Stack -- Array Implementation

```
#include <stdio.h>
#define SIZE 10
#define EMPTY -1

typedef struct stack {
    int items[SIZE];
    int top;
} stack;

void initialize(stack* stackPtr);
int full(stack* stackPtr);
int push(stack* stackPtr, int value);
int empty(stack* stackPtr);
int pop(stack* stackPtr);
int top(stack* stackPtr);

void initialize(stack* stackPtr) {
     stackPtr->top = -1;
}

int push(stack* stkPtr, int val) {
    if (full(stkPtr))
        return 0;

    stkPtr->items[stkPtr->top+1]= val;
    (stkPtr->top)++;
    return 1;
}

int full(stack* stkPtr) {
    return (stkPtr->top == SIZE - 1);
}

int empty(stack* stackPtr) {
    return (stackPtr->top == -1);
}

int pop(stack* stkPtr) {
    int retval;

    if (empty(stkPtr))
        return EMPTY;

    retval=stkPtr->items[stkPtr->top];
    (stkPtr->top)--;
    return retval;
}

int top(stack* stkPtr) {
    if (empty(stkPtr))
        return EMPTY;

    return stkPtr->items[stkPtr->top];
}
```

```
int main() {
    int i;
    stack mine;

    // create stack
    initialize(&mine);

    // push some items
    push(&mine, 4);
    push(&mine, 5);

    // pop an item
    printf("Popping %d\n",
            pop(&mine));


    // Push a couple more, test top.
    push(&mine, 22);
    push(&mine, 16);
    printf("At top now = %d\n",
            top(&mine));


    // Pop all three off.
    printf("Popping %d\n",
            pop(&mine));
    printf("Popping %d\n",
            pop(&mine));
    printf("Popping %d\n",
            pop(&mine));


    // Checking the empty function.
    if (empty(&mine))
        printf("Stack empty\n");


    // Fill the stack.
    for (i = 0; i<10; i++)
        push(&mine, i);


    // Check if full works.
    if (full(&mine))
        printf("Stack is full\n");


    // Pop everything back off.
    for (i = 0; i<10; i++)
        printf("popping %d\n",
                pop(&mine));

    return 0;
}
```

Stack -- Linked List Implementation

```
typedef struct stack {
  int data;
  struct stack *next;
} stack;

int push(stack **front, int num);
stack* pop(stack **front);
int empty(stack *front);
int top(stack *front);
void init(stack **front);

void init(stack **front) {
  *front = NULL;
}

int push(stack **front, int num) {
  stack *temp;
  temp=(stack *)malloc(sizeof(stack));

  if (temp != NULL) {
    temp->data = num;
    temp->next = *front;
    *front = temp;
    return 1;
  }
  else
    return 0;
}

stack* pop(stack **front) {
  stack *temp;
  temp = NULL;

  if (*front != NULL) {
    temp = (*front);
    *front = (*front)->next;
    temp -> next = NULL;
  }
  return temp;
}

int empty(stack *front) {
  if (front == NULL)
    return 1;
  else
    return 0;
}

int top(stack *front) {
  if (front != NULL) {
    return front->data;
  }
  else
    return -1;
}
```

```
int main() {

  stack *stack1, *temp;
  int tempval;

  // create stack
  init(&stack1);

  // push some items
  if (!push(&stack1, 3))
    printf("Push failed.\n");
  if (!push(&stack1, 5))
    printf("Push failed.\n");

  // pop an item
  temp = pop(&stack1);
  if (temp !=NULL)
    printf("Pop stack = %d\n",
           temp->data);

  // check empty
  if (empty(stack1))
    printf("Empty stack\n");
  else
    printf("Contains elements.\n");

  // check top
  tempval = top(stack1);
  if (tempval != -1)
    printf("Top of Stack = %d\n",
           tempval);

  // pop  some items
  temp = pop(&stack1);
  temp = pop(&stack1);
  if (temp != NULL)
    printf("Top of Stack = %d\n",
           temp->data);
  else
    printf("Popping empty stack.\n");

  return 0;
}
```