SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-IOT Including CS&BCT

COURSE NAME : 19CS304 - DATA STRUCTURES

II YEAR / III SEMESTER

Unit 1- LINEAR STRUCTURES AND TREES

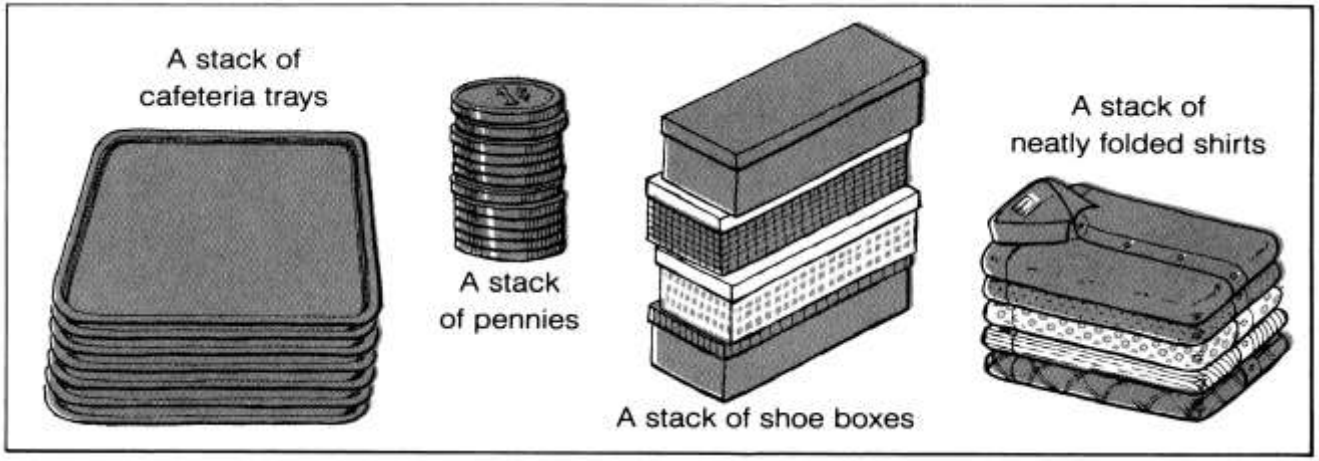
Topic 7 : Stack Based Implementation

Problem

- There are many real-life examples of a stack. Consider an example of plates stacked over one another in the canteen.
- The plate which is at the top is the first one to be removed, i.e. the plate which has been placed at the bottommost position remains in the stack for the longest period of time.

Stack Based Implementation

- It is an ordered group of homogeneous items of elements.
- Elements are added to and removed from the top of the stack (the most recently added items are at the top of the stack).
- The last element to be added is the first to be removed (**LIFO**: Last In, First Out).



Stack Based Implementation –Cont..

- Stack is a linear data structure which follows a particular order in which the operations are performed.
- The order may be LIFO(Last In First Out) or FILO(First In Last Out)

OR

- **What is Stack?**
- *Stack* is a **LIFO (Last In First Out)** data structure. It allows us to insert and remove an element in special order.
- Stack allows and removal stack.

Stack Based Implementation–Cont..

- Mainly the following three basic operations are performed in the stack:
- **Push:** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **Pop:** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
- **Peek or Top:** Returns top element of stack.
- **isEmpty:** Returns true if stack is empty, else false.
- **Time Complexities of operations on stack:**
- push(), pop(), isEmpty() and peek() all take O(1) time. We do not run any loop in any of these operations.

Stack Based Implementation

- **How to create stack data structure using array?**
- **Example:**
- int stack[SIZE]; // syntax //
- The above code creates a stack of integer. Where SIZE is the capacity of stack.
- we will need a variable to keep track of top element. Let, us declare a variable top to keep track of top element.
- int top = -1; // syntax //
- In the above code I have initialized top with -1. Which means there are no elements in stack.

Stack Based Implementation –Cont..

- **How to push elements to stack?**
- Inserting/Adding a new element to stack is referred as *Push operation* in stack.
- Step by step descriptive logic to push element to stack.
- If stack is out of capacity i.e. top >= SIZE,
- then throw "Stack Overflow" error. Otherwise move to below step.
- Increment top by one and push new element to stack using stack[++top] = data;
- (where data is the new element).

Stack Based Implementation –Cont..

- **How to pop elements from stack?**
- Removing an element from stack is referred as *Pop operation* in stack.
- Step by step descriptive logic to pop element from stack.
- If top < 0, then throw "Stack is Empty" error. Otherwise move down to below step.
- Return the top most element from stack and decrement top by one. Say return stack[top--];.

Stack Based Implementation –Cont..

- **How to find size of stack?**
- Finding size of stack is straightforward. Size of stack is represented by top. Since we have initialized top = -1;
- hence size of stack will be top + 1.

Applications of stack

1. Balancing of symbols
2. Infix to Postfix /Prefix conversion
3. Redo-undo features at many places like editors, photoshop.
4. Forward and backward feature in web browsers
5. Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.

Stack Creation and Initializes using C –Cont..

```
struct Stack {
    int top;
    unsigned capacity;
    int* array;
};
```

```
// function to create a stack of given capacity. It initializes size of
// stack as 0
struct Stack* createStack(unsigned capacity)
{
struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
        stack->array = (int*)malloc(stack->capacity * sizeof(int));
    return stack;
}
```

Stack Based Implementation –Cont..

```
/ Stack is full when top is equal to the last index
int isFull(struct Stack* stack)
{
   return stack->top == stack->capacity - 1;
}

// Stack is empty when top is equal to -1
int isEmpty(struct Stack* stack)
{
   return stack->top == -1;
}
```

Stack Based Implementation –Cont..

```
// Function to add an item to stack.  It increases top by 1
void push(struct Stack* stack, int item)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = item;
    printf("%d pushed to stack\n", item);
}

// Function to remove an item from stack.  It decreases top by 1
int pop(struct Stack* stack)
{
    if (isEmpty(stack))
        return INT_MIN;
    return stack->array[stack->top--];
}
```

Stack Based Implementation –Cont..

```
// Function to return the top from stack without removing it
int peek(struct Stack* stack)
{
   if (isEmpty(stack))
      return INT_MIN;
   return stack->array[stack->top];
}
 // Driver program to test above functions
int main()
{
   struct Stack* stack = createStack(100);

   push(stack, 10);
   push(stack, 20);
   push(stack, 30);

   printf("%d popped from stack\n", pop(stack));
    return 0; }
```

Activity

MCQ

1. Process of inserting an element in stack is called _____

   a)Create

   b)Push

   c)Evaluation

   d)Pop

2. Process of removing an element from stack is called_____

   a)Create

   b)Push

   c)Evaluation

   d)Pod

Advantages

1. Helps you to manage the data in a Last In First Out(LIFO) method which is not possible with Linked list and array.
2. When a function is called the local variables are stored in a stack, and it is automatically destroyed once returned.
3. A stack is used when a variable is not used outside that function.
4. It allows you to control how memory is allocated and deallocated.
5. Stack automatically cleans up the object.
6. Not easily corrupted
7. Variables cannot be resized.

Disadvantages

1. Stack memory is very limited.
2. Creating too many objects on the stack can increase the risk of stack overflow.
3. Random access is not possible.
4. Variable storage will be overwritten, which sometimes leads to undefined behavior of the function or program.
5. The stack will fall outside of the memory area, which might lead to an abnormal termination.

Assessment 1

1. List out the advantages of Stack based implementation

a)_____

b)_____

c)_____

d)_____

2. Identify the disadvantages of Stack based implementation

a)_____

b)_____

c)_____

d)_____

REFERENCES

1. M. A. Weiss, “Data Structures and Algorithm Analysis in C”, Pearson Education, 8th Edition, 2007. [Unit I, II, III, IV,V]

2. A. V. Aho, J. E. Hopcroft and J. D. Ullman, “Data Structures and Algorithms”, Pearson Education, 2nd Edition, 2007 [Unit IV].

3. A.M.Tenenbaum, Y. Langsam and M. J. Augenstein, “Data Structures using C”,PearsonEducation, 1st Edition, 2003.(UNIT I,II,V)

4.https://www.geeksforgeeks.org/stack-data-structure-introduction-program/

THANK YOU