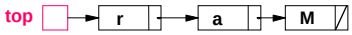
Стек

Дефиниране

```
struct stack                    // стек
{
        LINK top;               // връх на стека
};
typedef struct stack STACK;
```



Операции

1. **Създаване на празен стек**
2. **Включване на възел във върха на стека**
3. **Изключване на възел от върха на стека**
4. **Намиране на последния елемент в стека**
5. **Освобождаване на заетата памет от стек**

1. Създаване на празен стек

Алгоритъм

връх ← ПРАЗНО


```
void create (STACK *s)
{
        s->top = NULL;              // празен стек
}
```

2. Включване на възел във върха на стека

Алгоритъм

отдели памет за нов възел във временната връзка
ако отделянето на памет не е успешно
  неуспешна операция
в противен случай
  запази данните в новия възел
  добави новия възел във върха на стека
  успешна операция

```
int push (STACK *s, DATA key)
{
  LINK p;                           // временна връзка
  p = (LINK)malloc(sizeof(NODE));
  if (p == NULL)
  {
    printf("Няма достатъчно памет!\n");
    return FAILURE;
  }
  else
  {
    p->data = key;                  // запазва данните във възела
    p->next = s->top;               // добавя възела към върха
    s->top = p;
    return SUCCESS;
  }
}
```

3. Изключване на възел от върха на стека

Алгоритъм

ако стекът е празен
  неуспешна операция
в противен случай
  запази данните на стария връх
  насочи временната връзка към следващата
  компонента на стария връх
  освободи паметта, заета от стария връх
  насочи върха на стека към временната връзка
  успешна операция

```
int pop (STACK *s, DATA *key)
{
  if (s->top == NULL)
  {
    printf ("Празен стек!\n");
    key = NULL;
    return FAILURE;
  }
  else
  {
    LINK p;                  // временна връзка
    *key = s->top->data;     // запазва данните във върха
    p = s->top->next;        // придвижва временната връзка
    free (s->top);           // освобождава паметта
    s->top = p;              // насочва върха на стека
    return SUCCESS;
  }
}
```

4. Намиране на последния елемент в стека

Алгоритъм

ако стекът е празен
** неуспешна операция**
в противен случай
** запази данните на стария връх**
** успешна операция**

```
int look (STACK s, DATA *key)
{
  if (s.top == NULL)
  {
    printf ("Празен стек!\n");
    return FAILURE;
  }
  else
  {
    *key = s.top->data;          // запазва данните във върха
    return SUCCESS;
  }
}
```

5. Освобождаване на заетата памет от стек

Алгоритъм

насочи временната връзка към върха на стека
докато върха на стека не е ПРАЗНО
** придвижи върха на стека**
** освободи паметта, заета от предишния връх на стека**
** насочи временната връзка към върха на стека**

```
void freestack (STACK *s)
{
  LINK p;                          // временна връзка
  for (p=s->top; s->top != NULL; p = s->top)
  {
    s->top = s->top->next;       // придвижва временната връзка
    free (p);                    // освобождава паметта
  }
  printf ("Заетата памет е освободена от стека!\n");
}
```

Задача: Проверете съответствието между отварящи и затварящи скоби в израз чрез използване на стек от символи.

Алгоритъм

създай празен стек
въведи израз като символен низ
насочи временен указател към началото на низа
докато временният указател не е стигнал края на низа
повтаряй
** ако текущият символ е (**
**   включи символа във върха на стека**
** ако текущият символ е)**
**   изключи възела от върха на стека**
** придвижи временния указател**
ако стекът е празен
** успешна операция**
в противен случай
** неуспешна операция**