

# «ԾՐԱԳՐԱՎՈՐՄԱՆ ՀԻՄՈՒՆՔՆԵՐ» դասընթաց

այլ ոլորտներից դեպի տեխնոլոգիական ոլորտ  
սկսնակների համար



edu2020.am

## ԴԱՍ #14



ՀԱՅ-ՌՈՒՄԱԿԱՆ  
ՀԱՄԱԼՍԱՐԱՆ



ՀԱՅԱՍՏԱՆԻ ՀԱՆՐԱՊԵՏՈՒԹՅԱՆ  
ԲԱՐՁՐ ՏԵԽՆՈԼՈԳԻԱԿԱՆ  
ԱՐԴՅՈՒՆԱԲԵՐՈՒԹՅԱՆ ՆԱԽԱՐԱՐՈՒԹՅՈՒՆ

# Պահունակ (Stack)

- **Ստեկը** կառուցվածք է տվյալներ պահելու համար, որում տվյալների ավելացումը և հեռացումը կատարվում է հետևյալ կանոնով – **վերջին ավելացված էլեմենտը դուրս կգա առաջինը** (LIFO – Last In First Out)



# Պահունակ (Stack)

- **Ստեկը** կառուցվածք է տվյալներ պահելու համար, որում տվյալների ավելացումը և հեռացումը կատարվում է հետևյալ կանոնով – **վերջին ավելացված էլեմենտը դուրս կգա առաջինը** (LIFO – Last In First Out)
- Հիմնական գործողություններն են
  - push – ավելացնել էլեմենտ ստեկում
  - pop – ջնջել էլեմենտ ստեկից
  - top – վերադարձնել վերջին էլեմենտը
  - empty – վերադաձնում է true, եթե ստեկը դատարկ է, հակառակ դեպքում՝ false





# Պահույնակ (Stack)

```
class Stack {  
public:  
    Stack(int c);  
    Stack(Stack& st);  
    ~Stack();  
    bool empty();  
    int top();  
    void push(int obj);  
    void pop();  
private:  
    int stack_size;  
    int array_capacity;  
    int *array;  
};
```

<https://repl.it/@HaykAslanyan/stack>



# Պահույնակ (Stack)

```
int main() {  
    Stack st(5);  
    st.push(1);  
    st.push(2);  
    st.push(3);  
    std::cout << st.top() << "\n";  
    st.pop();  
    std::cout << st.top() << "\n";  
}
```



<https://repl.it/@HaykAslanyan/stack>



# Պահույնակ (Stack)

```
int main() {  
    Stack st(5);  
    st.push(1);  
    st.push(2);  
    st.push(3);  
    std::cout << st.top() << "\n";  
    st.pop();  
    std::cout << st.top() << "\n";  
}
```



<https://repl.it/@HaykAslanyan/stack>



# Պահույնակ (Stack)

```
int main() {  
    Stack st(5);  
    st.push(1);  
    st.push(2);  
    st.push(3);  
    std::cout << st.top() << "\n";  
    st.pop();  
    std::cout << st.top() << "\n";  
}
```



<https://repl.it/@HaykAslanyan/stack>



# Պահույնակ (Stack)

```
int main() {  
    Stack st(5);  
    st.push(1);  
    st.push(2);  
    st.push(3);  
    std::cout << st.top() << "\n";  
    st.pop();  
    std::cout << st.top() << "\n";  
}
```

1	2	3		
---	---	---	--	--



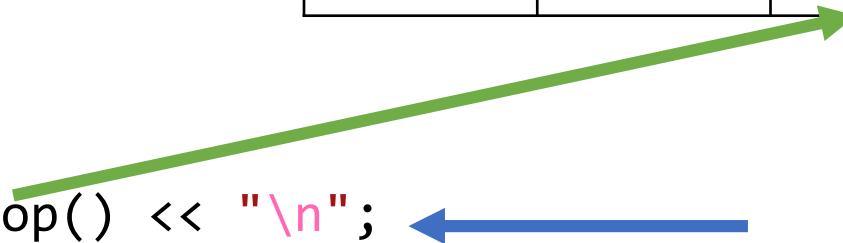
<https://repl.it/@HaykAslanyan/stack>





# Պահույնակ (Stack)

```
int main() {  
    Stack st(5);  
    st.push(1);  
    st.push(2);  
    st.push(3);  
    std::cout << st.top() << "\n";  
    st.pop();  
    std::cout << st.top() << "\n";  
}
```



<https://repl.it/@HaykAslanyan/stack>



# Պահույնակ (Stack)

```
int main() {  
    Stack st(5);  
    st.push(1);  
    st.push(2);  
    st.push(3);  
    std::cout << st.top() << "\n";  
    st.pop();  
    std::cout << st.top() << "\n";  
}
```

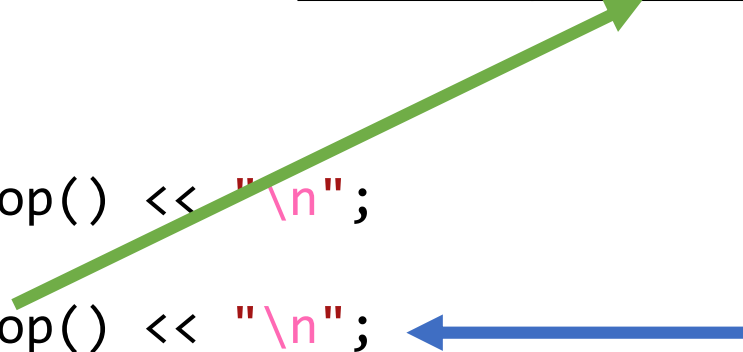


<https://repl.it/@HaykAslanyan/stack>



# Պահույնակ (Stack)

```
int main() {  
    Stack st(5);  
    st.push(1);  
    st.push(2);  
    st.push(3);  
    std::cout << st.top() << "\n";  
    st.pop();  
    std::cout << st.top() << "\n";  
}
```



<https://repl.it/@HaykAslanyan/stack>



# Պահույնակ (Stack)

```
class Stack {  
public:  
    Stack(int c);  
    Stack(Stack& st);  
    ~Stack();  
    bool empty();  
    int top();  
    void push(int obj);  
    void pop();  
private:  
    int stack_size;  
    int array_capacity;  
    int *array;  
};
```

// c should be greater than 0  
Stack::Stack(int c):  
 stack\_size(0),  
 array\_capacity(c),  
 array(new int[array\_capacity]) {}

<https://repl.it/@HaykAslanyan/stack>



# Պահույնակ (Stack)

```
class Stack {  
public:  
    Stack(int c);  
    Stack(Stack& st);  
    ~Stack();  
    bool empty();  
    int top();  
    void push(int obj);  
    void pop();  
private:  
    int stack_size;  
    int array_capacity;  
    int *array;  
};
```

```
// copy constructor  
Stack::Stack(Stack& st) :  
    stack_size(st.stack_size),  
    array_capacity(st.array_capacity),  
    array(new int[array_capacity]) {  
    for (int i = 0; i < stack_size; i++) {  
        array[i] = st.array[i];  
    }  
}
```

<https://repl.it/@HaykAslanyan/stack>





# Պահունակ (Stack)

```
class Stack {  
public:  
    Stack(int c);  
    Stack(Stack& st);  
    ~Stack();  
    bool empty();  
    int top();  
    void push(int obj);  
    void pop();  
private:  
    int stack_size;  
    int array_capacity;  
    int *array;  
};
```

```
Stack::~~Stack() {  
    delete [] array;  
}
```

<https://repl.it/@HaykAslanyan/stack>



# Պահունակ (Stack)

```
class Stack {  
public:  
    Stack(int c);  
    Stack(Stack& st);  
    ~Stack();  
    bool empty();  
    int top();  
    void push(int obj);  
    void pop();  
private:  
    int stack_size;  
    int array_capacity;  
    int *array;  
};
```

```
bool Stack::empty() {  
    return stack_size == 0;  
}
```

<https://repl.it/@HaykAslanyan/stack>



# Պահույնակ (Stack)

```
class Stack {  
public:  
    Stack(int c);  
    Stack(Stack& st);  
    ~Stack();  
    bool empty();  
    int top();  
    void push(int obj);  
    void pop();  
private:  
    int stack_size;  
    int array_capacity;  
    int *array;  
};
```

```
int Stack::top() {  
    assert (!empty());  
    return array[stack_size - 1];  
}
```

<https://repl.it/@HaykAslanyan/stack>



# Պահունակ (Stack)

```
class Stack {  
public:  
    Stack(int c);  
    Stack(Stack& st);  
    ~Stack();  
    bool empty();  
    int top();  
    void push(int obj);  
    void pop();  
private:  
    int stack_size;  
    int array_capacity;  
    int *array;  
};
```

```
void Stack::push(int obj) {  
    if (stack_size == array_capacity) {  
        return;  
    }  
    array[stack_size] = obj;  
    ++stack_size;  
}
```

<https://repl.it/@HaykAslanyan/stack>



# Պահունակ (Stack)

```
class Stack {  
public:  
    Stack(int c);  
    Stack(Stack& st);  
    ~Stack();  
    bool empty();  
    int top();  
    void push(int obj);  
    void pop();  
private:  
    int stack_size;  
    int array_capacity;  
    int *array;  
};
```

```
void Stack::pop() {  
    if (empty()) {  
        return;  
    }  
    --stack_size;  
}
```

<https://repl.it/@HaykAslanyan/stack>





# Պահույնակ (Stack)

```
int main() {  
    Stack st(5);  
    st.push(1);  
    st.push(2);  
    st.push(3);  
    std::cout << st.top() << "\n";  
    st.pop();  
    std::cout << st.top() << "\n";  
  
    Stack st2(st);  
    std::cout << st2.top() << "\n";  
}
```

<https://repl.it/@HaykAslanyan/stack>



# Պահունակի կիրառություն

- Փակագծերի ստուգում

```
void initialize( int *array, int n ) {  
    for ( int i = 0; i < n; ++i ) {  
        array[i] = 0;  
    }  
}
```



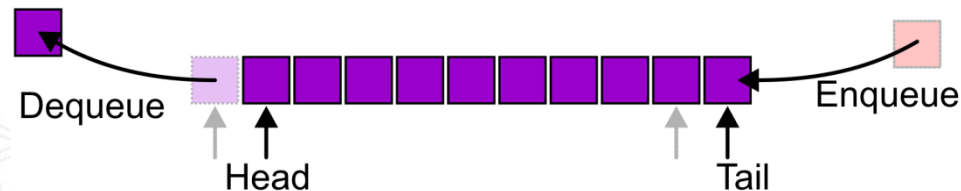
# Հերթ (Queue)

- **Հերթը** կառուցվածք է տվյալներ պահելու համար, որում տվյալների ավելացումը և հեռացումը կատարվում է հետևյալ կանոնով – **առաջին ավելացված էլեմենտը դուրս կգա առաջինը** (FIFO – first in first out)



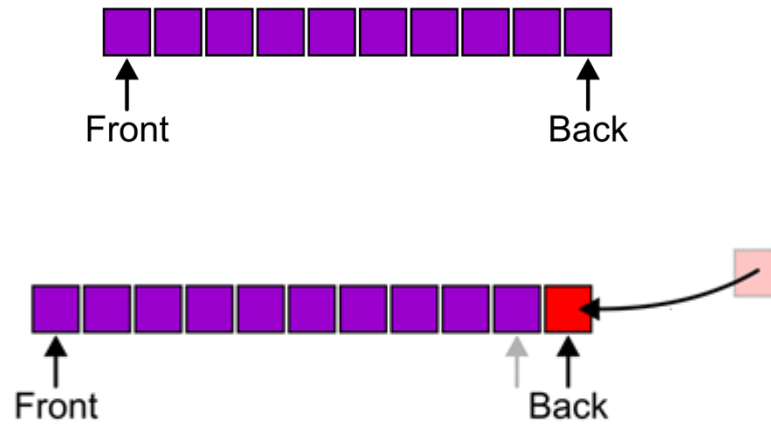
# Հերթ (Queue)

- **Հերթը** կառուցվածք է տվյալների պահելու համար, որում տվյալների ավելացումը և հեռացումը կատարվում է հետևյալ կանոնով – **առաջին ավելացված էլեմենտը դուրս կգա առաջինը (FIFO – first in first out)**
- Հիմնական գործողություններն են
  - enqueue (push) – ավելացնել էլեմենտ հերթում
  - dequeue (pop) – ջնջել էլեմենտ հերթից
  - front – վերադարձնել առաջին էլեմենտը
  - empty – վերադարձնում է true, եթե հերթը դատարկ է, հակառակ դեպքում՝ false

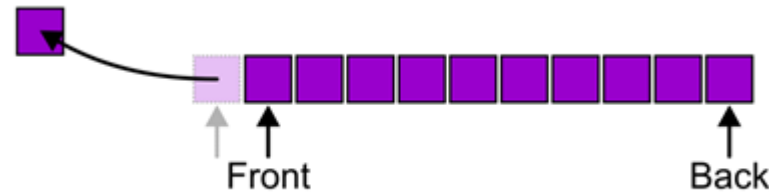


# Հերթ (Queue)

- enqueue (push)



- dequeue (pop)





# Հերթ (Queue)

```
class Queue {  
    public:  
        Queue(int n);  
        ~Queue();  
        bool empty();  
        int front();  
        void enqueue(int obj);  
        void dequeue();  
    private:  
        int queue_size;  
        int front_index; // first element index  
        int back_index; // last element index  
        int array_capacity;  
        int *array;  
};
```

<https://repl.it/@HaykAslanyan/queue>



# Հերթ (Queue)



```
int main() {  
    Queue queue(16);  
    for (int i = 1; i <= 16; i++) {  
        queue.enqueue(i);  
    }  
    std::cout << queue.front() << "\n";  
    for (int i = 1; i <= 5; i++) {  
        queue.dequeue();  
    }  
    std::cout << queue.front() << "\n";  
    queue.enqueue(17);  
    std::cout << queue.front() << "\n";  
}
```



<https://repl.it/@HaykAslanyan/queue>

# Հերթ (Queue)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

```
int main() {  
    Queue queue(16);  
    for (int i = 1; i <= 16; i++) { ←  
        queue.enqueue(i);  
    }  
    std::cout << queue.front() << "\\n";  
    for (int i = 1; i <= 5; i++) {  
        queue.dequeue();  
    }  
    std::cout << queue.front() << "\\n";  
    queue.enqueue(17);  
    std::cout << queue.front() << "\\n";  
}
```

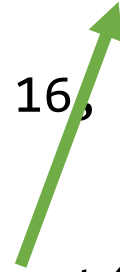


<https://repl.it/@HaykAslanyan/queue>

# Հերթ (Queue)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

```
int main() {  
    Queue queue(16);  
    for (int i = 1; i <= 16; i++) {  
        queue.enqueue(i);  
    }  
    std::cout << queue.front() << "\\n";  
    for (int i = 1; i <= 5; i++) {  
        queue.dequeue();  
    }  
    std::cout << queue.front() << "\\n";  
    queue.enqueue(17);  
    std::cout << queue.front() << "\\n";  
}
```



<https://repl.it/@HaykAslanyan/queue>

# Հերթ (Queue)

					6	7	8	9	10	11	12	13	14	15	16
--	--	--	--	--	---	---	---	---	----	----	----	----	----	----	----

```
int main() {  
    Queue queue(16);  
    for (int i = 1; i <= 16; i++) {  
        queue.enqueue(i);  
    }  
    std::cout << queue.front() << "\\n";  
    for (int i = 1; i <= 5; i++) { ←  
        queue.dequeue();  
    }  
    std::cout << queue.front() << "\\n";  
    queue.enqueue(17);  
    std::cout << queue.front() << "\\n";  
}
```



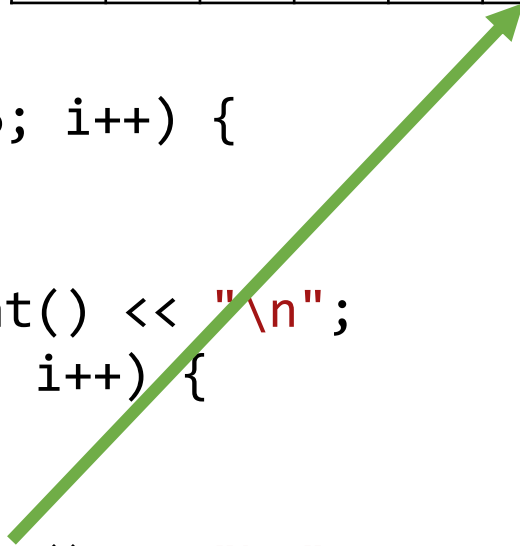
<https://repl.it/@HaykAslanyan/queue>



# Հերթ (Queue)

					6	7	8	9	10	11	12	13	14	15	16
--	--	--	--	--	---	---	---	---	----	----	----	----	----	----	----

```
int main() {  
    Queue queue(16);  
    for (int i = 1; i <= 16; i++) {  
        queue.enqueue(i);  
    }  
    std::cout << queue.front() << "\n";  
    for (int i = 1; i <= 5; i++) {  
        queue.dequeue();  
    }  
    std::cout << queue.front() << "\n";  
    queue.enqueue(17);  
    std::cout << queue.front() << "\n";  
}
```



# Հերթ (Queue)

17					6	7	8	9	10	11	12	13	14	15	16
----	--	--	--	--	---	---	---	---	----	----	----	----	----	----	----

```
int main() {  
    Queue queue(16);  
    for (int i = 1; i <= 16; i++) {  
        queue.enqueue(i);  
    }  
    std::cout << queue.front() << "\n";  
    for (int i = 1; i <= 5; i++) {  
        queue.dequeue();  
    }  
    std::cout << queue.front() << "\n";  
    queue.enqueue(17);  
    std::cout << queue.front() << "\n";  
}
```

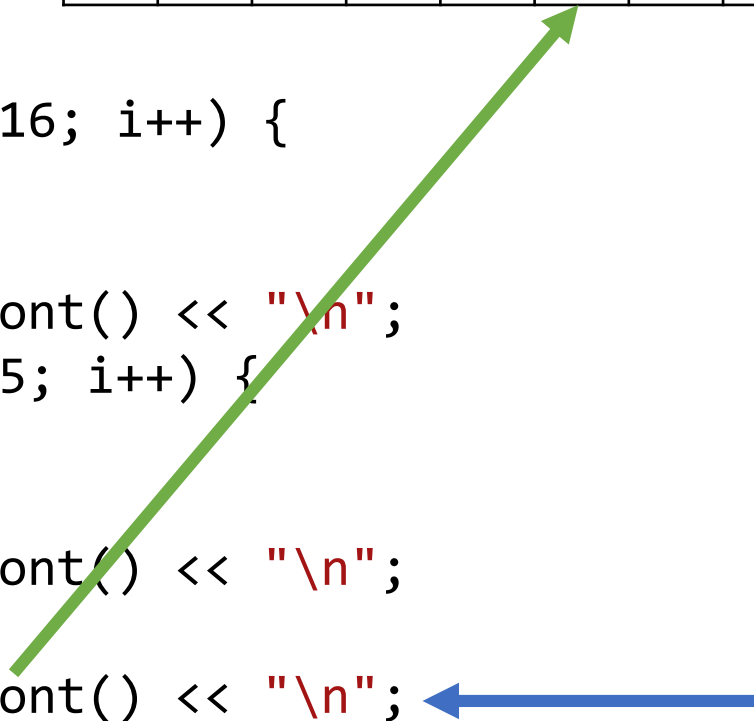


<https://repl.it/@HaykAslanyan/queue>

# Հերթ (Queue)

17					6	7	8	9	10	11	12	13	14	15	16
----	--	--	--	--	---	---	---	---	----	----	----	----	----	----	----

```
int main() {  
    Queue queue(16);  
    for (int i = 1; i <= 16; i++) {  
        queue.enqueue(i);  
    }  
    std::cout << queue.front() << "\n";  
    for (int i = 1; i <= 5; i++) {  
        queue.dequeue();  
    }  
    std::cout << queue.front() << "\n";  
    queue.enqueue(17);  
    std::cout << queue.front() << "\n";  
}
```



<https://repl.it/@HaykAslanyan/queue>

# Հերթ (Queue)

```
class Queue {  
public:  
    Queue(int n);  
    Queue(Queue& q);  
    ~Queue();  
    bool empty();  
    int front();  
    void enqueue(int obj);  
    void dequeue();  
private:  
    int queue_size;  
    int front_index; // first element index  
    int back_index; // last element index  
    int array_capacity;  
    int *array;
```

// n should be greater than 0  
Queue::Queue(int n) :  
 queue\_size(0),  
 front\_index(0),  
 back\_index(-1),  
 array\_capacity(n),  
 array(new int[array\_capacity]) {}

```
};
```

<https://repl.it/@HaykAslanyan/queue>



# Հերթ (Queue)

```
class Queue {
public:
    Queue(int n);
    Queue(Queue& q);
    ~Queue();
    bool empty();
    int front();
    void enqueue(int obj);
    void dequeue();
private:
    int queue_size;
    int front_index; // first element index
    int back_index; // last element index
    int array_capacity;
    int *array;
};
```

```
Queue::Queue(Queue& q) :
    queue_size(q.queue_size),
    front_index(q.front_index),
    back_index(q.back_index),
    array_capacity(q.array_capacity),
    array(new int[array_capacity]) {
    if (q.empty()) {
        return;
    }
    int i = front_index;
    do {
        array[i] = q.array[i];
        i++;
        if (i == array_capacity) {
            i = 0;
        }
    } while (i != back_index);
}
```

<https://repl.it/@HaykAslanyan/queue>



# Հերթ (Queue)

```
class Queue {  
public:  
    Queue(int n);  
    Queue(Queue& q);  
    ~Queue();  
    bool empty();  
    int front();  
    void enqueue(int obj);  
    void dequeue();  
private:  
    int queue_size;  
    int front_index; // first element index  
    int back_index; // last element index  
    int array_capacity;  
    int *array;  
};
```

```
Queue::~~Queue() {  
    delete [] array;  
}
```

<https://repl.it/@HaykAslanyan/queue>



# Հերթ (Queue)

```
class Queue {  
    public:  
        Queue(int n);  
        Queue(Queue& q);  
        ~Queue();  
        bool empty();  
        int front();  
        void enqueue(int obj);  
        void dequeue();  
    private:  
        int queue_size;  
        int front_index; // first element index  
        int back_index; // last element index  
        int array_capacity;  
        int *array;
```

```
bool Queue::empty() {  
    return queue_size == 0;  
}
```

```
};
```

<https://repl.it/@HaykAslanyan/queue>





# Հերթ (Queue)

```
class Queue {  
public:  
    Queue(int n);  
    Queue(Queue& q);  
    ~Queue();  
    bool empty();  
    int front();  
    void enqueue(int obj);  
    void dequeue();  
private:  
    int queue_size;  
    int front_index; // first element index  
    int back_index; // last element index  
    int array_capacity;  
    int *array;
```

```
int Queue::front() {  
    assert (!empty());  
    return array[front_index];  
}
```

```
};
```

<https://repl.it/@HaykAslanyan/queue>



# Հերթ (Queue)

```
class Queue {  
public:  
    Queue(int n);  
    Queue(Queue& q);  
    ~Queue();  
    bool empty();  
    int front();  
    void enqueue(int obj);  
    void dequeue();  
private:  
    int queue_size;  
    int front_index; // first element  
    int back_index; // last element  
    int array_capacity;  
    int *array;
```

```
void Queue::enqueue(int obj) {  
    if (queue_size == array_capacity) {  
        return;  
    }  
    ++back_index;  
    if (back_index == array_capacity) {  
        back_index = 0;  
    }  
    array[back_index] = obj;  
    ++queue_size;  
}
```

<https://repl.it/@HaykAslanyan/queue>



# Հերթ (Queue)

```
class Queue {  
public:  
    Queue(int n);  
    Queue(Queue& q);  
    ~Queue();  
    bool empty();  
    int front();  
    void enqueue(int obj);  
    void dequeue();  
private:  
    int queue_size;  
    int front_index; // first element  
    int back_index; // last element  
    int array_capacity;  
    int *array;
```

```
void Queue::dequeue() {  
    if (empty()) {  
        return;  
    }  
    --queue_size;  
    ++front_index;  
    if (front_index == array_capacity) {  
        front_index = 0;  
    }  
}
```



<https://repl.it/@HaykAslanyan/queue>

# Հերթ (Queue)

```
int main() {  
    Queue queue(16);  
    for (int i = 1; i <= 16; i++) {  
        queue.enqueue(i);  
    }  
    std::cout << queue.front() << "\\n";  
    for (int i = 1; i <= 5; i++) {  
        queue.dequeue();  
    }  
    std::cout << queue.front() << "\\n";  
    queue.enqueue(17);  
    std::cout << queue.front() << "\\n";  
}
```

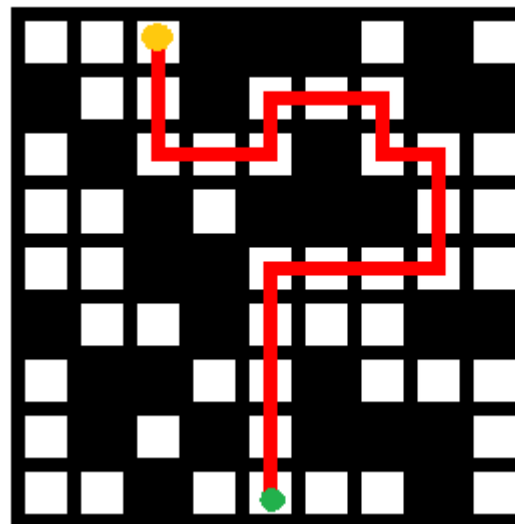






<https://repl.it/@HaykAslanyan/queue>

# Ալիքի ալգորիթ

Տրված է վանդակավոր ուղղանկյուն տախտակ (դաշտ):  
Վանդակները երկու տիպի են՝ ազատ և խոչընդոտ (կամ հող  
և ջուր)

Անհրաժեշտ է գտնել տրված երկու ազատ վանդակների միջև  
եղած ամենակարճ ճանապարհը, որն անցնում է  
բացառապես ազատ վանդակներով:



-  Ազատ
-  Խոչընդոտ
-  Սկիզբ
-  Վերջնակետ

# Ալիքի ալգորիթմ

Քայլերի հաջորդականություն.

- Սկսել տրված ազատ վանդակից՝ ավելացնել հերթի մեջ և այդ վանդակը ներկել (օրինակ վերագրել 1)
- Նշել նրա հարևան ազատ վանդակները (դա կհամարվի առաջին ալիքը)
- շարունակել նշելը արդեն նշված վանդակների հարևան ազատ վանդակները, մինչև ալիքի հասնելը տրված վերջնակետին:

Ալգորիթմը իրականացվում է հերթ (Queue) կառուցվածքի միջոցով:



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	0	0
0	0	0	0
0	1	1	0

Steps

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

Queue

{0, 0}				
--------	--	--	--	--

Կապույտ - դիտարկվող էլեմենտ

Կարմիր – հերթում գտնվող էլեմենտ

Կանաչ – արդեն դիտարկված էլեմենտ





# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	-1	-1
1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

Queue

{1, 0}	{0, 1}			
--------	--------	--	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	-1	-1
1	-1	-1	-1
2	-1	-1	-1
-1	-1	-1	-1

Queue

{0, 1}	{2, 0}			
--------	--------	--	--	--

Կապույտ - դիտարկվող էլեմենտ

Կարմիր – հերթում գտնվող էլեմենտ

Կանաչ – արդեն դիտարկված էլեմենտ



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	-1
2	-1	-1	-1
-1	-1	-1	-1

Queue

{2, 0}	{0, 2}			
--------	--------	--	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	-1
2	3	-1	-1
3	-1	-1	-1

Queue

{0, 2}	{3, 0}	{2, 1}		
--------	--------	--------	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	-1
2	3	-1	-1
3	-1	-1	-1

Queue

{3, 0}	{2, 1}			
--------	--------	--	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	-1
2	3	-1	-1
3	-1	-1	-1

Queue

{2, 1}				
--------	--	--	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	-1
2	3	4	-1
3	-1	-1	-1

Queue

{2, 2}				
--------	--	--	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ





# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	-1
2	3	4	5
3	-1	-1	-1

Queue

{2, 3}				
--------	--	--	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	6
2	3	4	5
3	-1	-1	6

Queue

{3, 3}	{1, 3}			
--------	--------	--	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	6
2	3	4	5
3	-1	-1	6

Queue

{1, 3}				
--------	--	--	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ



# Ալիքի ալգորիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	6
2	3	4	5
3	-1	-1	6

Queue

--	--	--	--	--

**Կապույտ** - դիտարկվող էլեմենտ

**Կարմիր** – հերթում գտնվող էլեմենտ

**Կանաչ** – արդեն դիտարկված էլեմենտ



# Ալիքի ակտրիթ

Road

0	0	0	1
0	1	1	0
0	0	0	0
0	1	1	0

Steps

0	1	2	-1
1	-1	-1	6
2	3	4	5
3	-1	-1	6

Queue

--	--	--	--	--



# Տնային աշխատանք

## Տնային աշխատանք 9-11

### Վարժություններ

- 00 [Նախազարգացում](#)
- 01 [Թվաբանություն և ճյուղավորում](#)
- 02 [Ցիկլեր և ստատիկ զանգվածներ](#)
- 03 [Դինամիկ զանգվածներ և ֆունկցիաներ](#)
- 04 [Դասեր](#)



# Շնորհակալություն. Հարցե՞ր