Stack ADT

* Is a list where inserts/deletes are done at one end
* LIFO: last in, first out (remove the most recent item inserted)

Example: Cafeteria trays, balancing braces, undo sequence of word processor, saving local variables with function call

Basic Operation:

* Push: add element to the top
* Pop: remove element from the top
* Top: retrieve top element

Example:

push(5);

push(6);

push(7);

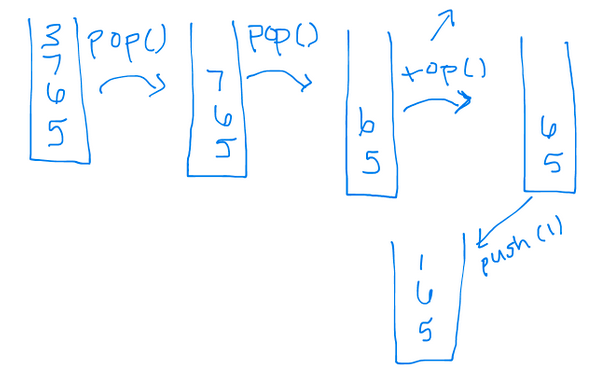
push(3);

pop();

pop();

cout<<top();

push(1);



How can we implement a stack?

We can use an array or a linked list

Array implementation

const int STACK\_CAPACITY=128;

typedef int dataType;

class Stack{

private:

//think about these

dataType a[STACK\_CAPACITY];

int myTop;

public:

Stack();

bool empty() const;

void push(const dataType &d);

void pop();

dataType top() const;

};

Stack::Stack(){

myTop=-1;

}

bool Stack:: empty() const{

return (myTop==-1);

}

void Stack::push(const dataType &d){

if (myTop<STACK\_CAPACITY-1){

myTop++;

a[myTop]=d;

}

else{

cerr<<"Stack full"; //displays message like cout but takes over, causes output to appear immediately--> unbuffered

}

}

void Stack::pop(){

if (!empty()){

myTop--;

}

else{

cerr<<"Empty\n";

}

}

dataType Stack:: top() const{

if (!empty())

return a[myTop];

else{

dataType garbage;

cerr<<"Empty";

return garbage;

}

}

Linked List Implementation

typedef int dataType;

class Node{

public:

dataType data;

Node \* next;

};

class Stack{

public:

//methods same

private:

Node \* myTop;

};

Stack::Stack(){

myTop=NULL;

}

bool Stack::empty() const{

return (myTop==NULL);

}

void Stack::push(const dataType & d){

Node \* newdata= new Node;

newdata->data=d;

newdata->next=myTop;

myTop=newdata;

}

void Stack::pop(){

if (!empty()){

Node \* temp=myTop;

myTop=myTop->next;

delete temp;

}

else{

cerr<<"Empty";

}

}

dataType Stack::top(){

if (!empty()){

return myTop->data;

}

else{

dataType garbage;

cerr<"Empty";

return garbage;

}

}

BIG OH TIMES

Applications of Stack

1. Balance of symbols in program

* [ ( ) ] is legal
* [ ( ] ) is illegal
* Algorithm: Make an empty stack. Read all characters until end of file
  + if character is an opening symbol, push it onto stack
  + if the character is a closing symbol
    - 1. If stack is empty🡪 ERROR
      2. Else pop element from stack. If the symbol popped is not the corresponding opening🡪 ERROR
* When at the end of the file, the stack should be empty (otherwise, report🡪 ERROR)

1. Evaluating Postfix Notation

operand, operand, operator

(((3+2)\*3)+(6\*4))

(3,2)+3\*6 4 \* +

3 2 + 3 \* 6 4 \* +

Alg :

1. Initialize empty stack
2. Repeat until end of expression
   * + - 1. set the token (literal/operator/variable)
         2. if it’s an operand, push it on stack, if it’s an operator

pop 2 values from the stack

apply the operator to these two values

push result on stack

3. When end of expression is encountered the result is on the stack (should be only value)