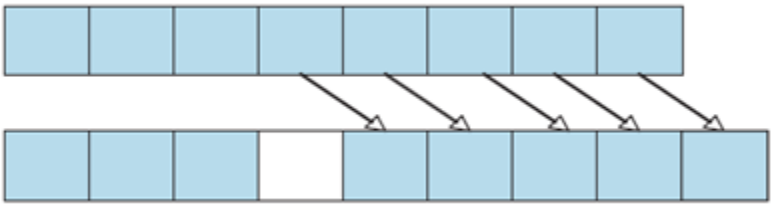
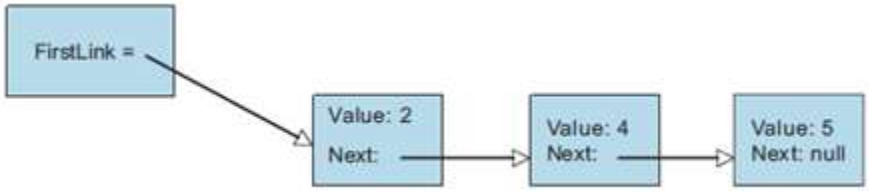
Worksheet 17: Linked List Introduction, List Stack

In Preparation: Read Chapter 6 to learn more about the Stack data type.
In Worksheet 14, you built a Stack using a dynamic array as the underlying container. A weakness of the Dynamic Array is that elements are stored in a contiguous block. As a consequence, when a new element is inserted into the middle of the collection, all the adjacent elements must be moved in order to make space for the new value.

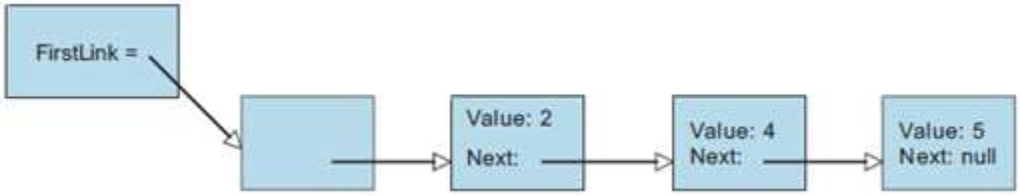


```
struct link {
   TYPE value;
   struct link * next;
};
```

An alternative approach is to use the idea of a *Linked List*. In a linked list each value is stored in a separate block of memory, termed a *link*. In addition to a value, each link contains a reference to the next link in sequence. As a data structure, a link can be described as shown at right.
We can visualize collection formed out of links as follows. A data field named firstLink will hold the first link in the collection. Each link refers to the next. The final link will have a **null** value in the next field:



The simplest data structure to create using links is a Stack. When a new element is pushed on the stack a new link will be created and placed at the front of the chain.



To remove a link the variable **firstLink** is simply changed to point to the next element in the chain. The space for the Link must then be freed.
The following is the beginning of an implementation of a **LinkedListStack** based on these ideas. Complete the implementation. Each operation should have constant time performance. Use an assertion to ensure that when a top or pop is performed the stack has at least one element. When you pop a value from the stack, make sure you free the link field.

```
struct link {
  TYPE value;
  struct link * next;
};
struct linkedListStack {
  struct link *firstLink;
}
void linkedListStackInit (struct linkedListStack * s)
{ s->firstLink = 0; }
void linkedListStackFree (struct linkedListStack *s)
        { while (! linkedListStackIsEmpty(s)) linkedListStackPop(s); }
void linkedListStackPush (struct linkedListStack *s, TYPE d) {
        struct link * newLink = (struct link *) malloc(sizeof(struct link));
        assert (newLink != 0);
/* Fix me */
        newLink->value = d;
        newLink->next = s->firstLink;
        s->firstLink = newLink;
}
TYPE linkedListStackTop (struct linkedListStack *s) {
/* Fix me */
        assert (!linkedListStackIsEmpty(s));
        return s->firstLink->value;
}
void linkedListStackPop (struct linkedListStack *s) {
/* Fix me */
        assert (!linkedListStackIsEmpty(s));

        struct link *temp;
        temp = s->firstLink;

        s->firstLink = s->firstLink->next;

        free(temp);
        temp = 0;
}


int linkedListStackIsEmpty (struct linkedListStack *s) {
/* Fix me */
        if (s->firstLink == 0){
                return 1;
        }else{
                return 0;
        }
}
```

Function Test Results

Create and Initialize Stack

Stack is Empty: TRUE

Push Stack: 5
Stack Contents = 5
Push Stack: 4
Stack Contents = 4 5
Push Stack: 3
Stack Contents = 3 4 5
Push Stack: 2
Stack Contents = 2 3 4 5
Push Stack: 1
Stack Contents = 1 2 3 4 5

Pop Stack: 1
Stack Contents = 2 3 4 5
Pop Stack: 2
Stack Contents = 3 4 5
Pop Stack: 3
Stack Contents = 4 5
Pop Stack: 4
Stack Contents = 5
Pop Stack: 5

Stack Contents =
Stack is Empty: TRUE

Free Stack