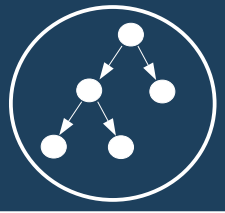


Data Structures and Algorithms

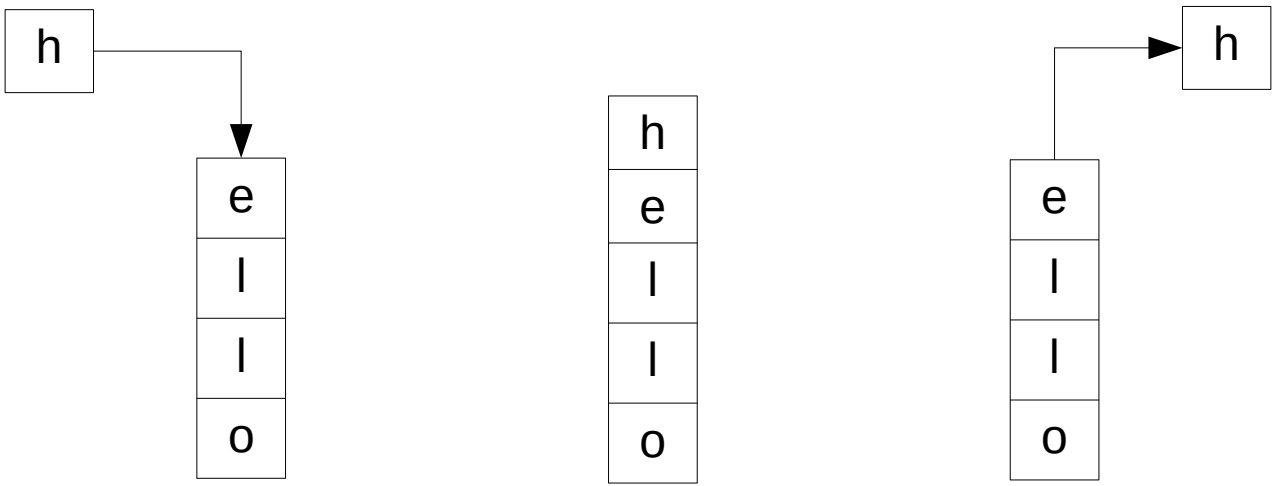
Структуры данных. Стек на основе массива

Стек

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

Поддерживаемые операции:

- Добавление элемента в вершину стека (push)
- Удаление элемента из вершины стека (pop)
- Получение элемента с вершины стека без удаления (peek)
- Получение размера стека (size)

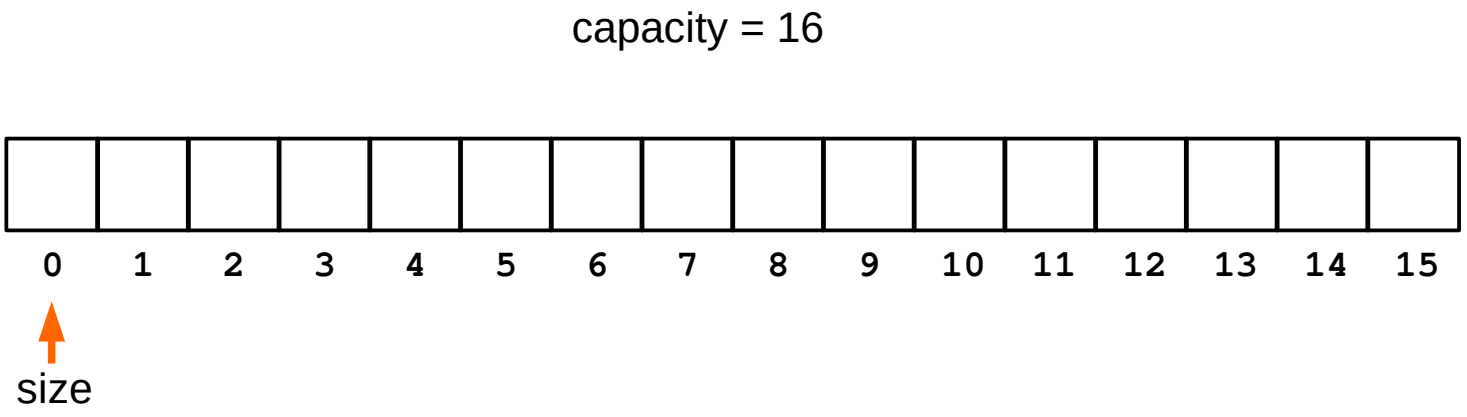


Реализация стека на основе массива

В качестве основы стека можно использовать массивы переменной длинны. В таком случае получение и добавления элемента в стек реализуются как изменение и получения элемента массива по индексу. Отдельно нужно будет рассмотреть случай необходимости увеличения размера стека. Для хранения индекса вершины стека можно использовать отдельную переменную, она же будет использована для вычисления размера стека.

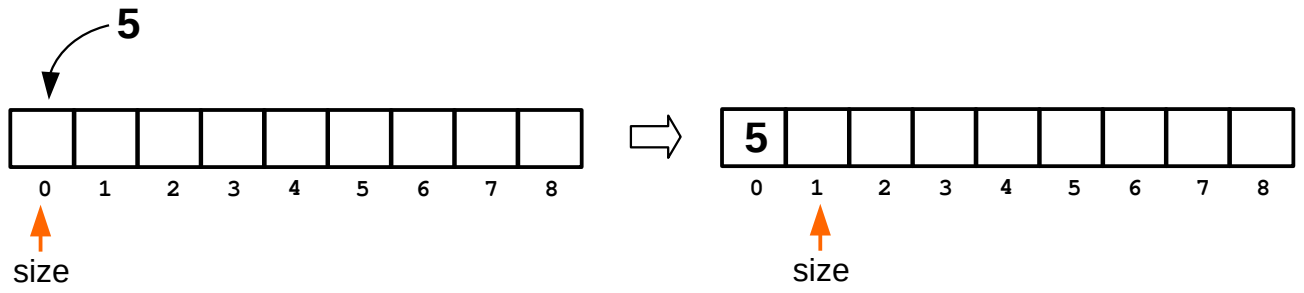
Реализации стека на основе массива

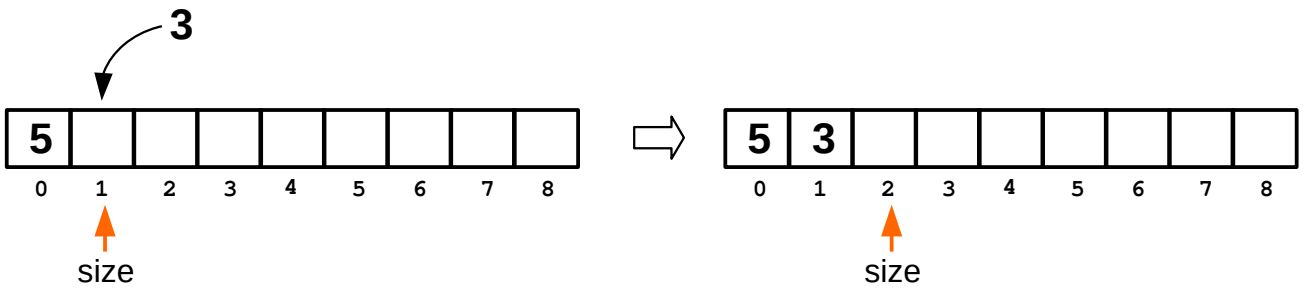
В качестве основы стека берем массив нужного типа данных. Его размер будем называть capacity(емкость). Также введем дополнительную переменную size(размер), она будет указателем на вершину стека (место для добавления элемента). При создании стека size устанавливается как индекс первого элемента в массиве.



Добавление значения в стек (размер меньше емкости)

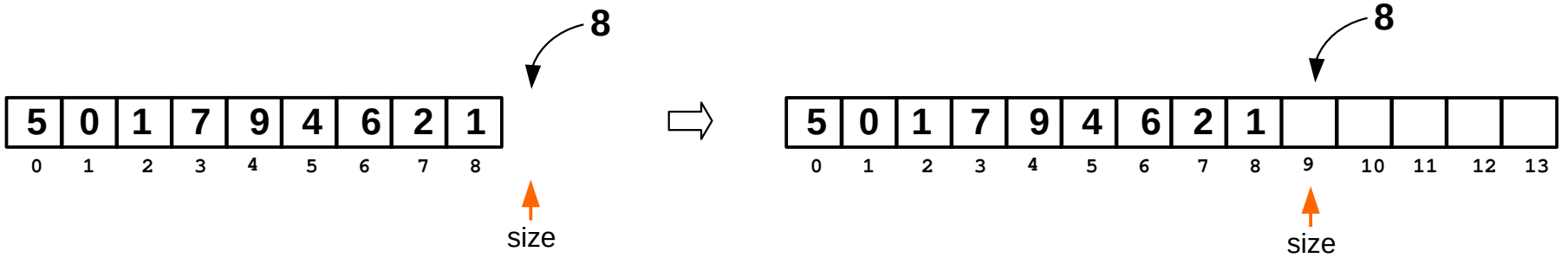
Если size меньше чем capacity, то добавляем элемент на индекс size и увеличиваем size на единицу.

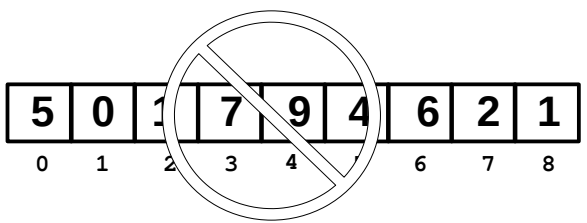




Добавление значения (размер равен емкости)

Если size равно capacity, то создаем новый массив размером (capacity * 3)/2 + 1. Копируем данные из базового массива в новый. Указываем, что теперь для хранения используется новый массив. Добавляем элемент на индекс size и увеличиваем size на единицу.

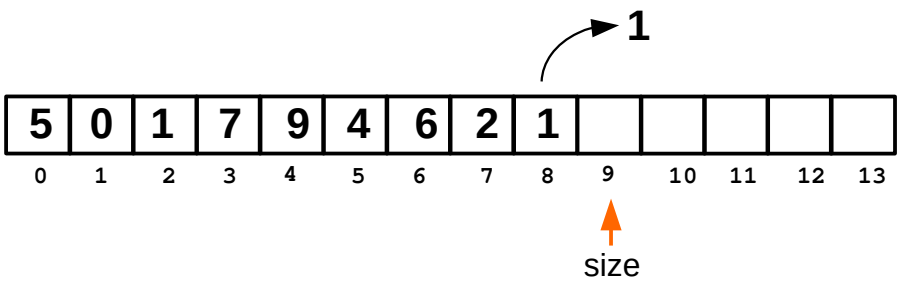




Освобождаем память занимаемую старым массивом.

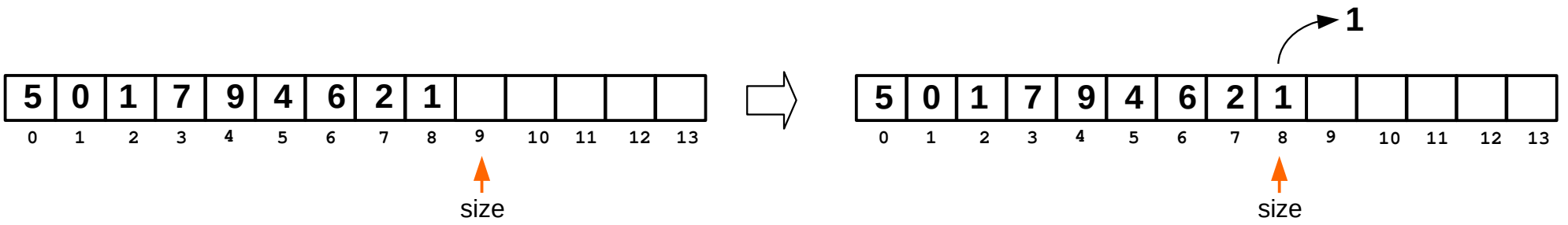
Получение элемента без удаления

При получении значения без удаления, сначала проверяют значение size. Если size равен первому индексу в массиве, то стек пуст. В противном случае возвращаем элемент по индексу size-1.



Получение элемента с удалением

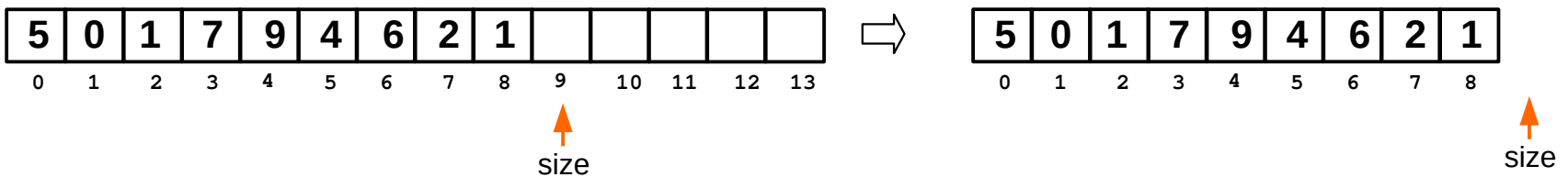
При получении значения с удалением сначала проверяют значение size. Если size равен первому индексу в массиве, то стек пуст. В противном случае уменьшаем size на единицу и возвращаем элемент по индексу size. При необходимости удаляем элемент по индексу size.

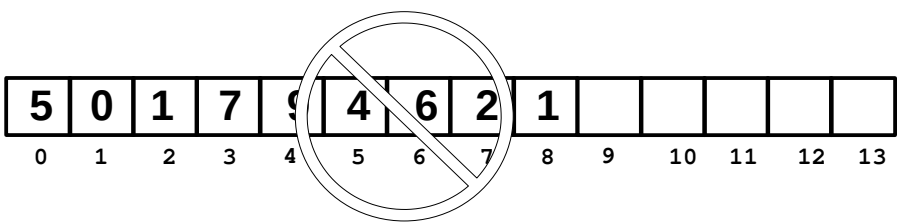


Уменьшение размера стека

В большинстве случаев стек на основе массива только увеличивает свою емкость. Автоматического уменьшения емкости не предусматривают. Для уменьшения емкости используют функцию, вызов которой осуществляется по желанию разработчика. В этой функции обычно устанавливают capacity равное size.

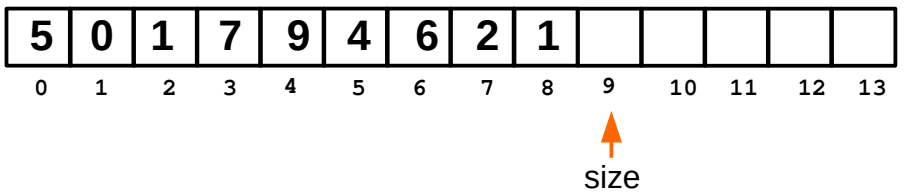
Создают новый массив размером size. Копируют данные из основного массива в новый. Указываем что новый массив теперь используется вместо основного.





Освобождаем память занимаемую старым массивом.

Получение размера стека



Для получения размера стека нужно использовать значение size.

Оценка сложности операций

Операция	Сложность операции в худшем случае
Вставка элемента	O(n)
Получение без удаления	O(1)
Получение с удалением	O(1)
Получение размера	O(1)

Python

Реализация на Python

Отсутствие массивов в Python

Так как в Python отсутствует поддержка массивов, то реализация стека на их основе становится бессмысленной.

Реализация на Java

Описание полей используемых для стека

```
class ArrayBasedStack {
    private Object[] dataArray;
    private int size;
    private int capacity;
    private final int DEFAULT_CAPACITY = 16;
    private final int MAX_STACK_SIZE = Integer.MAX_VALUE - 1;

    public ArrayBasedStack() {
        dataArray = new Object[DEFAULT_CAPACITY];
        capacity = dataArray.length;
        size = 0;
    }
```

Метод добавления

```
public void push(Object value) {
    if (size >= capacity) {
        boolean resizeResult = upResize();
        if (!resizeResult) {
            throw new RuntimeException("Cannot add element");
        }
    }
    dataArray[size] = value;
    size += 1;
}
```

Методы получения с удалением и без

```
public Object pop() {
    if (size == 0) {
        return null;
    }
    size -= 1;
    Object element = dataArray[size];
    dataArray[size] = null;
    return element;
}

public Object peek() {
    if (size == 0) {
        return null;
    }
    return dataArray[size - 1];
}
```

Метод для получения размера

```
public int size() {
    return size;
}
```

Метод для увеличения и уменьшения размера

```
public boolean upResize() {
     if (capacity >= MAX_STACK_SIZE) {
          return false;
     }
     long newCapacityL = (capacity * 3L) / 2L + 1L;
     int newCapacity = (newCapacityL < MAX_STACK_SIZE) ? (int) newCapacityL : MAX_STACK_SIZE;
     dataArray = Arrays.copyOf(dataArray, newCapacity);
     capacity = newCapacity;
     return true;
}

public void trimToSize() {
     dataArray = Arrays.copyOf(dataArray, size);
     capacity = dataArray.length;
}

public void clear() {
     dataArray = new Object[DEFAULT_CAPACITY];
     capacity = dataArray.length;
     size = 0;
}
```

Реализация на Fortran

Описание стека

```
type Array_Based_Stack
    integer,allocatable::data_array(:)
    integer::l_size, capacity

    contains
        procedure,pass::init
        procedure,pass::push
        procedure,pass::up_resize
        procedure,pass::pop
        procedure,pass::peek
        procedure,pass::get_size
        procedure,pass::trim_to_size
        procedure,pass::clear
        procedure,pass::show
end type Array_Based_Stack
```

Методы инициализации и увеличения размера

```
subroutine init(this)
    class(Array_Based_Stack)::this
    if(.not. allocated(this%data_array)) then
        allocate(this%data_array(1))
        this%capacity = 1
        this%l_size = 1
    end if
end subroutine init

subroutine up_resize(this)
    class(Array_Based_Stack)::this
    integer,allocatable::temp_array(:)
    integer::new_capacity
    new_capacity = (this%capacity * 3)/2 + 1
    allocate(temp_array,source = this%data_array)
    deallocate(this%data_array)
    allocate(this%data_array(new_capacity))
    this%data_array(1:this%l_size-1) = temp_array
    this%capacity = new_capacity
    deallocate(temp_array)
end subroutine up_resize
```

Метод добавления

```
subroutine push(this, data_value)
    class(Array_Based_Stack)::this
    integer, intent(in)::data_value
    if (this%l_size > this%capacity) then
        call this%up_resize()
    end if
    this%data_array(this%l_size) = data_value
    this%l_size = this%l_size + 1
end subroutine push
```

Методы получения с удалением и без

```
integer function pop(this,op_result)
    class(Array_Based_Stack)::this
    logical, intent(inout)::op_result
    if(this%l_size == 1) then
         op_result = .false.
         return
    end if
    this%l_size = this%l_size - 1
    pop = this%data_array(this%l_size)
    op_result = .true.
end function pop

integer function peek (this, op_result)
    class(Array_Based_Stack)::this
    logical, intent(inout)::op_result
    if(this%l_size == 1) then
        op_result = .false.
        return
    end if
    peek = this%data_array(this%l_size - 1)
    op_result = .true.
end function peek
```

Метод для получения длины

```
integer function get_size(this)
    class(Array_Based_Stack)::this
    get_size = this%l_size - 1
end function get_size
```

Методы уменьшения размера и очистки

```
subroutine trim_to_size(this)
    class(Array_Based_Stack)::this
    integer,allocatable::temp_array(:)
    allocate(temp_array,source = this%data_array(:this%l_size-1))
    this%data_array = temp_array
    this%capacity = size(this%data_array, dim = 1)
    deallocate(temp_array)

end subroutine trim_to_size

subroutine clear(this)
    class(Array_Based_Stack)::this
    if(allocated(this%data_array)) then
        deallocate (this%data_array)
        this%capacity = 0
        this%l_size = 0
    end if
end subroutine clear
```

Список литературы

1) Роберт Седжвик, Кевин Уэйн «Алгоритмы на java 4-е издание» Пер. с англ. - М. : ООО "И.Д. Вильямс", 2013. ISBN 978-5-8459-1781-2.