8.8 Destructors

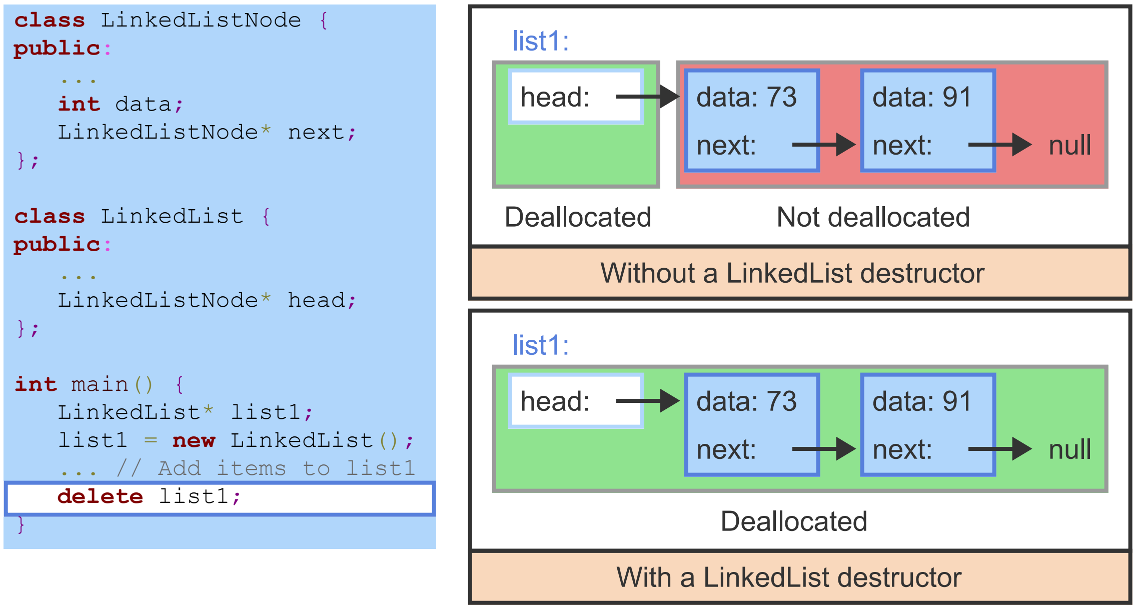
Overview

A ***destructor*** is a special class member function that is called automatically when a variable of that class type is destroyed. C++ class objects commonly use dynamically allocated data that is deallocated by the class's destructor.

Ex: A linked list class dynamically allocates nodes when adding items to the list. Without a destructor, the link list's nodes are not deallocated. The linked list class destructor should be implemented to deallocate each node in the list.

PARTICIPATION ACTIVITY 8.8.1: LinkedList nodes are not deallocated without a LinkedList class destructor. ✓

Start □ 2x speed



```
class LinkedListNode {
public:
   ...
   int data;
   LinkedListNode* next;
};

class LinkedList {
public:
   ...
   LinkedListNode* head;
};

int main() {
   LinkedList* list1;
   list1 = new LinkedList();
   ... // Add items to list1
   delete list1;
}
```

Feedback?

PARTICIPATION ACTIVITY 8.8.2: LinkedList class destructor. ✓

1) Using the delete operator to ✓

deallocate a LinkedList object automatically frees all nodes allocated by that object.

○ True

○ False

2) A destructor for the LinkedList class would be implemented as a LinkedList class member function. ✓

○ True

○ False

3) If list1 were declared without dynamic allocation, as shown below, no destructor would be needed. ✓

```
LinkedList list1;
```

○ True

○ False

Feedback?

Implementing the LinkedList class destructor

The syntax for a class's destructor function is similar to a class's constructor function, but with a "~" (called a "tilde" character) prepended to the function name. A destructor has no parameters and no return value. So the LinkedListNode and LinkedList class destructors are declared as `~LinkedListNode();` and `~LinkedList();`, respectively.

The LinkedList class destructor is implemented to free each node in the list. The LinkedListNode destructor is not required, but is implemented below to display a message when a node's destructor is called. Using delete to free a dynamically allocated LinkedListNode or LinkedList will call the object's destructor.

Figure 8.8.1: LinkedListNode and LinkedList classes.

```
#include <iostream>
using namespace std;

class LinkedListNode {
public:
   LinkedListNode(int dataValue) {
      cout << "In LinkedListNode constructor (" << dataValue << ")" << endl;
      data = dataValue;
   }

   ~LinkedListNode() {
      cout << "In LinkedListNode destructor (";
      cout << data << ")" << endl;
   }

   int data;
   LinkedListNode* next;
};

class LinkedList {
public:
   LinkedList();
   ~LinkedList();
   void Prepend(int dataValue);

   LinkedListNode* head;
};

LinkedList::LinkedList() {
   cout << "In LinkedList constructor" << endl;
   head = nullptr;
}

LinkedList::~LinkedList() {
   cout << "In LinkedList destructor" << endl;

   // The destructor deletes each node in the linked list
   while (head) {
      LinkedListNode* next = head->next;
      delete head;
      head = next;
   }
}

void LinkedList::Prepend(int dataValue) {
   LinkedListNode* newNode = new LinkedListNode(dataValue);
   newNode->next = head;
   head = newNode;
}
```

Feedback?

PARTICIPATION ACTIVITY

8.8.3: The LinkedList class destructor, called when the list is deleted, frees all nodes.

Start □ 2x speed

```
#include <iostream>
using namespace std;
```

Console:

```
In LinkedList constructor
In LinkedListNode constr
```

```
// ... LinkedListNode class omitted ...
// ... LinkedList class omitted, except for destructor, below ...

LinkedList::~LinkedList() {
   cout << "In LinkedList destructor" << endl;

   // The destructor deletes each node in the linked list
   while (head) {
      LinkedListNode* next = head->next;
      delete head;
      head = next;
   }
}

int main() {
   // Create a linked list
   LinkedList* list = new LinkedList;
   for (int i = 1; i <= 5; ++i)
      list->Prepend(i * 10);

   // Free the linked list.
   // The LinkedList class destructor frees each node.
   delete list;

   return 0;
}
```

```
In LinkedListNode constr
In LinkedListNode constr
In LinkedListNode constr
In LinkedListNode constr
In LinkedList destructor
In LinkedListNode destru
In LinkedListNode destru
In LinkedListNode destru
In LinkedListNode destru
In LinkedListNode destru
```

Feedback?

PARTICIPATION ACTIVITY | 8.8.4: LinkedList class destructor. ✓

1) After ~LinkedList() is called, the list's head pointer points to _____. ✓

○ null

○ the first node, which is now freed

○ the last node, which is now freed

2) When ~LinkedList() is called, ~LinkedListNode() gets called for each node in the list. ✓

○ True

○ False

3) If the LinkedList class were renamed to just List, the destructor ✓

function must be redeclared as

_____.

○ `void ~List();`

○ `~List();`

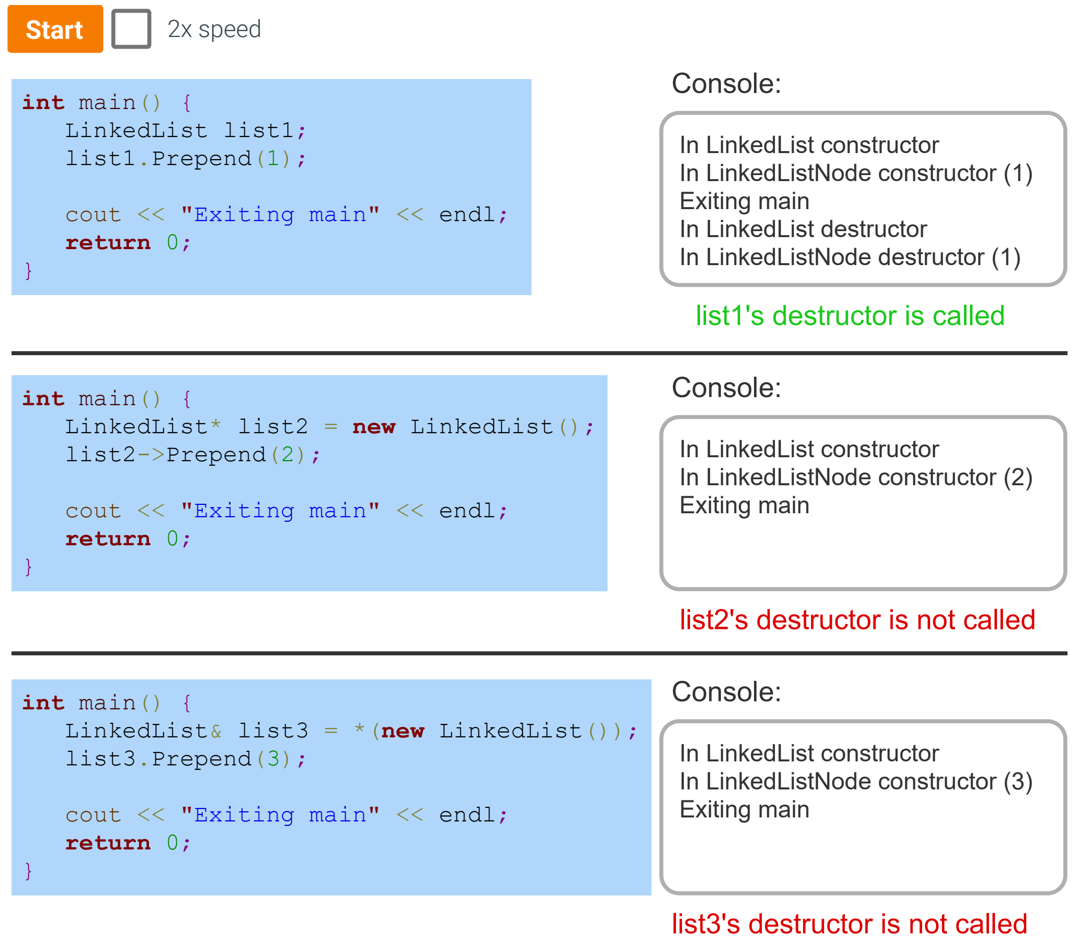
○ `List();`

Feedback?

When a destructor is called

Using the delete operator on an object allocated with the new operator calls the destructor, as shown in the previous example. For an object that is not declared by reference or by pointer, the object's destructor is called automatically when the object goes out of scope.

PARTICIPATION ACTIVITY | 8.8.5: Destructors are called automatically only for non-reference/pointer variables. ✓



```
int main() {
   LinkedList list1;
   list1.Prepend(1);

   cout << "Exiting main" << endl;
   return 0;
}
```

Console:

```
In LinkedList constructor
In LinkedListNode constructor (1)
Exiting main
In LinkedList destructor
In LinkedListNode destructor (1)
```

list1's destructor is called

```
int main() {
   LinkedList* list2 = new LinkedList();
   list2->Prepend(2);

   cout << "Exiting main" << endl;
   return 0;
}
```

Console:

```
In LinkedList constructor
In LinkedListNode constructor (2)
Exiting main
```

list2's destructor is not called

```
int main() {
   LinkedList& list3 = *(new LinkedList());
   list3.Prepend(3);

   cout << "Exiting main" << endl;
   return 0;
}
```

Console:

```
In LinkedList constructor
In LinkedListNode constructor (3)
Exiting main
```

list3's destructor is not called

Feedback?

PARTICIPATION ACTIVITY | 8.8.6: When a destructor is called. ✓

1) Both the constructor and destructor are called by the following code. ✓

```
delete (new LinkedList());
```

○ True

○ False

2) listToDisplay's destructor is called at the end of the DisplayList function. ✓

```
void DisplayList(LinkedList
listToDisplay) {
   LinkedListNode* node =
listToDisplay.head;
   while(node) {
      cout << node->data << " ";
      node = node->next;
   }
}
```

○ True

○ False

3) listToDisplay's destructor is called at the end of the DisplayList function. ✓

```
void DisplayList(LinkedList&
listToDisplay) {
   LinkedListNode* node =
listToDisplay.head;
   while(node) {
      cout << node->data << " ";
      node = node->next;
   }
}
```

○ True

○ False

Feedback?

CHALLENGE ACTIVITY | 8.8.1: Enter the output of the destructors. ✓

Start

Type the program's output.

```
#include <iostream>
using namespace std;

class IntNode {
   public:
      IntNode(int value) {
         numVal = value;
      }

      ~IntNode() {
         cout << numVal << endl;
      }

      int numVal;
};

int main() {
   IntNode* node1 = new IntNode(1);
   IntNode* node2 = new IntNode(3);
   IntNode* node3 = new IntNode(5);
   IntNode* node4 = new IntNode(7);

   delete node1;
   delete node2;
   delete node3;
   delete node4;

   return 0;
}
```

1

Check Next

Feedback?

CHALLENGE ACTIVITY | 8.8.2: Write a destructor ✓

Write a destructor for the CarCounter class that outputs the following. End with newline.

```
Destroying CarCounter
```

```
#include <iostream>
using namespace std;

class CarCounter {
   public:
      CarCounter();
      ~CarCounter();
   private:
      int carCount;
};

CarCounter::CarCounter() {
   carCount = 0;
}

/* Your solution goes here  */

int main() {
   CarCounter* parkingLot = new CarCounter();
   delete parkingLot;

```

Run

View your last submission ⌄

Feedback?

Exploring further:

- More on Destructors from msdn.microsoft.com
- Order of Destruction from msdn.microsoft.com