**[Computer science course notes](http://www.microveggies.com/csci)**

**Home**

[**Lecture12 - Stacks**](http://www.microveggies.com/csci/index.php/csci-2270-lecture-notes/5-lecture12-stacks)

**Details**

Written by Rhonda Hoenigman

 Published: 03 March 2015

 Hits: 486

Stacks are used when we want to process data in a last in, first out order (LIFO). This means that the last item added to the stack is the first item removed from the stack, much like a stack of cafeteria plates. Assume you have the following sentence and you want to store the individual words on a stack:

*"A liger it's pretty much my favorite animal."*

You add the words to the stack from the bottom to the top



**Push an item onto the stack**

When you add an item to a stack, you say "push" onto the stack.

**Pop an item off the stack**

When you remove an item from a stack, you say "pop" off the stack.

When you push and pop to and from the stack, you operate on the top of the stack only. After popping the words from this stack, you would have the sentence:

*"animal favorite my much pretty it's liger A"*

**Example of where stacks are used - Computer program execution**

When a computer program is running, information about currently active subroutines are stored on a call stack. As new subroutines become active, they are added to the stack, and as subroutines complete, they are popped off the stack. For example, consider the following simple program:

int addNums(int a, int b){

  return a + b;

}

void callAddNums(){

  int c = addNums(5, 6);

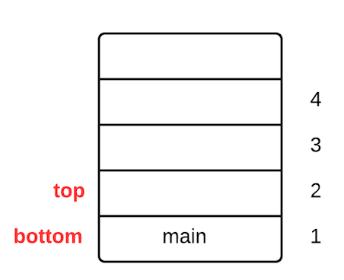
}

int main(){

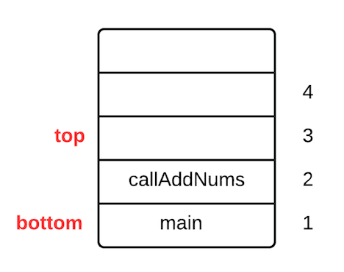
  callAddNums();

}

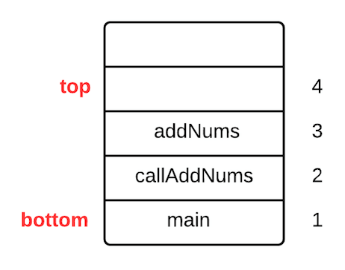
The program starts at the function main, and main is pushed onto the call stack:



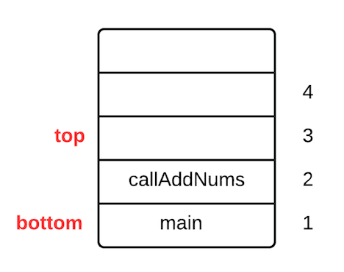
 Next, main calls callAddNums, which makes that subroutine active and it is pushed onto the stack:



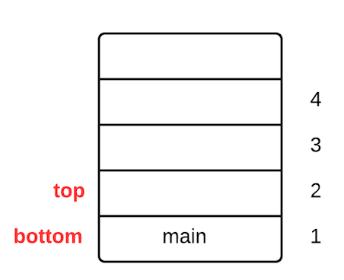
Next, callAddNums calls addNums, which makes that subroutine active and it is pushed onto the stack. At this point, all three subroutines are active because none of them has completed execution, and the call stack looks something like this:



When addNums() completes, it is popped off the stack, and the stack looks like.



Next, callAddNums() completes and it is popped off the stack:

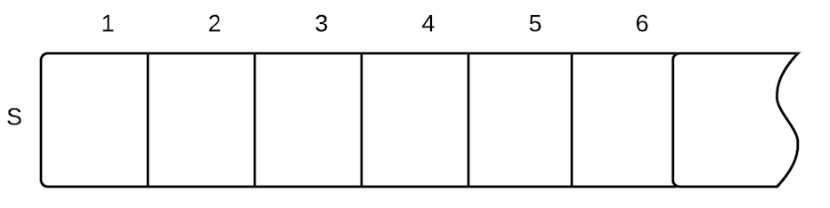


Finally, main() completes. It is popped off the stack and the program completes.

**Array implementation of a stack**

A stack can be implemented with other data structures, such as an array or a linked list. In the array implementation of a stack, items are stored in an array and the top of the stack refers to the index where the next item will be added.

Start with an empty stack, S



Initialize: top = 0 //Assumes a pseudocode convention of indexing starting at 1. In code, index would start at 0 and top would be initialized to -1

Pseudocode:

Push(S, v) //S is the stack, v is the value to push

  S.top = S.top + 1

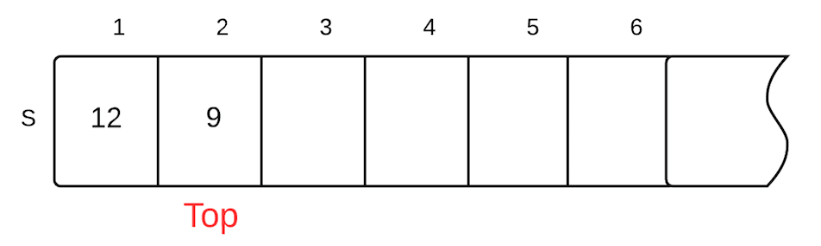
  S[top] = v

**Example:**

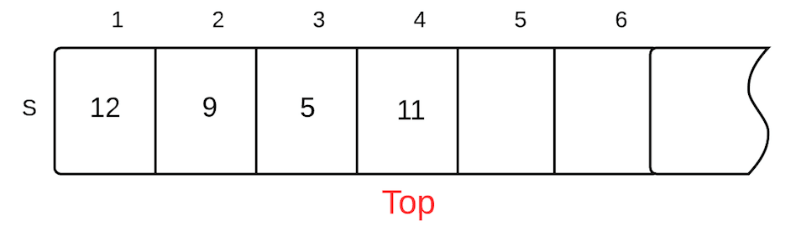
Push(S, 12)

Push(S, 9)

generates:



**Stack Terminology**



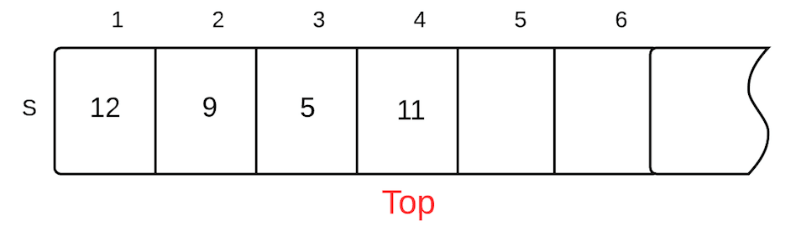
The stack S has 4 items, S[1...S.top] are the contents of the stack.

When S.top = 0, the stack is empty.

When S.top = n, the stack is full. (n is the size of the array)

When S.top > n, we say stack overflow. The pseudocode for the Push operation doesn't handle stack overflow.

**Remove an item from the stack**



Pop(S)

  if S.top == 0

    print "underflow error"

  else

    top = top - 1

  return S[top + 1]

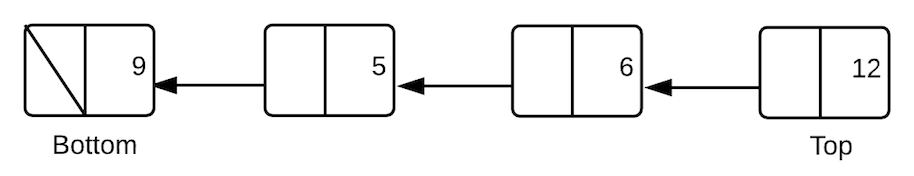
**Example:**

Use the diagram of stack S as reference

Pop(S) will set top to 3, and then return S[4], which is 11.

**Linked-list implementation of a stack**

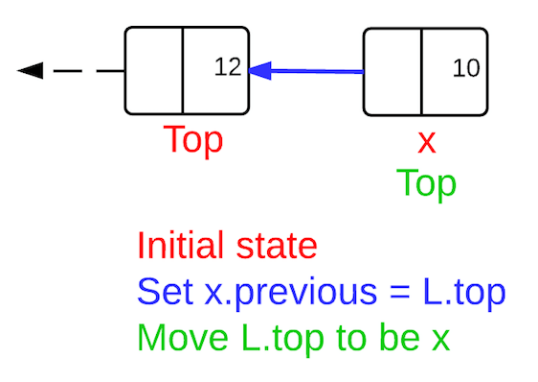
In the linked-list implementation of a stack, each item in the stack is stored in a node in a singly linked list. Each node contains a pointer to the previous node in the list. The node at the bottom of the list has a previous pointer to NULL.



 Push(L, x)  //L is the linked list, x is the new node

  x.previous = L.top

  L.top = x

Push operation on a linked-list stack

Pop(L)

   if L.top == NULL

      print "underflow"

   else

      x = L.top

      L.top = L.top.prev

   return x

[Back to Top](http://www.microveggies.com/csci/index.php/8-csci-2270-lecture-notes/5-lecture12-stacks#top)

© 2015 Computer science course notes