Алгоритмы и структуры данных
для начинающих

Стек

Определение стека

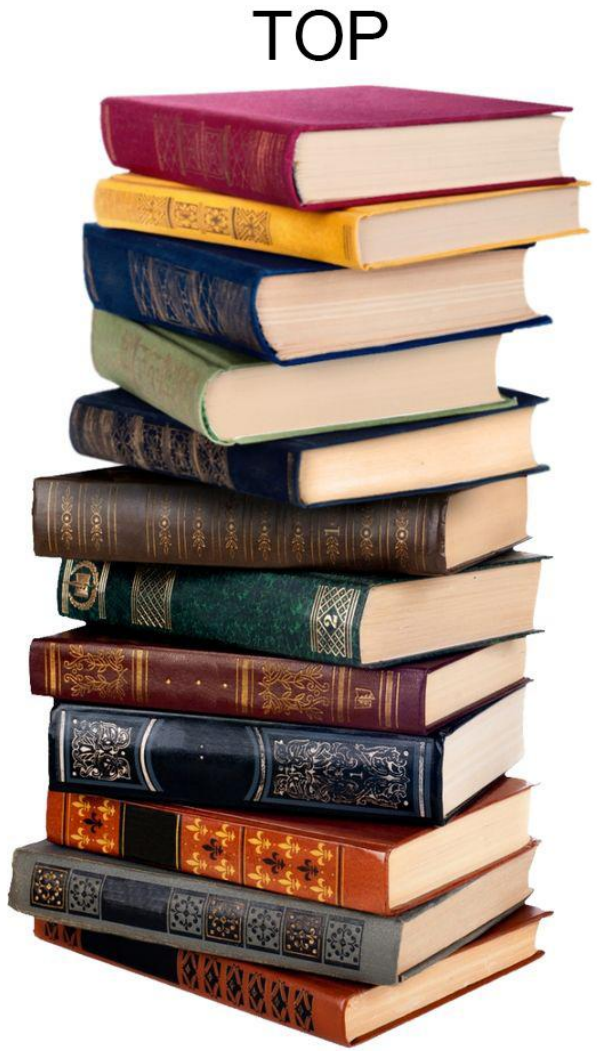
Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»).

Ограничения накладываемые на тип данных стек:

- нельзя получить доступ к произвольному элементу стека
- можно только добавлять или удалять элементы с помощью специальных методов
- нет метода Contains, как у списков
- нет итератора

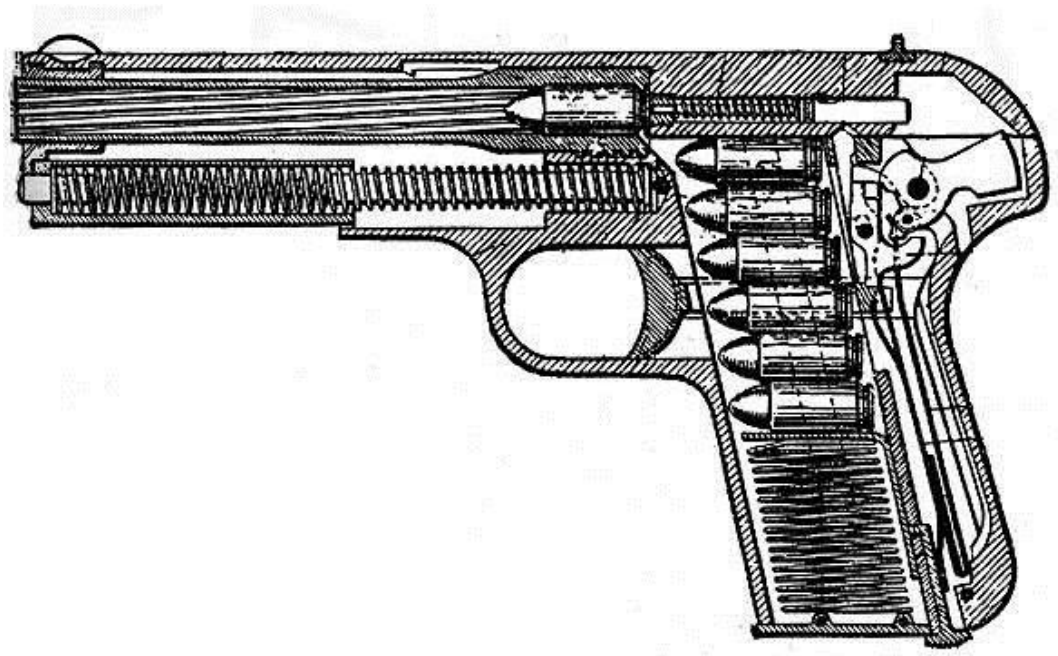
Основные операции для стека

- Создание стека
- Добавление элемента в стек
- Проверка стека на пустоту
- Считывание головного элемента
- Удаление головного элемента
- Очистка стека



Реализация стеков

Стеки могут реализовываться на различных структурах данных: на одномерных статических и динамических массивах, на линейных списках, с помощью файлов… В общем, стек можно реализовать на любой структуре, в которой есть возможность хранить несколько элементов и если есть то, куда можно сохранять данные.



Создание стека

```
struct List
{
   int x;                          //информационный элемент
   List *Next,*Head;                        //Голова стека и указатель на следующий элемент
};

struct stek
{
   int value;
   struct stek *next;              // указатель на следующий элемент списка (стека)
};
```

Добавление элемента в стек
(Добавляем элемент на вершину стека)

```
void Add(int x, List *&MyList)                    //Принимаем элемент стека и указатель на стек, при этом говорим, что принимаемый указатель будет сам по себе указателем
{   List *temp = new List;                        //Выделяем память для нового элемента
    temp->x = x;                                  //Записываем в поле x принимаемый в функцию элемент x
    temp->Next = MyList->Head;                    //Указываем, что следующий элемент это предыдущий
    MyList->Head = temp;                          //Сдвигаем голову на позицию вперед  }


void push(stek* &NEXT, const int VALUE)
{   stek *MyStack = new stek;                     // объявляем новую динамическую переменную типа stek
    MyStack->value = VALUE;                       // записываем значение, которое помещается в стек
    MyStack->next = NEXT;                         // связываем новый элемент стека с предыдущим
    NEXT = MyStack;                               // новый элемент стека становится его вершиной }
```

Отображение стека. Проверка стека на пустоту

```
void Show(List *MyList)                    //Нужен только сам стек
{
   List *temp = MyList->Head;              //Объявляем указатель и Указываем ему, что его позиция в голове стека
   //с помощью цикла проходим по всему стеку
   while (temp != NULL)                    //выходим при встрече с пустым полем
   {
      cout << temp->x << " ";              //Выводим на экран элемент стека
      temp = temp->Next;                   //Переходим к следующему элементу
   }
}
```

Удаление и возврат головного элемента
(Удаляем элемент с вершины стека и возвращает его)

```
int pop(stek* &NEXT)
{
   int temp = NEXT->value;              // извлекаем в переменную temp значение в вершине стека
   stek *MyStack = NEXT;                // запоминаем указатель на вершину стека, чтобы затем
                                        // освободить выделенную под него память
   NEXT = NEXT->next;                   // вершиной становится предшествующий top элемент
   delete MyStack;                      // освобождаем память, тем самым удалили вершину
   std::cout << temp;                   //Вывод текущего элемента на экран
   return temp;                         // возвращаем значение, которое было в вершине
}
```

Стек можно обсмотреть с противоположной стороны голове, но это требует очень многого с точки зрения затрат памяти, потому что каждый раз придется проходить от головы до последнего элемента ровно столько раз, сколько в стеке элементов.

Очистка стека

```
void ClearList(List *MyList)
{
while (MyList->Head != NULL)                    //Пока по адресу не пусто
   {
      List *temp = MyList->Head->Next;                  //Временная переменная для хранения адреса следующего элемента
      delete MyList->Head;                          //Освобождаем адрес обозначающий начало
      MyList->Head = temp;                           //Меняем адрес на следующий
   }
}
```

Заключение

- Классический пример использования стека — калькулятор в обратной польской, или постфиксной, записи. В ней оператор записывается *после* своих операндов. Т.е. вместо «4 + 2» мы запишем «4 2 +»

  push(4)

  push(2)

  push(pop() + pop())

- Примеры