栈的链式实现

链栈

typedef Stack_entry Node_entry;(用结点来存放栈的元素)

Whether the beginning or the end of the linked structure
will be the top of the stack?

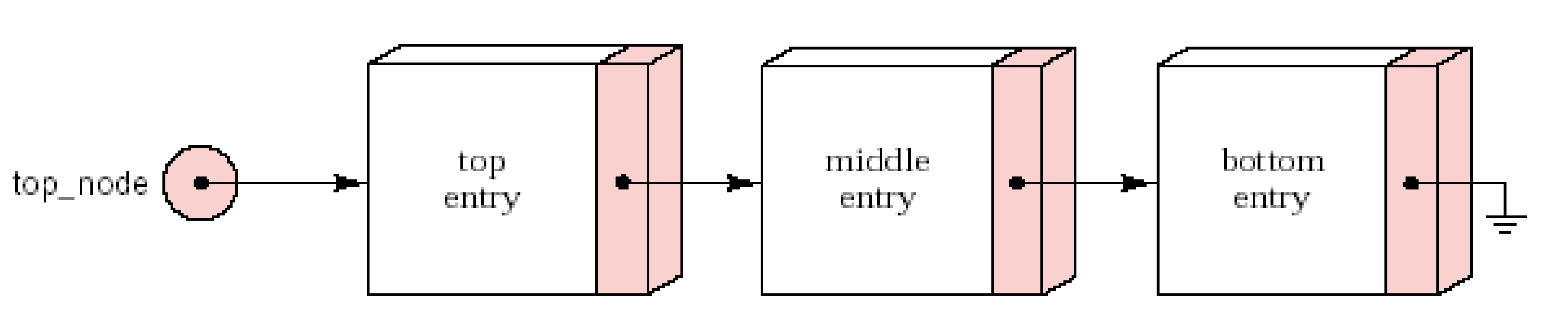


Figure 4.9. The linked form of a stack

The only information needed to keep track of the data in a linked stack is the location of its top.

链栈

- declaration of type Stack

```
class Stack {
public:
        Stack( );
        bool empty( ) const;
        Error_code push(const Stack_entry &item);
        Error_code pop( );
        Error_code top(Stack_entry &item) const;
protected:
        Node *top_node;
};
```

链栈

- pushing a linked stack（入栈）

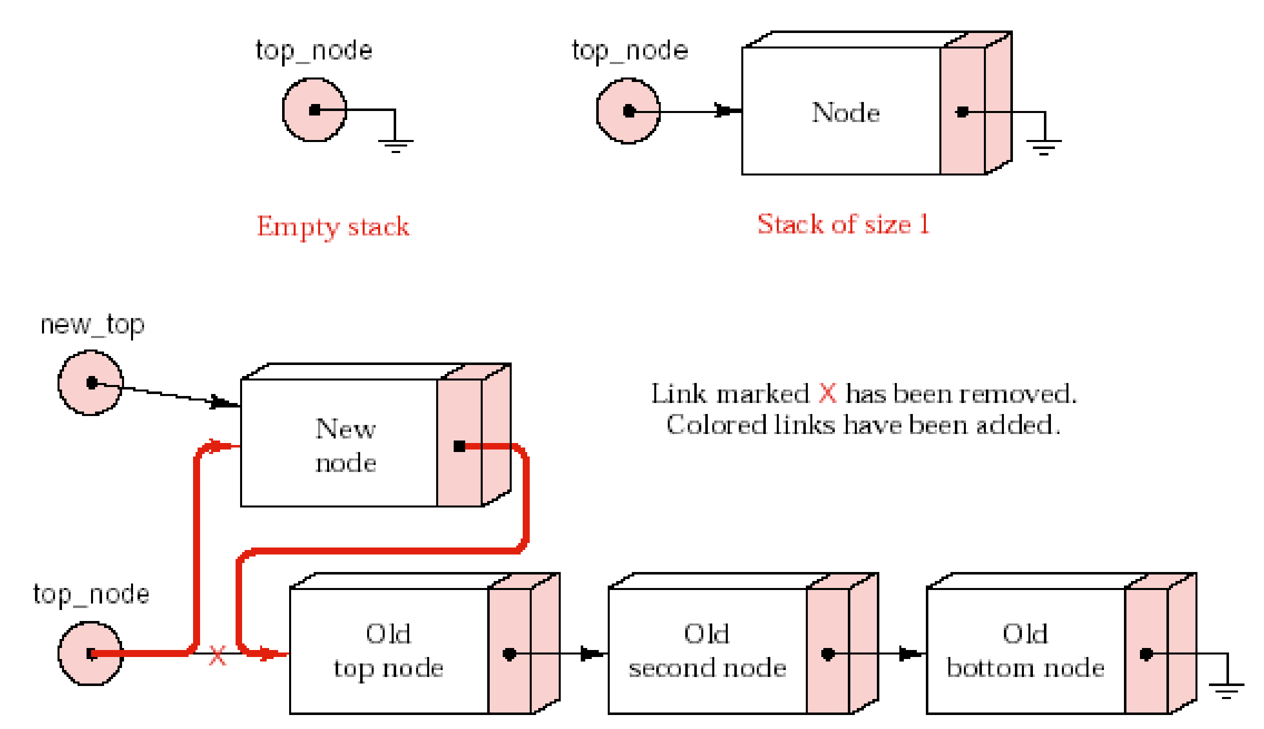


Figure 4.10. Pushing a node onto a linked stack

链栈

- □ pushing a linked stack（入栈）

```
Error_code Stack :: push(const Stack_entry &item)
/* Post: Stack_entry item is added to the top of the Stack; returns
success or returns a code of overflow if dynamic memory is
exhausted. */
{
Node *new_top = new Node(item, top_node);
if (new_top == NULL) return overflow;
top_node = new_top;
return success;
}
```

链栈

- ❑ popping a linked stack（出栈）

```
Error_code Stack :: pop( )
/* Post: The top of the Stack is removed. If the Stack is empty the
method returns underflow; otherwise it returns success. */
{
Node *old_top=top_node;
if (top_node==NULL) return underflow;
top_node=top_node->next;
delete old_top;
return success;
}
```

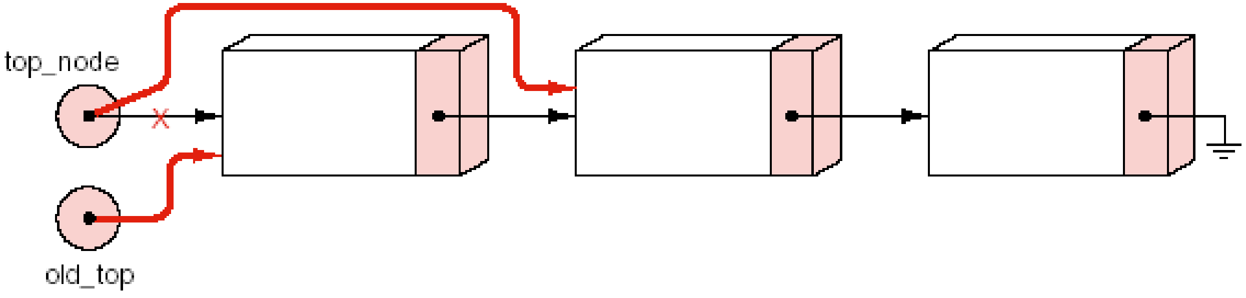


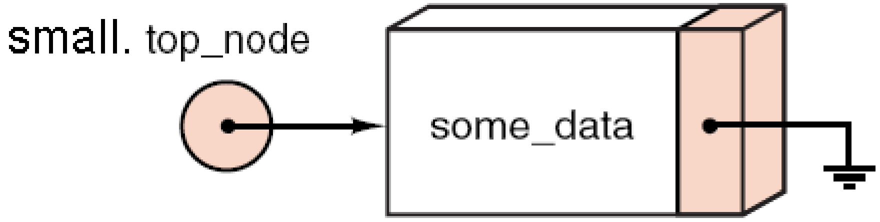
Figure 4.11. Popping a node from a linked stack

链栈的几点安全性修正

- Problem Example

```
for (int i = 0; i < 1000000; i++) {
        Stack small;
        small.push(some_data);
}
```

As soon as the object small goes out of scope, the data stored in small becomes garbage. Over the course of a million iterations of the loop, a lot of garbage will accumulate.



链栈的几点安全性修正

❑ The Destructor

C++ 中提供了析构函数，于对象死亡前系统自动调用，用于释放相关资源。

```
Stack :: ~ Stack( ) // Destructor
/* Post: The Stack is cleared. */
{
while (!empty( ))
            pop( );
}
```

链栈的几点安全性修正

❑

```
Stack outer_stack;
for (int i = 0; i < 1000000; i++) {
        Stack inner_stack;
        inner_stack.push(some_data);
        inner_stack = outer_stack;
}
```

存在的错误:

数据空间丢失

两个栈共享节点

inner_stack删除了outer_stack的内容，导致outer_stack.top_ node的指向无效

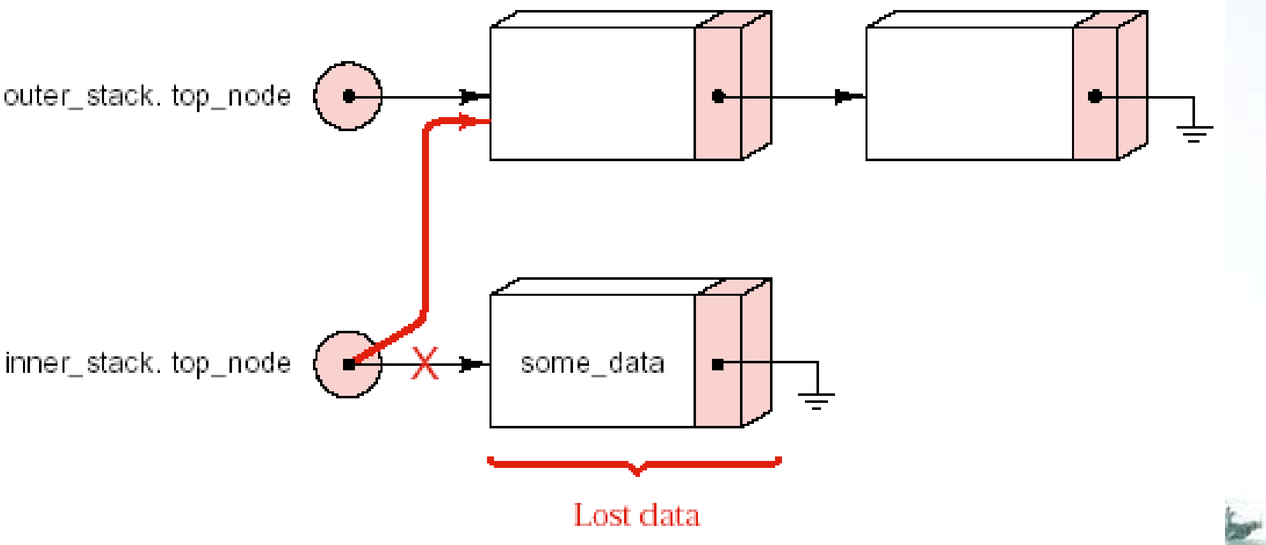


Figure 4.12. The application of bitwise copy to a Stack

链栈的几点安全性修正

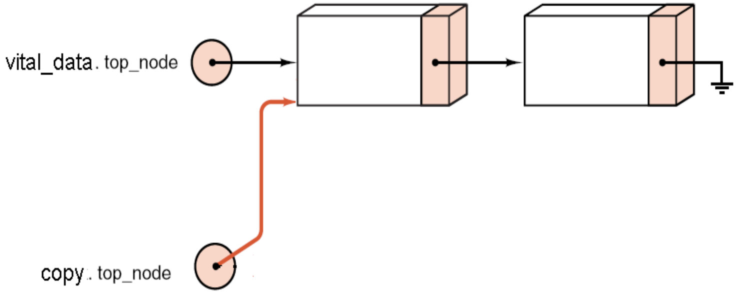
```
void Stack :: operator = (const Stack &original) // Overload assignment
/* Post: The Stack is reset as a copy of Stack original. */
{     Node *new_top, *new_copy, *original_node = original.top_node;
      if (original_node == NULL) new_top = NULL;
      else { // Duplicate the linked nodes
            new_copy = new_top = new Node(original_node->entry);
            while (original_node->next != NULL)      {
                  original_node = original_node->next;
                  new_copy->next = new Node(original_node->entry);
                  new_copy = new_copy->next;
                }
             }
      while (!empty( )) // Clean out old Stack entries
            pop( );
      top_node = new_top; // and replace them with new entries.
}
```

链栈的几点安全性修正

❑ 拷贝构造函数

- Problem example:

```
void destroy_the_stack (Stack copy)
{
}
int main( )
{
        Stack vital_data;
        ……
        destroy_the_stack(vital_data);
}
```



链栈的几点安全性修正

```
Main(){
      Stack p1;
      P1.push(somedata);
      Stack p2(p1);//Stack p2=p1;
      .....
}
```

解决方案：拷贝构造函数

Stack :: Stack(**const** Stack &original);

链栈的几点安全性修正

```
Stack :: Stack(const Stack &original) // copy constructor
/* Post: The Stack is initialized as a copy of Stack original. */
{
    Node *new_copy, *original_node = original.top_node;
    if (original_node == NULL) top_node = NULL;
    else { // Duplicate the linked nodes.
        top_node = new_copy = new Node(original_node->entry);
        while (original_node->next != NULL) {
        original_node = original_node->next;
        new_copy->next = new Node(original_node->entry);
        new_copy = new_copy->next;
  }}}
```

链栈的几点安全性修正

有了拷贝构造函数，赋值运算的重载可变为：

```
void Stack :: operator = (const Stack &original) {
    Stack new_copy(original);

    top_node = new_copy.top_node;
    }
```

修正后的链栈

```
class Stack {
    public:// Standard Stack methods
        Stack( );
        bool empty( ) const;
        Error_code push(const Stack_entry &item);
        Error_code pop( );
        Error_code top(Stack_entry &item) const;
    // Safety features for linked structures
        ~Stack( );
        Stack(const Stack &original);
        void operator = (const Stack &original);
    protected:
        Node *top_node;
    };
```