DATA STRUCTURES

Stacks

Prof. Khalil Al-Shqeerat

CS Dept. – Qassim University

OUTLINE

1. Stack definition
2. Stack as an Array
3. Stack as a Linked List

الـ Stack هو هيكل بيانات (Data Structure)
تخيّليه مثل رصّة صحون:
تحط صحن فوق = إضافة عنصر
تشيل الصحن اللي فوق = حذف عنصر
📌 المهم: كل العمليات تصير من الأعلى فقط (Top)

STACK DEFINITION

- Stack is a data structure in which addition of new element or deletion of an existing element always takes place at the top of stack.
- When an item is added to a stack, the operation is called **push**, and when an item is removed from the stack the operation is called **pop**.
- Stack is also called last-in-first-out (LIFO) list.
- A stack data structure can be maintained as an array or as a linked list.

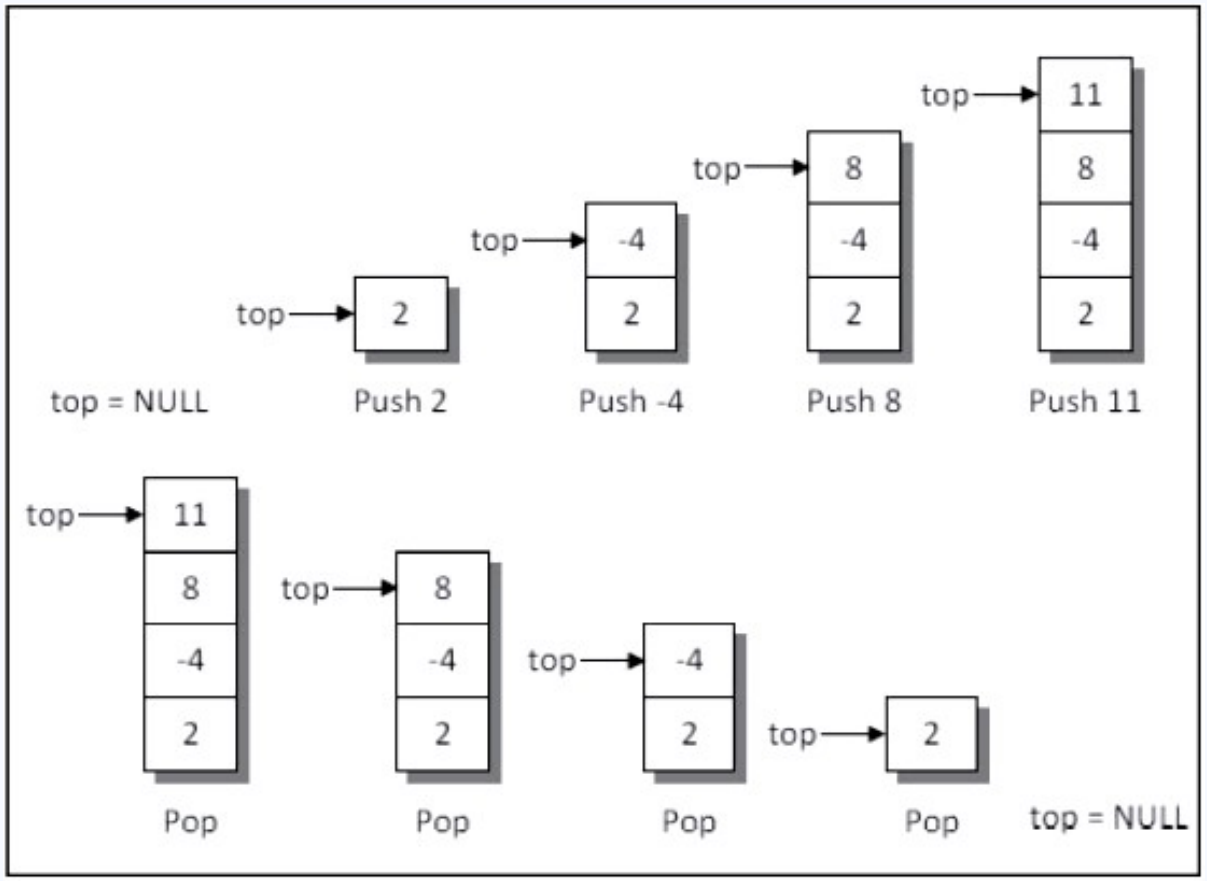
Push = Insert or add

Pop = Remove or delete

Last In, First Out
يعني:
آخر واحد دخل = أول واحد يطلع
مثال سريع:
Push A
Push B
Push C

Pop → C
Pop → B
Pop → A

INSERTION AND DELETION OF ELEMENTS IN STACK



STACK AS AN ARRAY

```
#include
using namespace std;
const int MAX = 10;
class stack
{
      private:
            int arr[MAX];
            int top;
      public:
            stack();
            void push (int item);
            int pop();

};
```

أقصى حجم للستاك = 10 عناصر

هذا هي المصفوفة اللي تخزن العناصر
حجمها 10

هذا أهم متغير
يمثل مؤشر (Index) لآخر عنصر في الستاك

عندنا 3 دوال:
Constructor → لإنشاء الستاك
push → لإضافة عنصر
pop → لحذف عنصر

```
// initializing a data member
stack:: stack()
{
top = -1;
}
```

لصفوفة تبدأ من index = 0
لا يكون top = -1 → يعني مافي أي عنصر

- The actual storage of stack elements is done in an array.
- The data member top is an index into array.
- To indicate that the stack is empty to begin with, the variable top is set with a value -1 in the constructor function.

ADDING AN ELEMENT TO THE STACK

```
void stack:: push (int item)
{
    if (top == MAX - 1)
    {
        cout << “Stack is full” << endl;
        return;
    }
    top++;
    arr[top] = item;
}
```

الـStack مليان، مانقدر نضيف زيادة

اذا لم يتحقق الشرط (نزيد قيمة التوب بحيث تساوي آخر عنصر تم اضافته)
ثم نضع القيمة في مكان التوب

- Every time an element is added to stack, it is verified whether such an addition is possible at all.
- If it is not, then the message ‘Stack is full’ is displayed.

EXTRACTING AN ELEMENT FROM THE STACK

```
int stack:: pop()
{
    if (top == -1)
    {
        cout << “Stack is empty” << endl;
        return NULL;
    }
    int data = arr[top];
    Arr[top] = 0;
    top--;
    return data;
}
```

هذا يعني ان الـStack فارغ
مافي عناصر عشان نطلعه

اذا لم يتحقق الشرط
نضع مكان آخر عنصر 0
ثم ننقص من قيمة التوب بحيث يساوي العنصر اللي قبله
ثم ترجع القيمة (طلعناه من الستاك) اخذنا قيمته وصفر القيمة داخل الستاك

- When we call **pop()** for the time, there are no elements present in the stack and top has a value -1 in it. Hence the ‘Stack is empty’ gets displayed

STACK AS A LINKED LIST

- Limitation of an array
  - its size cannot be increased or decreased once it is declared
  - Reserving either too much memory or too less memory for an array and in turn for a stack.
- This problem can be overcome if we implement a stack using a linked list.
- The pointer to the beginning of the list serves the purpose of the top of the stack

في الستاك باستخدام Array:

❌ الحجم ثابت:

```
int arr[10];
```

يعني:

لو احتجنا أكثر من 10 → ❌ ما نقدر

لو استخدمنا 3 فقط → ❌ ضيعنا ذاكرة

الحل: Linked List

بدل ما نحجز حجم ثابت، نستخدم Linked List

✔ الحجم يصير ديناميكي (يكبر ويصغر)

أول نود (Node) في الليست = Top of Stack

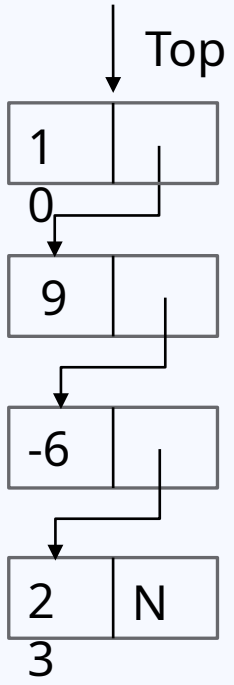
🔷 شكل الستاك:

Top → [30] → [20] → [10] → NULL

🔷 30 هو آخر عنصر دخل → صار في الأعلى

🔷 10 هو أول عنصر دخل → في الأسفل

STACK AS A LINKED LIST



IMPLEMENTATION STACK AS A LINKED LIST

```
#include
using namespace std;
class stack
{
    private:
        struct node
        {
            int data;
            node *link;
        } *top;
```

كل Node تحتوي:
data → القيمة
link → مؤشر للنود اللي بعدها

```
    public:
        stack();
        void push (int item);
        int pop();
        ~stack();
};
```

ننشئ نود جديدة فارغة
نضيف نود جديدة في البداية
نحذف نود موجودة
هذا يحذف كل العناصر من الذاكرة (عشان ما يصير Memory Leak)

```
// initializing a data member
stack:: stack()
{
top = NULL;
}
```

الستاك فاضي
ما فيه أي نود

- **top** pointer is set to NULL in the constructor to indicate that the stack is empty

ADDING A NEW NODE TO THE STACK

```
void stack:: push (int item){
//add node at the beginning
      node *temp;
      temp = new node;
      if (temp == NULL){
            cout << “Stack is full” << endl;
            return;}
      temp -> data = item;
      temp -> link = top;
      top = temp;
}
```

الشرح:

إنشاء نود جديدة:

```
node* newNode = new node;
```

نحط القيمة:

```
newNode->data = item;
```

نربطها باللي قبل:

```
newNode->link = top;
```

نحرك الـ top:

```
top = newNode;
```

- In every call to **push()** function a new node is created dynamically.
- As long as there is enough memory available for dynamic allocation temp would never become full.
- If value of temp happens to be NULL then that would be a stage where stack would become full.

- After, creating a new node, the data member **top** should point to the newly created item of the list. Hence we have assigned the address of this new node to **top**.

POP AN ELEMENT FROM THE STACK

```
int stack:: pop()
{
//Remove the first element in list
      if (top == NULL)
      {
            cout << “Stack is empty” << endl;
            return NULL;
      }
      node *temp;
      int item;
      temp = top;
      item = temp -> data;
      top = top -> link;
      delete temp;
      return item;
}
```

الشرح:

تحقق إذا فاضي:

top == NULL

نحفظ النود:

node* temp = top;

نأخذ القيمة:

int data = temp->data;

نحرك المؤشر:

top = top->link;

نحذف النود:

delete temp;

نرجع القيمة:

return data

- **pop()** function first checks whether or not a stack is empty.
  - If so, then a message ‘Stack is empty’ gets displayed.
  - If the stack is not empty then the topmost item gets removed from the list.

DEALLOCATING MEMORY

```
stack:: ~stack()
{
      if (top == NULL)
            return;
      node *temp;
      while (top != NULL)
      {
      temp = top;
      top = top -> link;
      delete temp;
      }
}
```

🔷 1. التحقق إذا الستاك فاضي

```
if (top == NULL)
return;
```

📌 إذا ما فيه عناصر:

ما نسوي شيء

نخرج مباشرة

🔷 2. تعريف مؤقت

```
node *temp;
```

📌 نستخدمه عشان:

نحفظ النود الحالية قبل حذفها

🔷 3. الحلقة (حذف كل العناصر)

```
while (top != NULL)
```

📌 نستمر إلى أن نحذف كل النودات

🔷 داخل الحلقة:

1. نحفظ النود الحالية

```
temp = top;
```

2. نحرك الـ top للنود التالية

```
top = top -> link;
```

📌 كأننا نقول:

انتقل للنود اللي بعدها قبل ما نحذف الحالية

3. نحذف النود القديمة

```
delete temp;
```

📌 تحرير الذاكرة