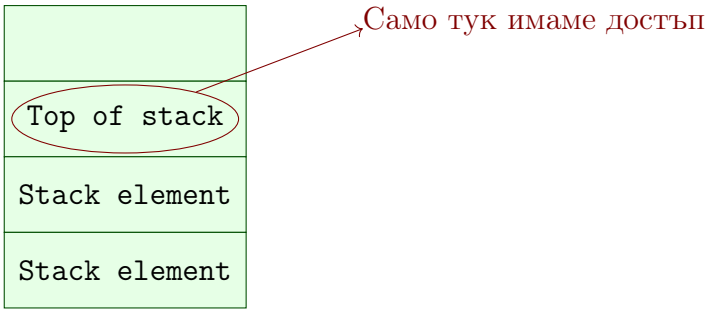
1 Стек

Стекът е абстрактен тип данни базиран на принципа **Last In First Out**. В стека винаги имаме достъп **единствено** до елемента добавен последен. Можем да го видим или да го премахнем, но по никакъв начин не можем да достъпим елементите добавени преди него. Пример за стек показва следната фигура:



Стекът поддържа следните операции:

1. *push(elem)* $\Rightarrow$ Добавя *elem* на върха на стека $O(1)$
2. *pop()* $\Rightarrow$ Премахва елемента на върха на стека $O(1)$
3. *top()* $\Rightarrow$ Връща елемента на върха на стека $O(1)$

Стековете имат много практически приложения. Методът, по който извикваме функции работи на принципа на стек. Undo/redo командите работят на принципа на стек. Много интересни задачи имат неочаквано лесни (за прилагане не за измисляне) решения използващи стек.
Първия проблем който ще разгледаме е как да реализираме стек с достъп до произволен елемент. Отговорът е прост - **никак!** Това не е стек.

1.1 Начини за реализация и std::stack

Има няколко начина по които можем да реализираме стек на *c++*. Стека може да е

1. Ограничен ⇒ имащ горна граница на елементите.
2. Динамичен ⇒ Можем да добавяме колкото искаме елементи (докато имаме памет де)

Също така представянията могат да са:

1. Чрез масив ⇒ Подобно на класа вектор, който реализирахме по ООП, но интерфейса позволява достъп само до последния елемент
2. Свързано представяне ⇒ Всеки елемент на стека има указател към следващ елемент.
   Примерно:

```
template<class T>
struct StackElement {
     T data;
     StackElement<T>* next;
}
```

Помислете как би изглеждала всяка от функциите написана за вид стек. Езикът *c++* ни предлага готова реализация на стек. Пример за използването ѝ:

```cpp
#include <stack>
#include <iostream>

int main() {
    std::stack<int> s;

    s.push(42);
    std::cout << s.top() << " ";
    s.push(6);
    std::cout << s.top() << " ";
    s.pop();
    std::cout << s.top();
    s.pop();            // -> Празен стек
    s.empty();          // True
    s.pop();            // -> Хвърля грешка
}
```

Какво ще изведе този код? STL (Standard Template Library) предлага доста алгоритми и структури от данни наготово, които ще разглеждаме с напредване на курса.

1.2 Балансиран низ от скоби

Един низ състоящ се от символите (и) наричаме балансиран ако на всяка отваряща скоба съответства затваряща и обратното. Примерно ((())) е балансиран низ а) не е балансиран.
От нас се иска при получен низ да върнем дали той е балансиран или не. Това може би е една от най - стандартните задачи свързани със стек. Идеята е следната: всеки път когато срещнем отваряща скоба я добавяме в стека. Всеки път когато срещнем затваряща скоба премахваме първия елемент на стека. Ако стека е празен при срещане на затваряща скоба то низът очевидно не е балансиран. Ако стекът има елементи в края на итерацията отново е очевидно, че низът не е балансиран.
Примерно решение:

```cpp
#include <stack>
#include <string>
#include <iostream>

bool isStringBalanced(const std::string& str) {
    std::stack<char> s;

    for(char currentChar : str) {
        if(currentChar == '('){
            s.push('(');
        }
        else if(currentChar == ')') {
            if(s.empty())
                return false;

            s.pop();
        }
    }
    return s.empty();
}
int main() {
    std::cout <<
        isStringBalanced("((()))") <<
        isStringBalanced("(())") <<
        isStringBalanced("())") <<
        isStringBalanced(")");
}
```

1.3 Обратен полски запис

Обратен полски запис наричаме запис, в който операторите "следват"(са след) числата. Примерно изразът $3 + 4$ в обратна полска нотация е $3\ 4\ +$. Или изразът $1 - 2 + 3$ би изглеждал $1\ 2\ -3+$. Обикновено първо са дадени числата а след това операциите които извършваме върху тях. А как би изглеждал израза $1 - (2 + 3)$. Тук първо трябва да извършим събирането и след това изваждането. Тоест израза се записва като $123 + -$. Предимството от тази нотация е, че нямаме нужда от скоби, което ускорява сметките.

Нашата задача е да напишем функция, която приема валиден стринг в обратна полска нотация и връща пресметнатия резултат.
Алгоритъмът, който от стандартен запис построява израз в обратен полски запис се нарича Shunting Yard.