Arrays, Stacks, & Queues

Arrays

An array is a linear data structure that stores elements of the same type in a **contiguous block of memory**. It is also known as a static array or a one-dimensional array. Here is a more detailed description of the array data structure:

- Elements of an array are of the same data type.
- Arrays are indexed, meaning each element in the array has a unique index that allows access to its value.
- The first element in an array is typically assigned index 0 and the last element is assigned the last index.
- Static arrays have a fixed size, which means once created, you cannot change its size.
- Arrays support basic operations such as accessing elements, modifying elements, and iterating over all elements.

Overall, arrays provide a simple and efficient way to work with collections of data.

Python C++

```py
arr = [0, 1, 2, 3]

arr[2] = "item"
#[0, 1, "item", 3]

size = len(arr)
```

Python lists are implemented as dynamic arrays under the hood. Initially, when you create a list with a few elements, Python allocates memory for a small array to hold those elements. As elements are added to the list and it reaches capacity, a growth strategy to resize the array often doubling the size of the array each time it needs to be resized. When the array needs to be resized, existing elements are copied to the newly created larger array in linear time.

How can Python lists store items of different types?

Python lists can store different item types because they are implemented using pointers to Python objects. Each element in a Python list is essentially a reference to a Python object, rather than the actual object itself. This allows lists to store objects of different types meaning the memory footprint of the list itself remains relatively small.

C++ arrays are static arrays. All the above-mentioned properties hold for these.

Note

In C++, vectors `std::vector` are dynamic arrays that are part of the Standard Template Library (STL).

```c++
#include <vector> // dont forget to add this (safer)

std::vector<int> nums = {10, 20, 30};

std::vector<int> arr;
arr.push_back(10);
arr.push_back(20);
```

Time Complexity

Operation	Static	Dynamic
Access	$O(1)$	$O(1)$
Remove/Insert	$O(n)$	$O(1)$ (Amortized)
Traverse	$O(n)$	$O(n)$

$O(n)$ memory requirements.

Useful methods

Python lists   C++ vectors

```python
arr.append(4) # Adds an item to the end of the list.
arr.extend([3,4,5]) # Add multiple items to the end of the list.
arr.insert(index, item) # Insert an item at a specific index.
arr.remove(item) # Remove the first occurrence of an item from the list.
arr.pop() # Remove and return the last item in the list.
arr.pop(index) # Remove and return an item at a specific index.
arr.reverse() # Reverse the order of the list in-place.
arr.index(item) # Return the index of the first occurrence of an item in the li

# Traverse
for i in range(len(arr)):
    print(arr[i])
```

Stack

A stack is a linear data structure that follows a particular order in which the operations are performed. The order is LIFO(Last In First Out). The principal operations of a stack are:

- Push
- Pop
- Peek
- isEmpty

Python C++

python

```python
# Stacks in python can be implemented using lists (dynamic arrays).
stack = []

# Stack operations
stack.append(10)  # Push
stack.append(20)
stack.append(30)
print(stack)  # [10, 20, 30]

item = stack.pop()  # Pop
print(item)  # 30
print(stack)  # [10, 20]

peek_item = stack[-1]  # Peek
print(peek_item)  # 20

is_empty = not stack  # isEmpty
print(is_empty)  # False
```

Queue

A queue is another linear data structure that follows a particular order in which the operations are performed. The order is FIFO(First In First Out). The principal operations of a queue are:

- Enqueue
- Dequeue
- Peek
- isEmpty

Python C++

python

```python
# Queues in python can be implemented using lists (dynamic arrays).
queue = [] # Enqueue
queue.append(10)
 queue.append(20)

# Dequeue
item = queue.pop(0)

# Peek
peek_item = queue[0]
print(peek_item)  # 20

# isEmpty
is_empty = not queue
print(is_empty)  # False

# Queues can also be implemented with Deque (double ended queue) in Python.
from collections import deque
queue = deque()
# Enqueue
queue.append(10)
queue.append(20)

# Dequeue
item = queue.popleft()

# Peek
peek_item = queue[0]
print(peek_item)  # 10

# isEmpty
is_empty = not queue
print(is_empty)  # False
```

Next page
Linked Lists

Previous page
Arrays, Stacks, & Queues

Next page
Markdown Examples

Runtime API Examples

This page demonstrates usage of some of the runtime APIs provided by VitePress.

The main `useData()` API can be used to access site, theme, and page data for the current page. It works in both `.md` and `.vue` files:

```md
<script setup>
import { useData } from 'vitepress'

const { theme, page, frontmatter } = useData()
</script>

## Results

### Theme Data
<pre>{{ theme }}</pre>

### Page Data
<pre>{{ page }}</pre>

### Page Frontmatter
<pre>{{ frontmatter }}</pre>
```

Results

Theme Data

```
{
  "nav": [
    {
      "text": "Home",
      "link": "/"
    },
    {
      "text": "Examples",
      "link": "/markdown-examples"
    }
  ],
  "sidebar": [
    {
      "text": "Data Structures & Algorithms",
      "items": [
        {
          "text": "Arrays, Stacks, & Queues",
          "link": "/algos/arrays"
        },
        {
          "text": "Linked Lists",
          "link": "/algos/linked-lists"
        },
        {
          "text": "Markdown Examples",
          "link": "/markdown-examples"
        },
        {
          "text": "Runtime API Examples",
          "link": "/api-examples"
        }
      ]
    }
  ],
  "socialLinks": [
    {
      "icon": "github",
      "link": "https://github.com/ajagekarakshay/computer-science-essentials"
    }
  ]
}
```

Page Data

```
{
  "title": "Runtime API Examples",
  "description": "",
  "frontmatter": {
    "outline": "deep"
  },
  "headers": [],
  "relativePath": "api-examples.md",
  "filePath": "api-examples.md"
}
```

Page Frontmatter

```
{
  "outline": "deep"
}
```

More

Check out the documentation for the full list of runtime APIs.

Previous page
Markdown Examples

Computer Science Essentials

Digital garden of concepts in data structures, algorithms, and programming.

My great project tagline

Markdown Examples API Examples

Feature A

Lorem ipsum dolor sit amet, consectetur adipiscing elit

Feature B

Lorem ipsum dolor sit amet, consectetur adipiscing elit

Feature C

Lorem ipsum dolor sit amet, consectetur adipiscing elit

Markdown Extension Examples

This page demonstrates some of the built-in markdown extensions provided by VitePress.

Syntax Highlighting

VitePress provides Syntax Highlighting powered by Shiki, with additional features like line-highlighting:

Input

md

~~~
```js{4}
export default {
  data () {
    return {
      msg: 'Highlighted!'
    }
  }
}
```
~~~

Output

js

```js
export default {
  data () {
    return {
      msg: 'Highlighted!'
    }
  }
}
```

Custom Containers

Input

```md
::: info
This is an info box.
:::

::: tip
This is a tip.
:::

::: warning
This is a warning.
:::

::: danger
This is a dangerous warning.
:::

::: details
This is a details block.
:::
```

Output

INFO

This is an info box.

TIP

This is a tip.

WARNING

This is a warning.

DANGER

This is a dangerous warning.

▶ **Details**

More

Check out the documentation for the full list of markdown extensions.

Previous page
Linked Lists

Next page
Runtime API Examples