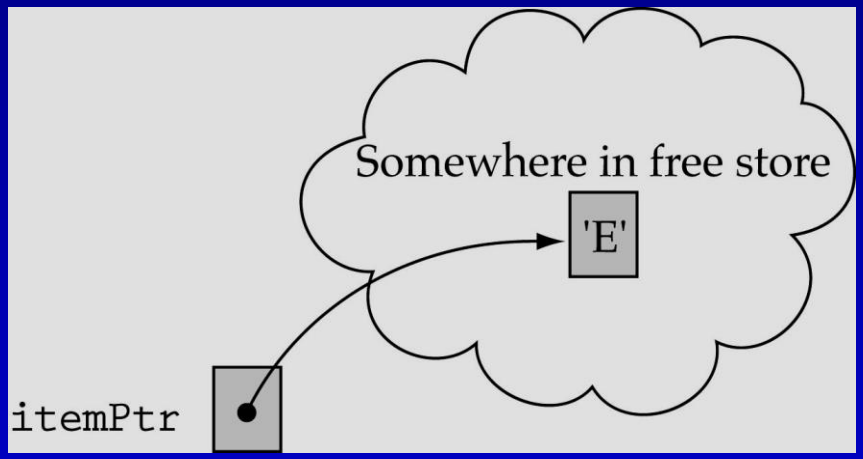
Dynamic allocation of each stack element

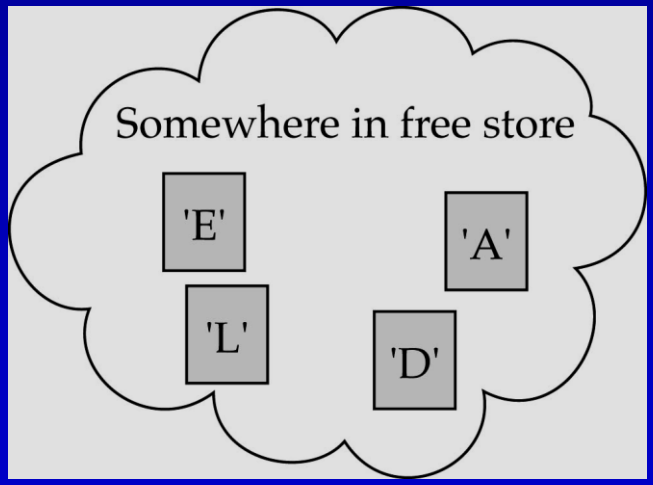
- Allocate memory for each new element dynamically



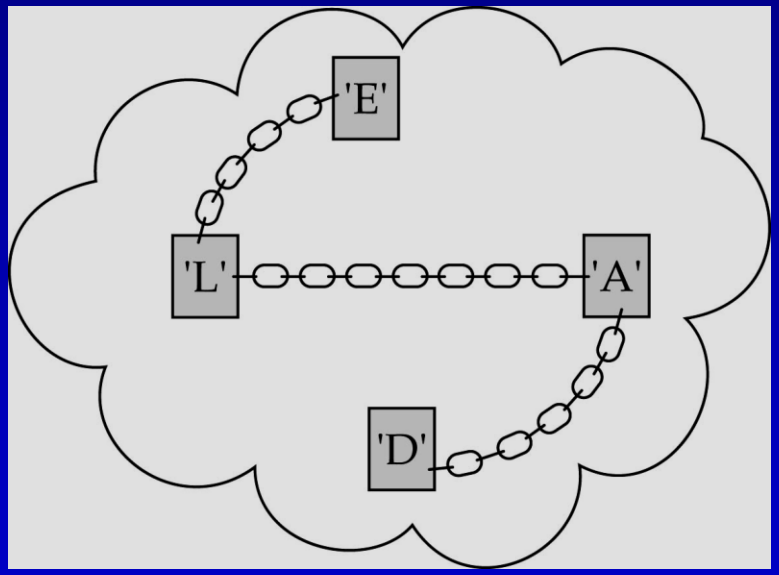

```
ItemType* itemPtr;
 ...
itemPtr = new ItemType;
*itemPtr = newItem;
```

Dynamic allocation of each stack element (cont.)

- How should we preserve the order of the stack elements?

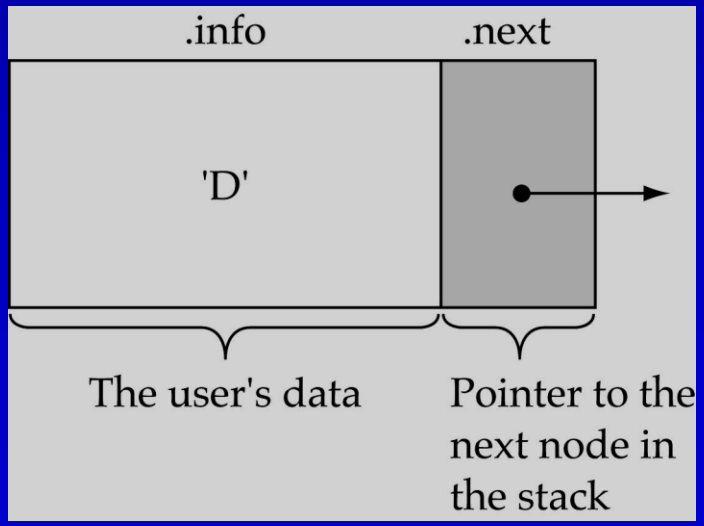


Chaining the stack elements together



Chaining the stack elements together (cont.)

- Each node in the stack should contain two parts:
  - info: the user's data
  - next: the address of the next element in the stack

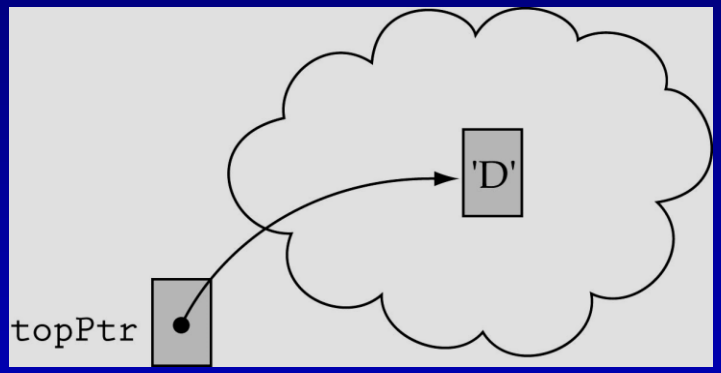


Node Type

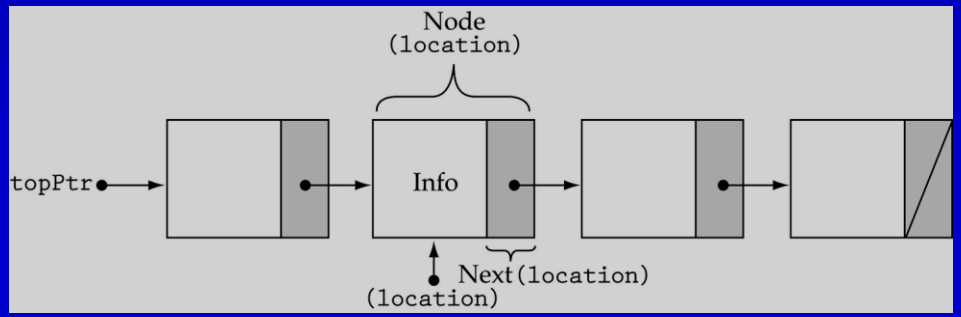
```
template<class ItemType>
 struct NodeType {
   ItemType info;
   NodeType* next;
 };
```

First and last stack elements

- We need a data member to store the pointer to the top of the stack



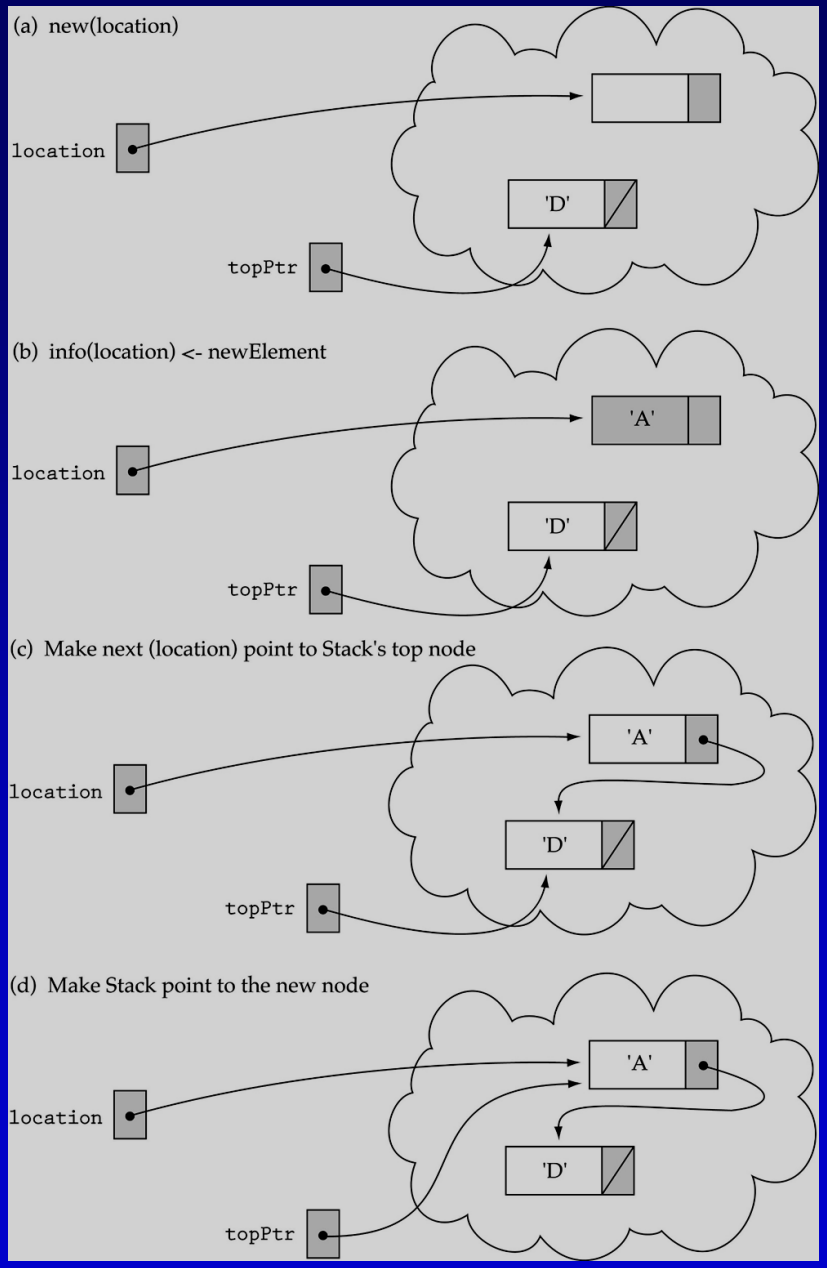
- The *next* element of the last node should contain the value *NULL*



Stack class specification

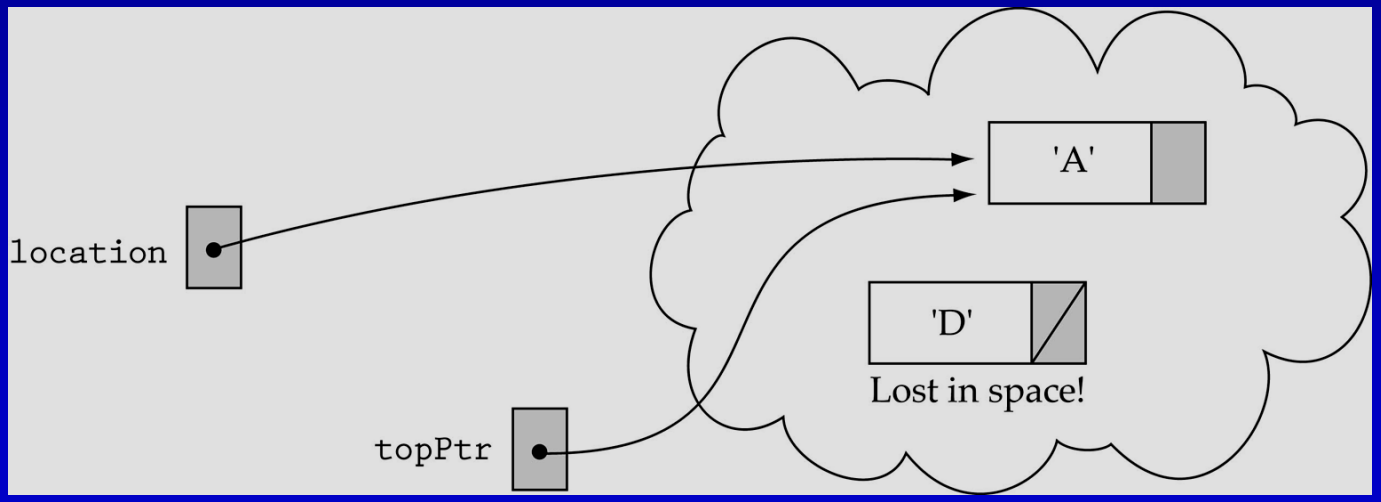
```
// forward declaration of NodeType (like function prototype)
template<class ItemType>
struct NodeType;

template<class ItemType>
class StackType {
 public:
    StackType();
    ~StackType();
    void MakeEmpty();
    bool IsEmpty();
    void Push(ItemType);
    void Pop(ItemType&);
 private:
    NodeType<ItemType>* topPtr;
};
```

Pushing on a non-empty stack

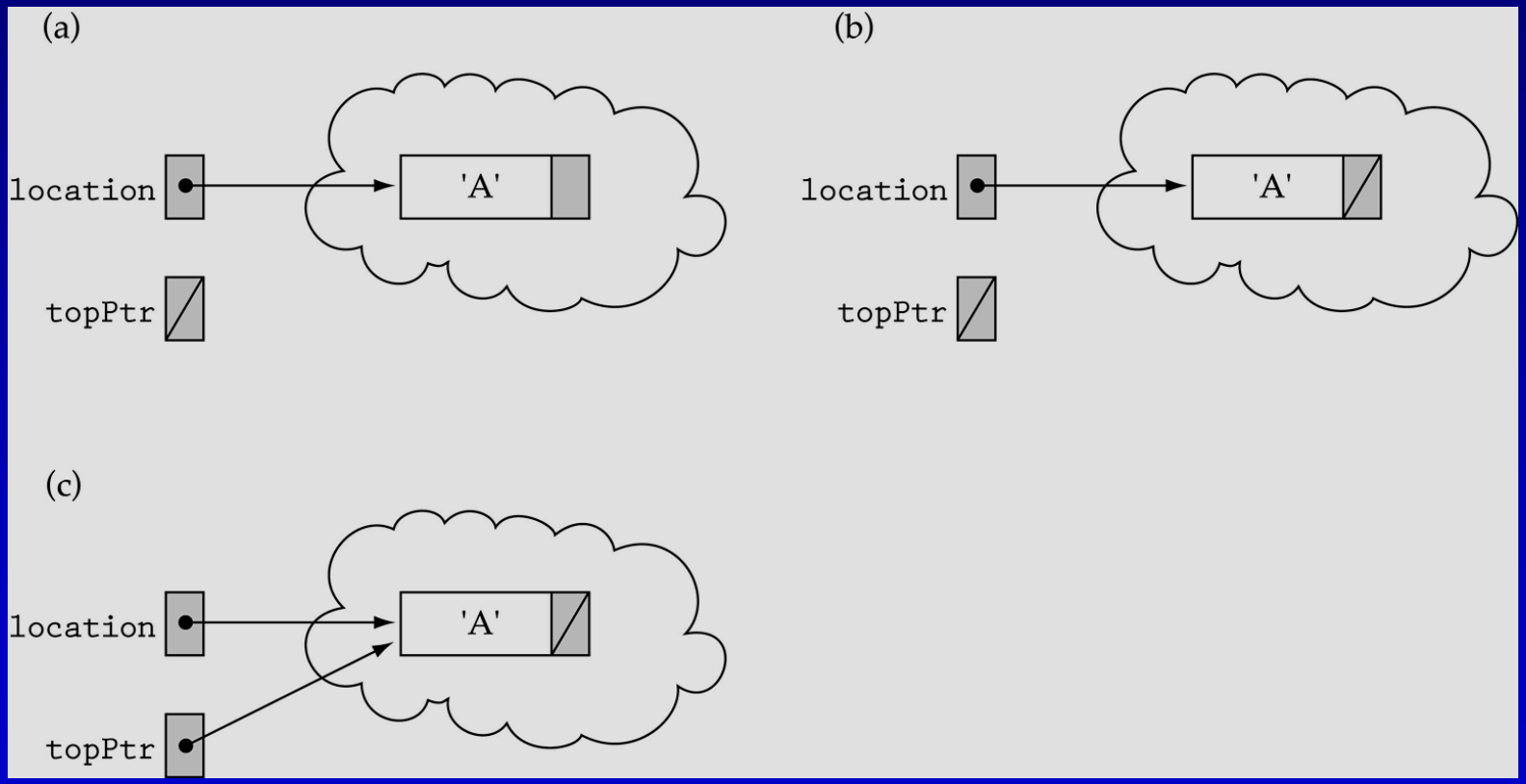


Pushing on a non-empty stack (cont.)

- The order of changing the pointers is very important !!



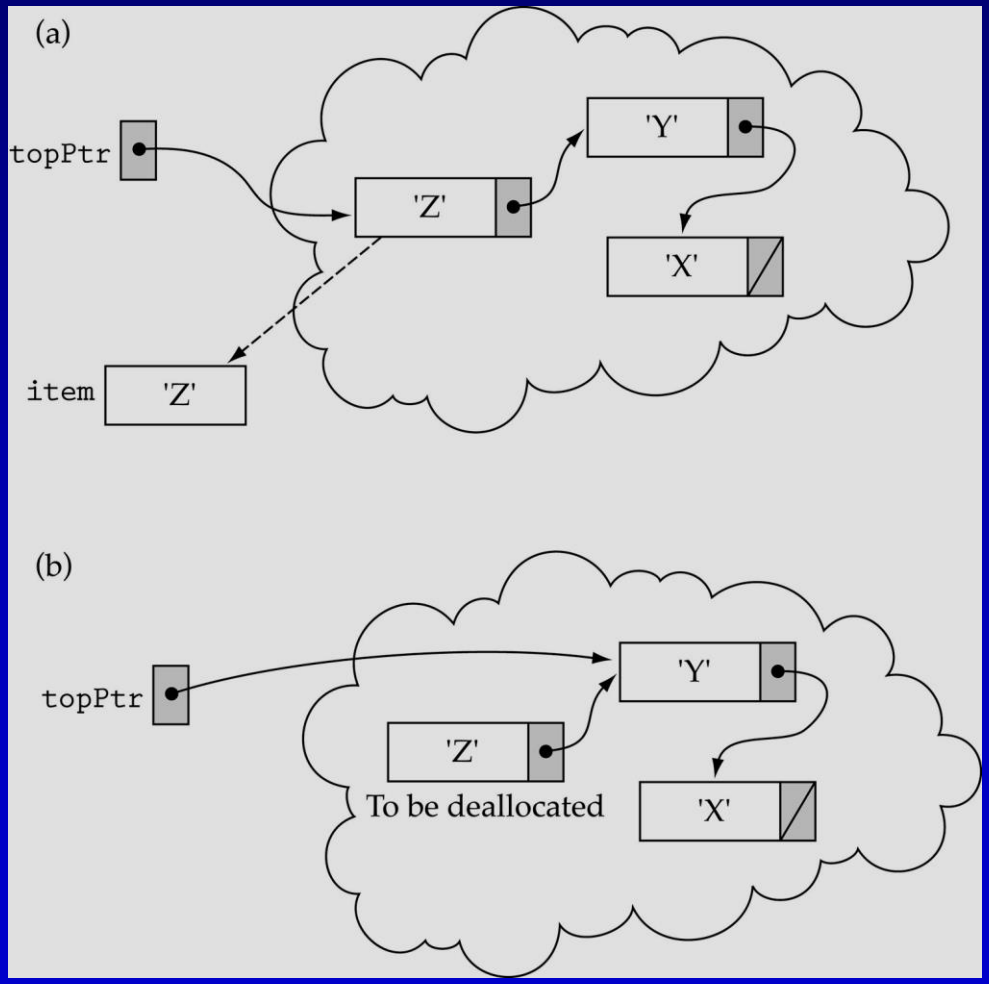
Pushing on an empty stack



Function Push

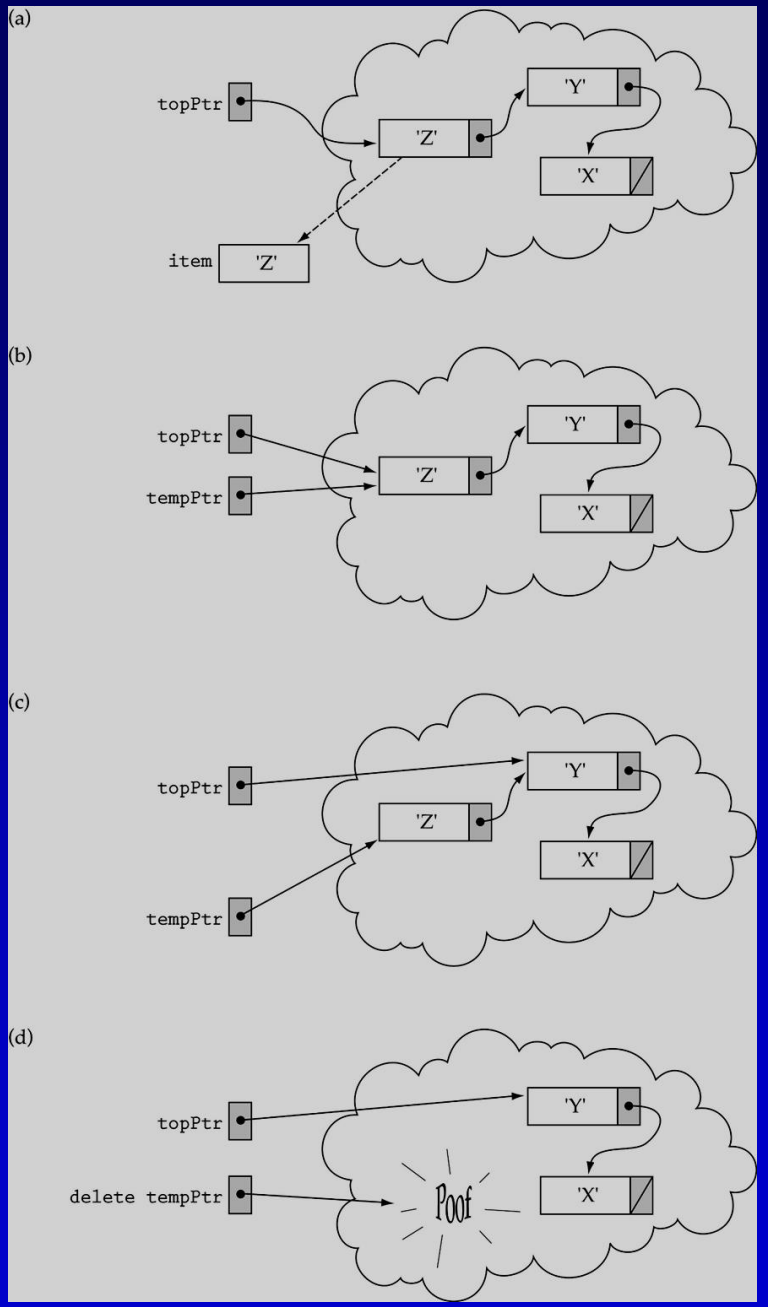
```
template <class ItemType>
void StackType<ItemType>::Push(ItemType
    item)
{
 NodeType<ItemType>* location;

 location = new NodeType<ItemType>;
 location->info = newItem;
 location->next = topPtr;
 topPtr = location;
}
```

Popping the top element



Popping the top element (cont.)

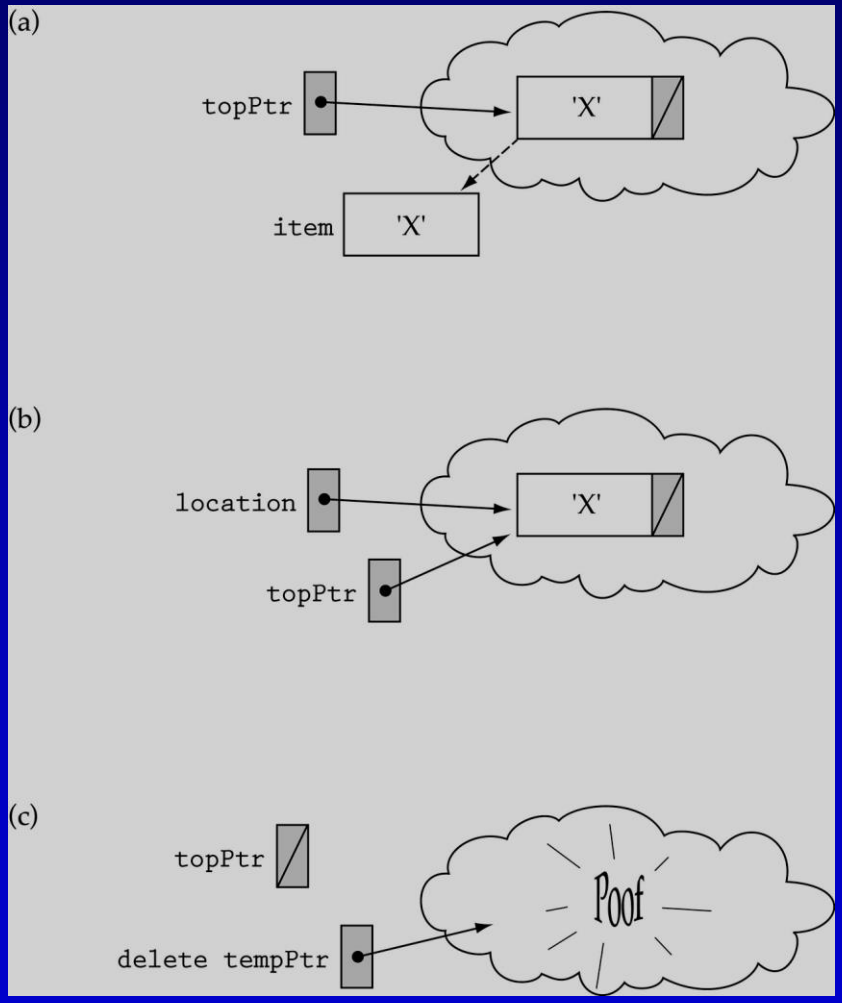
Need to use a temporary pointer



Function Pop

```
template <class ItemType>
void StackType<ItemType>::Pop(ItemType& item)
{
 NodeType<ItemType>* tempPtr;

 item = topPtr->info;
 tempPtr = topPtr;
 topPtr = topPtr->next;
 delete tempPtr;
}
```

Popping the last element on the stack



Other Stack functions

```
template<class ItemType>
StackType<ItemType>::StackType()
{
 topPtr = NULL;
}

template<class ItemType>
void StackType<ItemType>::MakeEmpty()
{
 NodeType<ItemType>* tempPtr;

 while(topPtr != NULL) {
   tempPtr = topPtr;
   topPtr = topPtr->next;
   delete tempPtr;
 }
}
```

Other Stack functions (cont.)

```
template<class ItemType>
bool StackType<ItemType>::IsEmpty()
{
 return(topPtr == NULL);
}

template<class ItemType>
StackType<ItemType>::~StackType()
{
 MakeEmpty();
}
```