Δομές Δεδομένων

Υλοποίηση Στοίβας σε γλώσσα C

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

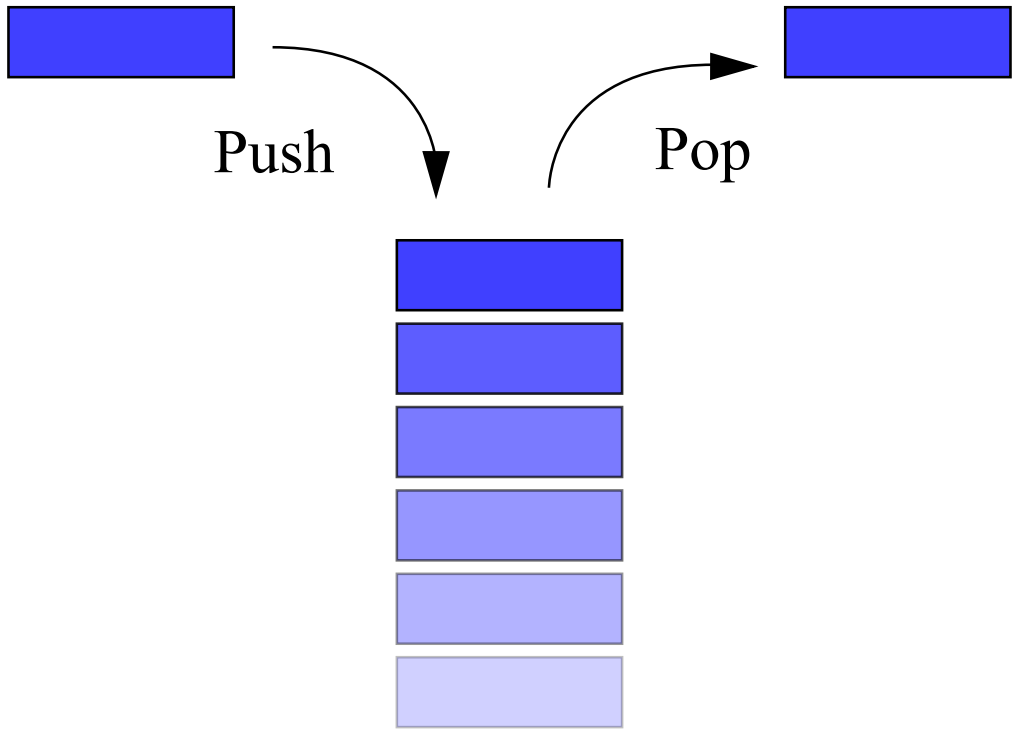
Στοίβα
ως ΑΤΔ

Μία στοίβα ώθησης προς τα κάτω (pushdown stack) είναι ένας ΑΤΔ που περιλαμβάνει τις εξής λειτουργίες:

- Εισαγωγή (ώθηση) ενός νέου στοιχείου, PUSH(S,x).
- Διαγραφή (απώθηση, pop) του τελευταίου στοιχείου που προστέθηκε στη στοίβα, POP(S).
- Έλεγχος κενής στοίβας, EMPTY(S).
- Επιστροφή μεγέθους στοίβας, SIZE(S).
- κ.τ.λ

Στοίβα

Η στοίβα ακολουθεί την αρχή **τελευταίο μέσα, πρώτο έξω** ή αλλιώς LIFO (last-in, first-out)



Αφού ορίζουμε μία στοίβα ως ΑΤΔ δεν μας ενδιαφέρει η ακριβής υλοποίηση της στοίβας. Μπορούμε για παράδειγμα να υλοποιήσουμε μια στοίβα με λίστα αλλά και με πίνακα.

Στοίβα

Μερικές προϋποθέσεις:

- Ξεχωριστή υλοποίηση με διασύνδεση
- Οι χρήστες δεν επιτρέπεται να πειράζουν την εσωτερική αναπάσταση
- Οι χρήστες πρέπει να μπορούν να φτιάξουν πολλές στοίβες ταυτόχρονα

Στοίβα

διασύνδεση σε C (stack.h)

```c
#ifndef _STACK_H
#define _STACK_H

typedef int item_type;
typedef struct _stack* stack;

stack stack_create();
void stack_destroy(stack s);

void stack_push(stack s, item_type elem);
void stack_pop(stack s);
item_type  stack_top(stack s);

int  stack_is_empty(stack s);
int  stack_size(stack s);
void stack_clear(stack s);
#endif
```

Υλοποίηση Στοίβας

Αναπαράσταση Στοίβας (stack.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"

struct _stack {
    item_type* array;
    int max_size;
    int top;
};
```

Υλοποίηση Στοίβας

Δημιουργία Στοίβας (stack.c)

```
static stack stack_create_maxsize(int max_size) {
    if (max_size <= 0) {
        fprintf(stderr, "Wrong stack size (%d)\n", max_size);
        abort();
    }

    stack s = (stack) malloc(sizeof(struct _stack));
    if (s == NULL) {
        fprintf(stderr, "Insufficient memory to initialize stack.\n");
        abort();
    }

    item_type *items;
    items = (item_type *) malloc(sizeof(item_type) * max_size);
    if (items == NULL) {
        fprintf(stderr, "Insufficient memory to initialize stack.\n");
        abort();
    }

    s->array = items;
    s->max_size = max_size;
    s->top = -1;
    return s;
}
```

Υλοποίηση Στοίβας

Δημιουργία Στοίβας (stack.c)

```
stack stack_create() { return stack_create_maxsize(64); }
```

Υλοποίηση Στοίβας

Καταστροφή Στοίβας (stack.c)

```
void stack_destroy(stack s) {
    if (s == NULL) {
        fprintf(stderr, "Cannot destroy stack\n");
        abort();
    }

    free(s->array);
    free(s);
}
```

Υλοποίηση Στοίβας

Διπλασιασμός Χωρητικότητας (stack.c)

```c
static
void stack_doublesize(stack s) {
    if (s == NULL) {
        fprintf(stderr, "Cannot double stack size\n");
        abort();
    }

    // create double the stack
    int new_max_size = 2 * s->max_size;
    item_type* new_array = (item_type*) malloc(sizeof(item_type) * (new_max_size));
    if (new_array == NULL) {
        fprintf(stderr, "Insufficient memory to double the stack.\n");
        abort();
    }

    // copy elements to new array
    int i;
    for(i = 0; i <= s->top; i++) {
        new_array[i] = s->array[i];
    }

    // replace array with new array
    free(s->array);
    s->array = new_array;
    s->max_size = new_max_size;
}
```

Υλοποίηση Στοίβας

Push (stack.c)

```
void stack_push(stack s, item_type elem) {
    // increase capacity if necessary
    if (s->top >= s->max_size-1) {
        stack_doublesize(s);
    }

    // push the element
    s->array[++s->top] = elem;
}
```

Υλοποίηση Στοίβας

Pop και Top(stack.c)

```
void stack_pop(stack s) {
    if (stack_is_empty(s)) {
        fprintf(stderr, "Can't pop element from stack: \
            stack is empty.\n");
        abort();
    }

    s->top--;
}

item_type stack_top(stack s) {
    if (stack_is_empty(s)) {
        fprintf(stderr, "Stack is empty.\n");
        abort();
    }

    return s->array[s->top];
}
```

Υλοποίηση Στοίβας

Empty and Size (stack.c)

```c
int stack_is_empty(stack s) {
    if (s == NULL) {
        fprintf(stderr, "Cannot work with NULL stack.\n");
        abort();
    }

    return s->top < 0;
}

int stack_size(stack s) {
    if (s == NULL) {
        fprintf(stderr, "Cannot work with NULL stack.\n");
        abort();
    }

    return s->top+1;
}
```

Υλοποίηση Στοίβας

Clear (stack.c)

```c
void stack_clear(stack s) {
    if (s == NULL) {
        fprintf(stderr, "Cannot work with NULL stack.\n");
        abort();
    }

    s->top = -1;
}
```

Δοκιμάζοντας την Στοίβα

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "stack.h"

int main() {
    stack s;
    s = stack_create();

    for(int i = 0; i < 1000; i++) {
        stack_push(s, i);
        assert(stack_top(s) == i);
        assert(stack_is_empty(s) == 0);
        assert(stack_size(s) == i+1);
    }

    for(int i = 999; i >= 0; i--) {
        assert(stack_top(s) == i);
        stack_pop(s);
    }

    stack_destroy(s);

    return 0;
}
```

Παράδειγμα Χρήσης

Μία κλασσική εφαρμογή στοίβας είναι ο έλεγχος αγκυλών και παρενθέσεων σε προγράμματα γραμμένα σε C ή Java.

Όταν δούμε αριστερή αγκύλη ή παρένθεση προσθέτουμε το σύμβολο στην στοίβα. Όταν δούμε δεξιά αγκύλη ή παρένθεση ελέγχουμε πως στην κορυφή της στοίβας υπάρχει το συμμετρικό του και το αφαιρούμε από την στοίβα.

Η απλή αυτή στρατηγική αυτή δεν λειτουργεί εαν έχουμε χρήση χαρακτήρων ή συμβολοσειρών που περιέχουν μέσα παρενθέσεις ή αγκύλες: π.χ `printf(")");`

Παράδειγμα Χρήσης

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "stack.h"

int main() {
    stack s;
    s = stack_create();

    int is_correct, c, top;
    is_correct = 1;

    while((c=getchar())!=EOF) {
        if (c == '(' || c == '{') {
            stack_push(s, c);
        } else if (c == ')') {
            if (stack_is_empty(s)) {
                is_correct = 0;
                break;
            }
            top = stack_top(s);
            stack_pop(s);
            if (top != '(') {
                is_correct = 0;
                break;
            }
        }
```

Example using Stack

```
            else if (c == '}') {
                if (stack_is_empty(s)) {
                    is_correct = 0;
                    break;
                }
                top = stack_top(s);
                stack_pop(s);
                if (top != '{') {
                    is_correct = 0;
                    break;
                }
            }
        }

        if (!stack_is_empty(s)) {
            is_correct = 0;
        }

        if (is_correct) {
            printf("OK\n");
        } else {
            printf("ERROR\n");
        }

        stack_destroy(s);

        return 0;
    }
```