Stack

Stack

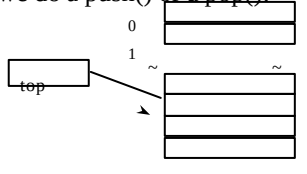
- Stacks are a specific kind of linked list (Constrained version of a linked list).
- Stacks have specific adds and removes called push and pop.
- Pushing nodes onto stacks is easily done by adding to the front of the list. Popping is simply removing from the front of the list.
- Last-in, first-out (LIFO) – New nodes can be added and removed only at the top.
- It would be wise to give return values when pushing and popping from stacks.
- Bottom of stack indicated by a link member to **`NULL`**

push & pop

- **`push`**
  - Adds a new node to the top of the stack
- **`pop`**
  - Removes a node from the top
  - Stores the popped value
  - Returns **`true`** if **`pop`** was successful

Stacks using arrays

- We use an array in which the data doesn't move when we do a push() or a pop():




```
push if not full:
insert data at cell top+1;
top++;
```

```
pop if not empty:
remove data from cell top;
top--;
```

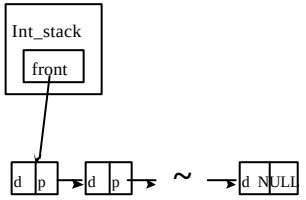
Stacks Using Linked Lists

- Data is added and removed from the same end (the head) of the list. Implementation is trivial using a singly linked list:

```
class Stack_node {
int datum;
Stack_node * next;
};
```

```
class Int_stack {
List_node * front;
};
```

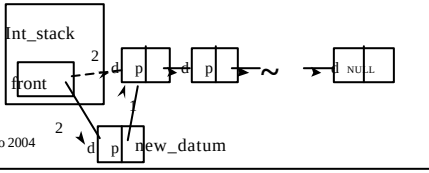
```
Int_stack s;
```



Stacks Using Linked Lists

- Here the interface uses the actual data, not pointers to nodes:

```
void push(int new_datum) {
Stack_node *ptr = new Stack_node;
   ptr -> datum = new_datum;
   ptr -> next = s.front;     /* step 1 */
   s.front = ptr;             /* step 2 */
}
```



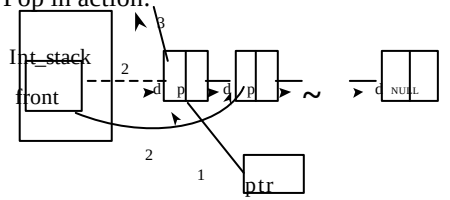
Stacks Using Linked Lists

- pop( ) must free the space occupied by the top node:

```
int pop(void)
{
    Stack_node *ptr = s.front;          /* step 1 */
    int result;
    if(s.front!=NULL) {
        result = s.front -> datum;
        s.front = s.front -> next;      /* step 2 */
        free(ptr);
        return result;                  /* step 3 */
    }
    return ERROR;          /* a special value to indicate empty stack */
}
```


Stacks Using Linked Lists

- Pop in action:



Example 10A

```
/* 10A
dynamic stack program */
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

struct stackNode {   /* self-referential structure */
   int data;
   struct stackNode *nextPtr;
};
typedef struct stackNode StackNode;
typedef StackNode *StackNodePtr;
void push( StackNodePtr *, int );
int pop( StackNodePtr * );
int isEmpty ( StackNodePtr );
void printStack( StackNodePtr );
void instructions( void );
```


Example 10A (cont')

```
int main()
{
   StackNodePtr stackPtr = NULL;  /* points to stack top */
   int choice, value;

   instructions();
   printf( "? " );
   scanf( "%d", &choice );

   while ( choice != 3 ) {

      switch ( choice ) {
         case 1:     /* push value onto stack */
            printf( "Enter an integer: " );
            scanf( "%d", &value );
            push(&stackPtr, value );
            printStack ( stackPtr );
            break;
```


Example 10A (cont')

```
         case 2:     /* pop value off stack */
            if ( !isEmpty( stackPtr ) )
               printf( "The popped value is %d\n",
                     pop(&stackPtr) );

            printStack ( stackPtr );
            break;
         default:
            printf( "Invalid choice. \n\n" );
            instructions();
            break;
      }

      printf( "? " );
      scanf ( "%d", &choice );
   }

   printf( "End of run. \n" );
   getch();
   return 0;
}
```


Example 10A (cont')

```
/* Print the instructions */
void instructions( void )
{
   printf( "Enter choice:\n"
           "1 to push a value on the stack\n"
           "2 to pop a value off the stack\n"
           "3 to end program \n" );
}

/* Insert a node at the stack top */
void push( StackNodePtr *topPtr, int info )
{
   StackNodePtr newPtr;
   (void *) newPtr = malloc ( sizeof( StackNode ) );
   if ( newPtr != NULL ) {
      newPtr->data = info;
      newPtr->nextPtr = *topPtr;
      *topPtr = newPtr;
   }
   else
      printf( "%d not inserted. No memory available. \n",
            info );
}
```

Example 10A (cont')

```
/* Remove a node from the stack top */
int pop( StackNodePtr *topPtr )
{   StackNodePtr tempPtr;
  int popValue;
  tempPtr = *topPtr;
  popValue = (*topPtr )->data;
  *topPtr = ( *topPtr )->nextPtr;
  free( tempPtr );
  return popValue;
}

/* Print the stack */
void printStack( StackNodePtr currentPtr )
{   if ( currentPtr == NULL )
      printf( "The stack is empty.\n\n" );
    else {
      printf( "The stack is: \n" );
      while ( currentPtr != NULL ) {
        printf( "%d --> ", currentPtr->data );
        currentPtr = currentPtr->nextPtr;
      }
      printf( "NULL\n\n" );
    }
}
```


Example 10A (cont')

```
/* Is the stack empty? */
int isEmpty ( StackNodePtr topPtr )
{
  return topPtr == NULL;
}
```


Output Result