2. The Stack ADT

The Stack ADT

TOS = 0

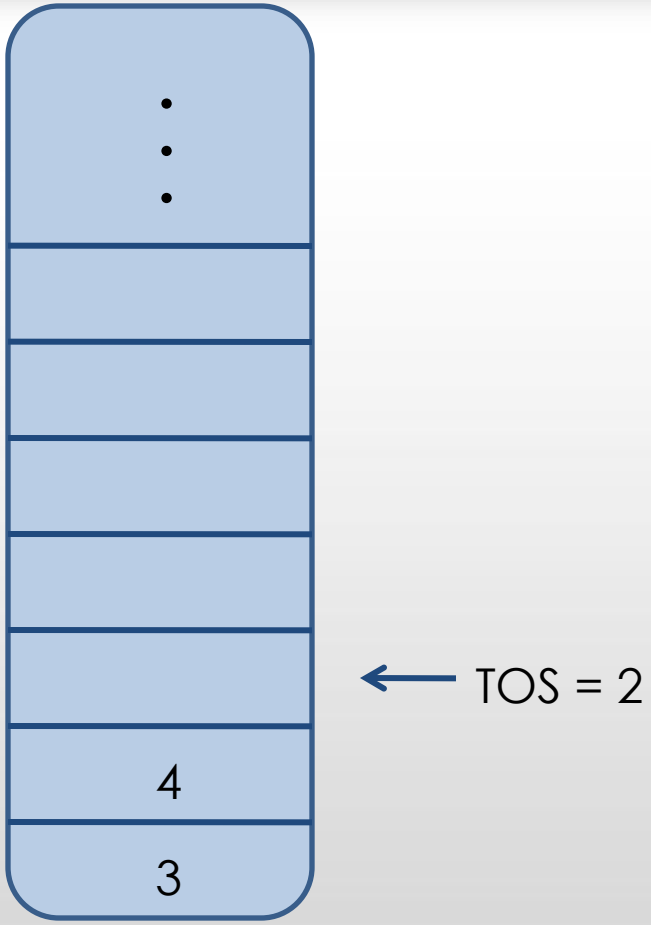
push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

The Stack ADT

TOS = 1

push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

The Stack ADT



push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

The Stack ADT

x = 4

TOS = 1

push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

The Stack ADT

x = 4

TOS = 2

push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

The Stack ADT

x = 24

TOS = 1

push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

The Stack ADT

x = 24

y = 3

TOS = 0

push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

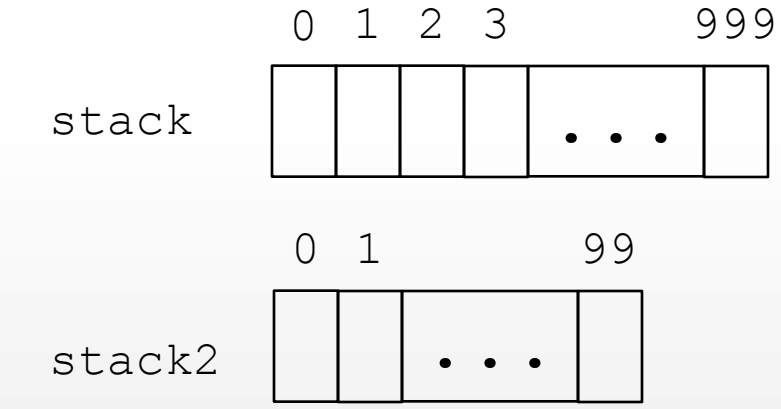
Array Implementation

```
#define LIMIT 1000
int stack[LIMIT];
int top=0;

void push(int x){
  if (top < LIMIT)
    stack[top++]=x;
  else {
    printf("stack overflow");
    exit(1);
  }
}
```

```
int pop(){
  if (top > 0 )
    return(stack[--top]);
  else {
    printf("stack underflow");
    exit(1);
  }
}
```

Array Implementation

```
#define LIMIT 1000
#define LIMIT2 100
int stack[LIMIT];
int top=0;
int stack2[LIMIT2];
int top2=0;

void push(int x){
  if (top < LIMIT)
    stack[top++]=x;
  else {
    printf("stack overflow");
    exit(1);
  }
}
```



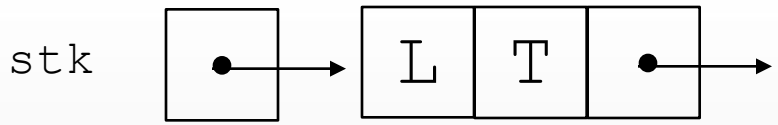

```
int pop(){
  if (top > 0 )
    return(stack[--top]);
  else {
    printf("stack underflow");
    exit(1);
  }
}
```

Array Implementation

```
typedef struct node{
  int limit;
  int top;
  int *array;
}stack;

stack *create(int size){
  stack *stk;

  stk = (stack *)malloc(sizeof(stack));
  stk->array = (int *)malloc(sizeof(int)*size);
  stk->top = 0;
  stk->limit = size;
  return stk;
}
```

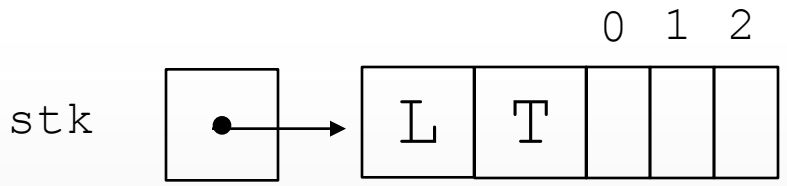
Array Implementation

```
typedef struct node{
  int limit;
  int top;
  int *array;
}stack;

stack *create(int size){
  stack *stk;

  stk = (stack *)malloc(sizeof(stack));
  stk->array = (int *)malloc(sizeof(int)*size);
  stk->top = 0;
  stk->limit = size;
  return stk;
}
```



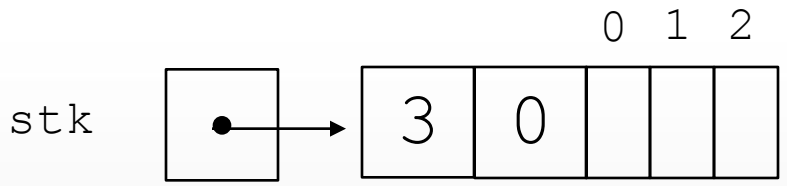
Array Implementation

```
typedef struct node{
  int limit;
  int top;
  int *array;
}stack;

stack *create(int size){
  stack *stk;

  stk = (stack *)malloc(sizeof(stack));
  stk->array = (int *)malloc(sizeof(int)*size);
  stk->top = 0;
  stk->limit = size;
  return stk;
}
```



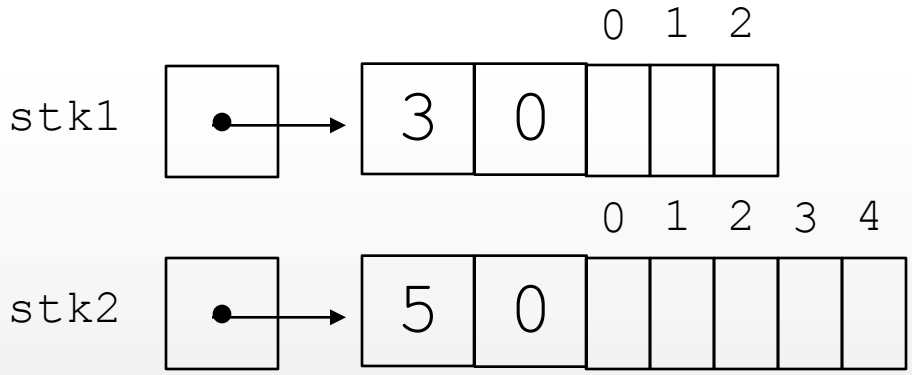
Array Implementation

```
typedef struct node{
  int limit;
  int top;
  int *array;
}stack;

stack *create(int size){
  stack *stk;

  stk = (stack *)malloc(sizeof(stack));
  stk->array = (int *)malloc(sizeof(int)*size);
  stk->top = 0;
  stk->limit = size;
  return stk;
}
```



Array Implementation

```
typedef struct node{
  int limit;
  int top;
  int *array;
}stack;

stack *create(int size){
  stack *stk;

  stk = (stack *)malloc(sizeof(stack));
  stk->array = (int *)malloc(sizeof(int)*size);
  stk->top = 0;
  stk->limit = size;
  return stk;
}
```



Array Implementation

```
void push(int x, stack *s){
  if (s->top < s->limit){
    s->array[s->top] = x;
    (s->top)++;
  }
  else {
    printf("stack overflow");
    exit(1);
  }
}
```

```
int pop(stack *s){
  if (____________){
    _____________
    ____________________________
  }
  else {
    printf("stack underflow");
    exit(1);
  }
}
```

Array Implementation

```
void push(int x, stack *s){
  if (s->top < s->limit){
    s->array[s->top] = x;
    (s->top)++;
  }
  else {
    printf("stack overflow");
    exit(1);
  }
}
```

```
int pop(stack *s){
  if (s->top > 0){
   (s->top)--;
   return(s->array[s->top]);
  }
  else {
    printf("stack underflow");
    exit(1);
  }
}
```

2. Stack ADT

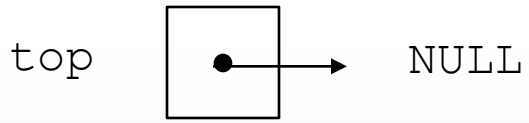
2.2 Linked-list Implementation

Linked list Implementation

- uses a singly linked list
- push - inserts at the front of the list
- pop - deletes the element at the front of the list and returns the value

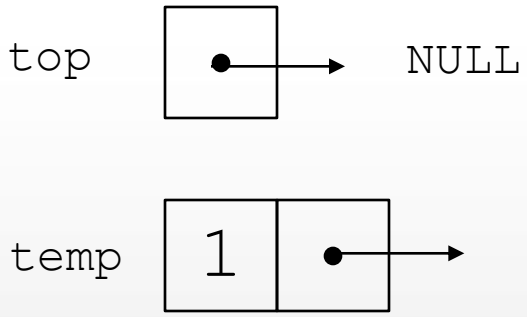
Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;

void push(int x, stack *top){
  stack *temp;

  temp=(stack *)malloc(sizeof(stack));
  if(temp==NULL){
    printf("stack overflow");
    exit(1);
  }
  temp->value=x;
  temp->next=top;
  top=temp;
}
```



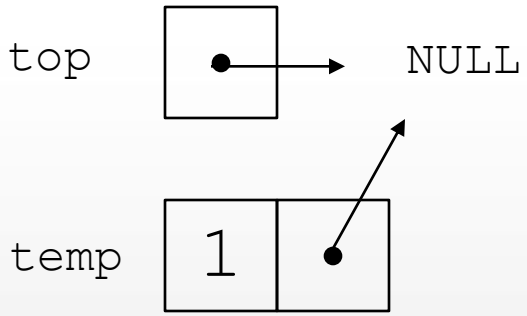
Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;

void push(int x, stack *top){
  stack *temp;

  temp=(stack *)malloc(sizeof(stack));
  if(temp==NULL){
    printf("stack overflow");
    exit(1);
  }
  temp->value=x;
  temp->next=top;
  top=temp;
}
```



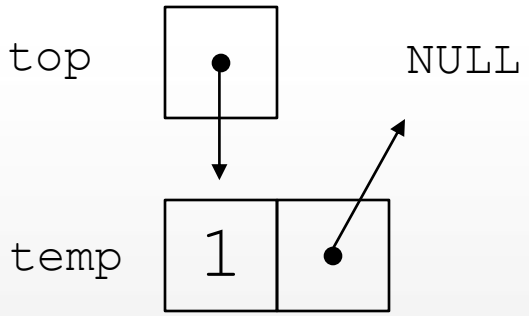
Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;

void push(int x, stack *top){
  stack *temp;

  temp=(stack *)malloc(sizeof(stack));
  if(temp==NULL){
    printf("stack overflow");
    exit(1);
  }
  temp->value=x;
  temp->next=top;
  top=temp;
}
```



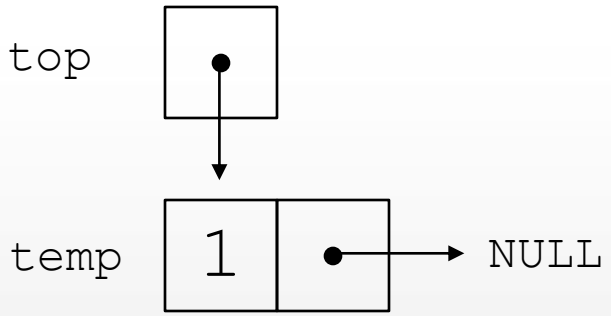
Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;

void push(int x, stack *top){
  stack *temp;

  temp=(stack *)malloc(sizeof(stack));
  if(temp==NULL){
    printf("stack overflow");
    exit(1);
  }
  temp->value=x;
  temp->next=top;
  top=temp;
}
```



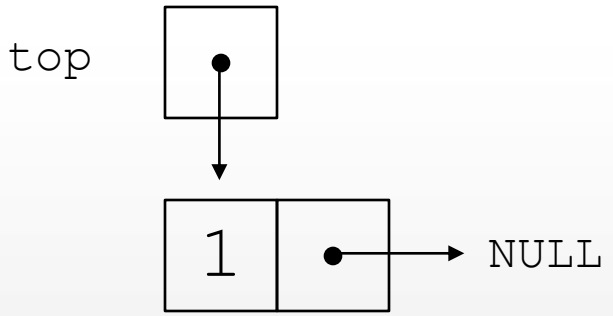
Linked-list Implementation

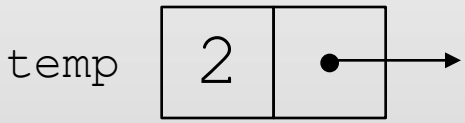
```
typedef struct node{
  int value;
  struct node *next;
}stack;

void push(int x, stack *top){
  stack *temp;

  temp=(stack *)malloc(sizeof(stack));
  if(temp==NULL){
    printf("stack overflow");
    exit(1);
  }
  temp->value=x;
  temp->next=top;
  top=temp;
}
```



Linked-list Implementation

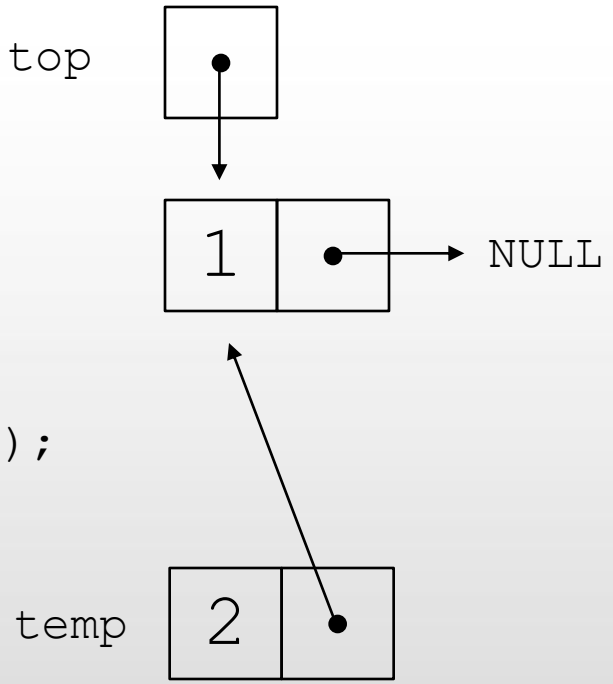
```
typedef struct node{
  int value;
  struct node *next;
}stack;

void push(int x, stack *top){
  stack *temp;

  temp=(stack *)malloc(sizeof(stack));
  if(temp==NULL){
    printf("stack overflow");
    exit(1);
  }
  temp->value=x;
  temp->next=top;
  top=temp;
}
```





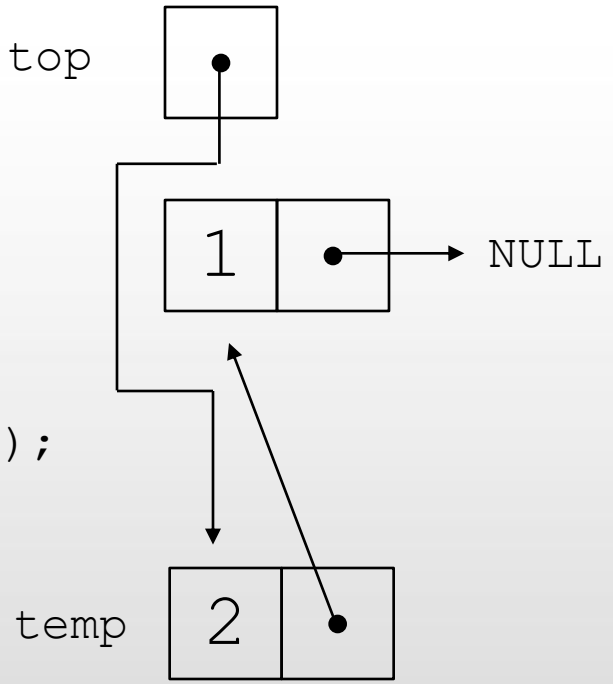
Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;

void push(int x, stack *top){
  stack *temp;

  temp=(stack *)malloc(sizeof(stack));
  if(temp==NULL){
    printf("stack overflow");
    exit(1);
  }
  temp->value=x;
  temp->next=top;
  top=temp;
}
```



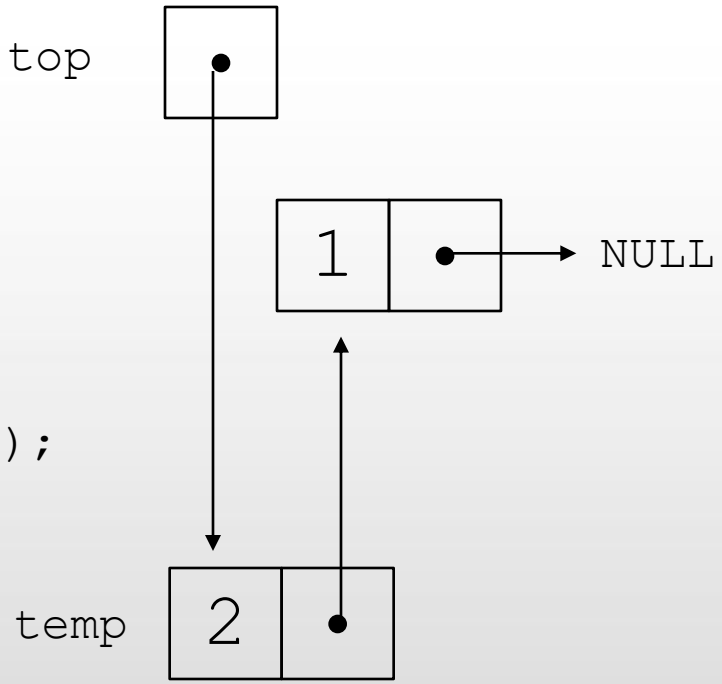
Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;

void push(int x, stack *top){
  stack *temp;

  temp=(stack *)malloc(sizeof(stack));
  if(temp==NULL){
    printf("stack overflow");
    exit(1);
  }
  temp->value=x;
  temp->next=top;
  top=temp;
}
```



Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;

void push(int x, stack *top){
  stack *temp;

  temp=(stack *)malloc(sizeof(stack));
  if(temp==NULL){
    printf("stack overflow");
    exit(1);
  }
  temp->value=x;
  temp->next=top;
  top=temp;
}
```



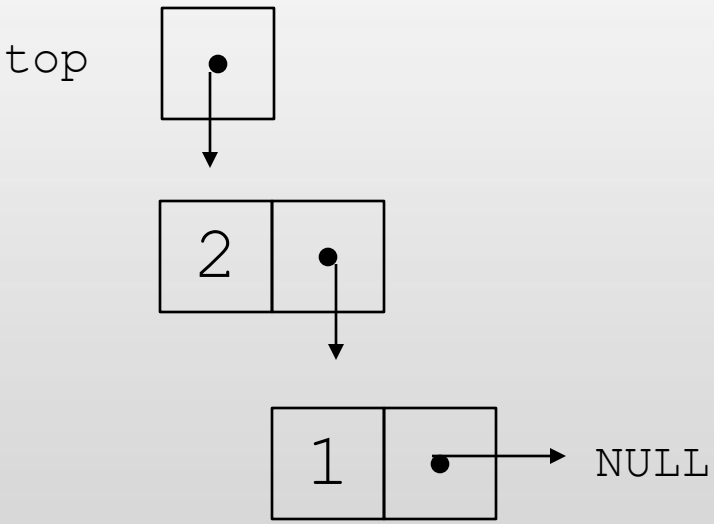
Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;
```

```
int pop(stack *top){
  stack *temp; int x;

  temp=top;
  if(temp==NULL){
    printf("stack underflow");
    exit(1);
  }
  top=top->next;
  x=temp->value;
  free(temp);
  return(x);
}
```

Linked-list Implementation

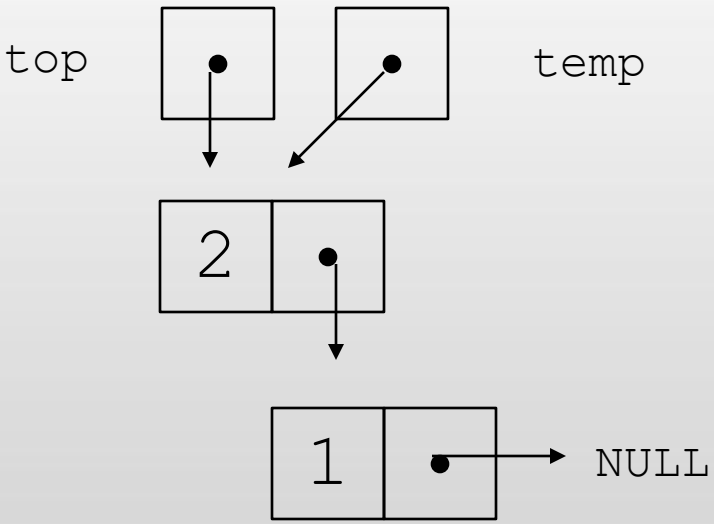
```
typedef struct node{
  int value;
  struct node *next;
}stack;
```




```
int pop(stack *top){
  stack *temp; int x;

  temp=top;
  if(temp==NULL){
    printf("stack underflow");
    exit(1);
  }
  top=top->next;
  x=temp->value;
  free(temp);
  return(x);
}
```

Linked-list Implementation

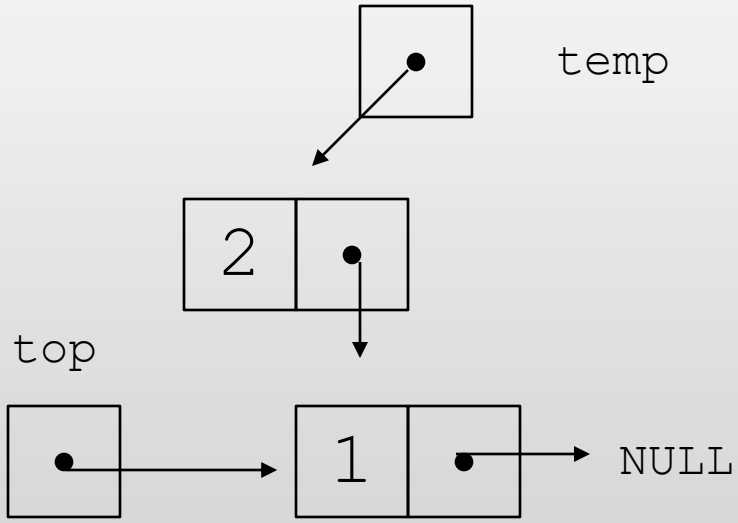
```
typedef struct node{
  int value;
  struct node *next;
}stack;
```




```
int pop(stack *top){
  stack *temp; int x;

  temp=top;
  if(temp==NULL){
    printf("stack underflow");
    exit(1);
  }
  top=top->next;
  x=temp->value;
  free(temp);
  return(x);
}
```

Linked-list Implementation

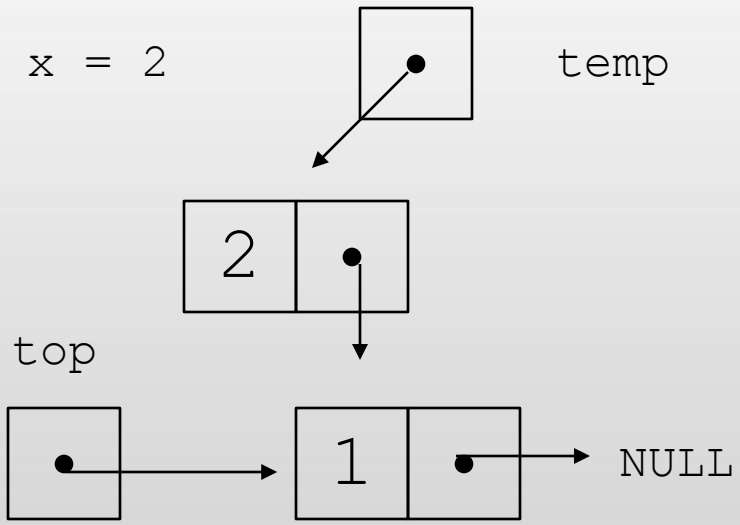
```
typedef struct node{
  int value;
  struct node *next;
}stack;
```




```
int pop(stack *top){
  stack *temp; int x;

  temp=top;
  if(temp==NULL){
    printf("stack underflow");
    exit(1);
  }
  top=top->next;
  x=temp->value;
  free(temp);
  return(x);
}
```

Linked-list Implementation

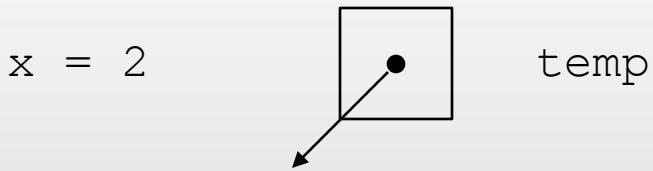
```
typedef struct node{
  int value;
  struct node *next;
}stack;
```

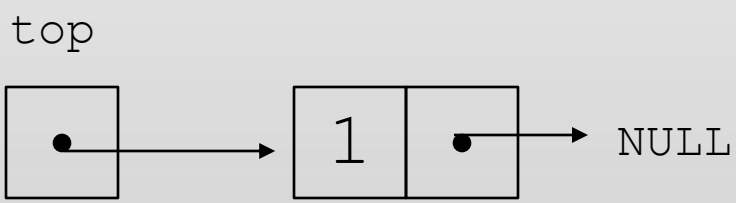



```
int pop(stack *top){
  stack *temp; int x;

  temp=top;
  if(temp==NULL){
    printf("stack underflow");
    exit(1);
  }
  top=top->next;
  x=temp->value;
  free(temp);
  return(x);
}
```

Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;
```

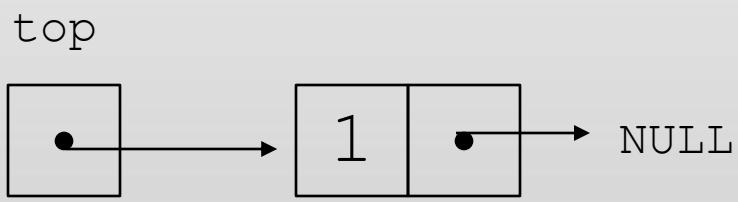





```
int pop(stack *top){
  stack *temp; int x;

  temp=top;
  if(temp==NULL){
    printf("stack underflow");
    exit(1);
  }
  top=top->next;
  x=temp->value;
  free(temp);
  return(x);
}
```

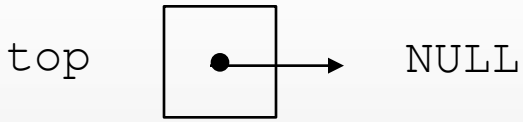
Linked-list Implementation

```
typedef struct node{
  int value;
  struct node *next;
}stack;
```



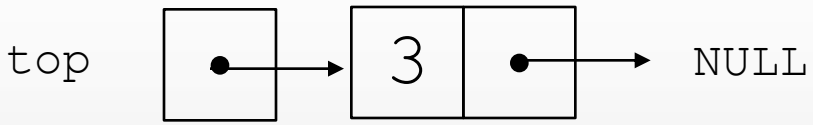

```
int pop(stack *top){
  stack *temp; int x;

  temp=top;
  if(temp==NULL){
    printf("stack underflow");
    exit(1);
  }
  top=top->next;
  x=temp->value;
  free(temp);
  return(x);
}
```

Linked-list Implementation



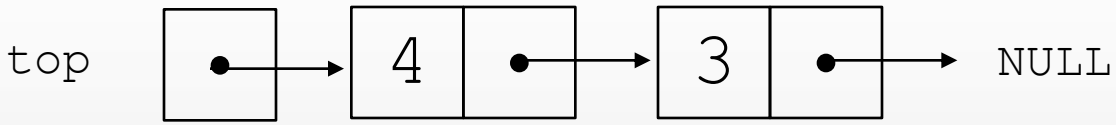
push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

Linked-list Implementation



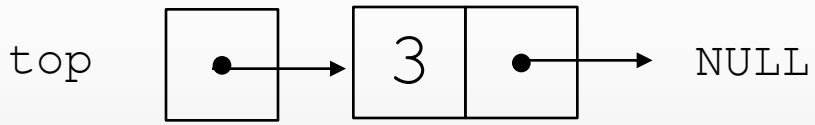
push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

Linked-list Implementation



push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

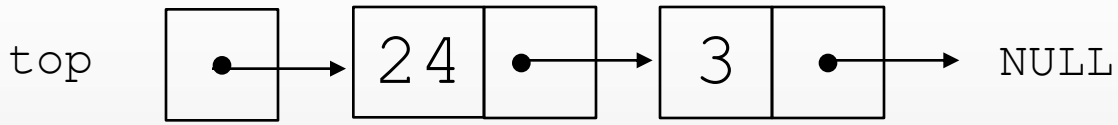
Linked-list Implementation



x = 4

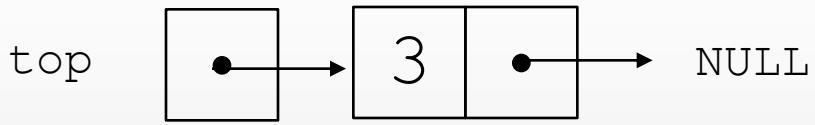
push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

Linked-list Implementation



push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

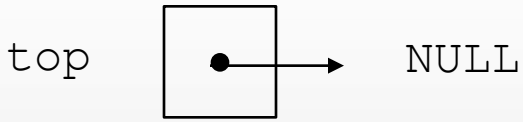
Linked-list Implementation



x = 24

push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

Linked-list Implementation



x = 24 , y = 3

push(3); push(4); x=pop(); push(24); x=pop(); y=pop();

Applications

- Balancing Symbols
  - Create an empty stack
  - Read characters until end of file
  - If character is an open symbol, push it onto the stack
  - If it is a close symbol and stack is empty, report an error otherwise, pop the stack
  - If symbol popped does not correspond to the opening symbol, report an error
  - At end of file, if stack is not empty, report an error

Example

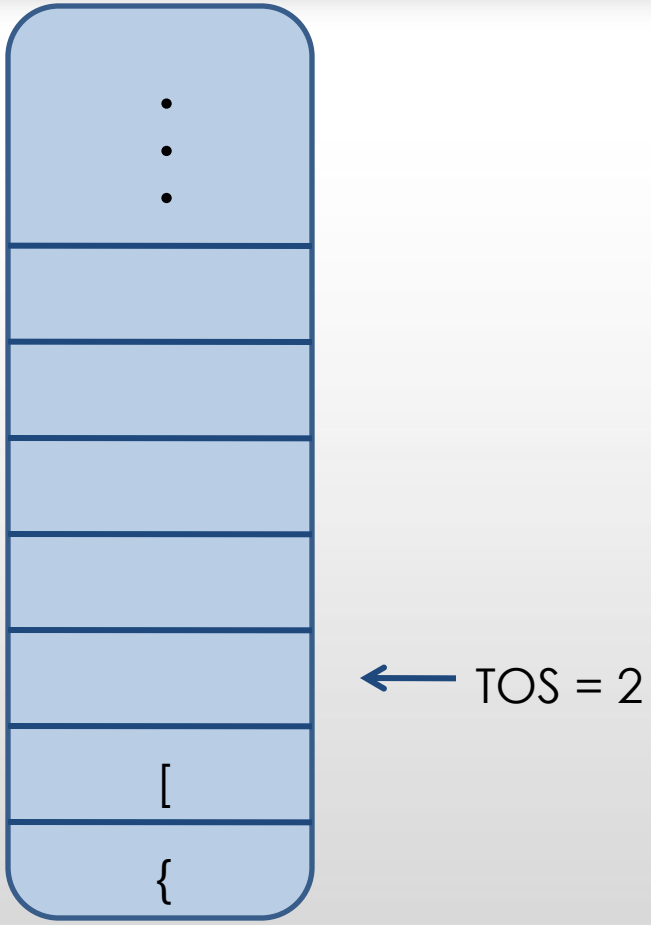
TOS = 0

Expression: **{ [(4 + 6) + 3] * 5 * 2**

Example

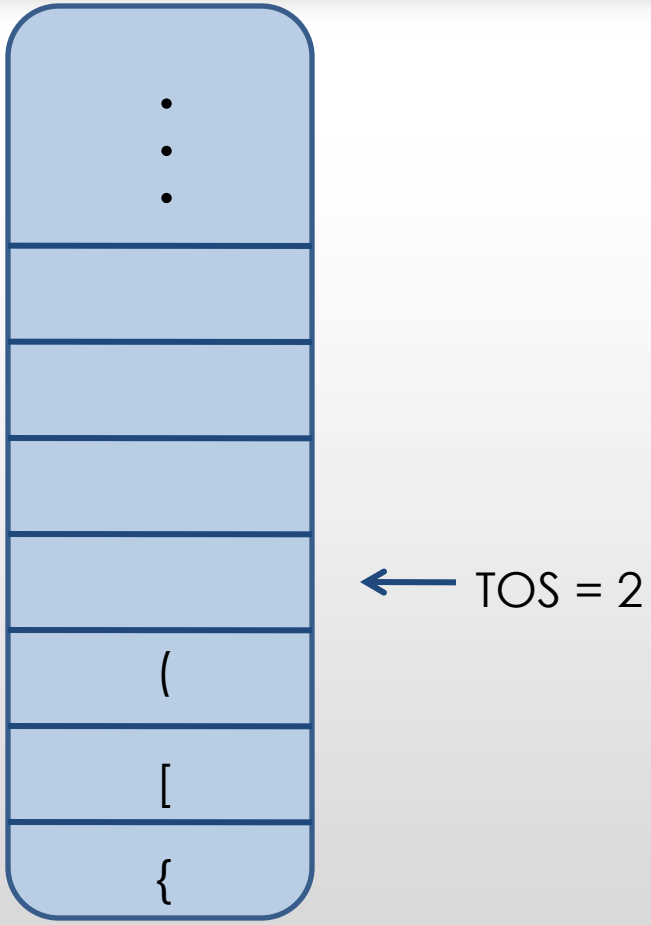
Expression: **{ [(4 + 6) + 3] * 5 * 2**

Example



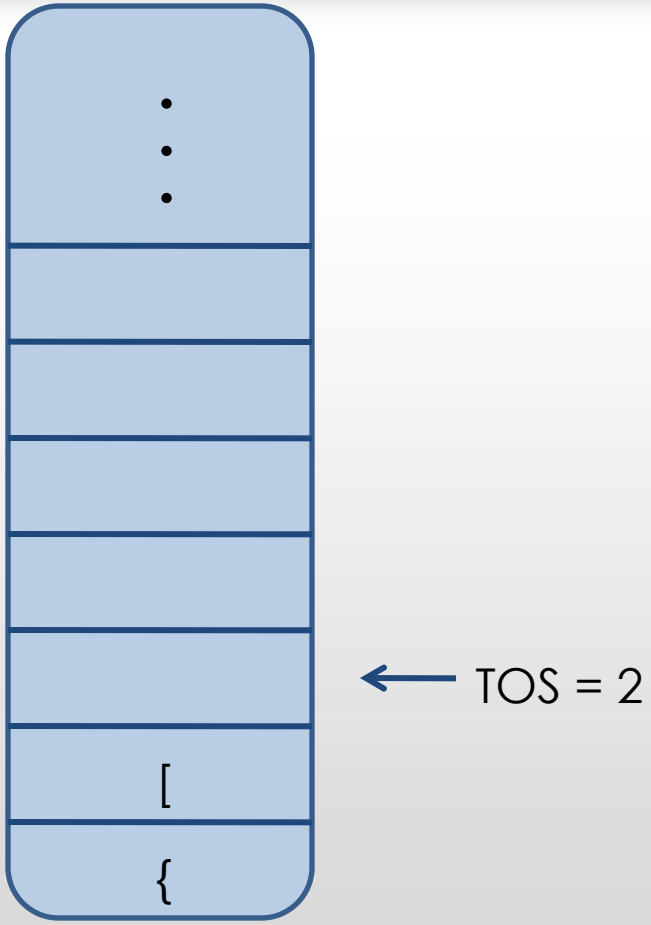
Expression: **{ [(4 + 6) + 3] * 5 * 2**

Example



Expression: **{ [(4 + 6) + 3] * 5 * 2**

Example



Expression: **{ [(4 + 6) + 3] * 5 * 2**

Example

Expression: **{ [(4 + 6) + 3] * 5 * 2**

Applications

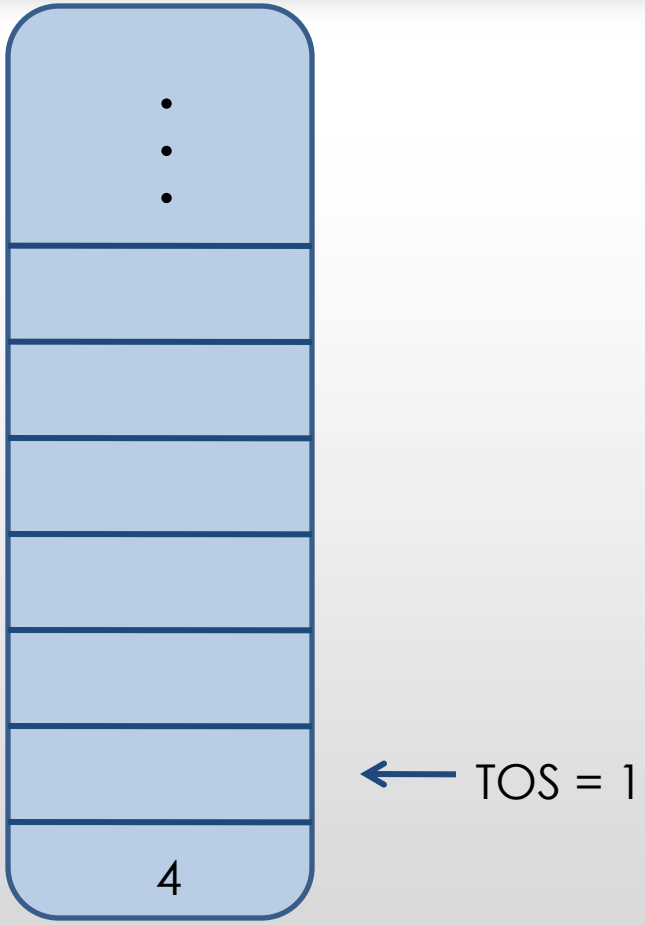
- Postfix Expressions
  - When a number is encountered in a given expression, it is pushed onto the stack
  - When an operator is seen, the operator is applied to the two numbers that are popped from the stack
  - The result is pushed onto the stack

Example

TOS = 0

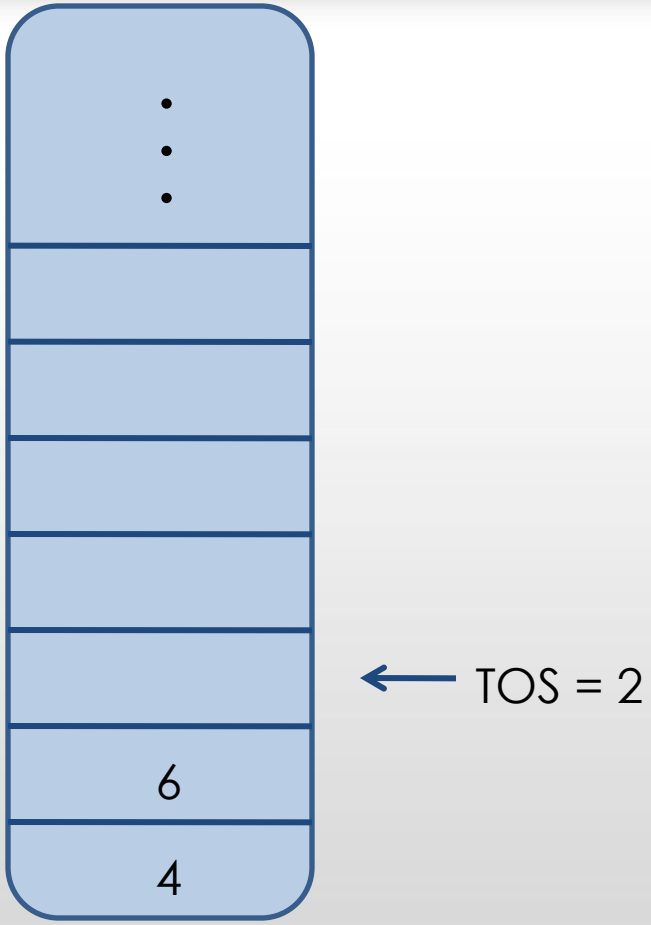
Expression: **4 6 + 3 5 + * 2 ***

Example



Expression: **4 6 + 3 5 + * 2 ***

Example



Expression: **4 6 + 3 5 + * 2 ***

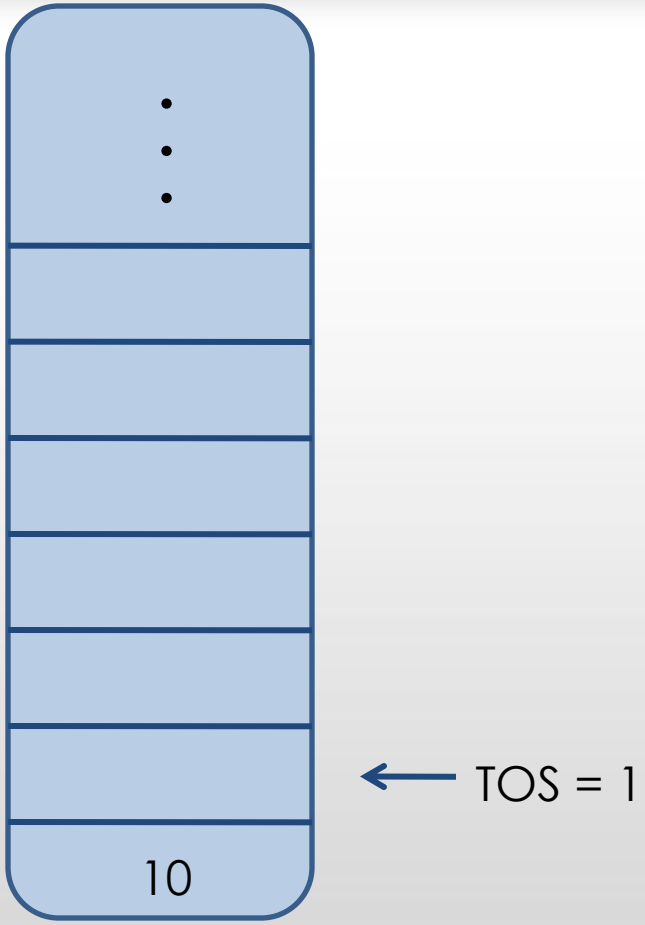
Example

Evaluate: 4 + 6

← TOS = 2

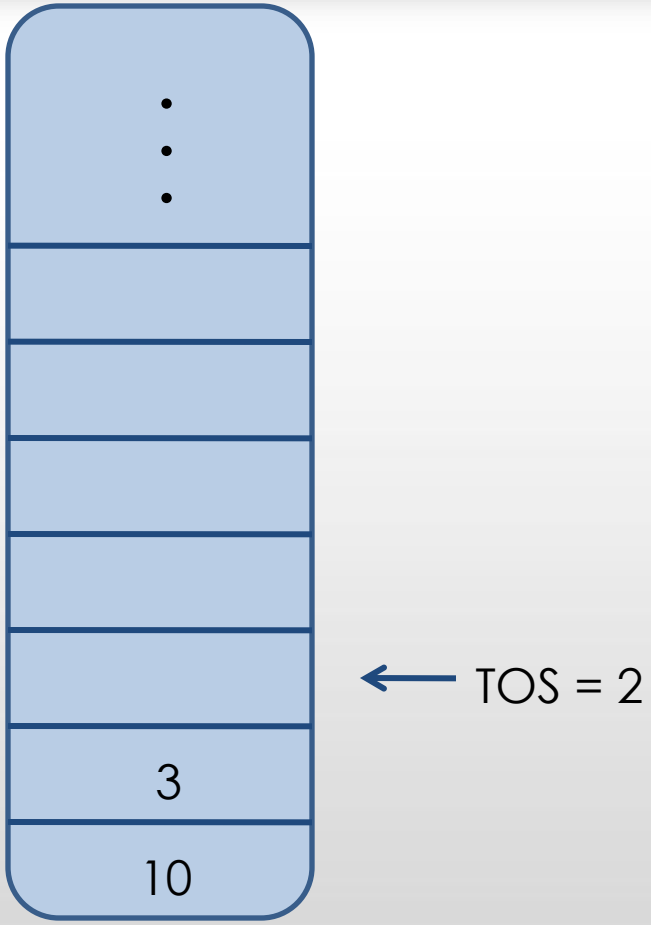
Expression: **4 6 + 3 5 + * 2 ***

Example



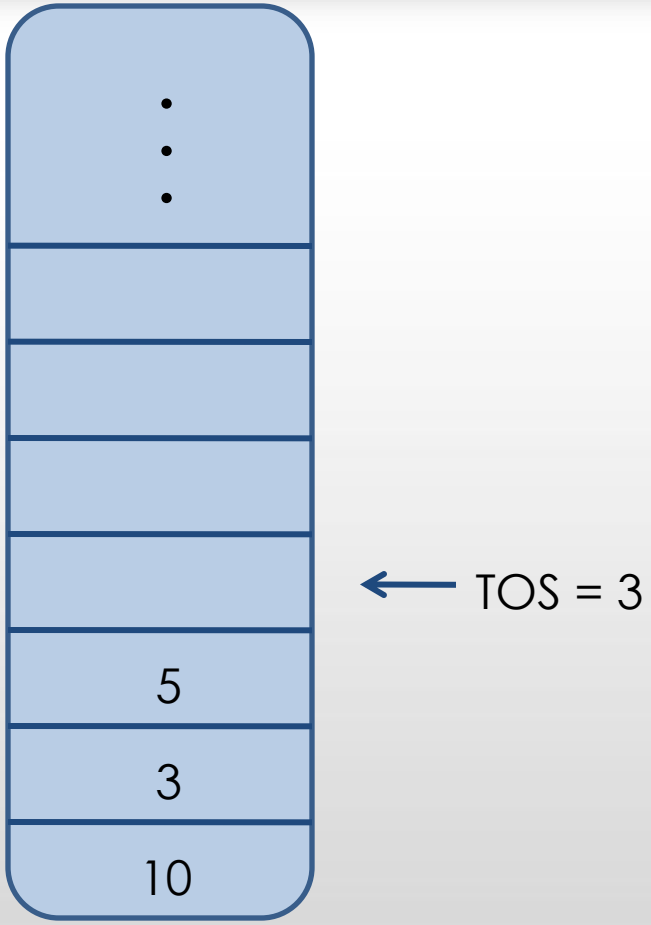
Expression: **4 6 + 3 5 + * 2 ***

Example



Expression: **4 6 + 3 5 + * 2 ***

Example



Expression: **4 6 + 3 5 + * 2 ***

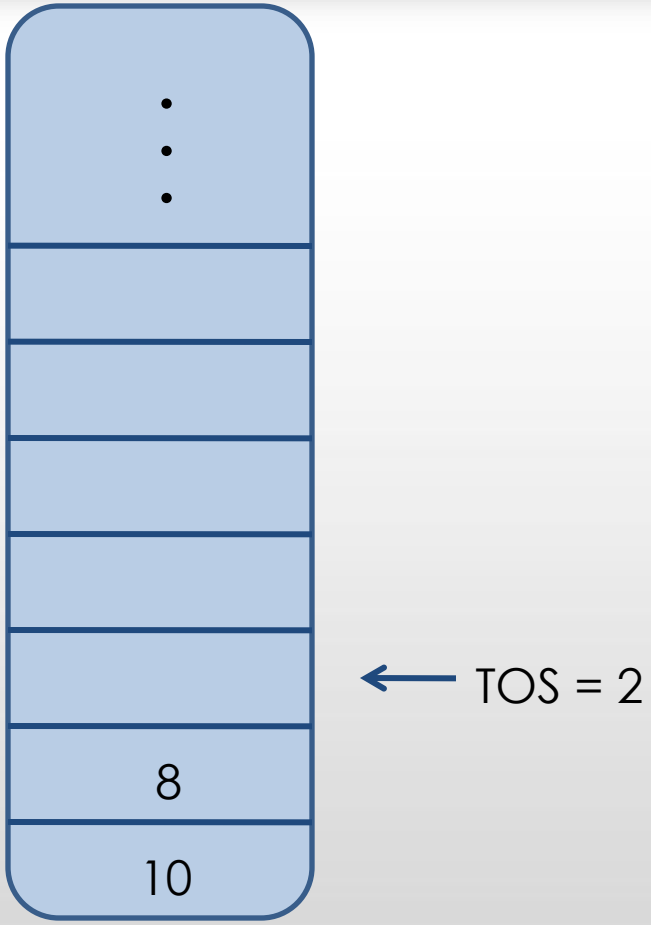
Example

Evaluate: 5 + 3

← TOS = 3

Expression: **4 6 + 3 5 + * 2 ***

Example



Expression: **4 6 + 3 5 + * 2 ***

Example

Evaluate: 10 * 8

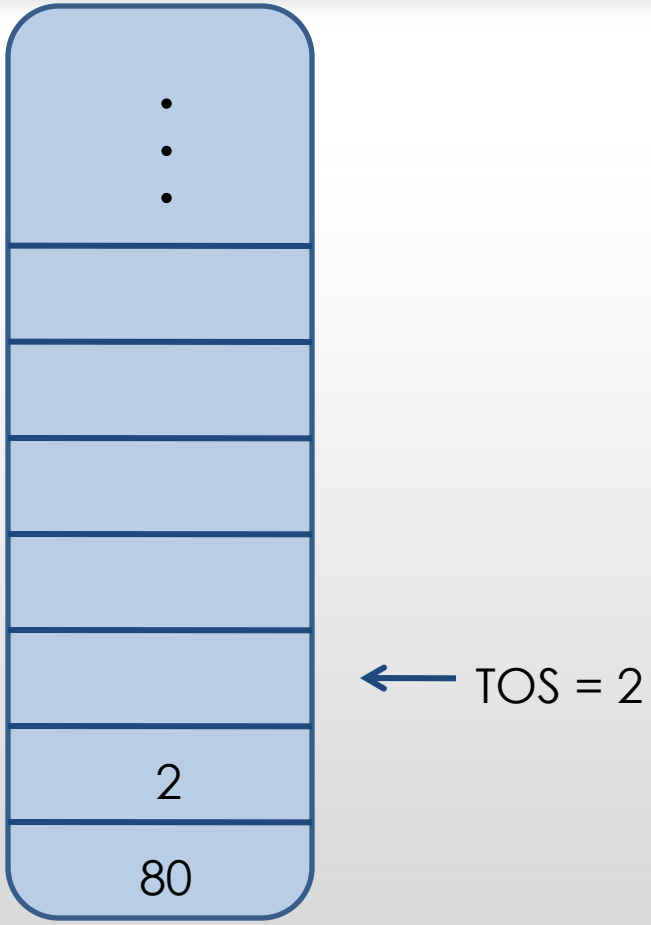
← TOS = 2

Expression: **4 6 + 3 5 + * 2 ***

Example

Expression: **4 6 + 3 5 + * 2 ***

Example



Expression: **4 6 + 3 5 + * 2 ***

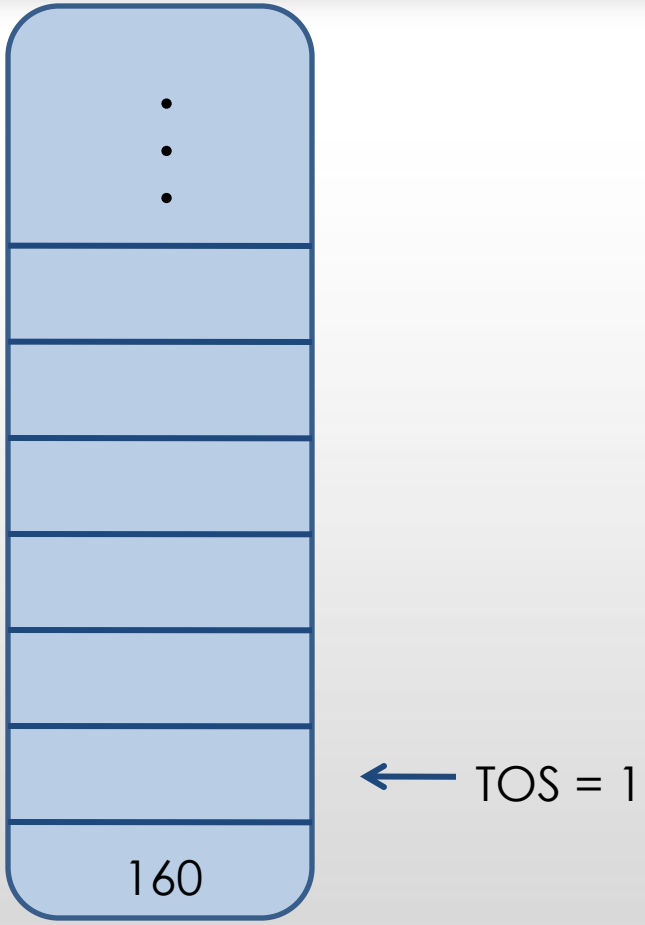
Example

Evaluate: 80 * 2

← TOS = 2

Expression: **4 6 + 3 5 + * 2 ***

Example



Expression: **4 6 + 3 5 + * 2 ***