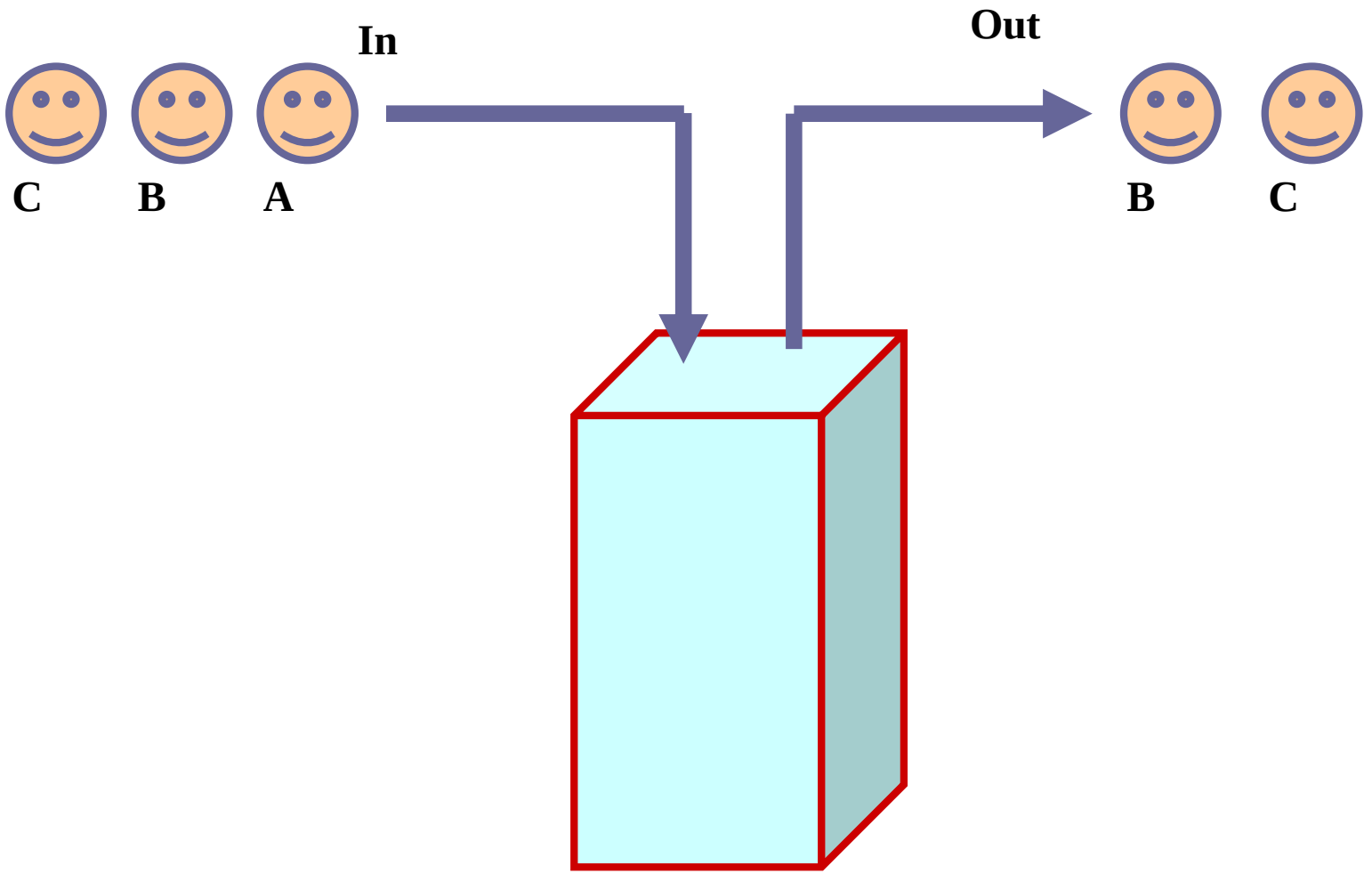
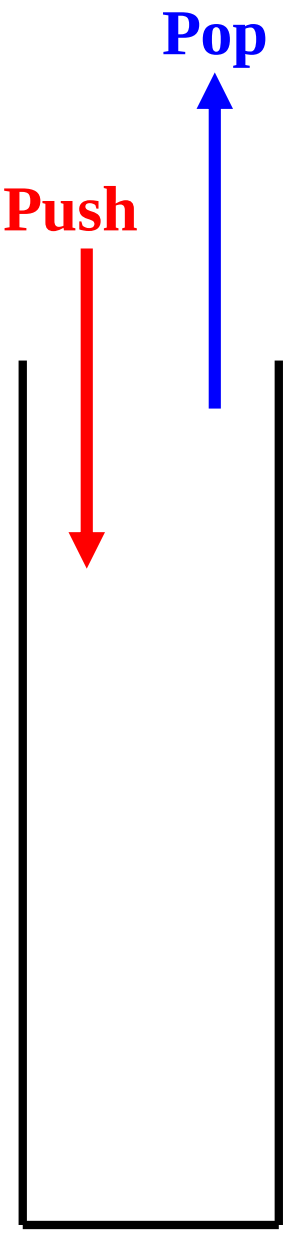
# Stack and Queue

# Stack

Data structure with Last-In First-Out (LIFO) behavior



# Typical Operations on Stack




isempty: determines if the stack has no elements

isfull: determines if the stack is full in case of a bounded sized stack

top: returns the top element in the stack

push: inserts an element into the stack

pop: removes the top element from the stack

push is like inserting at the front of the list

pop is like deleting from the front of the list

# Creating and Initializing a Stack

**Declaration**

```
#define MAX_STACK_SIZE 100
typedef struct {
    int key; /* just an example, can have
                any type of fields depending
                on what is to be stored */
}  element;
typedef struct {
    element list[MAX_STACK_SIZE];
    int top; /* index of the topmost element */
 } stack;
```

**Create and Initialize**

```
stack Z;
Z.top = -1;
```

# Operations

```
int isfull (stack *s)
{
    if (s->top >=
            MAX_STACK_SIZE – 1)
        return 1;
    return 0;
}
```

```
int isempty (stack *s)
{
    if (s->top == -1)
        return 1;
    return 0;
}
```

# Operations

```
element top( stack *s )
{
    return s->list[s->top];
}
```

```
void pop( stack *s )
{
    (s->top)--;
}
```

```
void push( stack *s, element e )
{
    (s->top)++;
    s->list[s->top] = e;
 }
```

# Application: Parenthesis Matching

- Given a parenthesized expression, test whether the expression is properly parenthesized
  - Examples:

    ( )( { } [ ( { } { } ( ) ) ] )  is proper

    ( ){ [ ]  is not proper

    ( { ) }  is not proper

    )( [ ]  is not proper

    ( [ ] ) )  is not proper

- Approach:
  - Whenever a left parenthesis is encountered, it is pushed in the stack
  - Whenever a right parenthesis is encountered, pop from stack and check if the parentheses match
  - Works for multiple types of parentheses ( ), { }, [ ]

# Parenthesis matching

```
while (not end of string) do
{
    a = get_next_token();
    if (a is ‘(‘ or ‘{‘ or ‘[‘) push (a);
    if (a is ‘)’ or ‘}’ or ‘]’)
    {
        if (is_stack_empty( ))
          { print (“Not well formed”); exit(); }
        x = top();
        pop();
        if (a and x do not match)
          { print (“Not well formed”); exit(); }
    }
}
if (not is_stack_empty( )) print (“Not well formed”);
```

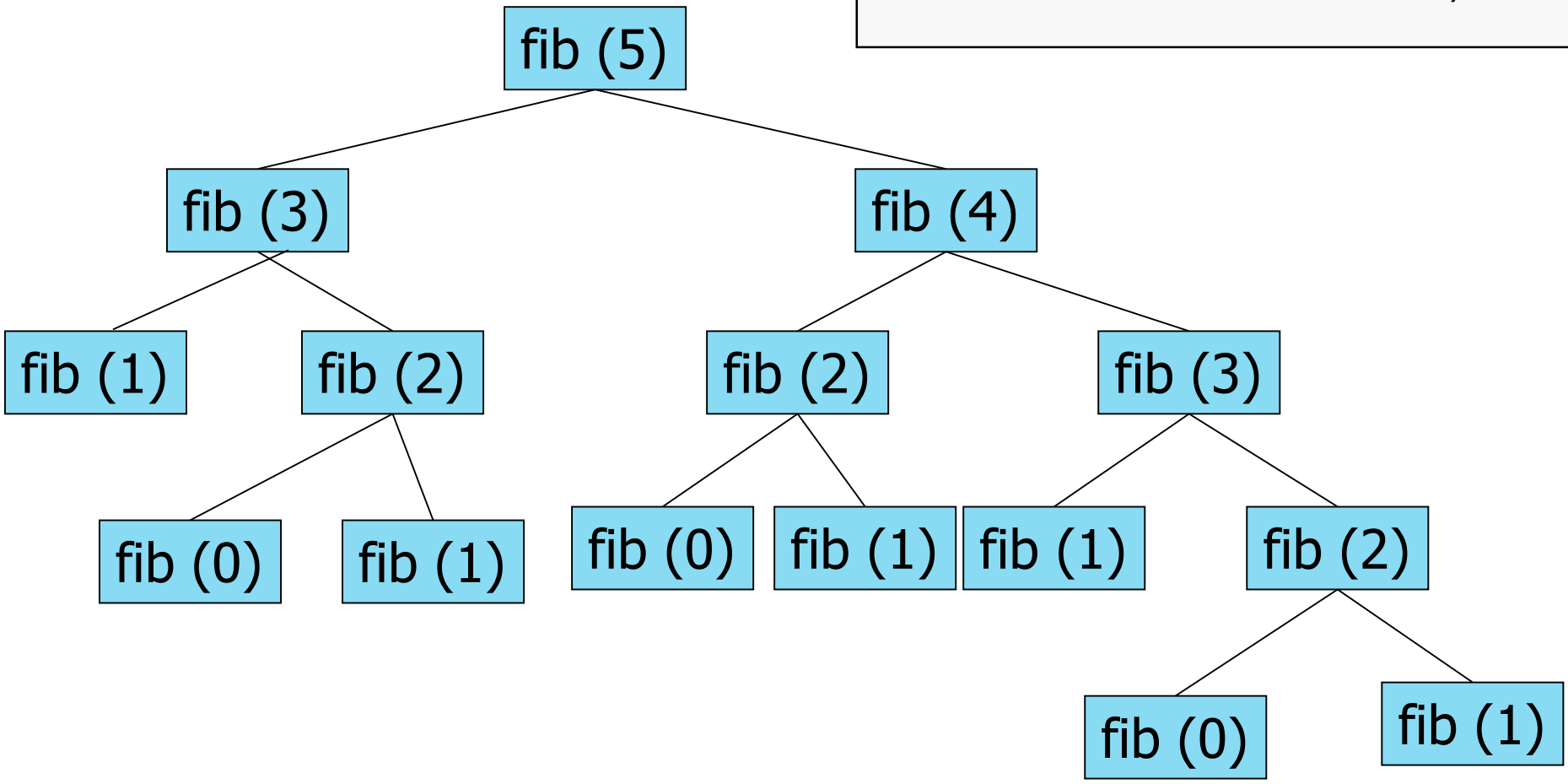
# Recursion can be implemented as a stack

**Fibonacci recurrence:**

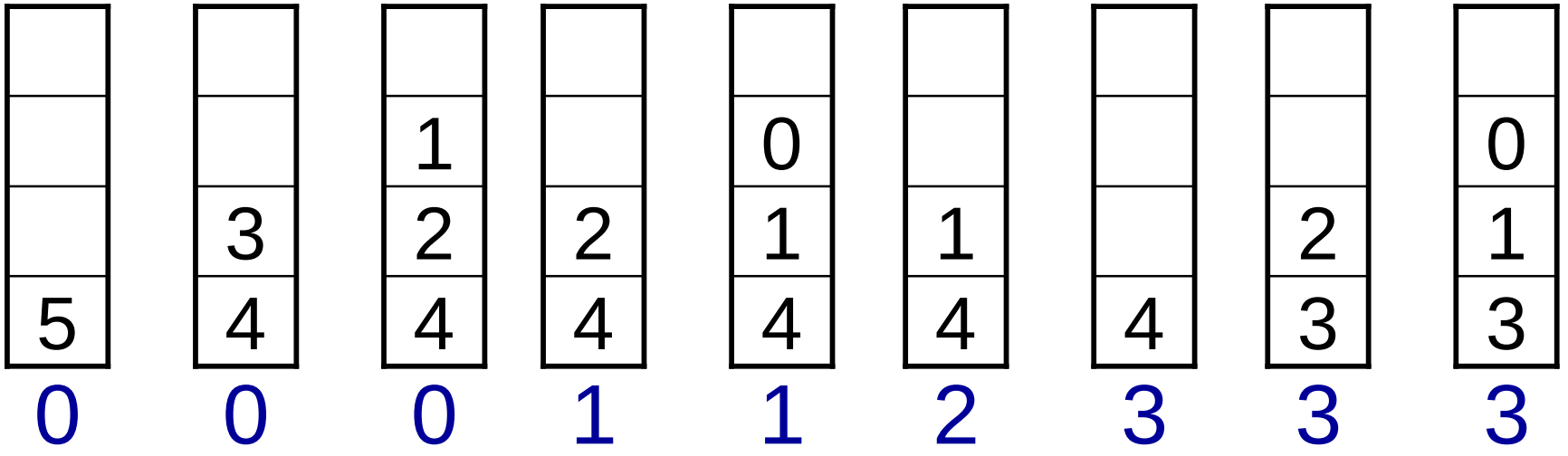
fib(n) = 1 if n =0 or 1;

= fib(n – 2) + fib(n – 1)

otherwise;



# Fibonacci Recursion Stack




Click to add text