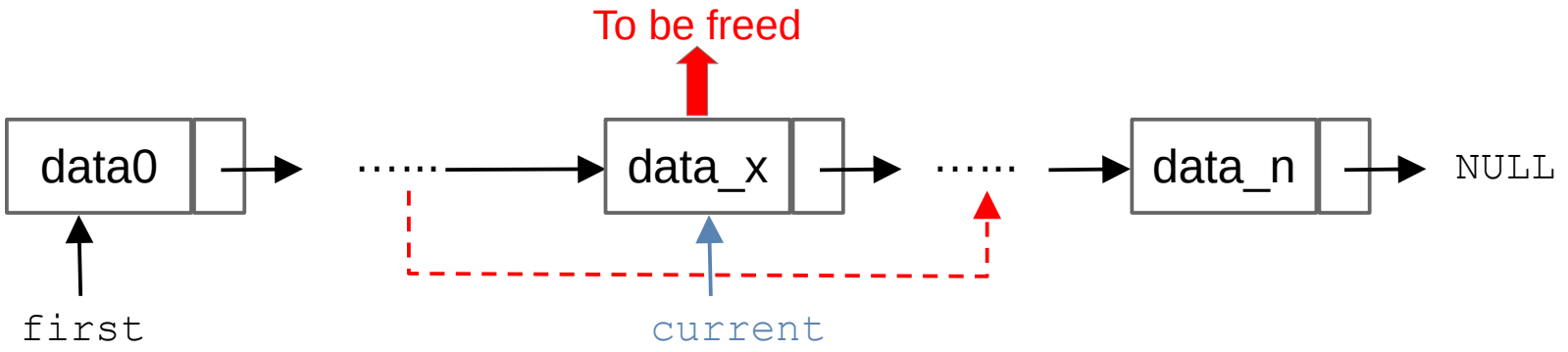
```
class LinkedList: public SimpleList {
Public:
    LinkedList(): SimpleList() { }
    ~LinkedList() = default;
    void Push_back(int x);
    void Push_front(string s, int x);  // TBD; add a node at the front of the list
    void Push_back(int x);       // add a note at the rear of the list
    void Push_back(string s, int x);   // TBD; add a note at the rear of the list
    void Delete(int x);          // delete a specific node with content "int x"
    void Clear();                // delete the whole list
    void Reverse();              // reverse the list: that is, 1->2->3 => 3->2->1
    void Delete(string s, int x); // delete a specific node with content "s and x"
};
```

Delete()



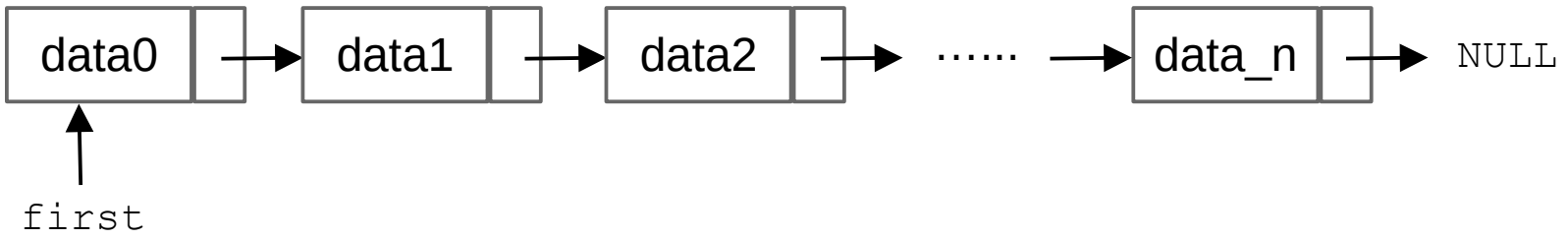

```
Node *current = first, *previous = nullptr;
while (current != nullptr && current->data != x) {
// traversing
    previous = current;
    current = current->next;
}
```

Delete()



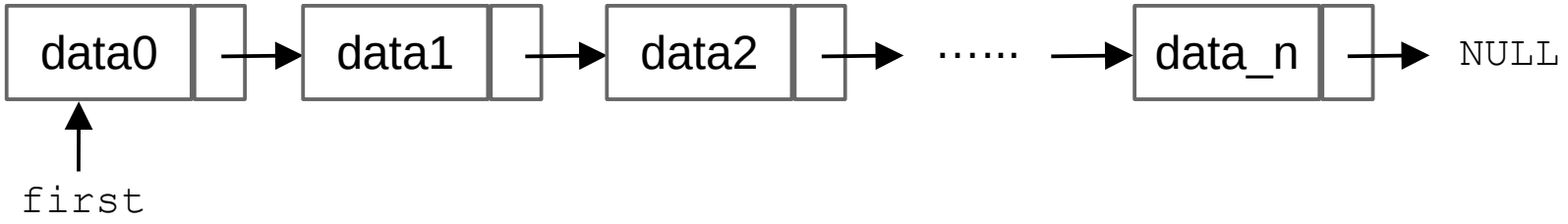

```
if (current == nullptr) {   // either can't find the node or the list is empty
    cout << x << " is not found in list." << endl;
} else if (current == first) {  // the node to delete is the first node of the list
    first = current->next;
    delete current;
    current = nullptr;
    size -= 1; // update the size here
} else {    // the other case:
    previous->next = current->next;
    delete current;
    current = nullptr;
    size -= 1; // update the size here
}
```

Clear()



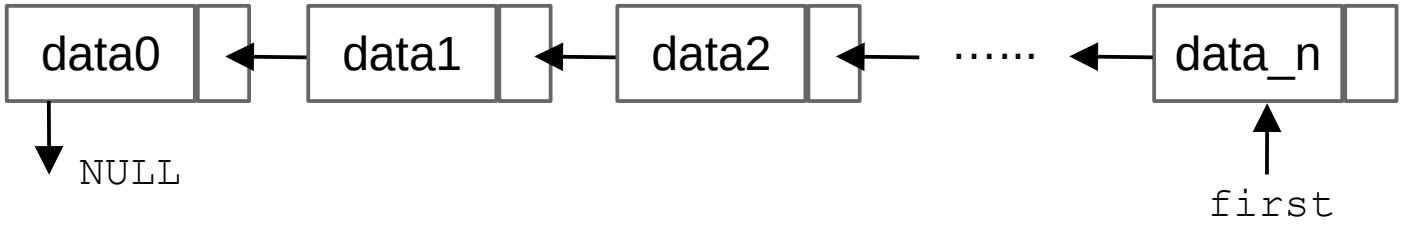

```
void LinkedList::Clear() {
    // delete all the list by deleting nodes one by one
    while (first != nullptr) { // Traversal
        Node *current = first;
        first = first->next;
        delete current;
        current = nullptr;
    }
    size = 0; // update the size here; an empty list has size 0
}
```

Reverse()



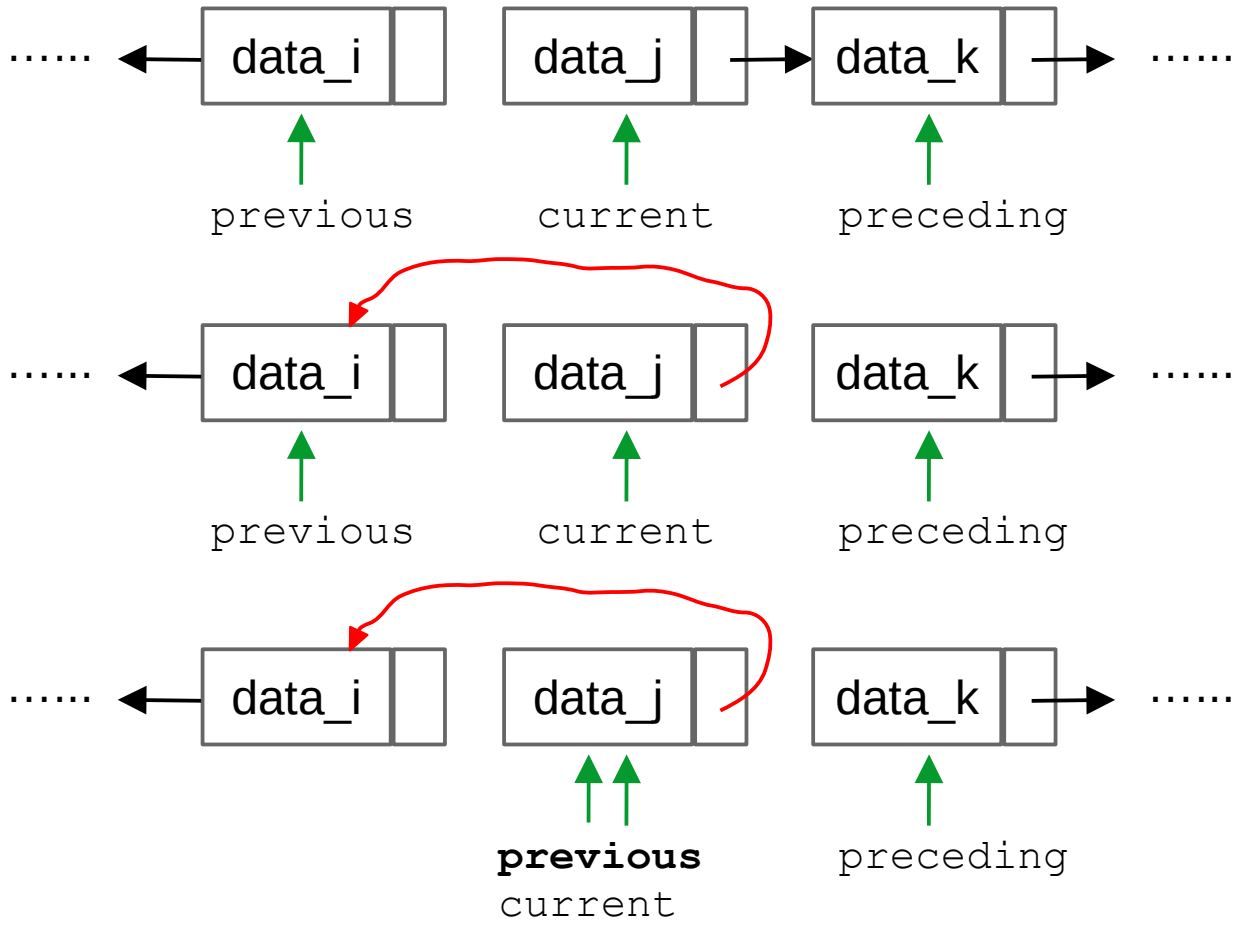

```
void LinkedList::Reverse() {
    if (first != nullptr && first->next != nullptr) {
        Node *previous = nullptr,
        *current = first,
        *preceding = first->next;

        while (preceding != nullptr) {
            current->next = previous;
            previous = current;
            current = preceding;
            preceding = preceding->next;
        }   // until preceding gets to nullptr
        current->next = previous;  // now current is at the last node
        first = current;          // update first to be current
    }
}
```

Reverse()

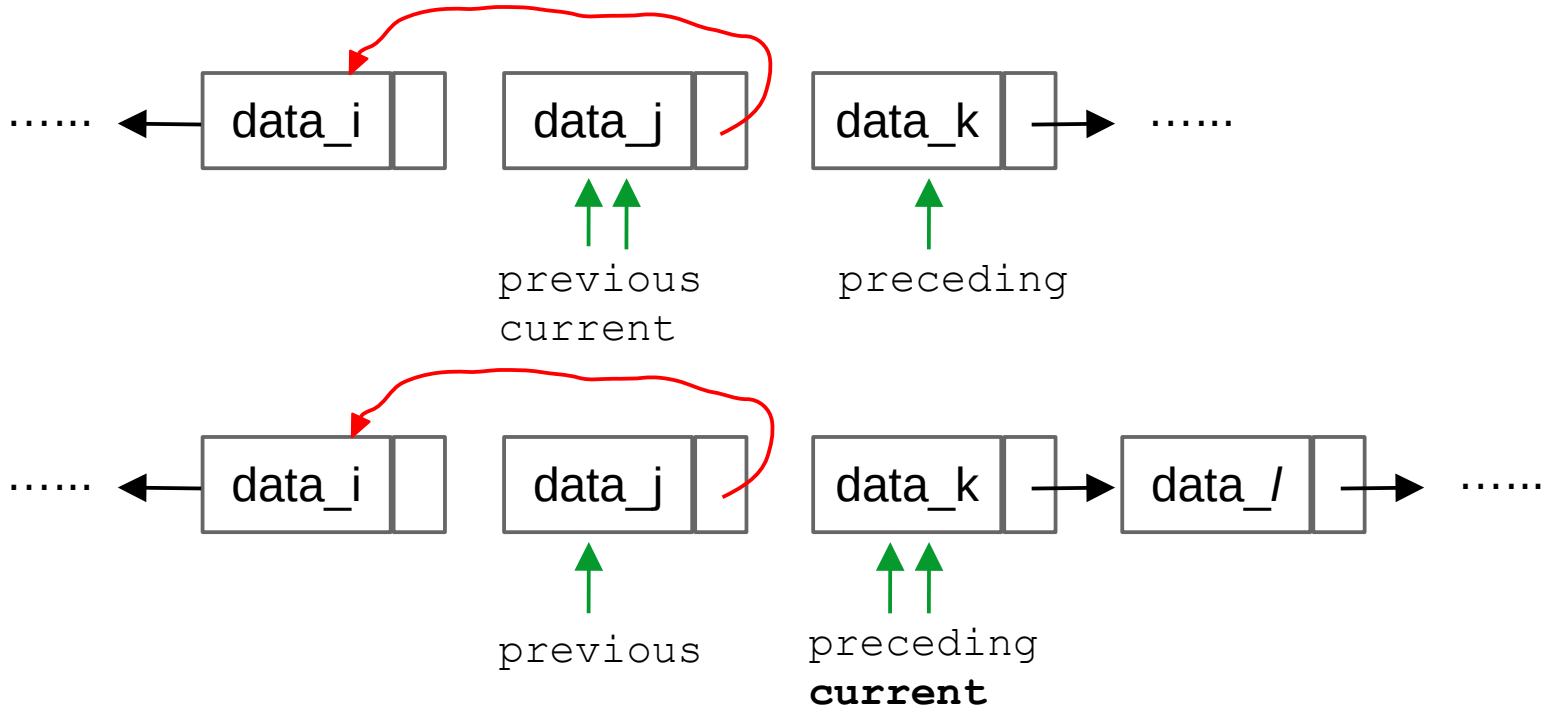



```
void LinkedList::Reverse() {
    if (first != nullptr && first->next != nullptr) {
        Node *previous = nullptr,
        *current = first,
        *preceding = first->next;

        while (preceding != nullptr) {
            current->next = previous;
            previous = current;
            current = preceding;
            preceding = preceding->next;
        }   // until preceding gets to nullptr
        current->next = previous;  // now current is at the last node
        first = current;          // update first to be current
    }
}
```

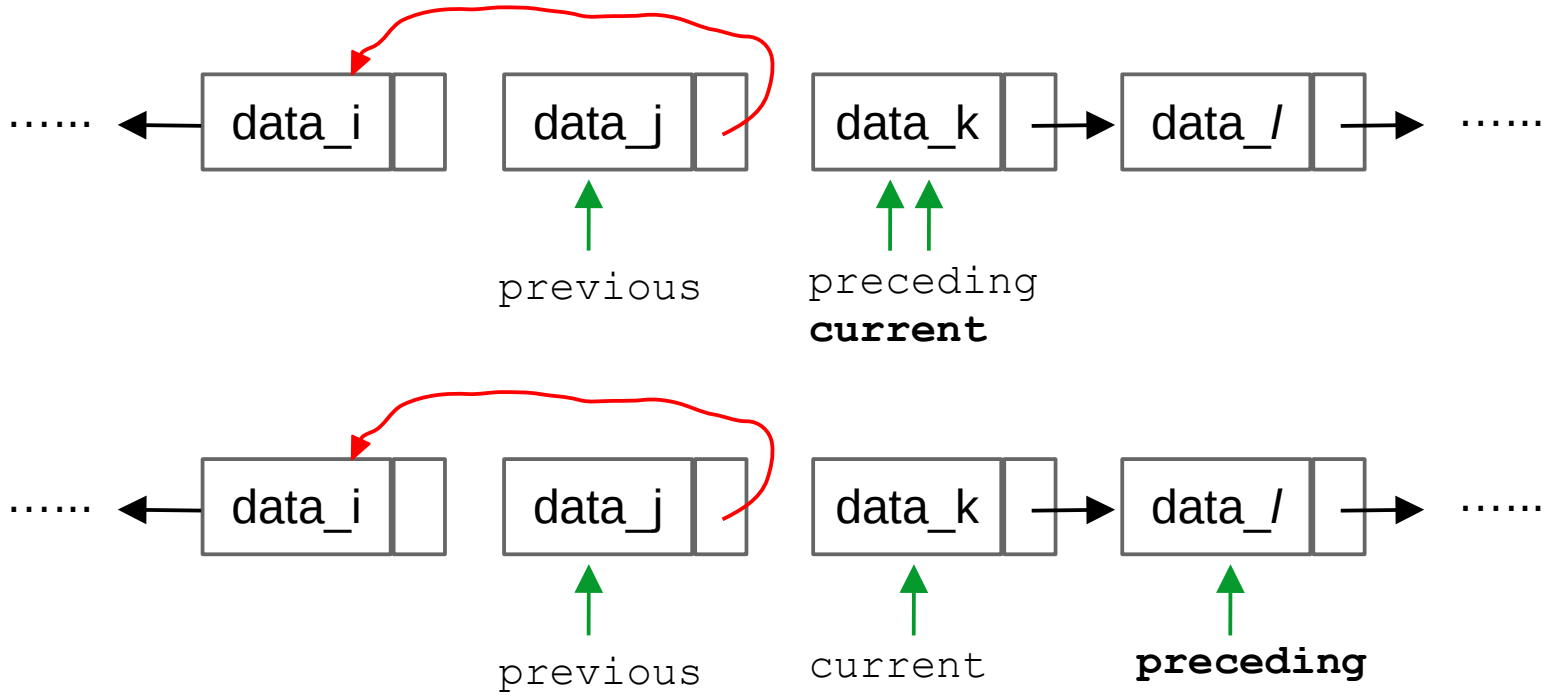
Illustration



Illustration



Illustration



Exercise (2%)

- Please implement:

```
void LinkedList::Delete(string s, int x);

void LinkedList::Clear();

void LinkedList::Push_back(string, int);

void LinkedList::Push_front(string, int);

void LinkedList::Reverse();

void SimpleList::PrintList();
```

```
int main() {
    LinkedList list;
    list.Push_back("B", 5);
    list.Push_back("A", 3);
    list.Push_front("C", 9);
    list.Push_front("N", 7);
    list.Delete("C", 9);
    list.Reverse();
    list.PrintList();
    list.Clear();
    list.PrintList();
    return 0;
}
```

```
A(3) B(5) N(7)
List is of size 3
List is empty
```

Anther Exercise (Double Linkedlist)

```
class Node : public BaseNode {
private:
    int data;
    Node *next;
public:
    Node() = default;
    Node(int a): BaseNode(""), data(a), next(nullptr) {};
    Node(string s, int a): BaseNode(s), data(a), next(nullptr) {};
    ~Node() = default;
    void printNode() { cout << nodeType << "(" << data << ")"; }
    friend class LinkedList;
    /* modify class Node */
};
```

Anther Exercise (Double Linkedlist) (contd.)

```
class LinkedList {
protected:
    int size;  // size: the size of the Linked list
    Node *first;  // pointing to the first node of the list
public:
    LinkedList(): size(0), first(nullptr) {};
    void PrintList();   // print content of all nodes in the list
    void Push_front(int x);   // add a node at the front of the list
    void Push_front(string s, int x);
           // add a node at the front of the list
    virtual void Push_back(int x);  // modify this declaration
           // add a note at the rear of the list
    virtual void Push_back(string s, int x); // modify this declaration
           // add a note at the rear of the list
};
```

Anther Exercise (Double Linkedlist) (contd.)

```
class twoEndLinkedList: public LinkedList {
private:
    Node *last;
public:
    twoEndLinkedList(): LinkedList(), last(nullptr) {};
    void Push_back(int x);
       // override Push_back(int x) in LinkedList
    void Push_back(string s, int x);
       // override Push_back(string s, int x) in LinkedList
    void printLast();
    // please implement this member function outside the class definition
};
```

Anther Exercise (Double Linkedlist) (contd.)

```
int main() { // sample main()
    twoEndLinkedList list;
    list.PrintList();
    list.printLast();
    list.Push_back("A", 5);
    list.Push_back("B", 3);
    list.Push_front("C", 9);
    list.Push_back("D", 11);
    list.PrintList();
    list.printLast();
    return 0;
}
```

```
List is empty.
Last node: none
C(9)A(5)B(3)D(11)
List is of size 4
Last node: D(11)
```