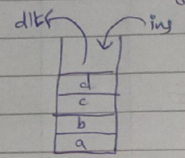
03 = STACK

* linear ds.
* LIFO. eg → collection of plates.
* ins & dlt at top

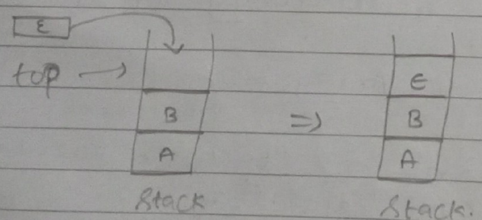


* Basic operation →
1) PUSH → ins
2) pop → dlt.

* othr () →
- peek() → Get the top data element of stack without removing it.
- isFull() → Check if stack is full. (ins)
- isempty() → Check if the stack is empty. (dlt)

* push operation =

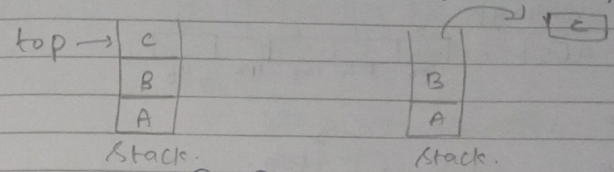
Process of putting a new element onto stack.



* when you are ins~ an element | push
→ increment [illegible].

* pop operation => pop → decrement [illegible].

* top-ൽ [illegible] element [illegible] [illegible] [illegible] pop [illegible] ~ [illegible], cannot choose at random.



* Prgm, (pop)

```
int pop (int data) {
    if (! isempty()) {
        data = stack[top];   → stack-ന്റെ top-ൽ ഉള്ള element dlt.
        top = top-1;
        return data;
    } else {
        printf("stack is empty");
}.
```

* prgm for push =

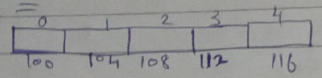
```
void push(int data) {
    if (! isfull()) {
        top = top+1;
        stack[top] = data;   → data ins [illegible].
    } else {
        printf("stack full");
}.
```

eg → [illegible] element 'c' → ins [illegible] => top = 4 (at print)
so top = top+1 => 4+1 = 5.
∴ stack[top] = data
=> stack[5] = 'c';

⇒ Stack implementation =

① using arrays =

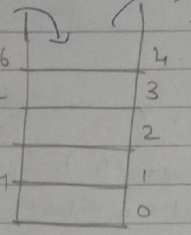
int a[5] ·



int → size = 2.

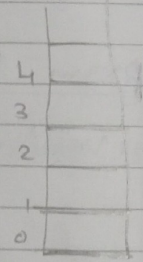
Stack

int stack[5]



* Prgm

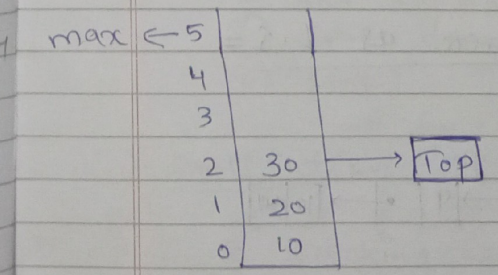
```
#define N 5;
int stack[N];
int top = -1;        → empty stack.
void main() {
    push();
    pop();
    peek();
    display();
}
void push() {
    int x;
    printf("Entr data");
    scanf("%d", &x);
    if (top == N-1) {        N=5, here top=4
        print("overflow");   is N-1.
    }else {
        top++;  (push operation)
        stack[top] = x;
    }
}
```



full

```
void pop() {
    int item;
    if (top == -1) {        (stack empty)
        printf("underflow");
    }else {
        item = stack[top]
        top--;
        print("%d", item);
    }
}
void peek() {        (top-ல உள்ள element-ஐ display செய்வது)
    if (top == -1) {
        printf("underflow");
    }else {
        printf("%d", stack[top]);
    }
}
void display() {        (stack-ல் உள்ள elements-ஐ
    for (int i=top; i>=0; i--) {        print செய்வது)
        printf("%d", stack[i]);
    }
}
```

⇒ Representation of stack as arrays =



here max = 5

top[2] = 30

There is room for 3 more items in the stack.

* Stack can be maintained in a prgm by using a array of elemnts & pointer Variables Top & MAX.
* Top pointer contains the location of the top elemnts of the stack & max gives the max no. of elemnts in stack.
* Stack is empty if Top = -1 or Top = NULL.

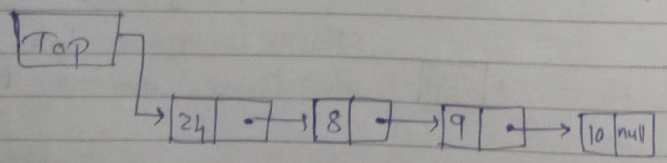
→ (A) for push

PUSH (STACK, TOP, MAX, ITEM)
1. If TOP = MAX then print : OVERFLOW. & Return
2. Set TOP = TOP + 1
3. Set STACK[TOP] = ITEM
4. return / Stop

→ (A) for pop

POP (STACK, TOP, ITEM)
1. If TOP = 0, then print UNDERFLOW & Return
2. Set ITEM = STACK[TOP]
3. Set TOP = TOP - 1
4. return / stop.

⇒ Representation of stack as L.S =



* disadv of using an array to represent a stack is that the array must be specified to have a fixed size.
* If the size of the array can't be determined in advance, we can use L.S representation of stack.
* HEAD pointer of L.S is used as the Top of stack.

→ push operation as L.S =
PUSH (Top, Item)
1) start.
2) create new node
3) Data(newnode) = item
4) if top = null, set top = newnode & set next(top) = null else stp 5.
5) next(new node) = top
6) Set top = newnode
7) stop.

→ pop operation as L.S =
1) start
2) if top = null goto stp 3 else stp 4
3) print stack underflow
4) Set temp = top → element
5) set top = top → next
6) free (temp)
7) stop.

→ Application of stack =

* Back tracking ⇒ is an algorithmic technique

for solving probs recursively by trying to build a soln incrementally.

* polish notation
* Recursion
* Evaluating exp.

⇒ polish notation = a+b, a+b, a-b, a/b } exp.

* Algebraic exp can be written using 3 separate bt equivalent notations namely infix, postfix & prefix notations.

a) Infix notation =
operator symbol is placed b/2 its 2 operands in most arithmetic (o).
eg → (A+B).

b) Prefix notation = (polish notation)
notation in which operator is placed bfre its 2 operands.
eg → A+B ⇒ +AB.
exp → prefix

c) postfix notation = (reverse polish notation)
operator placed after the operands.
eg → A+B ⇒ AB+
exp → postfix

⇒ Recursion using Stack =

A () that calls itself → recursive () &
this technique → recursion.

eg →
```
int main() {
    fact(5);
}
void fact(n) {
    fact(n-1);
}
```