C++ серверная часть. введение в АИСД

...

Сервер - компьютер или ПО, обслуживающая запросы др. устройств через специализированную сеть.

Мы будем исп-ть localhost.

localhost - (*простыми словами*) это ваш компьютер.

Схема работы <localhost server>.

<server request>

Запросы на сервер - это сообщения от клиентов, запрашивающие данные или услуги у сервера. Подобные запросы отправляются как правило через сеть, а сервер их обрабатывает и возвращает ответ клиенту.

HTTP - протокол передачи данных.

- 200[OK]
- 404[Not Found]
- 500[Internal Server Error]
- 302[Found]

<server http work>

Чтобы работать с HTTP на языке C++, можно использовать стандартные средства языка или сторонние библиотеки. Файлы заголовков:

заголовки **<sys/socket.h>**, **<arpa/inet.h>**, **<unistd.h>** для SOCKET.

SOCKET - абстракция по обмену данными между устройствами через сеть в сетевом программировании.

<connect to server>

socket()

connect() <if return -1 (FALSE CONNECTION)>

sockaddr_in - это структура, используемая для представления адреса сокета в семействе протоколов IPv4. В ней содержатся поля для IP-адреса и порта.

<CLS>

create-localhost-server - создадим на Python (так проще).

```
{
    host = ‘127.0.0.1’
    port = 8080
}
```

<python-code>1

```python
import socket

def main():
    host = '127.0.0.1'
    port = 8080
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}")
    while True:
        client_socket, client_address= server_socket.accept()
        print(f"Connection from {client_address}")
        response = code
        client_socket.sendall(response.encode())
        client_socket.close()

if __name__ == "__main__":
    main()
```

<python-code>2

```
code = """HTTP/1.1 200 OK
Content-Type: text/html

<html>
<head><title>Simple HTML Page</title></head>
<body>
<h1>Hello, World!</h1>
<p>This is a simple HTML page served by a Python server.</p>
</body>
</html>
"""
```

АИСД

Что такое АИСД?

АИСД - это алгоритмы и структуры данных. область информатики, изучающая методы организации, хранения и обработки данных с целью эффективного решения различных задач.

В АИСД входят алгоритмы (*последовательность действий для решения задачи*) и структуры данных (*способы организации информации для эффективного доступа и модификации*).

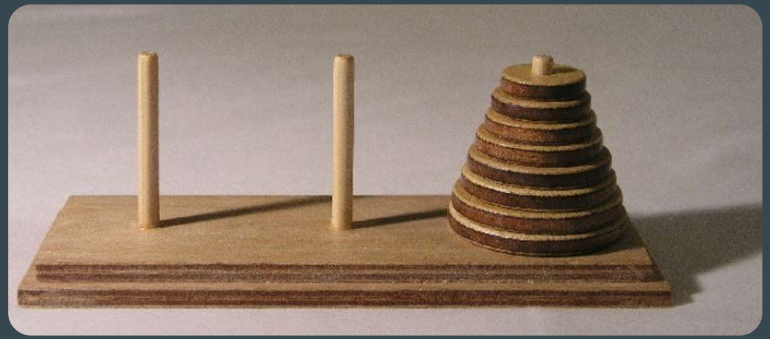
СТЕК ДАННЫХ

СТЕК ДАННЫХ

Стек данных - это структура данных, где элементы добавляются и удаляются только с одного конца (вершины) по принципу "последний вошел - первый вышел".

Last in, First out (LIFO)

ПРИМЕРЫ ИЗ ЖИЗНИ



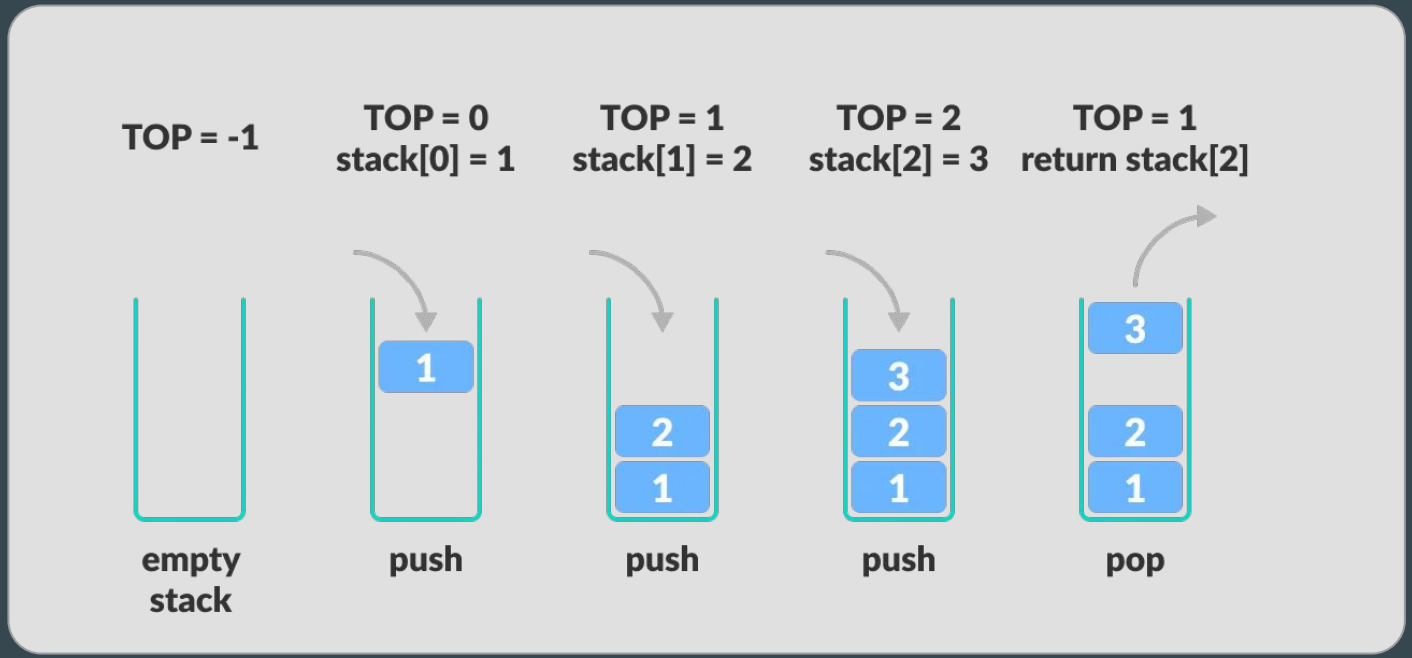
<stack data operations>

- **Push**: Добавляет элемент в вершину стека.
- **Pop**: Удаляет элемент из вершины стека и возвращает его значение.
- **Top**: Получает значение элемента на вершине стека без его удаления.
- **IsEmpty**: Проверяет, пуст ли стек.
- **IsFull**: Проверяет, заполнен ли стек (если у него фиксированный размер).
- **Size**: Получает количество элементов в стеке.
- **Clear**: Удаляет все элементы из стека.
- **Peek**: Просматривает значение элемента на определенной позиции (не удаляя его).
- **Copy**: Создает копию стека.
- **Reverse**: Переворачивает порядок элементов в стеке.

<basic stack class>

```
class Stack {
private:
    T stack[max_size];
    int top;
public:
    Stack() : top(-1) {}
};
```

<stack methods>



<stack class dynamic memory>

```
template <typename T>
class Stack {
private:
    T* stack;         // указатель на массив элементов
    size_t capacity; // текущая ёмкость стека
    size_t top; // индекс верхнего элемента

public:
    Stack() : stack(nullptr), capacity(0), top(0) {}
    ~Stack() { delete[] stack;} // освобождение динамической памяти
};
```

<stack-CDM> (method push)

```
void push(const T& value) {
        if (top == capacity) {
            // увеличение ёмкости стека в два раза при необходимости
            // выделение новой памяти
            // копирование старых данных
            // освобождение старой памяти
            // обновление указателя на данные
        }
        // после - добавление нового элемента
}
```

<stack-CDM> (method pop #1)

```
void pop() {
        if (top > 0) {
            --top; // уменьшение индекса верхнего элемента
        } else {
            std::cerr << "Stack is empty!\n";
        }
}
```

<stack-CDM> (method pop #2)

```
void pop() {
    if (is_empty()) {return;}
    --top;
}
```

<stack-CDM> (method is_empty)

```
bool is_empty() const {
    // проверка что наш top = 0 или не равен
}
```

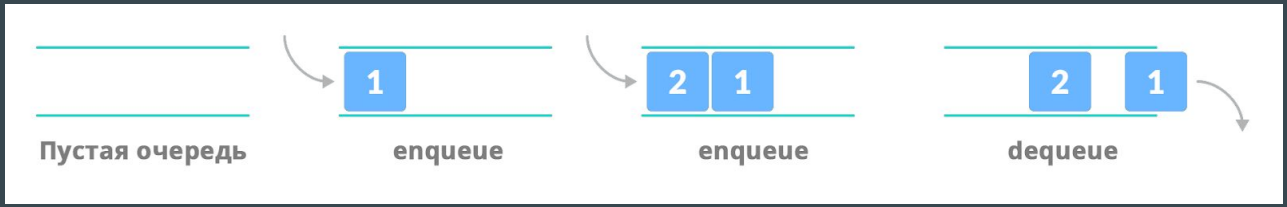
<stack-CDM> (method size)

```
size_t size() const {
    // вернуть просто top
}
```

ОЧЕРЕДЬ.

ОЧЕРЕДЬ

Очередь - структура данных FIFO (*первым пришёл, первым вышел*), где элементы добавляются в конец и удаляются из начала.



ОЧЕРЕДЬ ДАННЫХ

1. **enqueue (добавление в конец):** Добавляет элемент в конец очереди.
2. **dequeue (удаление из начала):** Удаляет элемент из начала очереди и возвращает его значение.
3. **front (первый элемент):** Возвращает значение первого элемента без его удаления.
4. **isEmpty (пуста ли очередь):** Проверяет, пуста ли очередь.
5. **size (размер очереди):** Возвращает количество элементов в очереди.

<base class queue>

```
class Queue {
private:
    T data[MAX_SIZE];
    size_t front;
    size_t rear;
    size_t count;

public:
    Queue() : front(0), rear(0), count(0) {}
```

ДЕК.

ДЕК

Дек (двусторонняя очередь) — это структура данных, поддерживающая вставку и удаление элементов с обоих концов.

ДЕК

- **push_back(value):** Добавляет элемент в конец дека.
- **push_front(value):** Добавляет элемент в начало дека.
- **pop_back():** Удаляет последний элемент из дека.
- **pop_front():** Удаляет первый элемент из дека.
- **front():** Возвращает ссылку на первый элемент дека.
- **back():** Возвращает ссылку на последний элемент дека.
- **size():** Возвращает количество элементов в деке.
- **empty():** Проверяет, пуст ли дек.
- **clear():** Удаляет все элементы из дека.
- **resize(new_size):** Изменяет размер дека, увеличивая или уменьшая его до указанного размера.

<base class deque>

```
template <typename T, int capacity>
class Deque {
private:
    T data[capacity];
    int front;
    int back;
    int size;

public:
    Deque() : front(0), back(0), size(0) {}
```

<deque> (method is_empty)

```
bool is_empty() const {
        return size == 0;
}
```

<deque> (method clear)

```
void clear() {
        front = back = size = 0;
}
```

<deque> (method size)

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