Warning: this example demonstrates implementation of a linked list base stack ADT. In practice, however, we must prefer to use `std::stack<T>`.

Stack.h

```
#ifndef STACK_H
#define STACK_H
#include<iostream>
class Stack
{
private:
   // representaion
   struct Node                      // member type of Stack
   {
      int data;
      struct Node *next;
      Node(int data, Node *next = nullptr) : data(data), next(next) {}
   };

   Node *head;                      // points to first node in the list; otherwise nullptr
   int nodeCount;                   // number of nodes

   // helper members
   void releaseResources();         // code common to both dtor + copy assignment
   void copyToThis(const Stack & s); // code common to both copy ctor + copy assignment
public:
   Stack();                        // default ctor
   Stack(const Stack & s);         // copy ctor
   Stack& operator=(const Stack & s); // copy assignment
   ~Stack();                           // dtor
   bool empty() const;
   void push(int);
   void pop();
   int top() const;
   int size()const;

   // examples of friend functions
   friend std::ostream & operator<<(std::ostream & out, const Stack::Node & node);
   friend std::ostream & operator<<(std::ostream & out, const Stack & s);
};
// examples of  members implemented inline
// inline member functions must be defined in the same header file
inline Stack::Stack() : head{ nullptr }, nodeCount{ 0 } {}
inline int Stack::size() const { return nodeCount; }
#endif
```

Stack.cpp

```cpp
#include<iostream>
#include<cassert>
#include "Stack.h"

bool Stack::empty() const
{
    return head == nullptr;
}

Stack::Stack(const Stack& s) : head{ nullptr }, nodeCount{ 0 }
{
    copyToThis(s);
}

Stack & Stack::operator=(const Stack & s) // allow multi-assignment
{
    if (this == &s)  return *this; // handle self assignment
    this->releaseResources();
    copyToThis(s);
    return *this;
}

Stack::~Stack()
{
    releaseResources();
}

void Stack::push(int val)
{
    head = new Node(val, head);
    ++nodeCount; // update size
}

void Stack::pop()
{
    assert(!empty());
    Node* temp = head;
    head = head->next;
    delete temp;
    --nodeCount; // update size
}

int Stack::top() const
{
    assert(!empty());
    return head->data;
}
```

Stack.cpp (Continued)

```
48 void Stack::releaseResources()
49 {
50    while (!empty())
51    {
52       this->pop();
53    }
54    /*
55    Node *temp;
56    while(head != nullptr)
57    {
58    temp = head;
59    head = head->next;
60    delete temp;
61    }
62    */
63 }
64
65 void Stack::copyToThis(const Stack & s)
66 {
67    if (s.head == nullptr)
68       head = nullptr;
69    else
70    {
71       head = new Node(s.head->data); // we'll keep head intact during construction
72       Node* temp1 = s.head->next;
73       Node* temp2 = head;
74       while (temp1 != nullptr)
75       {
76          temp2->next = new Node(temp1->data);// so next = nullptr
77          temp2 = temp2->next;
78          temp1 = temp1->next;
79       }
80    }
81 }
82
83 std::ostream & operator<<(std::ostream & out, const Stack::Node & node)
84 {
85    out << node.data;
86    return out;
87 }
88
89 std::ostream & operator<<(std::ostream & out, const Stack & s)
90 {
91    std::cout << "[";
92    for (Stack::Node * node = s.head; node != nullptr; node = node->next)
93    {
94       std::cout << *node << " "; // uses operator<< above
95       // or std::cout << (node->data) << " ";
96    }
97    std::cout << "]  size: " << s.nodeCount << "\n";
98    return out;
99 }
```

Quick Test Driver Code

```
#include<iostream>
#include<cassert>
#include "Stack.h"
using namespace std;

// sample test driver code
int main()
{
   Stack s;
   int choice;
   do
   {
      cout << "Stack Menu\n==========";
      cout <<  "\n1: push"
               "\n2: pop"
               "\n3: top"
               "\n4: copy"
               "\n5: assign"
               "\n6: print stack"
               "\n0: exit";
      cout << "\nEnter the number of your choice: ";
      cin >> choice;
      switch (choice)
      {
      case 0:
         break;
      case 1:
         int x;
         cin >> x;
         s.push(x);
         break;
      case 2:
         s.pop();
         break;
      case 3:
         cout << "top: " << s.top() << endl;
         break;
      case 4: { // copy ctor
         Stack s2(s);
         cout << s2 << endl;
         break; }
      case 5: { // assignment=
         Stack s3;
         s3 = s;
         cout << s3 << endl;
         break; }
      case 6:
         cout << s << endl;
         cout << "\n";
         break;
      default:
         cout << "\nBad choice. Try again.\n";
         break;
      }
   } while (choice != 0);
   return 0;
}
```