Comparison between the array-based and the linked implementation: "Which is **always** better?" is a wrong question!

Array-based implementation	Linked implementation

```
typedef struct stack{
        int top;
        StackEntry entry[MAXSTACK];
} Stack;
```

-All the space is reserved even the stack is empty (wasting memory)

-Stack gets full even if the memory is not!

```
typedef struct stacknode{
    StackEntry entry;
    struct stacknode *next;
}StackNode;

typedef struct stack{
   StackNode *top;
   int  size;
}Stack;
```

-Extra field `next` for every new node.

-The stack size is as large as the memory you have!

Comparison between the array-based and the linked implementation: "Which is **always** better?" is a wrong question!

	Array-based implementation	Linked implementation
`Pop`	?(1)	?(1)
`Push`	?(1)	?(1)
`CreateStack`	?(1)	?(1)
`StackSize`	?(1)	?(1)
`TraverseSack`	?(N)	?(N)
`ClearStack`	?(1)	?(N)

Then, there are advantages and disadvantages for every implementation. Which one is better **really** depends on the application.E.g.,

If `ClearStack` is extensively used then, may be array-based implementation is better.

If the memory is very limited, then may be the linked implementation is better.

The rule is: Know very well the pros and cons of each implementation and decide based on your application needs.