TIHOMIR DIMOV

WELCOME TO MY OFFICIAL BLOG

03.10.13

by tdimov

DOUBLY LINKED LIST C# IMPLEMENTATION

Hello, friends!

In this post I am going to show you what doubly linked list is and how to implement it using C#.

So… What doubly linked list is and why we use it in computer programming.

Definition:

A doubly linked list is a linked data structure that is sequentially linked nodes where every node contains information about his next and his

SEARCH

Search …

RECENT POSTS

- Simple web chat with ASP.NET SignalR October 11, 2013
- Moving Flashing Smiles in Circle August 2, 2013
- Качеството на висшето образование в България July 6, 2013
- Hangman Game May 28, 2013

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

Close and accept

(node) and third field is a pointer to the previous element (node). For more information about doubly linked list you can find **here**.

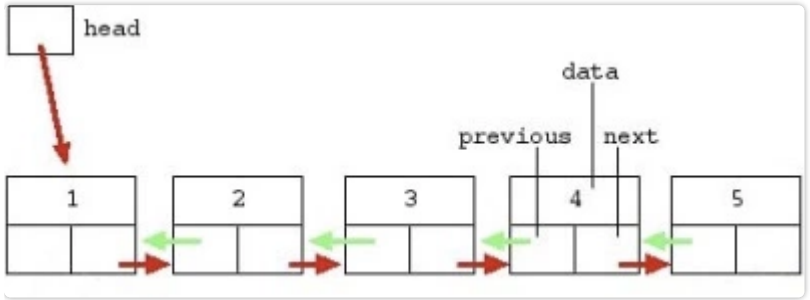
Advantages and disadvantages:

– We can add node quickly of arbitrary position in the list;

– The searching of a node by index or by content is slow operation, because we have to pass over every node sequentially as we start from the beginning of the list;

– Removing of node is also slow operation, because we have to pass over every node again;

The below figure show how a doubly linked list looks like:



With this simple explanations know we know what doubly linked list is let me start to show how to implement it using C#.

For the implementation we need two classes for instance called **DoublyLinkedList** and **DoublyLinkedListNode**. The first class is our main class and the second one which describe one certain node of the list. The implementation of the **DoublyLinkedListNode** class has to be inside the **DoublyLinkedList** class.

CATEGORIES

- Hardware
- Other
- Software

LINKS

- Telerik Academy
- The New Boston

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy
Close and accept

DoublyLinkedListNode class:

```
   private class DoublyLinkedListNode
{
private object element;
private DoublyLinkedListNode next;
private DoublyLinkedListNode previous;

   public DoublyLinkedListNode(object element)
{
this.element = element;
this.next = null;
this.previous = null;
}

   public DoublyLinkedListNode(object element,
DoublyLinkedListNode prevNode)
{
this.element = element;
this.previous = prevNode;
prevNode.next = this;
}

   public object Element
{
get { return this.element; }
set { this.element = value; }
}

   public DoublyLinkedListNode Next
{
get { return this.next; }
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy
Close and accept

```
   public DoublyLinkedListNode Previous
{
get { return this.previous; }
set { this.previous = value; }
}
   }
```

Here we have three fields for an element, a next node and a previous node. Two constructors, first we use to make the first element of the list and second which we use to make the following nodes. Also we have properties about an element, next node and previous node with get and set accessories.

DoublyLinkedList class:

```
   class DoublyLinkedList
{
private DoublyLinkedListNode head;
private DoublyLinkedListNode tail;
private int count;

   public DoublyLinkedList()
{
this.head = null;
this.tail = null;
this.count = 0;
}

   public int Count
{
get { return this.count; }
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy
Close and accept

```
   public object this[int index]
{
get
{
if (index >= count || index < 0)
{
throw new ArgumentOutOfRangeException("Out of range!");
}
DoublyLinkedListNode currentNode = this.head;
for (int i = 0; i < index; i++)
{
currentNode = currentNode.Next;
}
return currentNode.Element;
}
set
{
if (index >= count || index < 0)
{
throw new ArgumentOutOfRangeException("Out of range!");
}
DoublyLinkedListNode currentNode = this.head;
for (int i = 0; i < index; i++)
{
currentNode = currentNode.Next;
}
currentNode.Element = value;
}
}
   //more code here...
}
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

Close and accept

In this class we have three fields again – head, tail and count, one constructor and two properties. The head field contains our first node in the list, the tail field contains the last node in the list and a count field which contains the number of elements in the list. By constructor initially we make empty list. By property **Count** we get number of the elements in the list and **this[int index]** we get or set the element on the specified position (it is called indexer).

Here in this class we will write several methods:

– Add(object) – add element in the end of list;

– Insert(object, position) – add element by given position in the list;

– Remove(object) – remove the passed node in the function;

– RemoveAt(position) – remove element of current position;

– IndexOf(object) – return position of an element;

– Contains(object) – checks if an element exists;

– Clear() – remove all nodes from the list;

Add a node in the list (Add(object)).

```
   public void Add(object item)
{
if (this.head == null)
{
this.head = new DoublyLinkedListNode(item);
this.tail = this.head;
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy
Close and accept

```
{
DoublyLinkedListNode newItem = new DoublyLinkedListNode(item,
tail);
this.tail = newItem;
}
count++;
}
```

Here we consider two cases: when the list is empty and when the list is not empty. The purpose here is to add every new node at the end of the list and after adding every variable (head, tail and count) to have correct values.

Insert a node in the list (Insert(object, position)).

```
   public void Insert(object item, int index)
{
count++;
if (index >= count || index < 0)
{
throw new ArgumentOutOfRangeException("Out of range!");
}
DoublyLinkedListNode newItem = new
DoublyLinkedListNode(item);
int currentIndex = 0;
DoublyLinkedListNode currentItem = this.head;
DoublyLinkedListNode prevItem = null;
while (currentIndex < index)
{
prevItem = currentItem;
currentItem = currentItem.Next;
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy
Close and accept

```
if (index == 0)
```

```
{
newItem.Previous = this.head.Previous;
newItem.Next = this.head;
this.head.Previous = newItem;
this.head = newItem;
}
else if (index == count - 1)
{
newItem.Previous = this.tail;
this.tail.Next = newItem;
newItem = this.tail;
}
else
{
newItem.Next = prevItem.Next;
prevItem.Next = newItem;
newItem.Previous = currentItem.Previous;
currentItem.Previous = newItem;
}
}
```

Insert current node on the given position. Here we increase count, make our new element and searching its position in the list. When the current position is found and after some checks about where we want to add the new element we redirect his next and previous pointers to its next and previous nodes.

Remove a node from list by element (Remove(object)).

```
   public void Remove(object item)
{
int currentIndex = 0;
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy
Close and accept

```
while (currentItem != null)
```

```
{
if ((currentItem.Element != null &&
currentItem.Element.Equals(item)) ||
(currentItem.Element == null) && (item == null))
{
break;
}
prevItem = currentItem;
currentItem = currentItem.Next;
currentIndex++;
}
if (currentItem != null)
{
count--;
if (count == 0)
{
this.head = null;
}
else if (prevItem == null)
{
this.head = currentItem.Next;
this.head.Previous = null;
}
else if (currentItem == tail)
{
currentItem.Previous.Next = null;
this.tail = currentItem.Previous;
}
else
{
currentItem.Previous.Next = currentItem.Next;
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

Close and accept

```
}
```

First we check if the current node exists and if the node exists remove it. Then we consider four cases:

– The list remains empty after the deleting;

– The node is at the beginning of the list (it doesn't have a previous). Here we make head to shows a node after the deleted one;

– The node is at the end of the list (it doesn't have a next). Here we make tail to shows a node before the deleted one.

– The node is between head and tail.

Remove a node from the list by index (RemoveAt(index)).

```
   public void RemoveAt(int index)
{
if (index >= this.count || index < 0)
{
throw new ArgumentOutOfRangeException("Out of range!");
}

   int currentIndex = 0;
DoublyLinkedListNode currentItem = this.head;
DoublyLinkedListNode prevItem = null;
while (currentIndex < index)
{
prevItem = currentItem;
currentItem = currentItem.Next;
currentIndex++;
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy
Close and accept

```
this.head = null;
}
else if (prevItem == null)
{
this.head = currentItem.Next;
this.head.Previous = null;
}
else if (index == count - 1)
{
prevItem.Next = currentItem.Next;
tail = prevItem;
currentItem = null;
}
else
{
prevItem.Next = currentItem.Next;
currentItem.Next.Previous = prevItem;
}
count--;
}
```

Remove a node from list by index is almost the same like remove a node from list by element only with one small difference. We have to check if the index exists and if it doesn't exist throw an appropriate exception with appropriate message.

Return index of the element (IndexOf(object)).

```
   public int indexOf(object item)
{
int index = 0;
DoublyLinkedListNode currentItem = this.head;
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

Close and accept

```
if (((currentItem.Element != null) && (item ==
currentItem.Element)) ||
((currentItem.Element == null) && (item == null)))
{
return index;
}
index++;
currentItem = currentItem.Next;
}
return -1;
}
```

Search for given element in the list. And if the element is found return index of the element or if the element is not found return -1;

Checks whether element exists (Contains(object)).

```
   public bool Contains(object element)
{
int index = indexOf(element);
bool contains = (index != -1);
return contains;
}
```

Here we call the IndexOf(object) and if the return index is different of -1 this means that the element which we search exists and method returns true, else returns false if the element doesn't exist.

Clear the list (Clear()).

```
   public void Clear()
{
this.head = null;
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy
Close and accept

With this method we remove all elements from the list as give for head and tail null and for count 0.

Conclusion

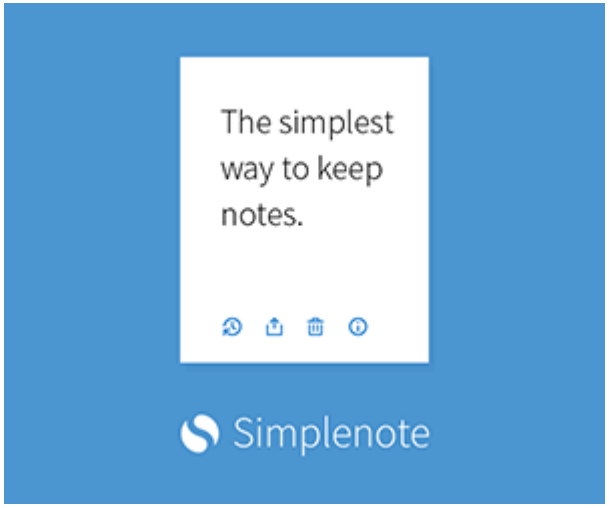
So this was shortly the doubly linked list implementation. I hope that the post was useful for you. If you have some questions you know how to connect with me. Bye bye 🙂

CODE

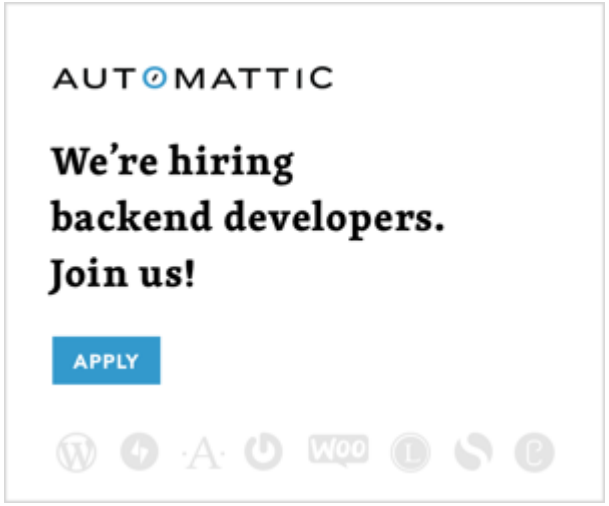
Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

Close and accept

Advertisements



REPORT THIS AD



REPORT THIS AD

Share this:

Facebook 1

Like

Be the first to like this.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

Close and accept

Related

How to define a class in C#
In "Software"

Pretty Cool Horizontal Navigation Bar
In "Software"

Hangman Game
In "Software"

This entry was posted in Software. Bookmark the permalink.

← PRETTY COOL HORIZONTAL NAVIGATION BAR

WELCOME SAMSUNG GALAXY SIV →

4 THOUGHTS ON "DOUBLY LINKED LIST C# IMPLEMENTATION"

Vlad
June 10, 2013
00:24

The material is shown really nicely.

Thank you for the example.

Reply

highfive
July 23, 2013
14:50

if I want to use strings as elements in this list, do I have to somehow determine the element as string or its universal? couse it seems that it is possible to put strings in the list, but when I try to print currentItem.Element the result is System.char[]

Reply

But why you need to print the

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

Close and accept

July 23, 2013
22:07

(integer, double, person, animal, etc)

Reply

highfive
July 23, 2013
22:42

just to check if all data was red correcly. and I need to print all in my case words (strings) from this douby linked list to a file

LEAVE A REPLY

Enter your comment here...

Blog at WordPress.com.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

Close and accept