COP 3502 – Computer Science 1

Lecture 07

Dr. Sahar Hooshmand
sahar.hooshmand@ucf.edu

Department of Computer Science

Slides modified from Dr. Ahmed, with permission

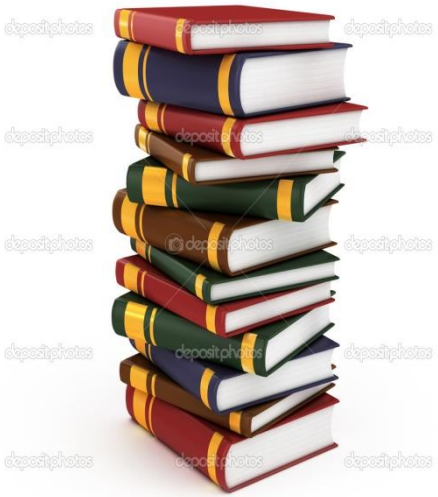
Stack

Content

- Stack
  - Applications of Stack
  - Stack Operations and Implementation
  - Examples and Simulation
  - Expression Evaluation, Infix to Postfix

What is Stack?

- Stack is another Abstract Data typoe
- A stack is a Last In, First Out (LIFO) ADT
- Anything added goes on the top
- Anything removed from the stack is taken from the “top”
- Things are removed in the reverse order from that in which they were inserted
- Analogy:
  - Stack of plates
  - Which one will you remove first?
  - Another example pile of books
  - Can you think other examples?



Why stack?

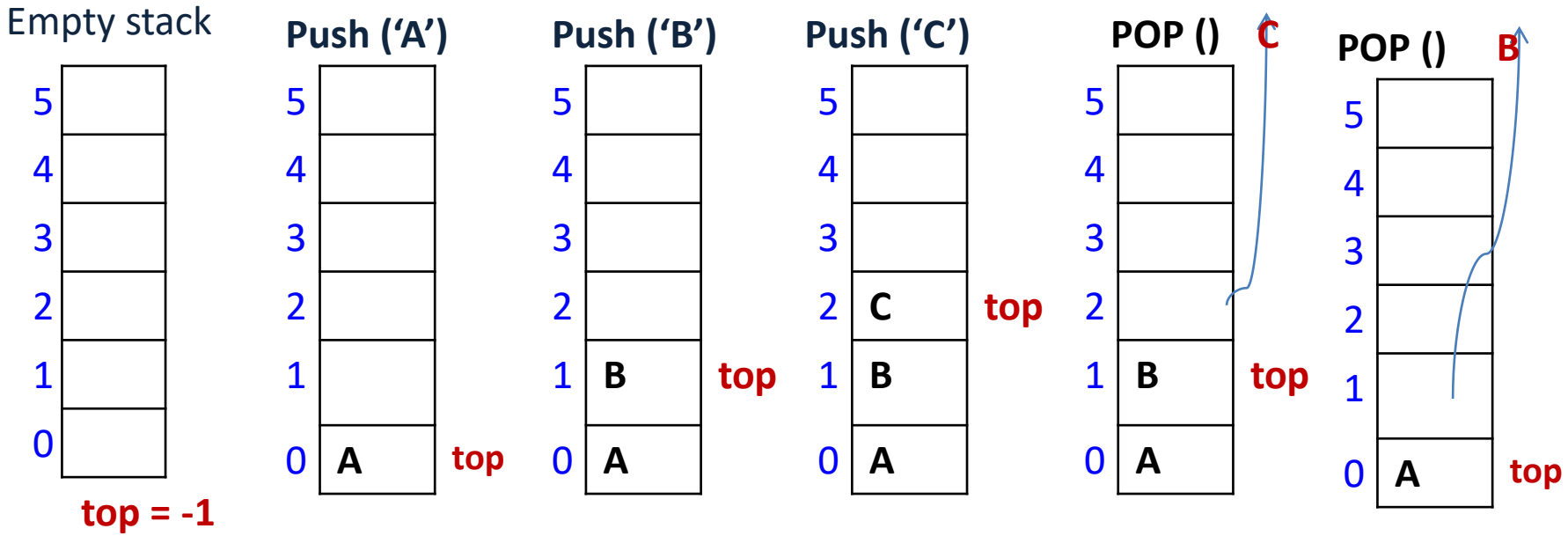
- Variety of applications of stack:
  - Program execution (e.g., function call, recursion)
    - F1() calls F2(), then F2() calls F3(). Where F3() will return?
  - Evaluating expressions (e.g., a / b - c + d * e - a * c)
  - Undo operation (back button of your browser?)
  - Searching path in a graph
  - Validating the balance of parenthesis (we will see example)

Stack operations

- Basic stack operations
  - **Push :** Add a new item
  - **Pop :** get and remove the item that was added most recently
  - **IsEmpty:** Determine whether a stack is empty
  - **IsFull:** Determine whether a stack is full
  - **Peek:** Get the most recently added item without deleting it

Example of stack operations

- Max_Size = 6, Stack[Max_Size], top = -1
- Push(char x): Stack[++top] = x,
- POP(): return Stack[top--], Peek(): return Stack[top]



- Discussion: POP(), POP()
- Stack Overflow? Or Stack Underflow?

Stack Implementations

- There can be various ways stacks can be implemented.
  - Using global top variables and global array (not the best solution as it is difficult to implement multiple stacks. Also, global variables should be avoided, if possible) //we will go through this in the class
  - Using structure and array and variables within the structure (we will go through this in the class)
    - Then use local variable of structure, modify push, pop functions to take the reference of the structure variable so that it will work on the passed stack
    - This can facilitate creating many stacks
  - Using linked list (you should do it at home)
    - Very easy as you just need to insert at the head while pushing, and remove from the head while popping. (Both of these operations are the easiest operation of insertion and deletion in linked list)

Example Stack Implementation- global variable

```
void push(int x)
 {
     if( top >= Max_Size-1)
        printf("Stack Overflow“);
    else
   {
       Stack[++top] = x;
       printf(“Inserted“);
   }
}
```

```
int pop()
{
    if(top < 0)
   {
       printf("Stack Underflow“);
       return -9999;
    }
    else
       return Stack[top--];
}
```

```
int isEmpty()
{
    if(top < 0)
   {
       printf( "Stack is empty“);
       return 1;
   }
   else
   {
       printf( "Stack is not empty“);
       return 0;
    }
}
```

```
int isFull()
{
      what do you
       think?
}
```

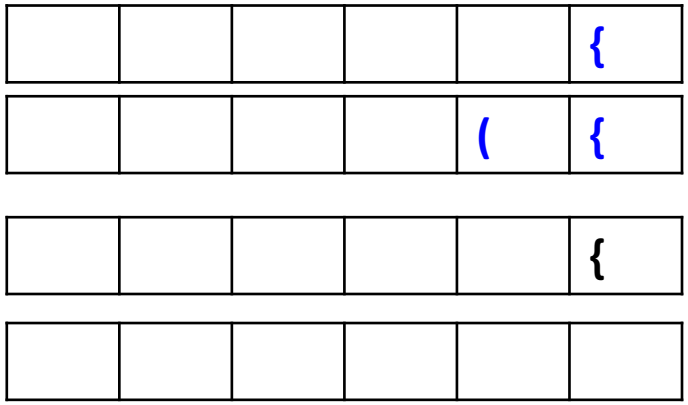
```
int isFull()
{
    if ( top >= Max_Size-1)
   {
          printf(“Full”);
          return 1;
    }
    else
    {
          [          ]
          return 0;
}
```

Checking the balance of parenthesis

- Is it balanced?: ({ }[ ])
- How about this one?: (({ } [( ))])
- Simple counting is not enough to check balance
- You can do it with a stack.
- Read the string char by char left to right,
  - If you see a (, [, or {, push it on the stack
  - If you see a),], or }, pop the stack and check whether you got the corresponding (, [, or {
  - When you reach the end, check that the stack is empty. If it is empty, it is balanced.

Simulation

- Input: { ( ) }
- Read {, so push (‘ {‘)
- Read (, so push (‘(‘).
- Read), so pop. popped item is ‘(‘ which matches).
- Read }, so pop; popped item is { which matches }.
- End of string; stack is empty, so the string is valid.



Activity: Using Stack, check whether the following is balanced or not : “{ ( ) }) ”

-Assume a stack of Max_Size = 6

-Draw the stack for each step and show the value of “top”

-If any decision is made (valid/invalid), then write the reason

Exercise

Write a program that will take an expression with parenthesis as input and return whether the parenthesis in the expression is balanced or not.

- Use stack for the solution.

Expression Evaluation

- Infix Expression:
  - **A + B * C**
  - **(A + B) *C – D * E**
  - Easily understandable by human
  - Hard to parse in a computer program and difficult to evaluate by a program
- Postfix:
  - **Operators** come after the **operand**
  - ***A + B → A B +***
  - **A + B * C → *A B C * +***
  - Operators are ALWAYS in correct evaluation order.

Infix to Postfix

- Consider we have two elements:
  1. An empty expression string, called Postfix
  2. An empty operator stack

Converting Infix Exp. to Postfix Exp.

- Main Steps (Assuming the expression is valid):
- Start reading the Infix expression from left to right
- **If we encounter an operand** we will append it to the Postfix string, **[Operand doesn't go to stack]**
- **If we encounter an operator let's say current operator:**
- The current operator is compared to the operator that is at the top of the stack
- If the priority of the current operator is **HIGHER** than the priority of the operator at the top of the stack, then the current operator is pushed at the top of the stack.
- If the priority of the current operator is **same or lower** than the priority of the operator at the top of the stack, we keep POPing operators from the stack until the priority of the current operator is higher than the operator at the top of the stack. POPed operators are appended to the Postfix in the order they are POPed. **Finally,** the current operator is PUSHED to Stack.
- **If we encounter an open parenthesis:** immediately pushed in the stack
- **If we encounter a closing parenthesis:** POP until an open parenthesis (inclusive) found in the stack. Append the popped items to the Postfix except the parenthesis.
- ***Priority of the operators is as follows (high to low):***
  - ***1. power ^***
  - ***2. * / %***
  - ***3. + -***