```
// $Id: linked_tstack.cpp,v 1.27 2015-01-27 17:54:09-08 - - $

//
// linked_tstack - show the linked list implementation of a stackk
//
// Deleting pointers in nodes is probably a bad idea here, since
// that will prevent the stack from sharing pointee objects with
// other data structures.
//

#include <cstddef>
#include <cstring>
#include <iostream>
#include <stdexcept>
#include <string>
#include <vector>
using namespace std;

/////////////////////////////////////////////////////////////////
// deleter classes
/////////////////////////////////////////////////////////////////

template <typename Type>
struct deleter {
   void operator() (const Type& p) {
      cout << "deleter(" << &p << ")" << endl;
   }
};

template <typename Type>
struct ptr_deleter {
   void operator() (const Type& p) {
      cout << "ptr_deleter(" << &p << ")" << endl;
      delete p;
   }
};

template <typename Type>
struct array_ptr_deleter {
   void operator() (const Type& p) {
      cout << "array_ptr_deleter(" << &p << ")" << endl;
      delete[] p;
   }
};

```

```

////////////////////////////////////////////////////////////////////
// linked_tstack.h
////////////////////////////////////////////////////////////////////

template <typename Type, class Deleter = deleter<Type>>
class linked_tstack {
   private:
      struct node {
         Type item;
         node* link;
         node (Type item, node* link): item(item), link(link) {}
      };
      node* top_ = nullptr;
      int size_ = 0;
      linked_tstack (const linked_tstack&) = delete;
      linked_tstack& operator= (const linked_tstack&) = delete;
   public:
      linked_tstack(): top_(nullptr), size_(0) {}
      ~linked_tstack();
      void push (const Type&);
      void pop();
      Type& top() { return top_->item; }
      const Type& top() const { return top_->item; }
      size_t size() const { return size_;}
      bool empty() const { return size_ == 0;}
};

////////////////////////////////////////////////////////////////////
// linked_tstack.cpp
////////////////////////////////////////////////////////////////////

template <typename Type, class Deleter>
linked_tstack<Type, Deleter>::~linked_tstack() {
   while (not empty()) pop();
}

template <typename Type, class Deleter>
void linked_tstack<Type, Deleter>::push (const Type& item) {
   top_ = new node (item, top_);
   ++size_;
}

template <typename Type, class Deleter>
void linked_tstack<Type, Deleter>::pop() {
   node* tmp = top_;
   top_ = top_->link;
   Deleter() (tmp->item);
   delete tmp;
   --size_;
}

```

```

////////////////////////////////////////////////////////////////
// main.cpp
////////////////////////////////////////////////////////////////

int main (int argc, char** argv) {
   vector<string> args (&argv[1], &argv[argc]);

   // First, with stack<string>:
   cout << "First:";
   linked_tstack<string> stkstr;
   for (string arg: args) {
      cout << " " << arg;
      stkstr.push (arg);
   }
   cout << endl;
   while (stkstr.size() > size_t (argc / 2)) {
      cout << "popping: " << stkstr.top() << endl;
      stkstr.pop();
   }

   // Second, with stack<string*>:
   cout << endl << "Second:";
   linked_tstack<string*, ptr_deleter<string*>> stkpstr;
   for (string arg: args) {
      string* str = new string (arg);
      cout << " " << *str;
      stkpstr.push (str);
   }
   cout << endl;
   while (stkpstr.size() > size_t (argc / 2)) {
      string* top = stkpstr.top();
      cout << "popping: " << top << "->" << *top << endl;
      stkpstr.pop();
   }

   // Finally, with stack<char[]>
   cout << endl << "Third:";
   linked_tstack<char*, array_ptr_deleter<char*>> argvstk;
   for (char** argi = &argv[1]; argi != &argv[argc]; ++argi) {
      char* str = new char[strlen (*argi) + 1];
      strcpy (str, *argi);
      cout << " " << str;
      argvstk.push (str);
   }
   cout << endl;

   return 0;
}

/*
//TEST// alias grind='valgrind --leak-check=full --show-reachable=yes'
//TEST// grind linked_tstack this is some test data for the stack \
//TEST//       >linked_tstack.out 2>&1
//TEST// mkpspdf linked_tstack.ps linked_tstack.cpp* linked_tstack.out*
*/

```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: starting linked_tstack.cpp
linked_tstack.cpp:
     $Id: linked_tstack.cpp,v 1.27 2015-01-27 17:54:09-08 - - $
g++ -g -O0 -Wall -Wextra -rdynamic -std=gnu++11 linked_tstack.cpp -o linked_tstack -lglut -lGLU -lGL -lX11 -lrt -lm
rm -f linked_tstack.o
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ mkc: finished linked_tstack.cpp
```

```
==9970== Memcheck, a memory error detector
==9970== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==9970== Using Valgrind-3.9.0 and LibVEX; rerun with -h for copyright info
==9970== Command: linked_tstack this is some test data for the stack
==9970==
First: this is some test data for the stack
popping: stack
deleter(0x4e7d640)
popping: the
deleter(0x4e7d5f0)
popping: for
deleter(0x4e7d5a0)
popping: data
deleter(0x4e7d550)

Second: this is some test data for the stack
popping: 0x4e7daf0->stack
ptr_deleter(0x4e7db40)
popping: 0x4e7da50->the
ptr_deleter(0x4e7daa0)
popping: 0x4e7d9b0->for
ptr_deleter(0x4e7da00)
popping: 0x4e7d910->data
ptr_deleter(0x4e7d960)

Third: this is some test data for the stack
array_ptr_deleter(0x4e7e040)
array_ptr_deleter(0x4e7dfa0)
array_ptr_deleter(0x4e7df00)
array_ptr_deleter(0x4e7de60)
array_ptr_deleter(0x4e7ddc0)
array_ptr_deleter(0x4e7dd20)
array_ptr_deleter(0x4e7dc80)
array_ptr_deleter(0x4e7dbe0)
ptr_deleter(0x4e7d8c0)
ptr_deleter(0x4e7d820)
ptr_deleter(0x4e7d780)
ptr_deleter(0x4e7d6e0)
deleter(0x4e7d500)
deleter(0x4e7d4b0)
deleter(0x4e7d460)
deleter(0x4e7d410)
==9970==
==9970== HEAP SUMMARY:
==9970==     in use at exit: 0 bytes in 0 blocks
==9970==   total heap usage: 50 allocs, 50 frees, 787 bytes allocated
==9970==
==9970== All heap blocks were freed -- no leaks are possible
==9970==
==9970== For counts of detected and suppressed errors, rerun with: -v
==9970== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```