GeeksQuiz

Computer science mock tests for geeks

Search

- Home
- Latest Questions
- Articles
- C/C++ Programs
- Contribute
- Books

Subscribe

Stack | Set 1 (Introduction)

March 7, 2013

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).

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: Get the topmost item.

How to understand a stack practically?
There are many real life examples of stack. Consider the simple example of plates stacked over one another in 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. So, it can be simply seen to follow LIFO/FILO order.

Implementation:
There are two ways to implement a stack:
Using array
Using linked list

Using array:

```
// C program for array implementation of stack
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
```

```
// A structure to represent a stack
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 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;  }

// 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--];
}

// Function to get top item from stack
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));

    printf("Top item is %d\n", peek(stack));

    return 0;
}
```

Pros: Easy to implement. Memory is saved as pointers are not involved.
Cons: It is not dynamic. It doesn't grow and shrink depending on needs at runtime.

```
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
Top item is 20
```

Linked List Implementation:

```
// C program for linked list implementation of stack
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// A structure to represent a stack
struct StackNode
{
    int data;
    struct StackNode* next;
};

struct StackNode* newNode(int data)
{
    struct StackNode* stackNode =
              (struct StackNode*) malloc(sizeof(struct StackNode));
    stackNode->data = data;
    stackNode->next = NULL;
    return stackNode;
}

int isEmpty(struct StackNode *root)
{
    return !root;
}
```

```
void push(struct StackNode** root, int data)
{
    struct StackNode* stackNode = newNode(data);
    stackNode->next = *root;
    *root = stackNode;
    printf("%d pushed to stack\n", data);
}

int pop(struct StackNode** root)
{
    if (isEmpty(*root))
        return INT_MIN;
    struct StackNode* temp = *root;
    *root = (*root)->next;
    int popped = temp->data;
    free(temp);

    return popped;
}

int peek(struct StackNode* root)
{
    if (isEmpty(root))
        return INT_MIN;
    return root->data;
}

int main()
{
    struct StackNode* root = NULL;

    push(&root, 10);
    push(&root, 20);
    push(&root, 30);

    printf("%d popped from stack\n", pop(&root));

    printf("Top element is %d\n", peek(root));

    return 0;
}
```

Output:

```
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
Top element is 20
```

Pros: The linked list implementation of stack can grow and shrink according to the needs at runtime.
Cons: Requires extra memory due to involvement of pointers.

Applications of stack:

Balancing of symbols:
Infix to Postfix/Prefix conversion
Redo-undo features at many places like editors, photoshop.
Forward and backward feature in web browsers
Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem.
Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver

We will cover the implementation of applications of stack in separate posts.

Quiz: Stack Questions

References:
http://en.wikipedia.org/wiki/Stack_%28abstract_data_type%29#Problem_Description

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Questions:

- Binary Heap
- Delete all occurrences of a given key in a linked list
- How to create mergable stack?
- Deque | Set 1 (Introduction and Applications)
- A data structure for n elements and O(1) operations
- Convert left-right representation of a bianry tree to down-right
- Print level order traversal line by line
- C Program for Red Black Tree Insertion

Anindya Srivastava • 9 months ago

Quiz: Stack Questions should lead to:
http://geeksquiz.com/data-stru...
and not
http://geeksquiz.com/stack/

2 ▲ | ▼ • Reply • Share ›

guest • a month ago

do'nt we have to allocate memory to root pointer also. struct StackNode* root= (struct StackNode*) malloc(sizeof(struct StackNode)); before using it in pop function

▲ | ▼ • Reply • Share ›

Phagocyte • 2 months ago

Can somebody explain what "return !root" does? Thanks.

▲ | ▼ • Reply • Share ›

irresolute ➦ Phagocyte • 2 months ago

as you might be knowing that the integral equivalence of "NULL" is "0" so when the root is set to NULL then the "return !root" can be visualized as "return !0" i.e. any non zero value will be returned which is also the value of boolean "true" ...and if its false then "0" will be returned ...

▲ | ▼ • Reply • Share ›

Manish Kumar • 2 months ago

Can someone explain the need for
struct StackNode* stackNode = (struct StackNode*) malloc(sizeof(struct StackNode));

▲ | ▼ • Reply • Share ›

irresolute ➦ Manish Kumar • 2 months ago

go through the linked lists . once again.

▲ | ▼ • Reply • Share ›

Sagar • 3 months ago

what is the advantage of using the createStack() in array implementation and NewNode() in LL implementation, when we can do this simply in main() ?

▲ | ▼ • Reply • Share ›

Thiago Fambrought • 3 months ago

why the top of stack is initialized with -1 and not with zero ?

▲ | ▼ • Reply • Share ›

Sagar ➦ Thiago Fambrought • 3 months ago

Because initially there are no element in stack. and when we push a element in the stack, the top is initialized with 0(or 1 is added). and so this way we can directly refer to it (top element) using 0(top) as index.

• Reply • Share ›

Anil Kumar Gurindapalli • 5 months ago

1. Consider an initially empty stack S. A program loops through an array of 10 integers in the order a[0], a[1], ..., a[9], and for each i, it pushes a[i] onto the stack S if i is even, and it sets a[i] equal to the value popped from the stack S if i is odd. After the program is complete, what can we say about the array?
A. the array is unchanged B. the elements of the array are reversed C. a[i] == a[i+1] for each even value of i D. None of the above

• Reply • Share ›

arjun_gowm • 5 months ago

Shall any one explain the difference between these two or they are same??
1)struct Stack* stack
2)struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack));

• Reply • Share ›

Rajat Kumar Seth ➚ arjun_gowm • 5 months ago

1) struct Stack *stack;
This line means that stack is a poiter variable which is struct Stack type..

Means the syntax
is-
<datatype> *<pointer variable="">;

This line only defined a pointer but didn't allocated the Memory..

2) it's correct..
Poiter also defined and Memory also allocated.. :)

• Reply • Share ›

arjun_gowm ➚ Rajat Kumar Seth • a month ago

Thanks for ur reply..

• Reply • Share ›

arjun_gowm ➚ Rajat Kumar Seth • 5 months ago

Thanks **@Rajat Kumar Seth** ,But in above program if I modified struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack)); in struct Stack* createStack module to struct Stack *stack; it runs perfectly .So How it happens without memory being allocated ?

• Reply • Share ›

Rajat Kumar Seth ➚ arjun_gowm • 5 months ago

Your Option '1)' will require another line to allocate the memory..

i told that

1) struct Stack *stack;

will define a pointer but will not allocate the memory..

so you have to use another line to allocate memory..

So, 1) will Become:--

1) struct Stack *stack;

stack=(struct Stack*)malloc(sizeof(struct Stack));

▲ ▼ • Reply • Share ›

arjun_gowm ➚ Rajat Kumar Seth • 5 months ago

Once again Thanks Rajat Kumar Seth , I haven't use the dynamic memory allcation method, just
deleted (struct Stack*)malloc(sizeof(struct Stack));
in struct Stack* stack = (struct Stack*) malloc(sizeof(struct Stack)); in createstack module in the implementaion using array program given above,but it runs perfectly ???Thanks in advance..

▲ ▼ • Reply • Share ›

Rajat Kumar Seth ➚ arjun_gowm • 5 months ago

It works..
But the Program is implemented wrong.. Coz either we shoud use fully array.. Means stack should also be defined as array.. Like-
MAXSIZE=50;
stack[MAXSIZE]; //All these are global Declaration
peek=-1;

Then We would have created stack
as-

```
void inss(int n)
{
if(peek==MAXSIZE-1)
{
printf("Overflow");
exit(0);
}
peek++;
```

```
stack[peek]=n;
}
```

• Reply • Share ›

arjun_gowm ➚ Rajat Kumar Seth • 5 months ago

Thanks **@Rajat Kumar Seth** , sorry I didn't get it.Is the above program implemented in a wrong way?Please elaborate me..

• Reply • Share ›

sumantmann ➚ arjun_gowm • 2 months ago

hey arjun! , do you remember in linked list we created new nodes as
struct linklist *newnode;
and for allocating memory to them we did
newnode=(struct linklist*)malloc(sizeof(struct linklist));
here we are using
Struct Stack *stack=(struct Stack*)malloc(sizeof(struct Stack));
because if in future if we require to create more than one stack we can easily do that.hope you understood it.

• Reply • Share ›

arjun_gowm ➚ sumantmann • a month ago

Got it @sumantmann ,Thanks for ur reply...

• Reply • Share ›

Meet • 5 months ago

@GeeksforGeeks

Program can be written without using structure also to implement stack using array. You have made program complex. Will you please tell me why you have used structure to implement the program?

• Reply • Share ›

Ankit Marothi ➚ Meet • 5 months ago

That must be to make the program modular for bigger use.

• Reply • Share ›

Meet • 6 months ago

Program can be written without using structure also to implement stack using array. You have made program complex. Will you please tell me why you have used structure to implement the program?

• Reply • Share ›

mithun • 6 months ago

can anybody explain...."unsigned capacity".....it should be ny data type like...int..flot...

• Reply • Share ›

Kartik Nagpal ➚ mithun • 5 months ago

unsigned capacity; // it is shorthand for unsigned int capacity

unsigned int is 4*sizeof(char), i.e. 4 bytes and can take up values in range [0, (2 ^ 32) -1]

Note: unsigned really is a shorthand for unsigned int, and so defined in standard C.

• Reply • Share ›

Ankit Marothi ➚ Kartik Nagpal • 5 months ago

Usually, signed bits are kept as the MSB(most significant bit) of any data type, this means that the data can be either positive or negative. When you are sure that the data is going to be positive and you need a larger storage like in the example integer. You can use unsigned. Like for characters if you need one byte, the typical range goes from -256 to 255(I am not sure if the number is exactly correct) so you can store only 255 characters if you are going to store only positive values, if you know before hand that only positive characters are going to be stored you can used unsigned so the -256 will be added to 255, so now your range of positive characters increases to 255 + 256. Here, since capacity can only be positive the author has used unsigned.

• Reply • Share ›

Kartik Nagpal ➚ Ankit Marothi • 5 months ago

Yes, indeed this roughly explains difference btw signed and unsigned values. But the range you specified that can be represented with a single byte is mathematically incorrect.

Let, k = 1 byte = 8 bits
signed range -> [-(2^(k-1)), 2^(k-1) - 1] -> [-128, 127]
unsigned range -> [0, (2^k)-1] -> [0, 255]

And coming down to bits itself, its not just about the MSB, 2's complement is the standard way to represent signed values, that is used by almost every computer manufactured today.

2's Complement:
The 2's complement data type specifies the representation for each negative integer so that when the ALU adds it to the representation of the positive integer of the same magnitude, the result will be representation for 0. Eg.,
Since 01101 is the representation of +13, 10011 is chosen as representation for -13.
Such that, 01101 + 10011 = 100000 = 00000 (ignoring carry-out)

• Reply • Share ›

amateur • 8 months ago

can someone please explain INT_MIN which is returned if stack is empty in pop operation?

• Reply • Share ›

GeeksforGeeks Mod ➚ amateur • 8 months ago

INT_MIN is typically used to indicate minus infinite. It is a macro defined in limits.h.

5 • Reply • Share ›

Ankit Marothi ➚ amateur • 5 months ago

INT_MIN is the lowest limit that can be assigned to a integer. The header file limits.h contains the limits for primitive data types for C. So, it contains the lowest limit for the integer values. In a, linux system you can find this file in the location /usr/local/include/limits.h (generally)

1 • Reply • Share ›

amateur ➚ Ankit Marothi • 5 months ago

ohk, thank you! :)

• Reply • Share ›

Guest • 8 months ago

<script src="http://ideone.com/e.js/yNJ09l" type="text/javascript"></script>

• Reply • Share ›

kannan • 8 months ago

hi

• Reply • Share ›

- # Categories

  - Articles (124)
    - Algorithms (24)
    - C (16)
    - C++ (17)
    - Data Structures (29)
    - DBMS (1)
    - Interview Experiences (6)

- Java (2)
- Operating Systems (1)
- Puzzle (12)
- Searching and Sorting (10)

- Programs (35)
- Quizzes (2,106)
  - Aptitude (2)
  - Computer Science Quizzes (2,103)
    - Algorithms (147)
    - C (207)
    - C++ (129)
    - Computer Organization and Architecture (1)
    - Data Structures (132)
    - DBMS (2)
    - GATE (1,406)
    - Java (51)
    - Operating Systems (28)
  - Web technologies (1)

Recent Discussions

- Vidhi

  Process P2 will start it's last I/O operation...

  Operating Systems | Process Synchronization | Question 4 · 1 day ago

- xerosanyam

  This article is slightly less clear in one...

  A Programmer's approach of looking at Array vs. Linked List · 1 day ago

- Aditya Goel

  Please include Heap overflow condition as well....

  Queue | Set 2 (Linked List Implementation) · 1 day ago

- Aditya Goel

  We can initialize rear as -1 with no change in...

  Queue | Set 1(Introduction and Array Implementation) · 1 day ago

- Sg

  Implementing this question leaves an extra...

  GATE | GATE-CS-2015 (Set 3) | Question 22 · 2 days ago

- jyuan92

  What if I want to handle the situation that,...

  Remove comments from a given C/C++ program · 2 days ago

-