CANTHO UNIVERSITY

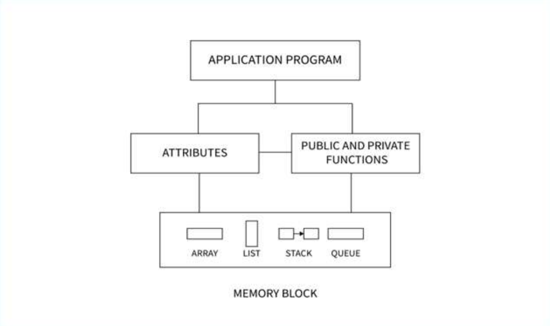
Chapter 2: Common abstract data types

Lâm Hoài Bảo - FSE – CICT
Trương Minh Thái – FSE - CICT

CANTHO UNIVERSITY

Content

- List abstract data type (LIST)
- Stack abstract data type (STACK)
- Queue abstract data type (QUEUE)

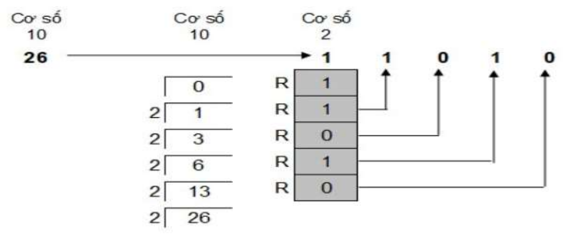


Stack

- Stack ADT
- Implementation
- Example
- Summary

Example of decimal to binary conversion

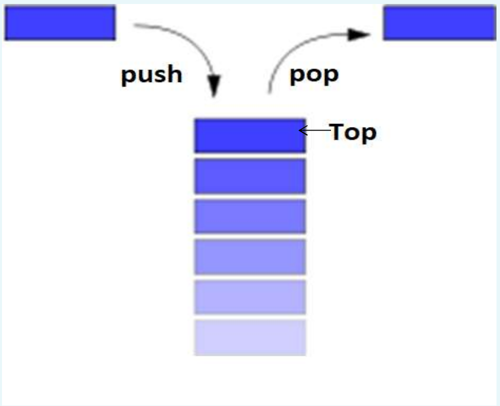
- Algorithm: divide by 2, the remainder is written from left to right.
- Example $(26)_{10} = (11010)_2$



CANTHO UNIVERSITY

Definition

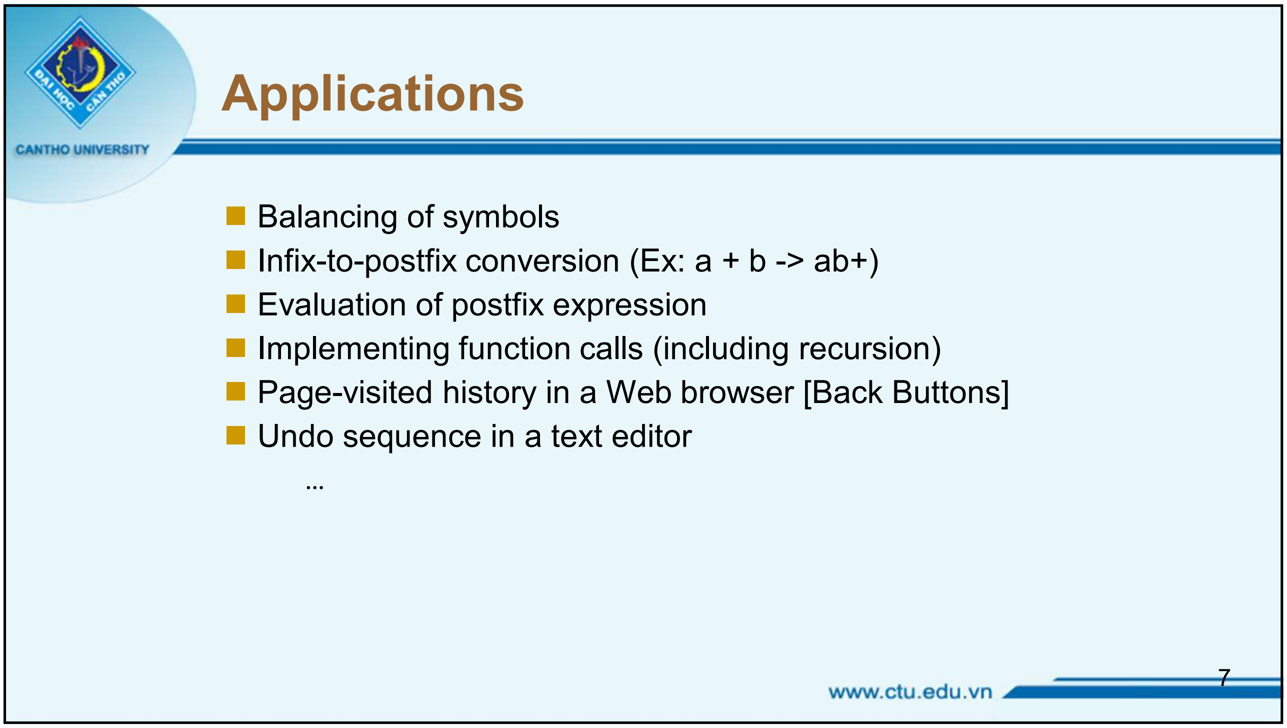
- Stack is a special list that only allows the **push** or **pop** of an element at a particular position called the **top**
- LIFO (Last In First Out) or FILO(First In Last Out) → **Stack**



CANTHO UNIVERSITY

Operators

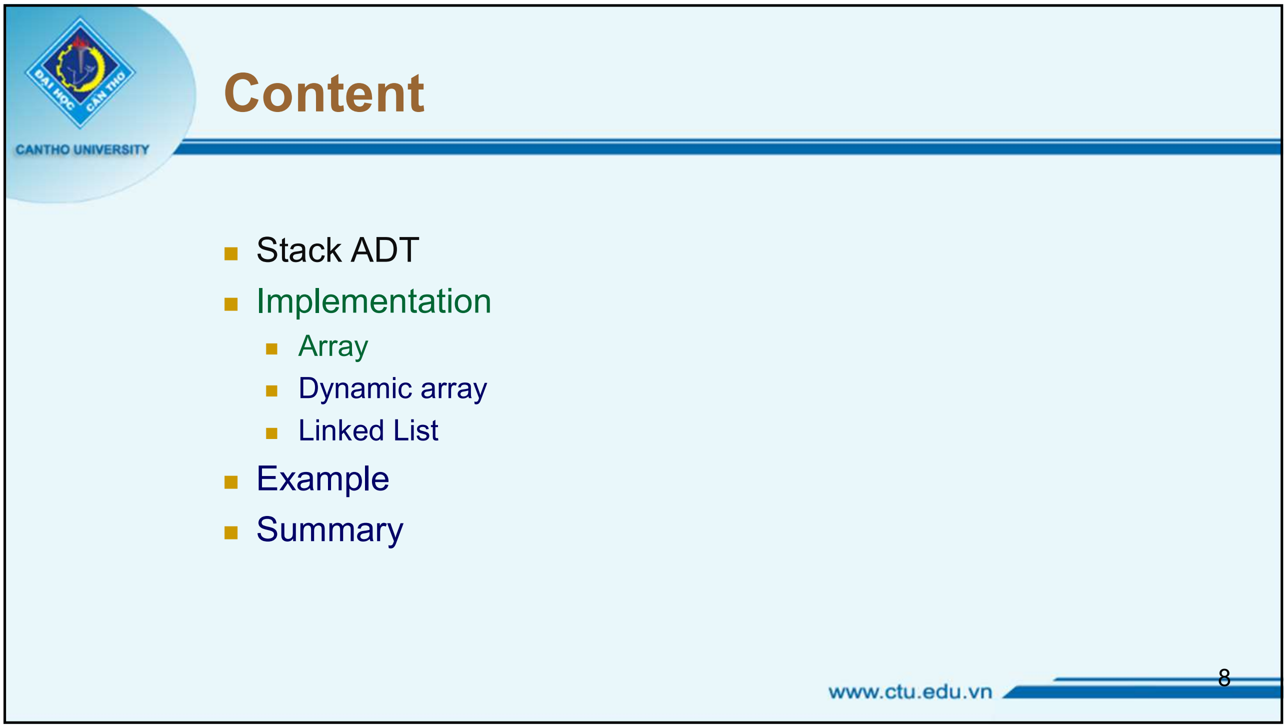
Operator	Description	~List operator
`makeNull(&S)`	Initialize an empty stack	`makenull(&S)`
`isEmpty(S)`	Check whether a stack is empty?	`empty(S)`
`isFull(S)`	Check whether a stack is full?	full(S)
`push(x, &S)`	Push x to the top	`insertLast(x, &S)` (`insertFirst(x, &S)`)
`pop(&S)`	Return the element at the top and remove it	`deleteLast(&S)` (`deleteFirst(&S)`)



Applications

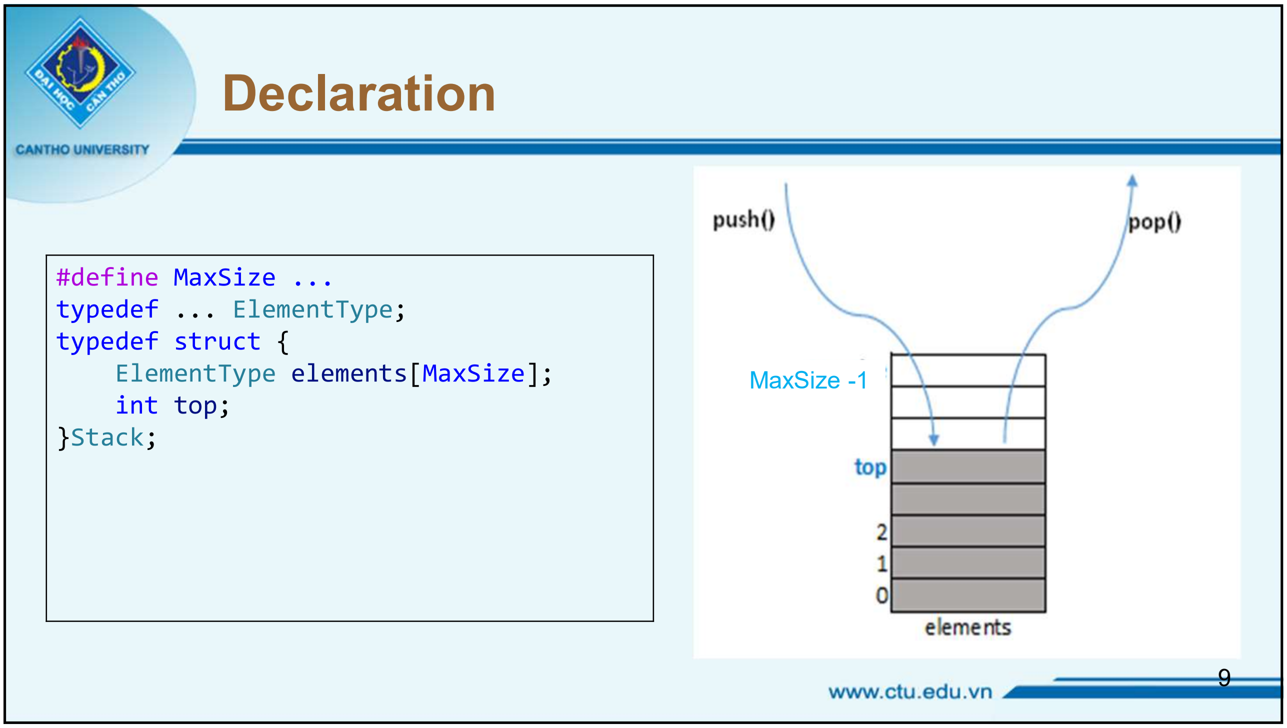
- Balancing of symbols
- Infix-to-postfix conversion (Ex: a + b -> ab+)
- Evaluation of postfix expression
- Implementing function calls (including recursion)
- Page-visited history in a Web browser [Back Buttons]
- Undo sequence in a text editor

  …



Content

- Stack ADT
- Implementation
  - Array
  - Dynamic array
  - Linked List
- Example
- Summary

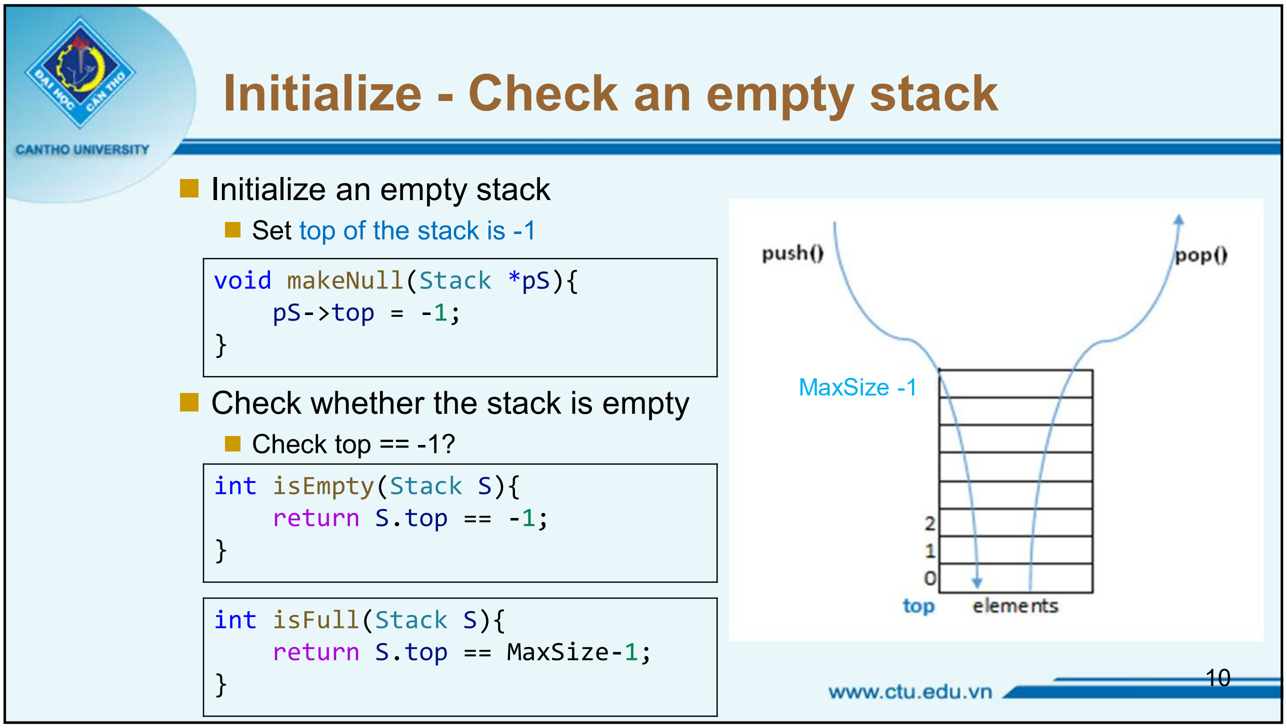


CANTHO UNIVERSITY

Declaration

```
#define MaxSize ...
typedef ... ElementType;
typedef struct {
    ElementType elements[MaxSize];
    int top;
}Stack;
```

push()
pop()
MaxSize -1
top
2
1
0
elements



CANTHO UNIVERSITY

Initialize - Check an empty stack

- Initialize an empty stack
  - Set top of the stack is -1

```
void makeNull(Stack *pS){
    pS->top = -1;
}
```

- Check whether the stack is empty
  - Check top == -1?

```
int isEmpty(Stack S){
    return S.top == -1;
}
```

```
int isFull(Stack S){
    return S.top == MaxSize-1;
}
```

push()
pop()
MaxSize -1
2
1
0
top
elements

CANTHO UNIVERSITY

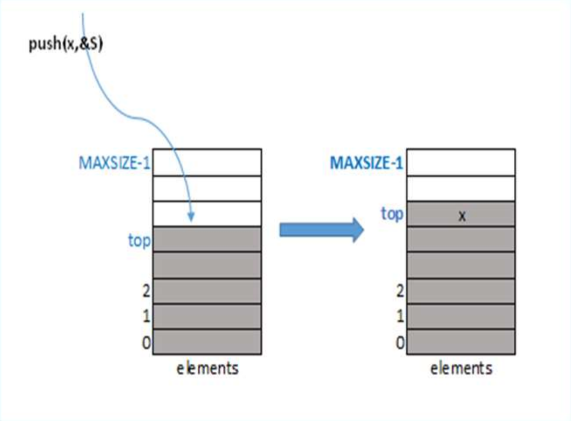
Push a new element to the top

- Algorithm

```
ALGORITHM push(x, *pS):
    if (isFull(*pS)): #check stack full?
        raiseError("Stack is full")
    else:
        pS->top ← pS->top + 1
        pS->elements[pS->top] ← x
```

T(n) = O(1)

- Stack is full when top == MAXSIZE - 1



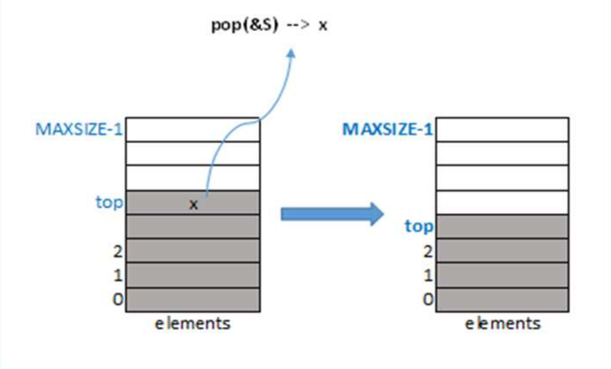
CANTHO UNIVERSITY

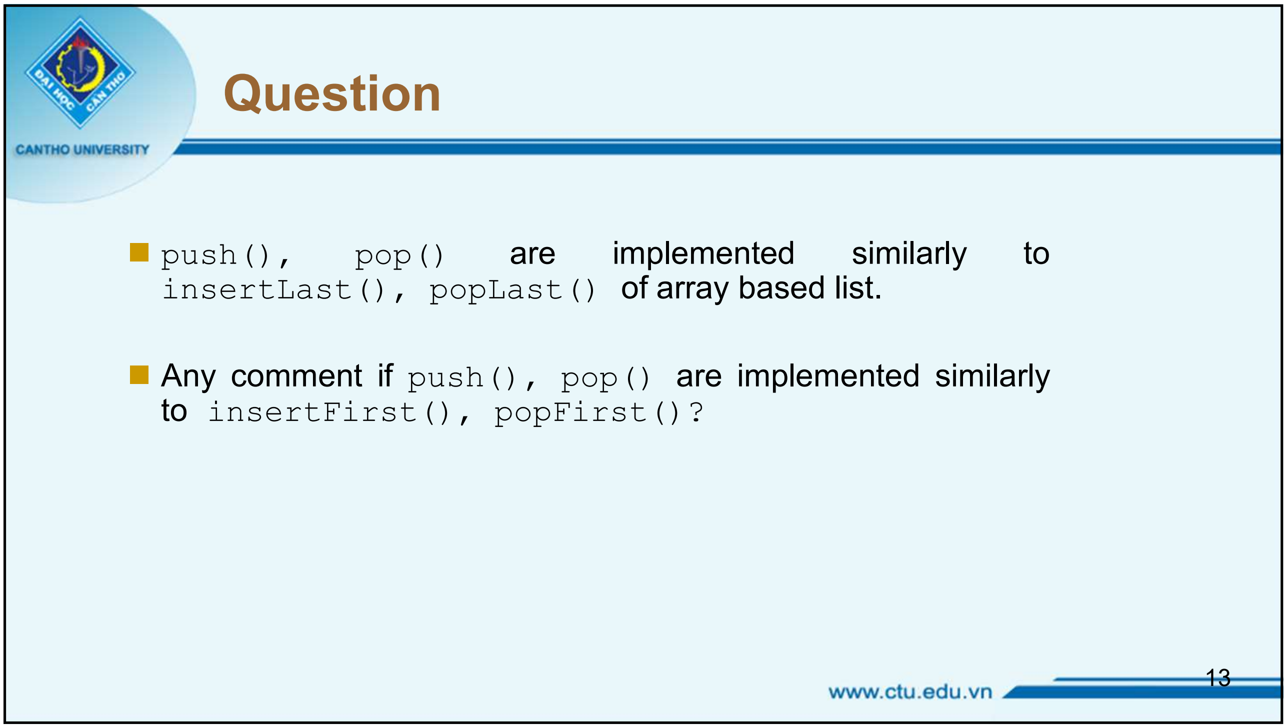
Get the top element and remove it

- Algorithm

```
ALGORITHM pop(*pS):
    if (isEmpty(*pS)):
        return ERROR
    else:
        x ← pS->elements[pS->top]
        pS->top ← pS->top - 1
        return x
```

- T(n) = O(1)

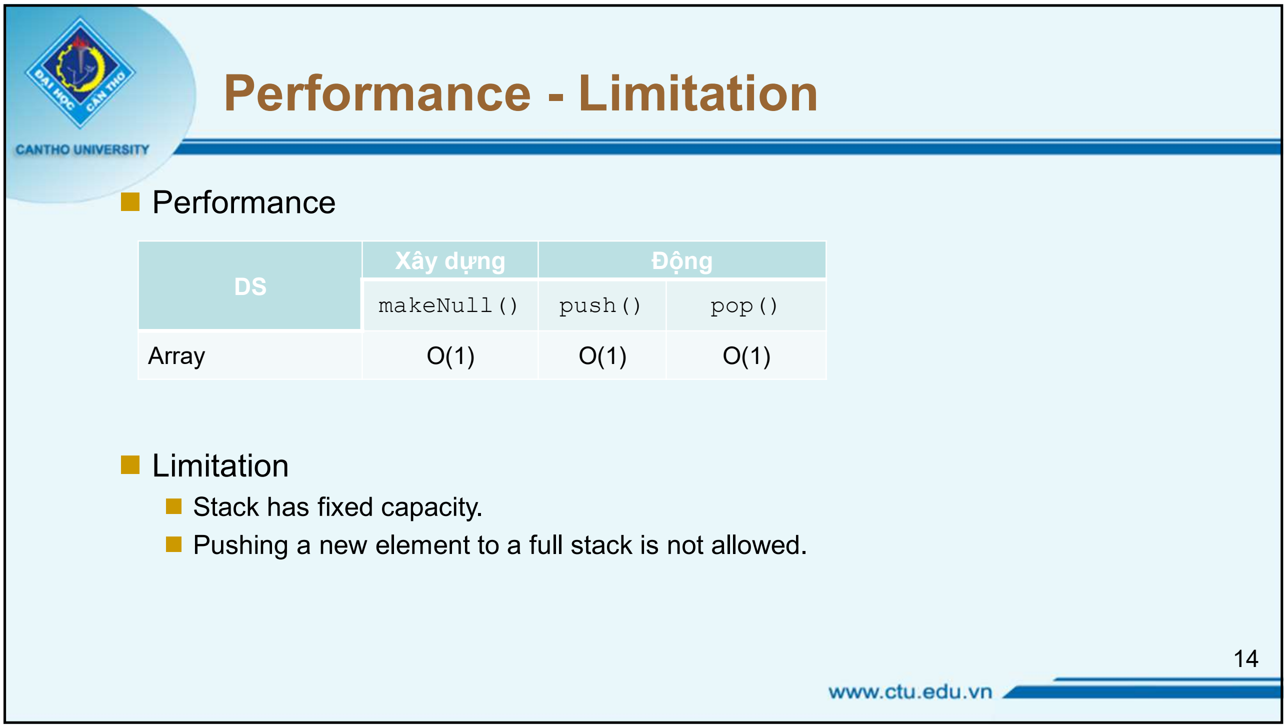




CANTHO UNIVERSITY

Question

- `push()`, `pop()` are implemented similarly to `insertLast()`, `popLast()` of array based list.
- Any comment if `push()`, `pop()` are implemented similarly to `insertFirst()`, `popFirst()`?



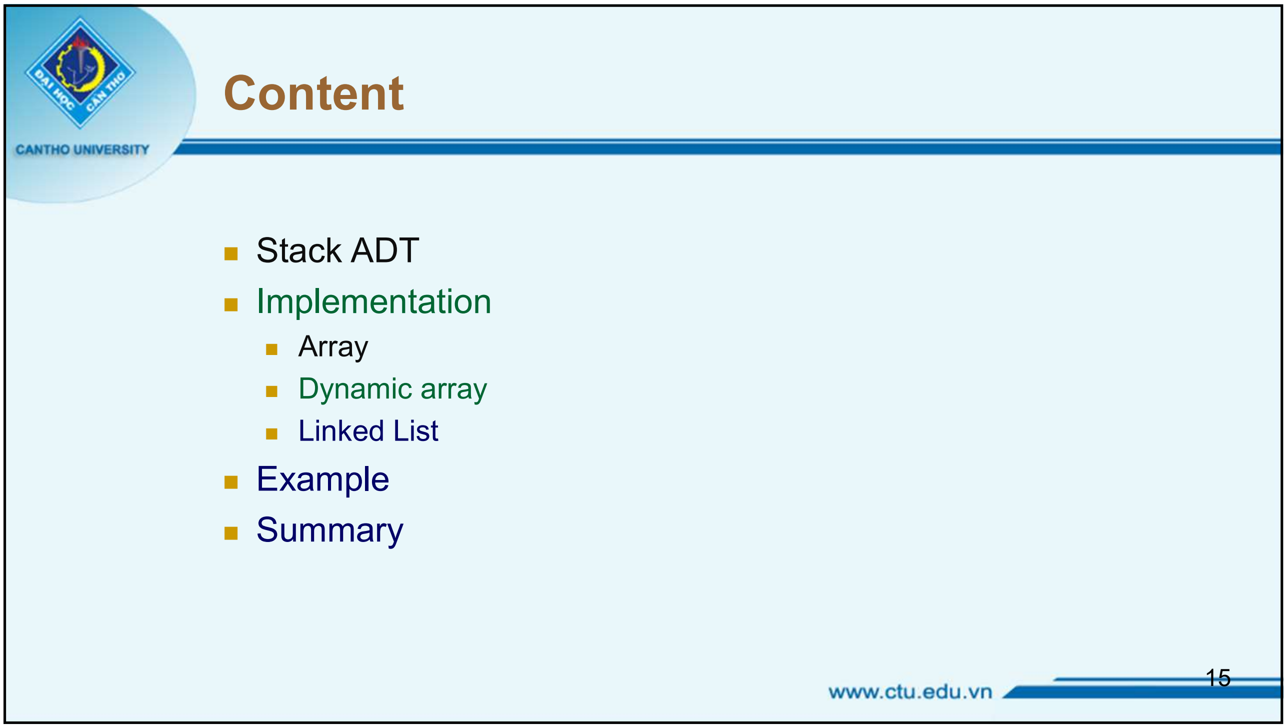
CANTHO UNIVERSITY

Performance - Limitation

- Performance

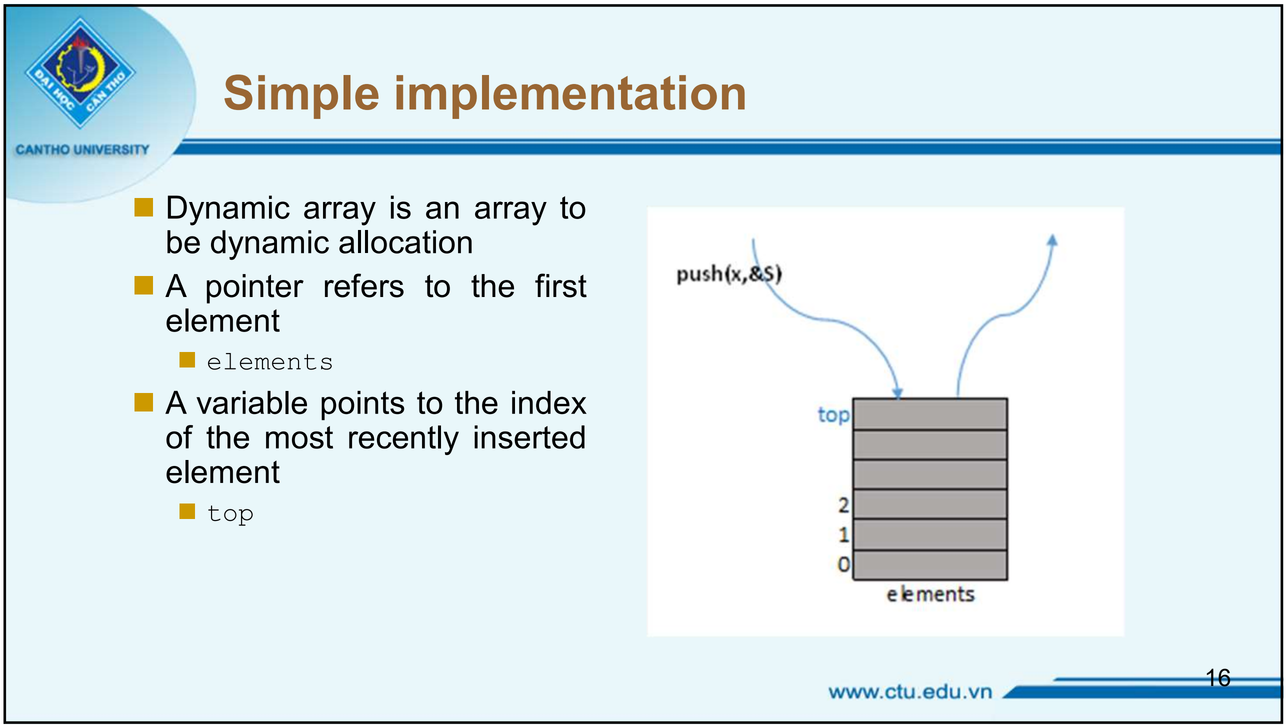
DS	Xây dựng	Động	
	`makeNull()`	`push()`	`pop()`
Array	O(1)	O(1)	O(1)

- Limitation
  - Stack has fixed capacity.
  - Pushing a new element to a full stack is not allowed.



Content

- Stack ADT
- Implementation
  - Array
  - Dynamic array
  - Linked List
- Example
- Summary



Simple implementation

- Dynamic array is an array to be dynamic allocation
- A pointer refers to the first element
  - `elements`
- A variable points to the index of the most recently inserted element
  - `top`

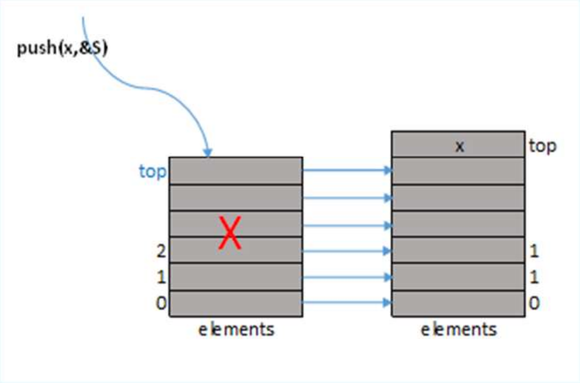
Simple array based implementation

- Push a new element:
  - Allocate a new array
  - Copy elements to new memory location
  - Release the old memory

  → **T(n) = O(n)**

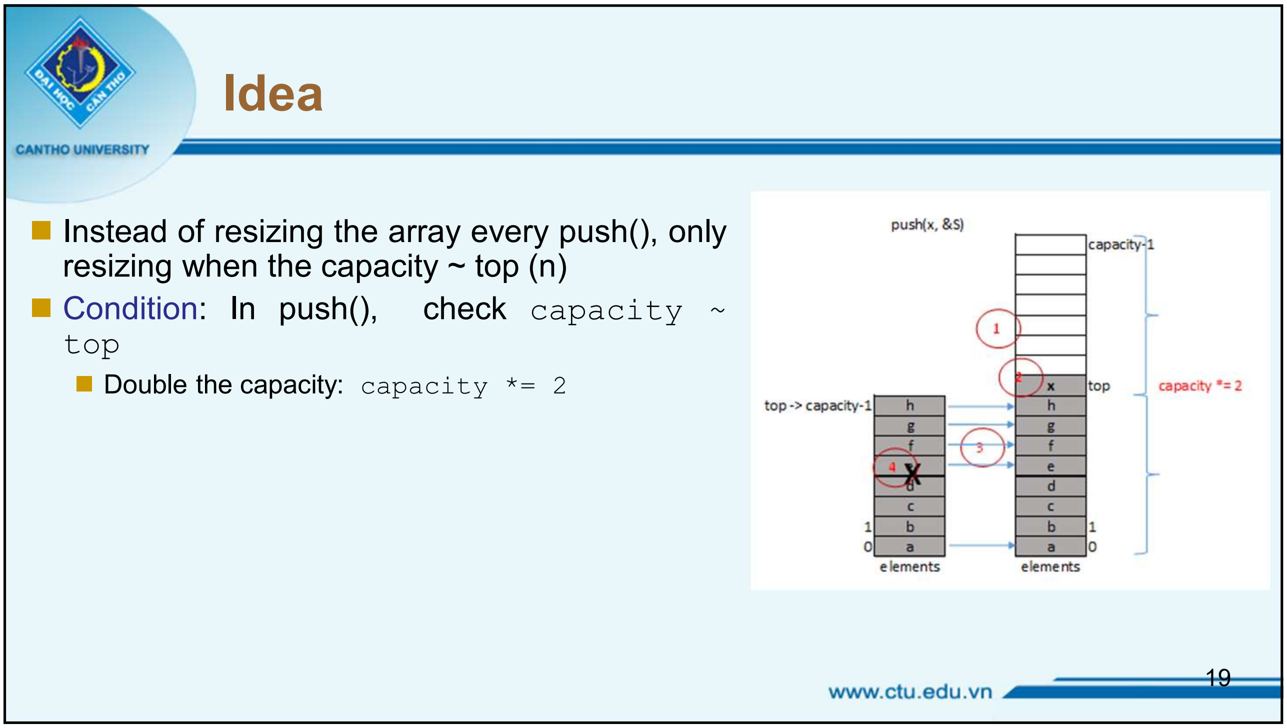
- If there are n contiguous push()
  T(n) = O(n^2)

Can we do better?



Exercise 5.1

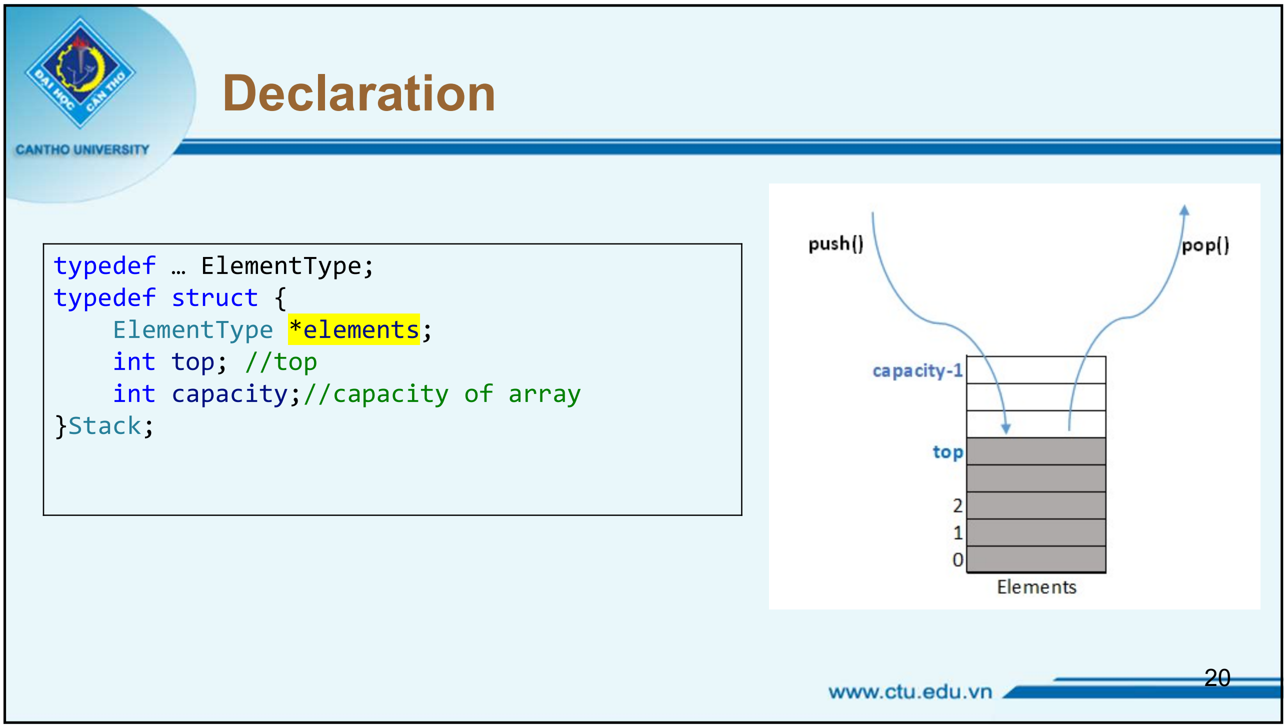
- Write a function to convert integer n to binary number (using operations on the stack)



CANTHO UNIVERSITY

Idea

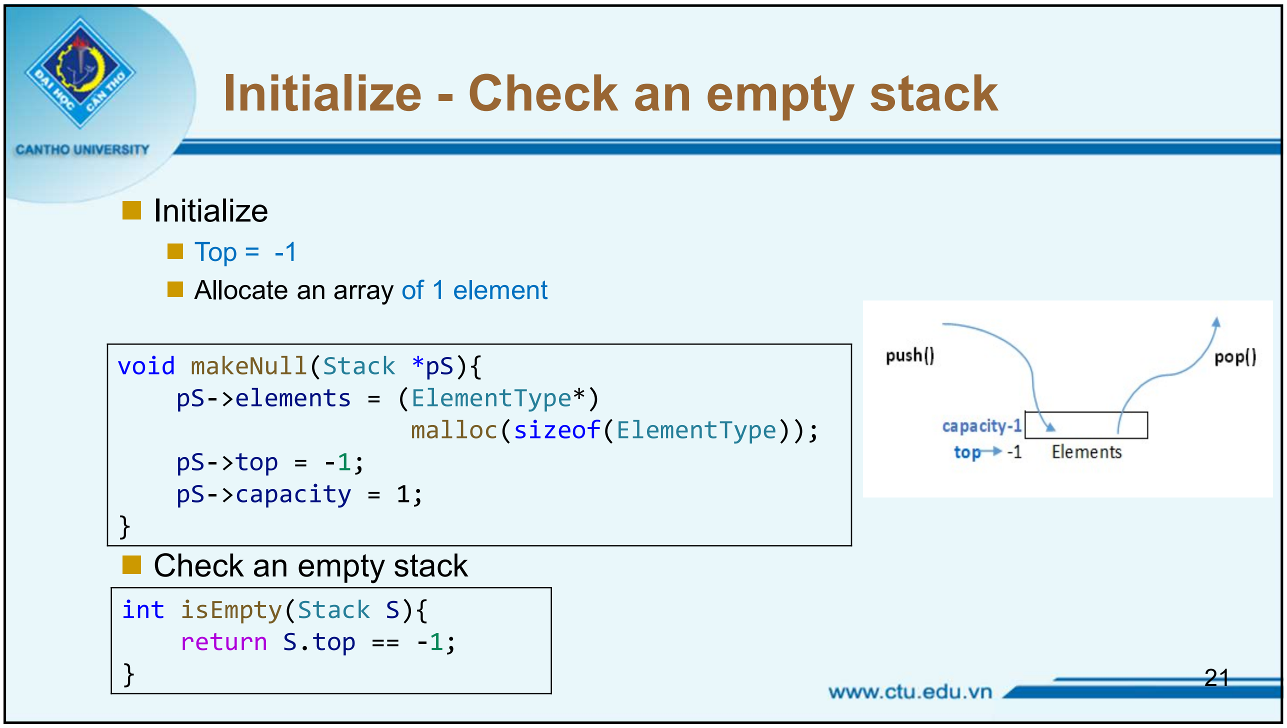
- Instead of resizing the array every push(), only resizing when the capacity ~ top (n)
- Condition: In push(), check `capacity ~ top`
  - Double the capacity: `capacity *= 2`



CANTHO UNIVERSITY

Declaration

```
typedef … ElementType;
typedef struct {
    ElementType *elements;
    int top; //top
    int capacity;//capacity of array
}Stack;
```



Initialize - Check an empty stack

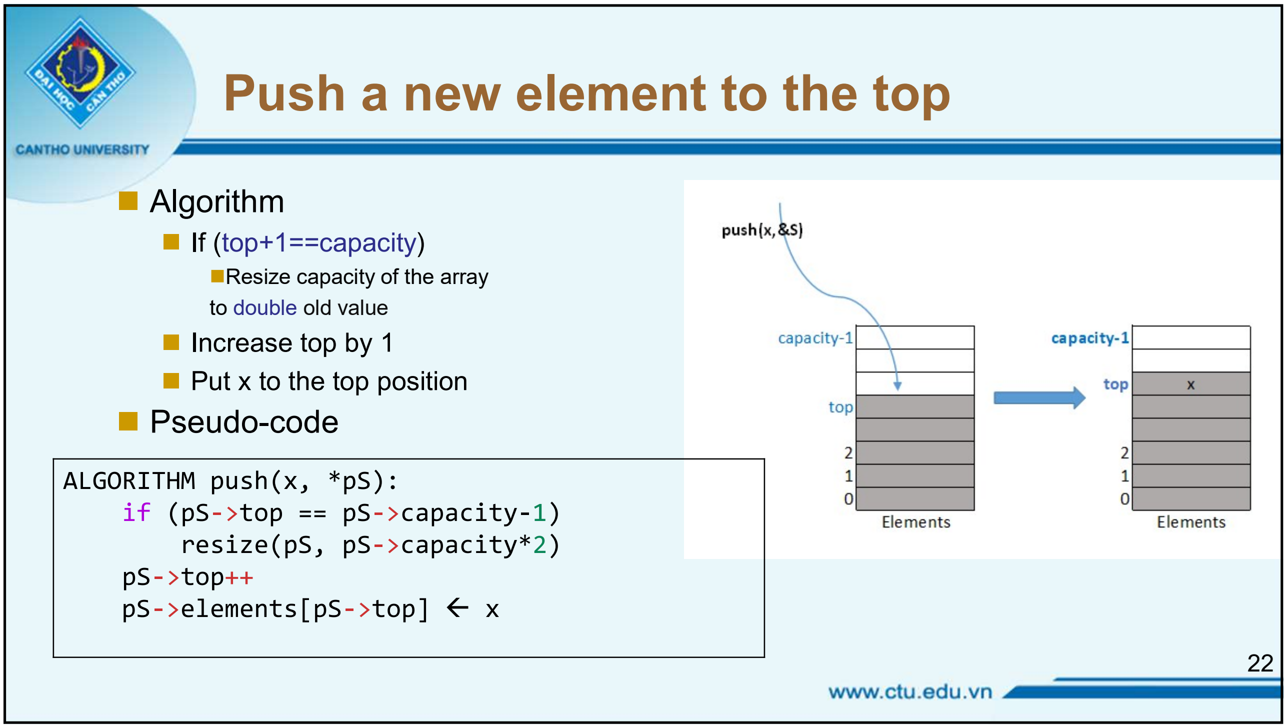
- Initialize
  - Top = -1
  - Allocate an array of 1 element

```
void makeNull(Stack *pS){
    pS->elements = (ElementType*)
                    malloc(sizeof(ElementType));
    pS->top = -1;
    pS->capacity = 1;
}
```

- Check an empty stack

```
int isEmpty(Stack S){
    return S.top == -1;
}
```

push()
pop()
capacity-1
top
-1
Elements



Push a new element to the top

- Algorithm
  - If (top+1==capacity)
    - Resize capacity of the array to double old value
  - Increase top by 1
  - Put x to the top position
- Pseudo-code

```
ALGORITHM push(x, *pS):
    if (pS->top == pS->capacity-1)
        resize(pS, pS->capacity*2)
    pS->top++
    pS->elements[pS->top] ← x
```

push(x,&S)
capacity-1
top
2
1
0
Elements
capacity-1
top
x
2
1
0
Elements

CANTHO UNIVERSITY

Resize capacity

- Algorithm

```
ALGORITHM resize(*pS, newCapacity):
    A 🡐 pS->elements
    pS->capacity 🡐 newCapacity
    pS->elements 🡐
                malloc(newCapacity*|ElementType|)
    for i=0 to pS->top - 1:
        pS->elements[i] 🡐 A[i]
    free(A)
```

→ T(n) = O(n)

CANTHO UNIVERSITY

Amortized analysis

- In the worst case `push()` costs O(n)
- However, the capacity only resizes if n = 2^i
  - n contiguous `push()` cost (1 + 2 + 4 + … + n) = 2n = O(n)
  - ~ in average, each operator costs O(1) or each *operator costs O(1) amortized*
- Few `push()` is linear, but O(1) per each operator (O(1) *amortized*)

CANTHO UNIVERSITY

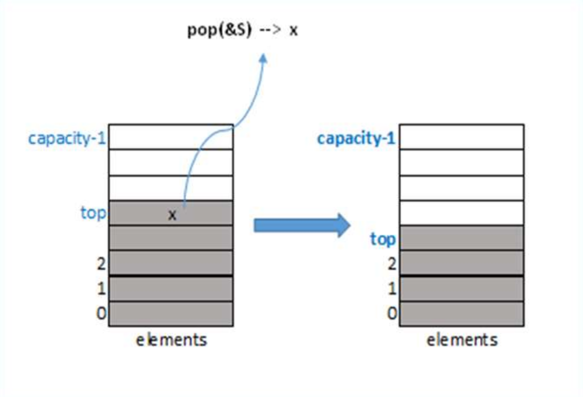
Amortized analysis

- A series of operators -> estimate the cost per operator
  - Example: A series of n insert, lookup, remove from a database
- An operator costs T(n) amortized if k continuous operators cost <= kT(n)
- T(n) amortized ~ the average cost of k continuous operators
  - `push()` to a dynamic array based stack costs O(1) amortized

CANTHO UNIVERSITY

Get and remove the top element

- Algorithm
  - If (top+1 <= capacity/4)
    - Resize capacity of the array to a half of old value
  - Let x be the element at top
  - Decrease top by 1
  - **RETURN** x
- T(n) = O(1)
- Why capacity/4?



CANTHO UNIVERSITY

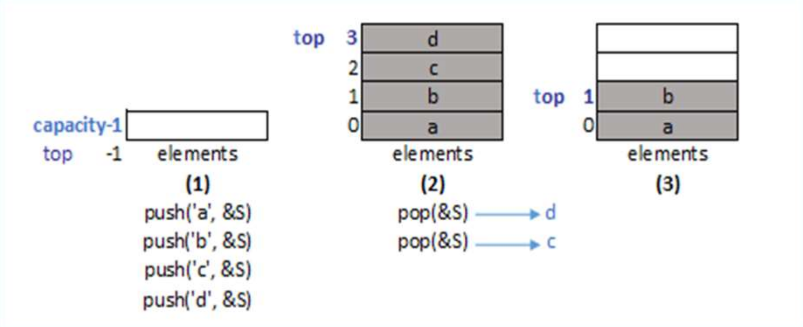
Example

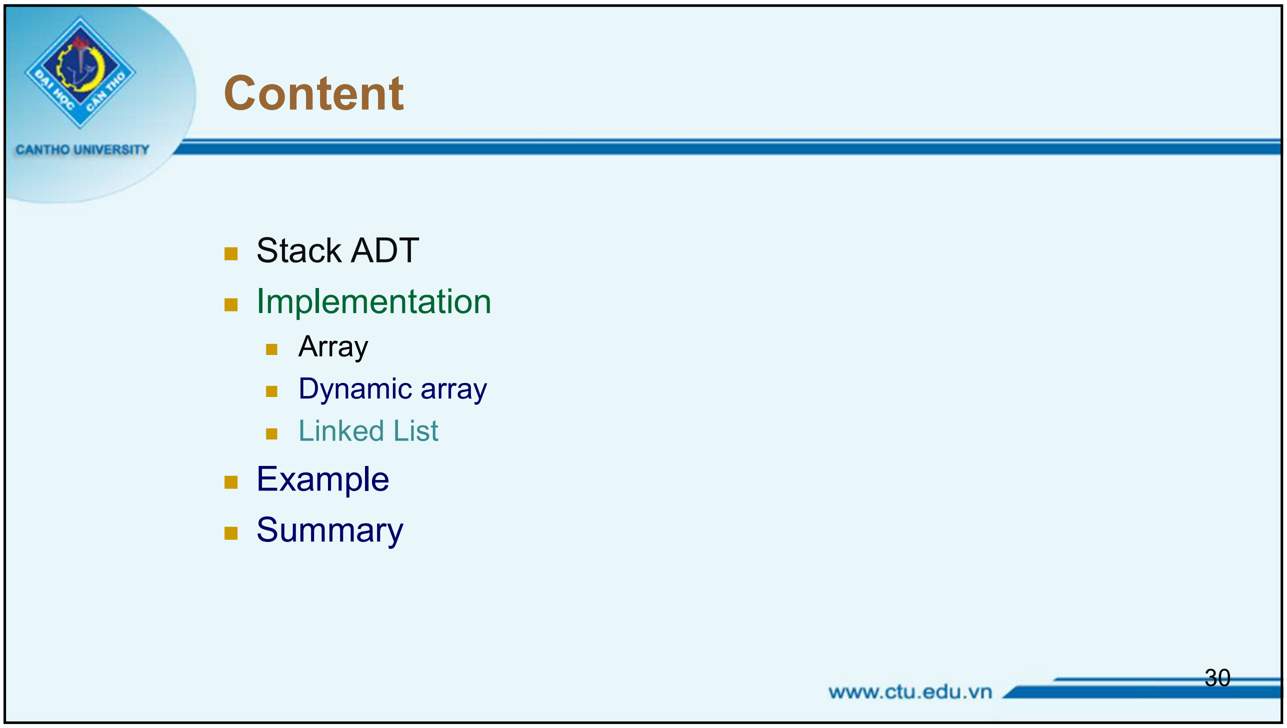
- Give an empty stack **(1)**, describe the stack after
  - Call **(2)**
    - push(‘a’, &S)
    - push(‘b’, &S)
    - push(‘c’, &S)
    - push(‘d’, &S)
  - Call **(3)**
    - pop(&S)
    - pop(&S)

CANTHO UNIVERSITY

Example

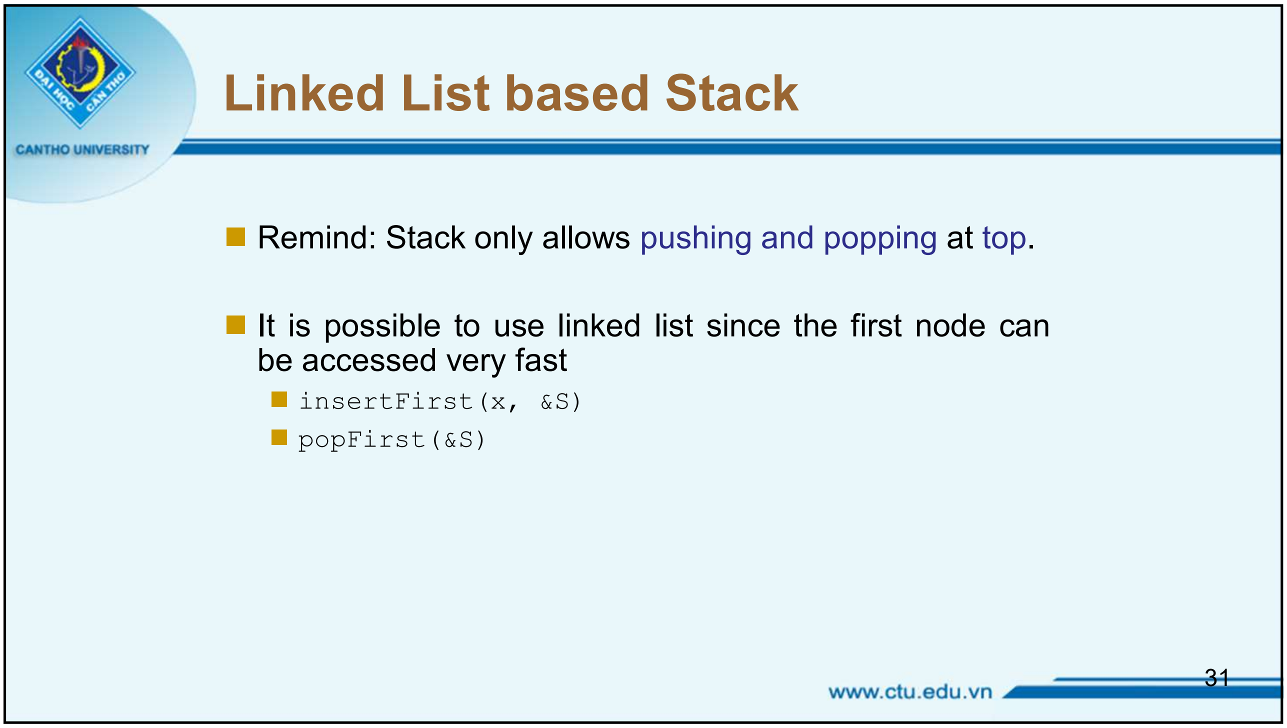
- Give an empty stack **(1)**, describe the stack after
  - Call **(2)**
    - push(‘a’, &S)
    - push(‘b’, &S)
    - push(‘c’, &S)
    - push(‘d’, &S)
  - Call **(3)**
    - pop(&S)
    - pop(&S)





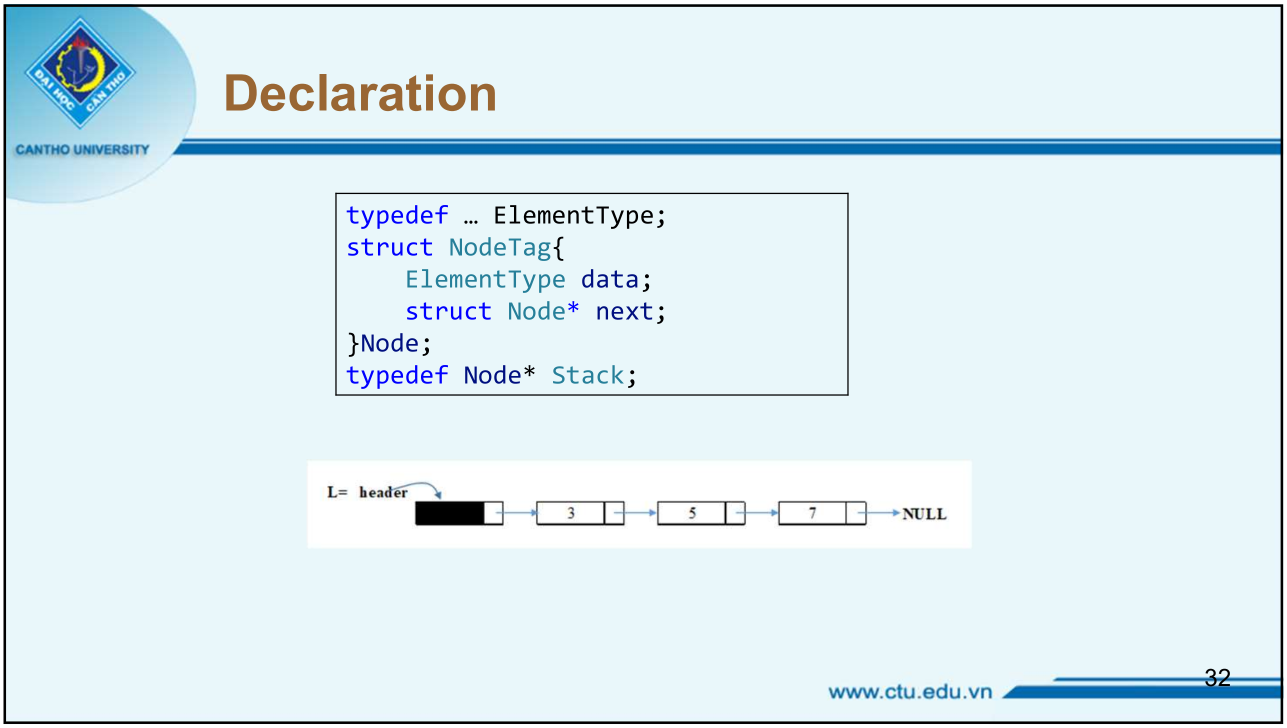
Content

- Stack ADT
- Implementation
  - Array
  - Dynamic array
  - Linked List
- Example
- Summary



Linked List based Stack

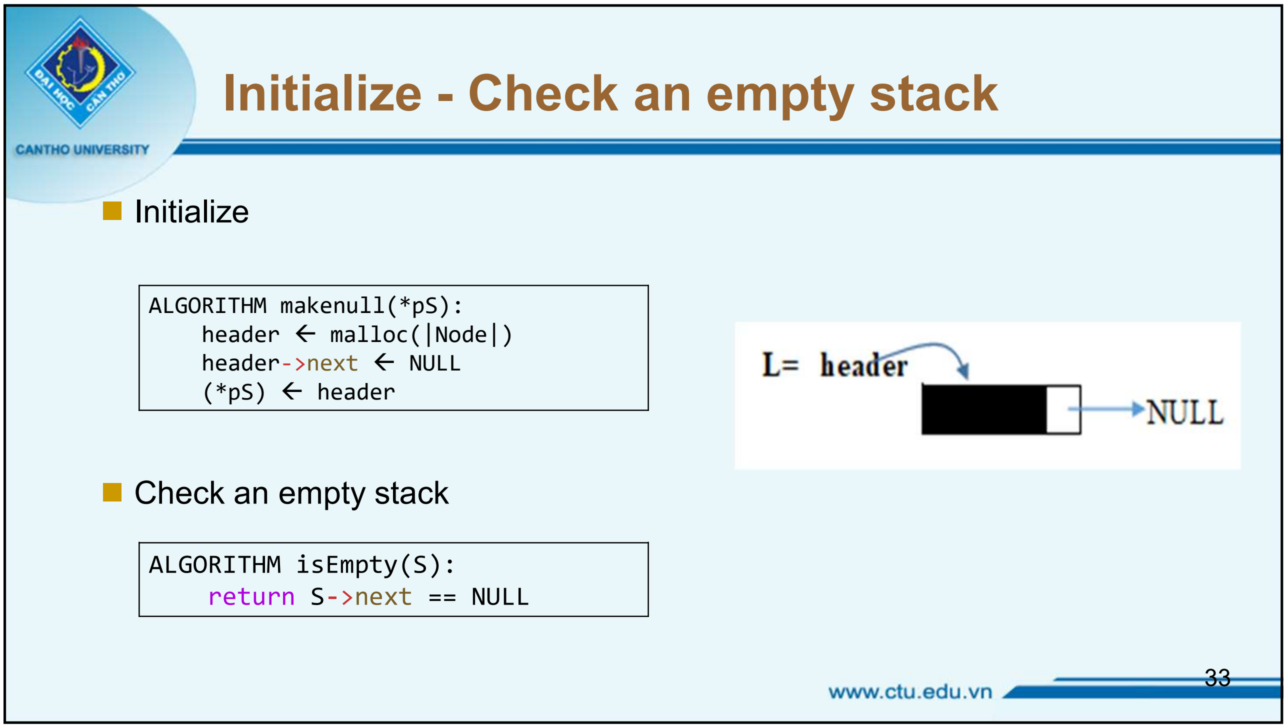
- Remind: Stack only allows pushing and popping at top.
- It is possible to use linked list since the first node can be accessed very fast
  - `insertFirst(x, &S)`
  - `popFirst(&S)`



Declaration

```
typedef … ElementType;
struct NodeTag{
    ElementType data;
    struct Node* next;
}Node;
typedef Node* Stack;
```

L= header
3
5
7
NULL



Initialize - Check an empty stack

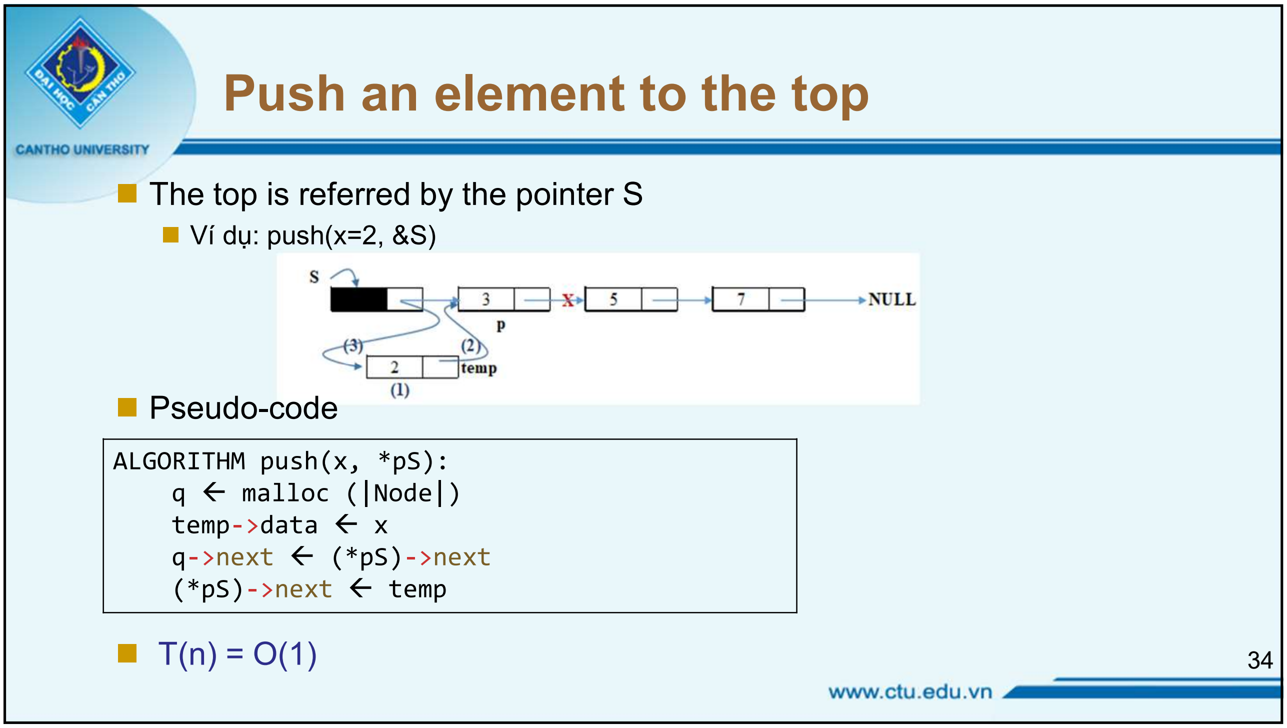
- Initialize

```
ALGORITHM makenull(*pS):
    header ← malloc(|Node|)
    header->next ← NULL
    (*pS) ← header
```

L= header
NULL

- Check an empty stack

```
ALGORITHM isEmpty(S):
    return S->next == NULL
```



Push an element to the top

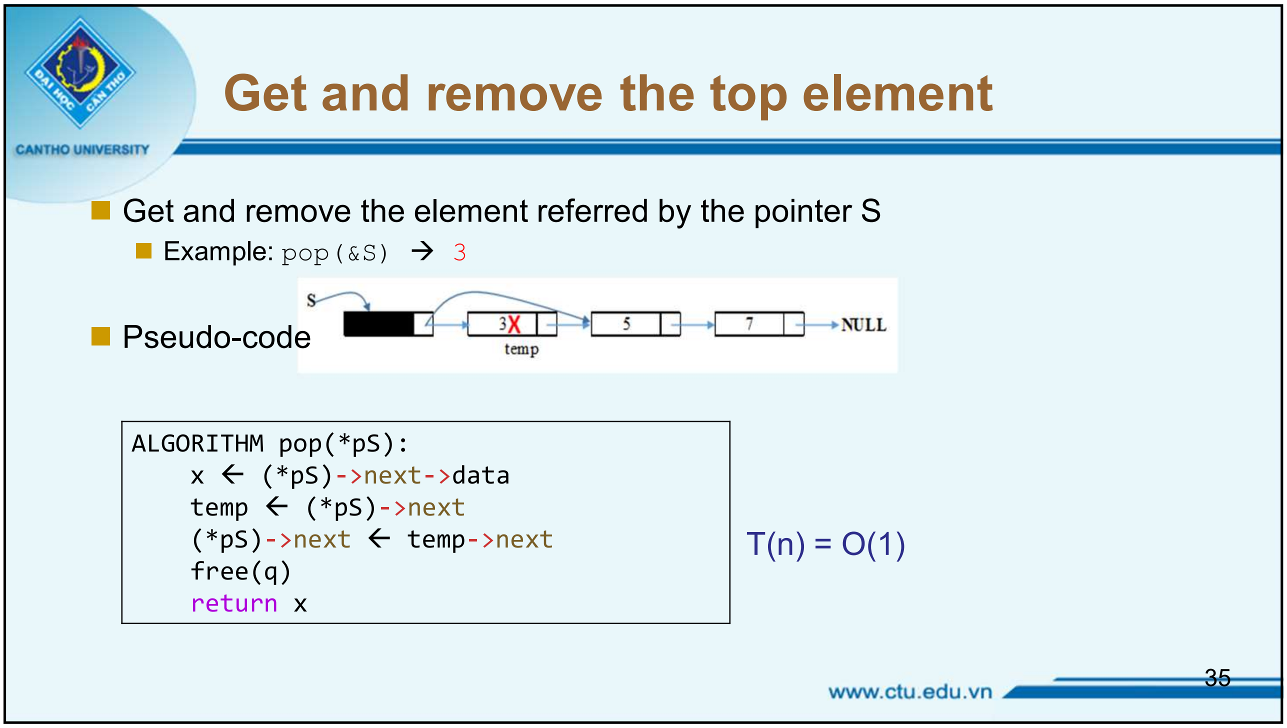
- The top is referred by the pointer S
  - Ví dụ: push(x=2, &S)

- Pseudo-code

```
ALGORITHM push(x, *pS):
    q ← malloc (|Node|)
    temp->data ← x
    q->next ← (*pS)->next
    (*pS)->next ← temp
```

- T(n) = O(1)

34



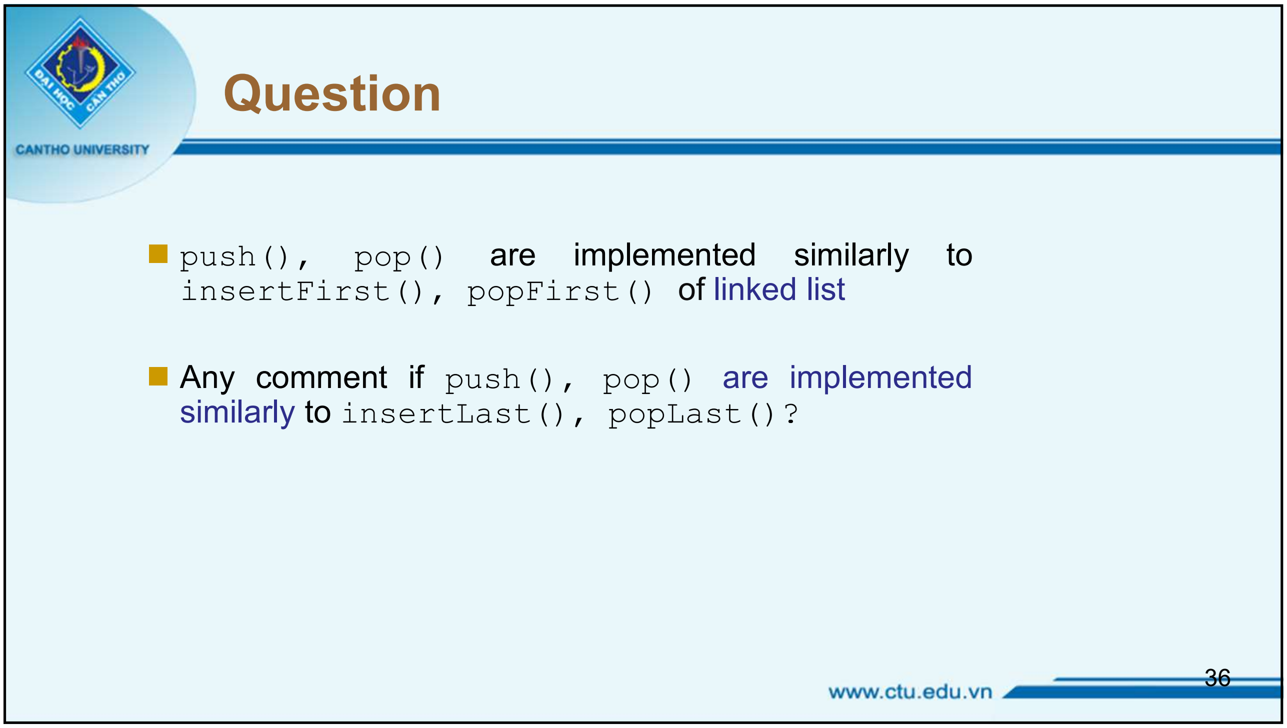
Get and remove the top element

- Get and remove the element referred by the pointer S
  - Example: pop(&S) → 3
- Pseudo-code

```
ALGORITHM pop(*pS):
    x ← (*pS)->next->data
    temp ← (*pS)->next
    (*pS)->next ← temp->next
    free(q)
    return x
```

T(n) = O(1)

35

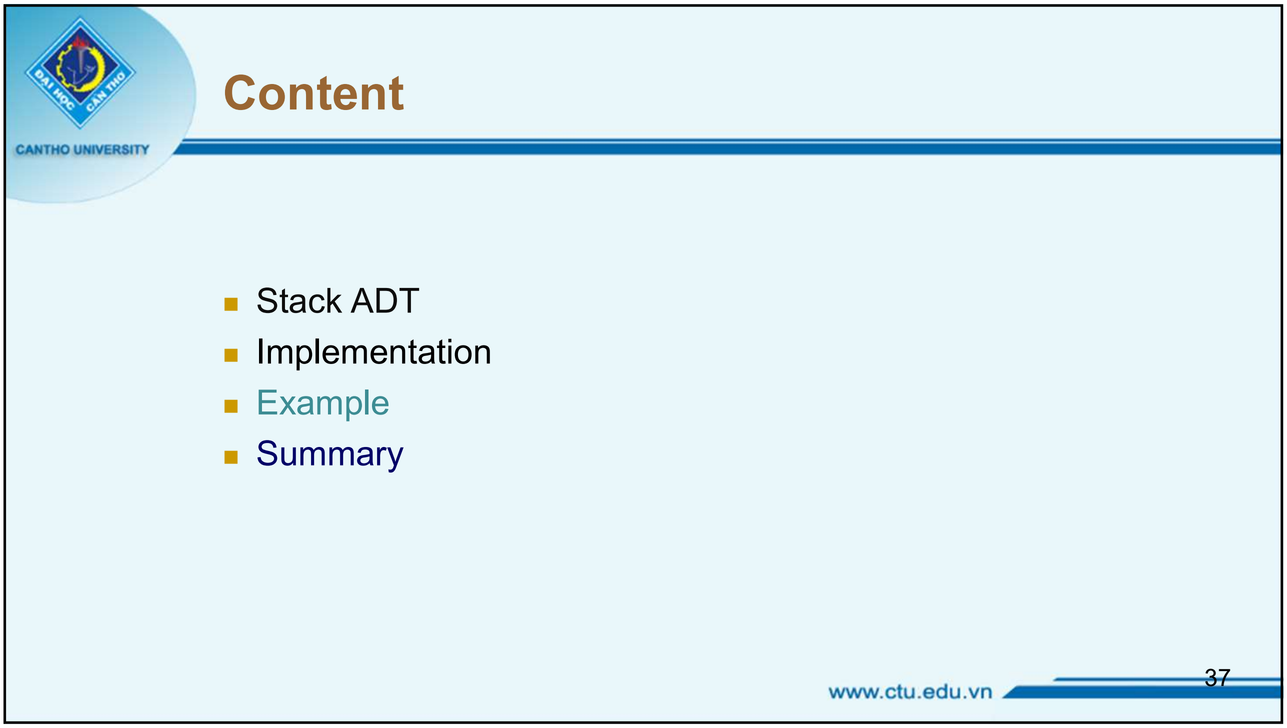


CANTHO UNIVERSITY

Question

- `push(), pop()` are implemented similarly to `insertFirst(), popFirst()` of linked list
- Any comment if `push(), pop()` are implemented similarly to `insertLast(), popLast()`?

www.ctu.edu.vn 36

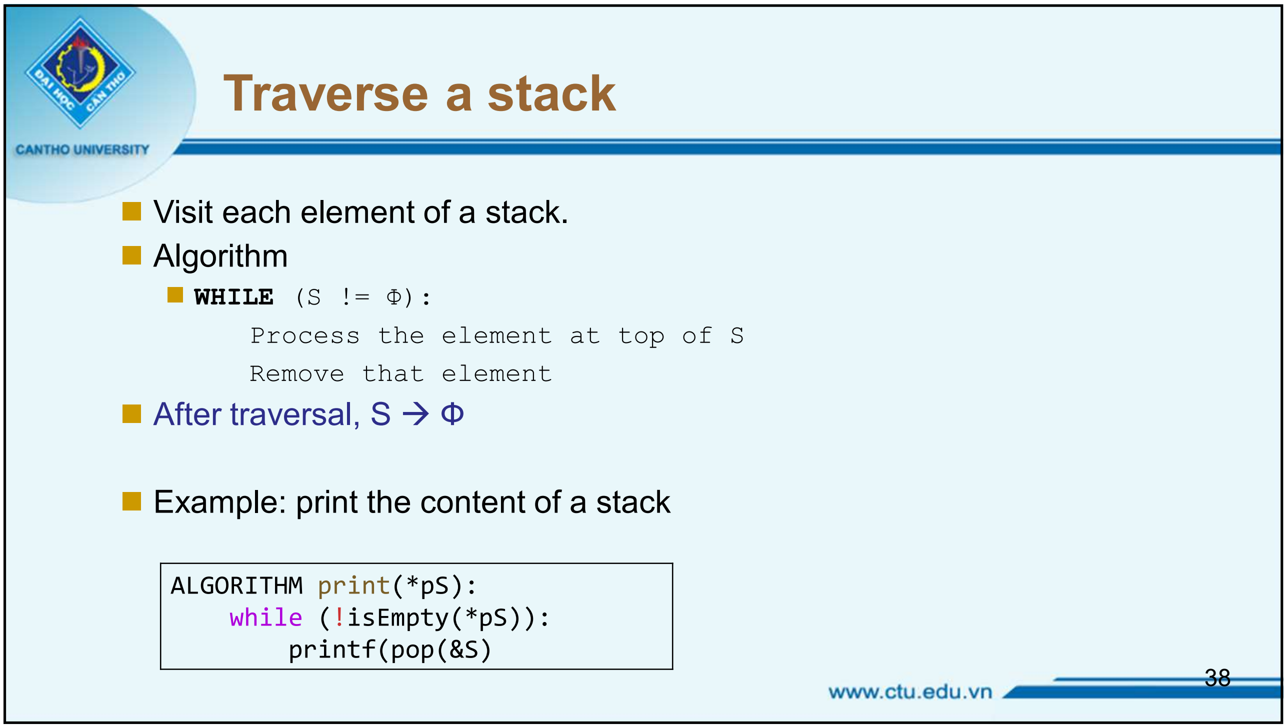


CANTHO UNIVERSITY

Content

- Stack ADT
- Implementation
- Example
- Summary

www.ctu.edu.vn 37



CANTHO UNIVERSITY

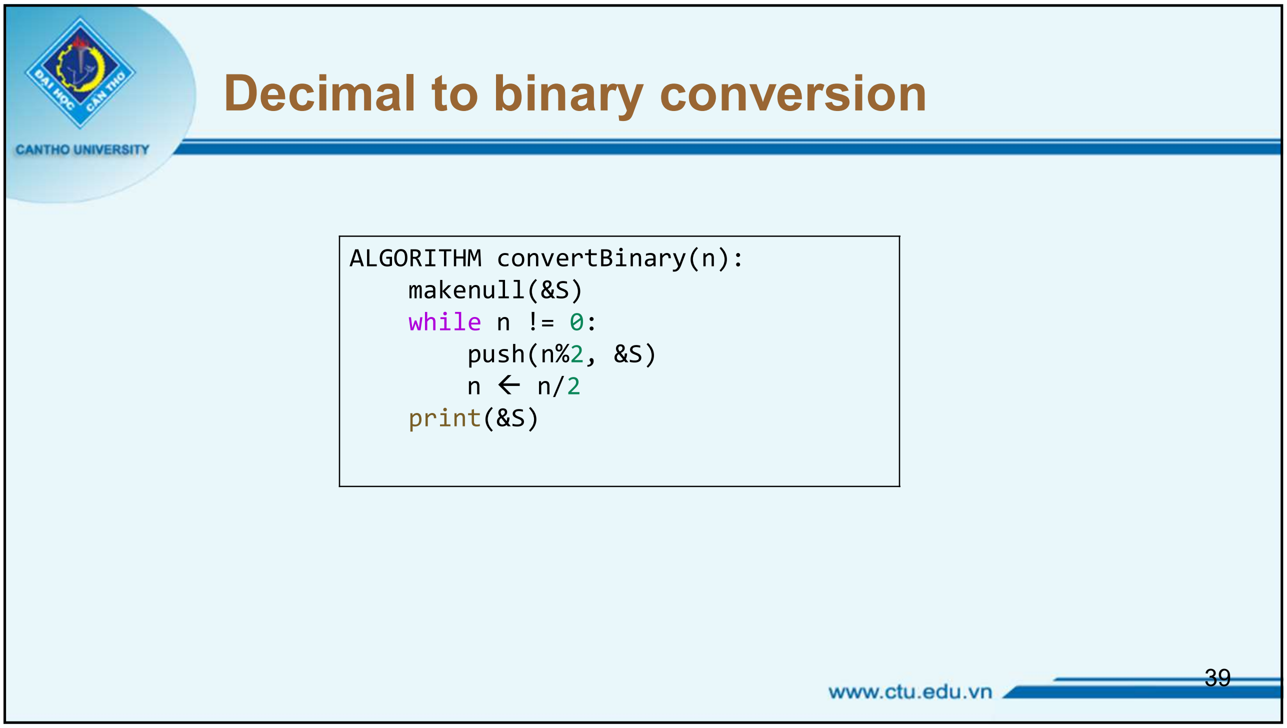
Traverse a stack

- Visit each element of a stack.
- Algorithm
  - **WHILE** (S != Φ):

```
        Process the element at top of S
        Remove that element
```

- After traversal, S → Φ
- Example: print the content of a stack

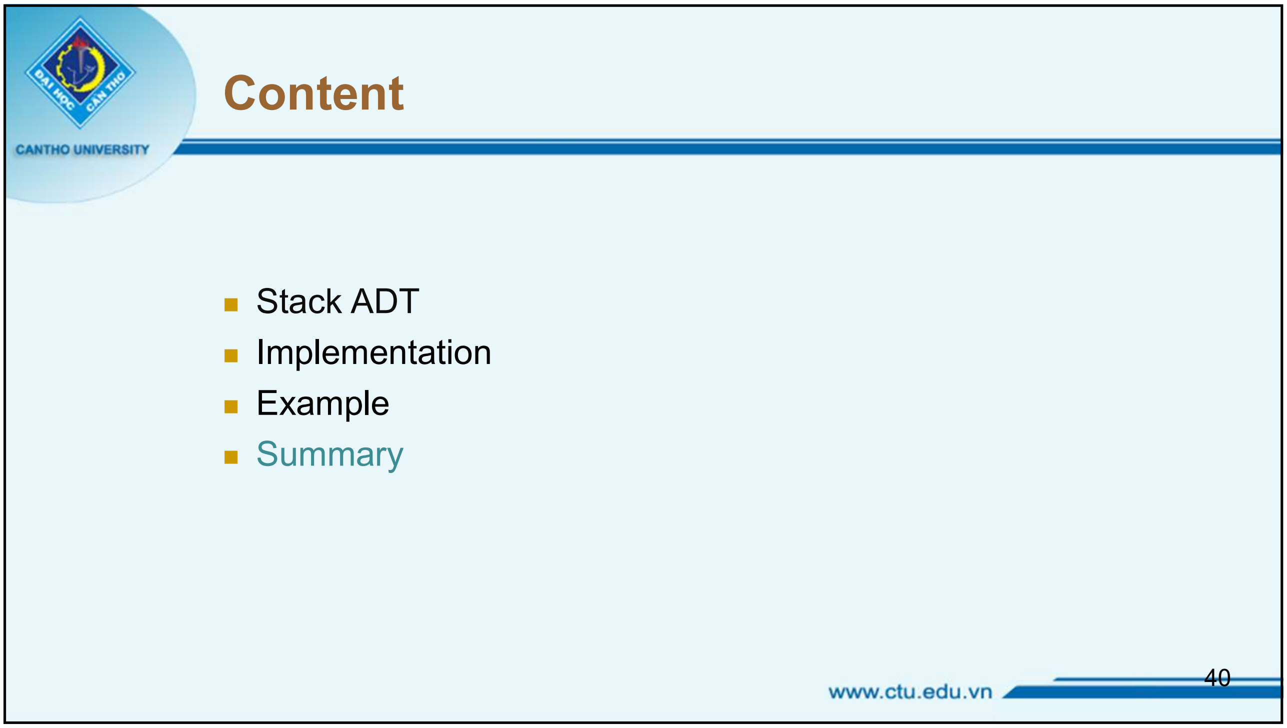
```
ALGORITHM print(*pS):
    while (!isEmpty(*pS)):
        printf(pop(&S)
```



CANTHO UNIVERSITY

Decimal to binary conversion

```
ALGORITHM convertBinary(n):
    makenull(&S)
    while n != 0:
        push(n%2, &S)
        n ← n/2
    print(&S)
```



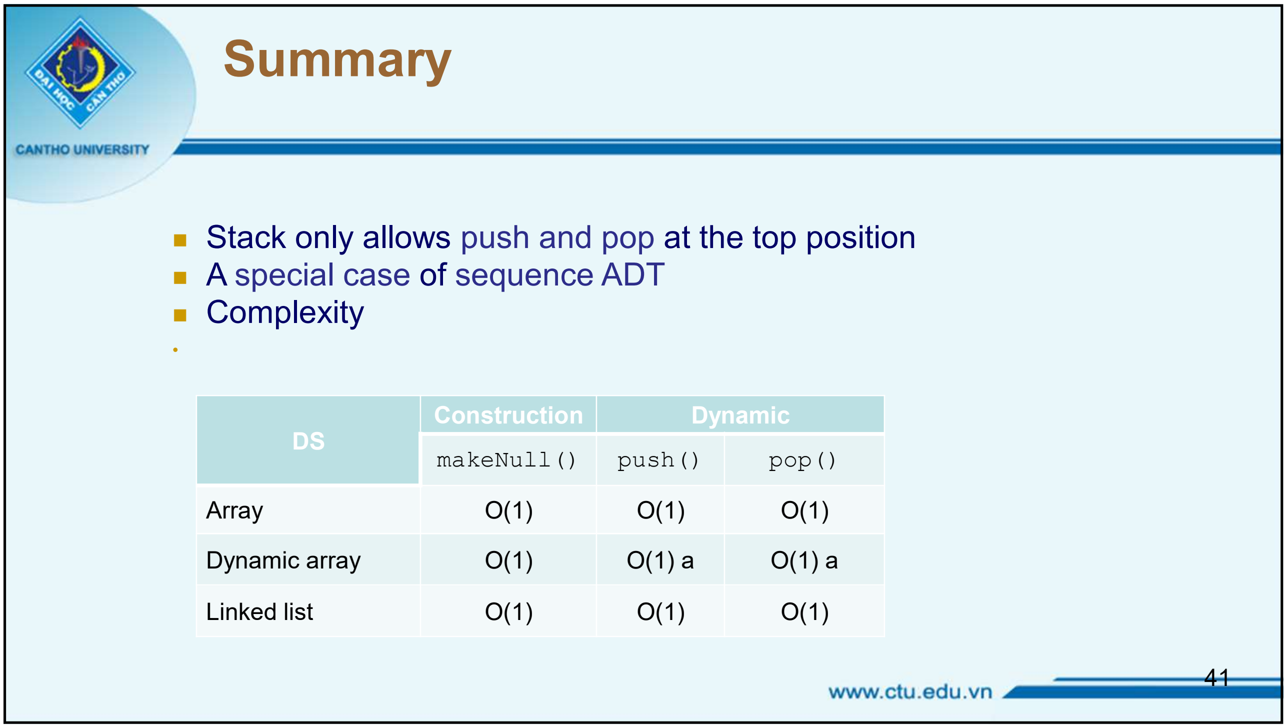
CANTHO UNIVERSITY

Content

- Stack ADT
- Implementation
- Example
- Summary

www.ctu.edu.vn 40

40



CANTHO UNIVERSITY

Summary

- Stack only allows push and pop at the top position
- A special case of sequence ADT
- Complexity

DS	Construction	Dynamic	
	`makeNull()`	`push()`	`pop()`
Array	O(1)	O(1)	O(1)
Dynamic array	O(1)	O(1) a	O(1) a
Linked list	O(1)	O(1)	O(1)

www.ctu.edu.vn 41

41

Exercise 5.2

```
int Fibonacci(int n)
{
    if (n == 1 || n == 2)
        return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

$$f(n) = f(n-1) + f(n-2)$$

$$f(1) = 1;\ \ f(2) = 1$$

Exercise 5.3

```
int C(int k, int n) {
        if (k == 0 || k == n) return 1;
        if (k == 1) return n;
        return C(k - 1, n - 1) + C(k, n - 1);
}
```

$$C_n^k + C_n^{k+1} = C_{n+1}^{k+1}$$

$$C_n^0 = C_n^n = 1$$

$$C_n^1 = C_n^{n-1} = n$$

CANTHO UNIVERSITY

Q&A

www.ctu.edu.vn