Languages

Properties

The following table summarizes all available integer types and their properties:

Type specifier	Equivalent type	Width in bits by data model				
		C++ standard	LP32	ILP32	LLP64	LP64
`short`	`short int`	at least **16**	**16**	**16**	**16**	**16**
`short int`						
`signed short`						
`signed short int`						
`unsigned short`	`unsigned short int`					
`unsigned short int`						
`int`	`int`	at least **16**	**16**	**32**	**32**	**32**
`signed`						
`signed int`						
`unsigned`	`unsigned int`					
`unsigned int`						
`long`	`long int`	at least **32**	**32**	**32**	**32**	**64**
`long int`						
`signed long`						
`signed long int`						
`unsigned long`	`unsigned long int`					
`unsigned long int`						
`long long`	`long long int` (C++11)	at least **64**	**64**	**64**	**64**	**64**
`long long int`						
`signed long long`						
`signed long long int`						
`unsigned long long`	`unsigned long long int` (C++11)					
`unsigned long long int`						

Specifier	Common Equivalent	Signing	Bits	Bytes	Minimum Value	Maximum Value
int8_t	signed char	Signed	8	1	−128	127
uint8_t	unsigned char	Unsigned	8	1	0	255
int16_t	short	Signed	16	2	−32,768	32,767
uint16_t	unsigned short	Unsigned	16	2	0	65,535
int32_t	long	Signed	32	4	−2,147,483,648	2,147,483,647
uint32_t	unsigned long	Unsigned	32	4	0	4,294,967,295
int64_t	long long	Signed	64	8	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	unsigned long long	Unsigned	64	8	0	18,446,744,073,709,551,615

```cpp
#include <iostream>
#include <limits>

using namespace std;

int main() {
    int lowest_int = std::numeric_limits<int>::lowest();
    int min_int = std::numeric_limits<int>::min();
    int max_int = std::numeric_limits<int>::max();
    unsigned int lowest_uint = std::numeric_limits<unsigned int>::lowest();
    unsigned int min_uint = std::numeric_limits<unsigned int>::min();
    unsigned int max_uint = std::numeric_limits<unsigned int>::max();
    cout << lowest_int << " " << min_int << " " << max_int << "\n";
    cout << lowest_uint << " " << min_uint << " " << max_uint << "\n";
    return 0;
}
```

Success #stdin #stdout 0.01s 5480KB

stdin

```
Standard input is empty
```

stdout

```
-2147483648 -2147483648 2147483647
0 0 4294967295
```

```cpp
#include <iostream>
#include <limits>

using namespace std;

int main() {
    float lowest_float = std::numeric_limits<float>::lowest();
    float min_float = std::numeric_limits<float>::min();
    float max_float = std::numeric_limits<float>::max();
    cout << lowest_float << " " << min_float << " " << max_float << "\n";
    return 0;
}
```

Success #stdin #stdout 0.01s 5520KB

stdin

```
Standard input is empty
```

stdout

```
-3.40282e+38 1.17549e-38 3.40282e+38
```

```cpp
#include <iostream>
#include <limits>

using namespace std;

int main() {
    int min_int = std::numeric_limits<int>::min();
    int max_int = std::numeric_limits<int>::max();
    unsigned int min_uint = std::numeric_limits<unsigned int>::min();
    unsigned int max_uint = std::numeric_limits<unsigned int>::max();
    cout << min_int << " " << max_int << "\n";
    cout << min_uint << " " << max_uint << "\n";
    cout << min_int - 1 << " " << max_int + 1 << "\n"; // UB
    cout << min_uint - 1 << " " << max_uint + 1 << "\n";
    return 0;
}
```

Success #stdin #stdout 0.01s 5528KB

stdin

```
Standard input is empty
```

stdout

```
-2147483648 2147483647
0 4294967295
2147483647 -2147483648
4294967295 0
```

```
1. a = (10**100000000)
2. b = (10**100000000)
3. print(a * b > 0)
4.
5.
```

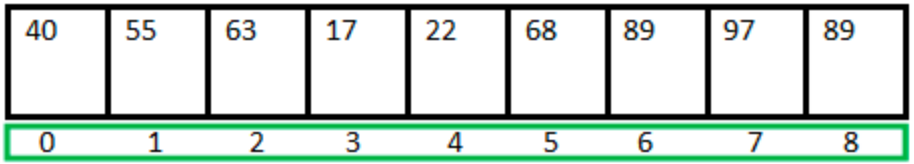
Time limit exceeded #stdin #stdout 5s 21620KB

stdin

```
Standard input is empty
```

stdout

```
Standard output is empty
```



Array Length = 9
First Index = 0
Last Index = 8

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a_old[8]; // old way
    array<int, 8> a_new; // new way
    sort(a_old, a_old + 8);
    sort(a_new.begin(), a_new.end());
    return 0;
}
```

Success #stdin #stdout 0.01s 5504KB

stdin

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```python
a = [1, 2, 3]
b = list({1, 2, 3})
a.append(4)
b.append(4)
print(a)
print(b)
```

Success #stdin #stdout 0.04s 9440KB

stdin

```
Standard input is empty
```

stdout

```
[1, 2, 3, 4]
[1, 2, 3, 4]
```

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    vector<int> a = {1, 2, 3};
    a.push_back(4);
    for (auto elem: a) {
        cout << elem << " ";
    }
}
```

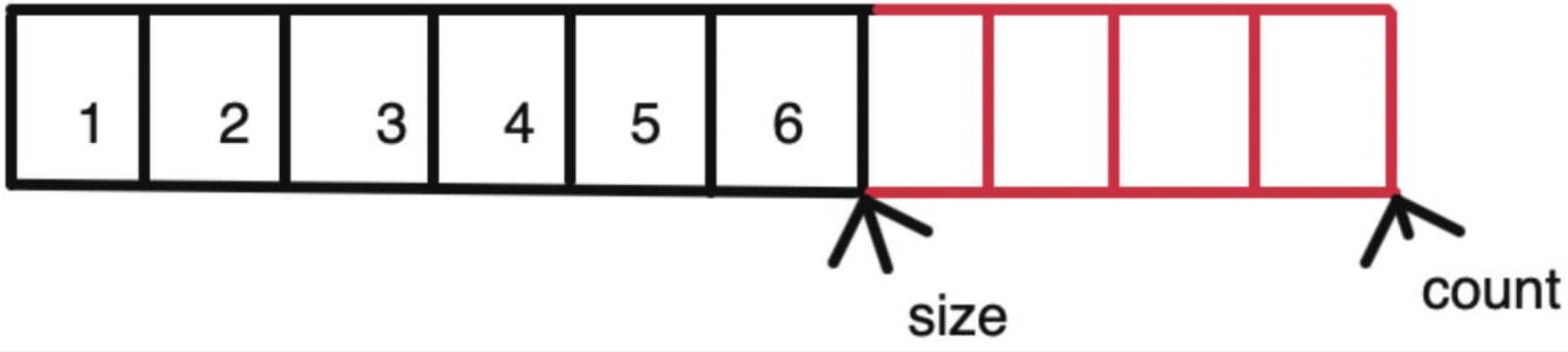
Success #stdin #stdout 0.01s 5544KB

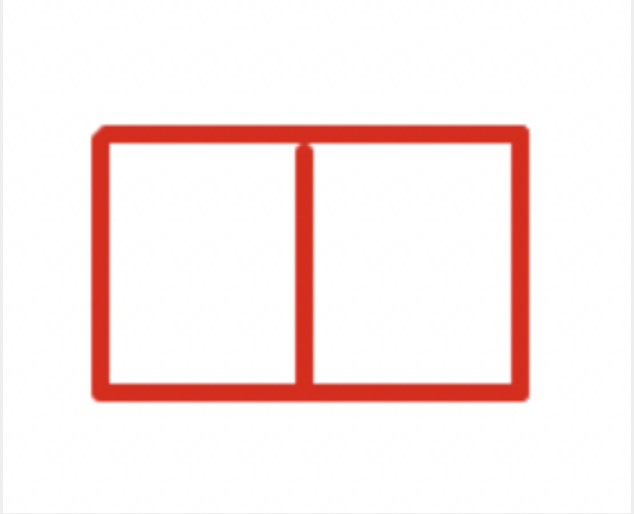
stdin

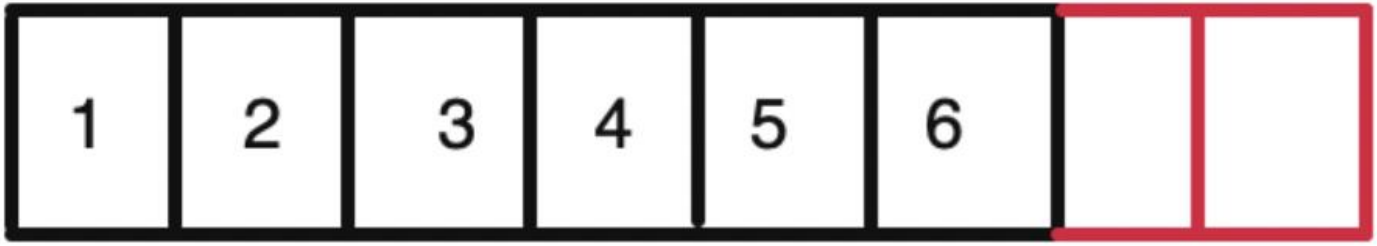
```
Standard input is empty
```

stdout

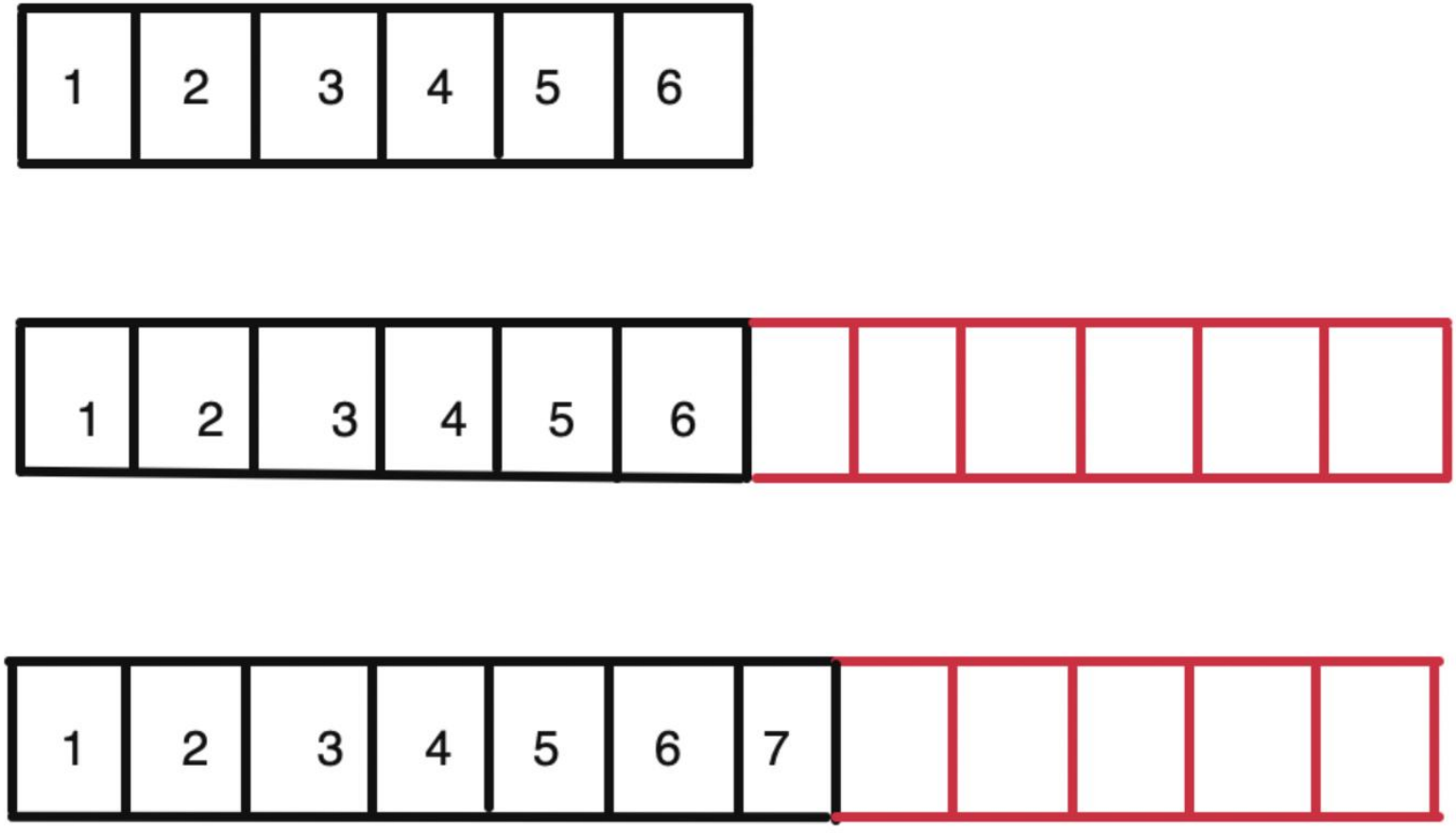
```
1 2 3 4
```


1
2
3
4
5
6
size
count




1
2
3
4
5
6


1
2
3
4
5
6
7


1
2
3
4
5
6
1
2
3
4
5
6
1
2
3
4
5
6
7

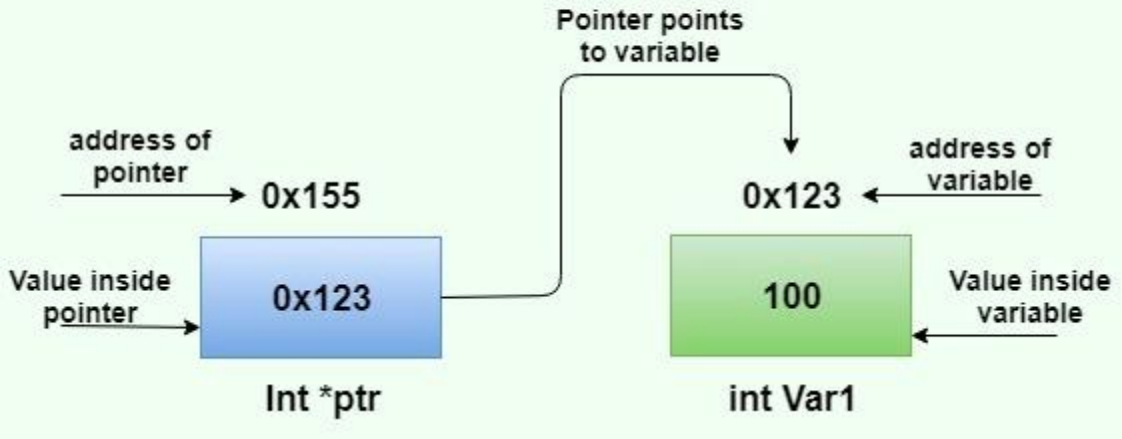
Some knowledge

- $1 + 2 + 4 + .. + 2^i = 2^{(i + 1)} - 1$
- $1 + 2 + 4 + .. + 2^{(\log(n))} = 2^{(\log(n) + 1)} - 1 = n * 2 - 1 = O(n)$

Vector

- In the first option we just insert element ⇒ O(1)
- In the second option we need to remove old array of size n, create new array of size 2n and insert element ⇒ O(n)
- We will increase array, when size = 2, 4, 8, 16, 32 …
- Asymptotic = O(1) * n + ΣO(i) for i in (2, 4, 8, …) = O(n)
- Amortized analysis

Pointers in C++



```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int a = 10;
    int *ptr_a = &a;
    cout << ptr_a << "\n";
    *ptr_a = 11;
    cout << a << "\n";
    int **ptr_ptr_a = &ptr_a;
    cout << ptr_ptr_a << "\n";
    cout << *ptr_ptr_a << "\n";
    cout << **ptr_ptr_a << "\n";
    return 0;
}
```

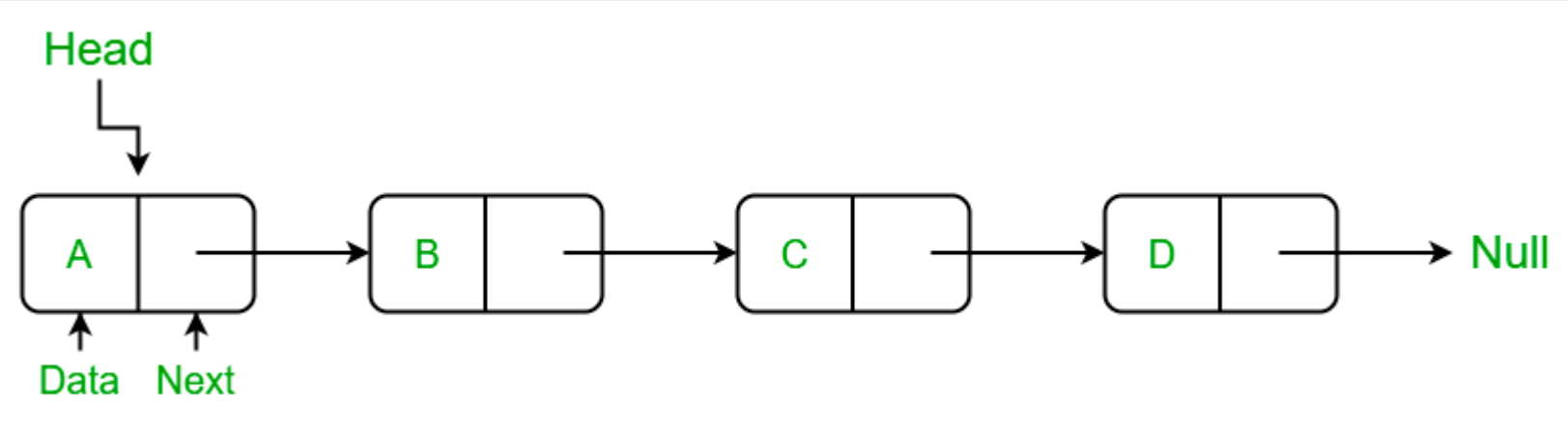
Success #stdin #stdout 0.01s 5408KB

stdin

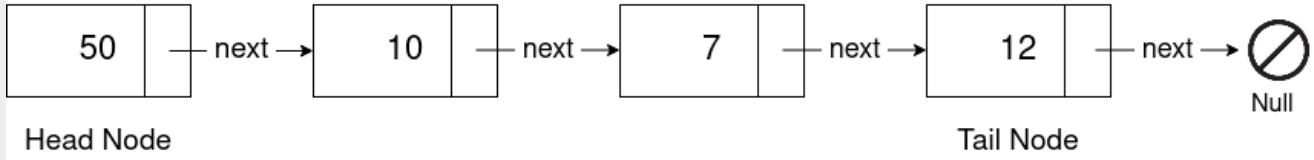
```
Standard input is empty
```

stdout

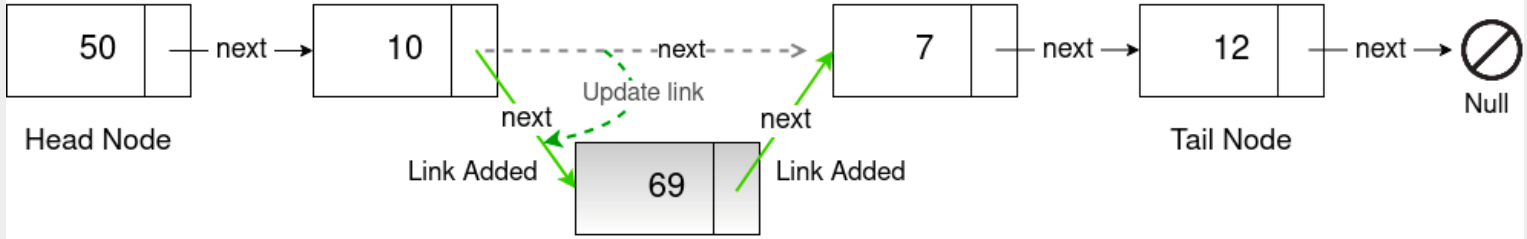
```
0x7fff0f2a39dc
11
0x7fff0f2a39e0
0x7fff0f2a39dc
11
```


Head
A
B
C
D
Null
Data
Next

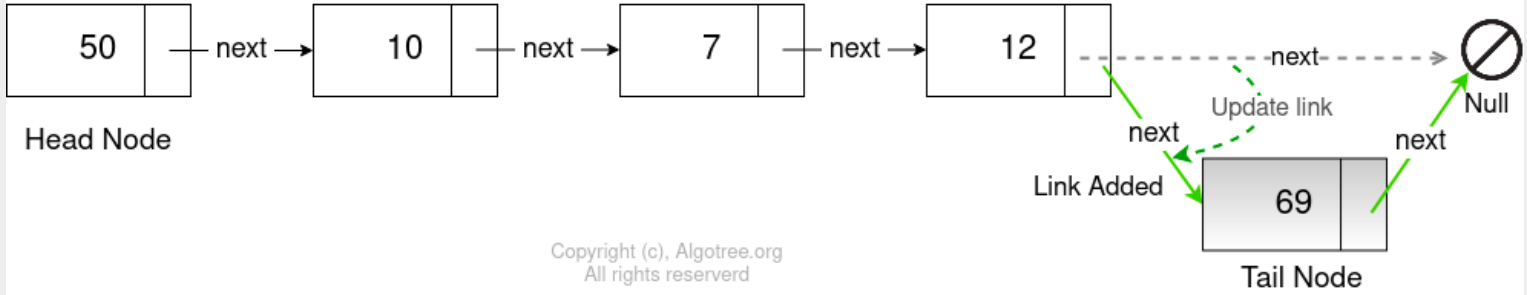
Singly Linked List with integer data



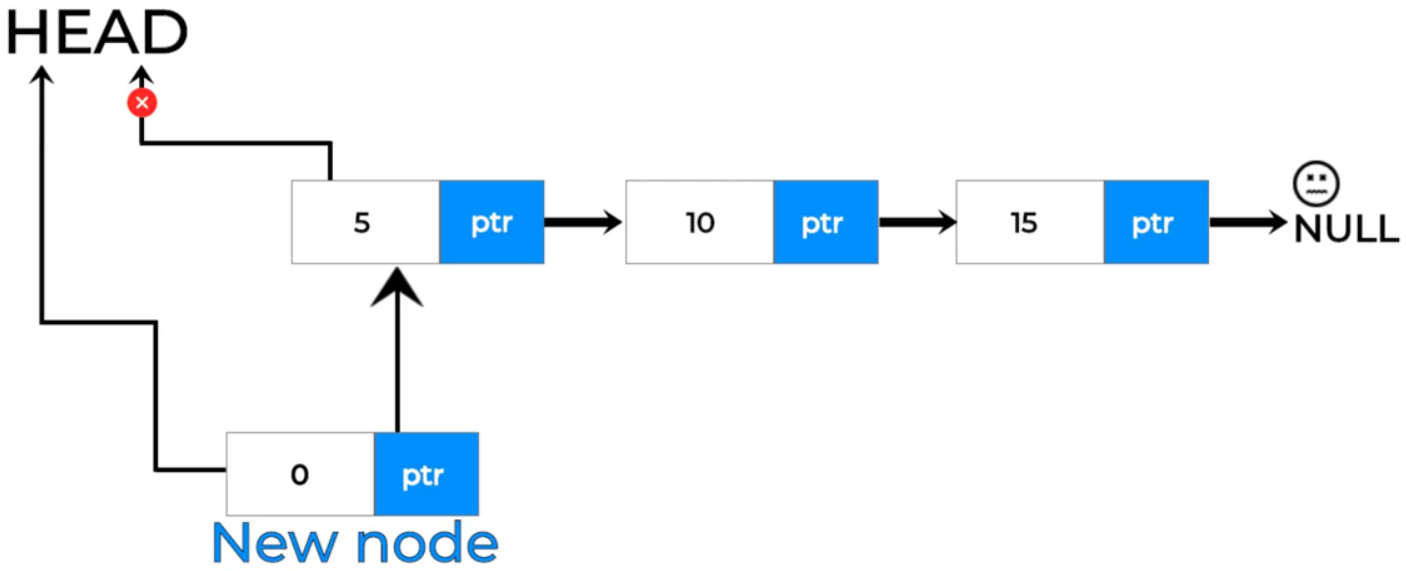
Singly Linked List *insert* operation



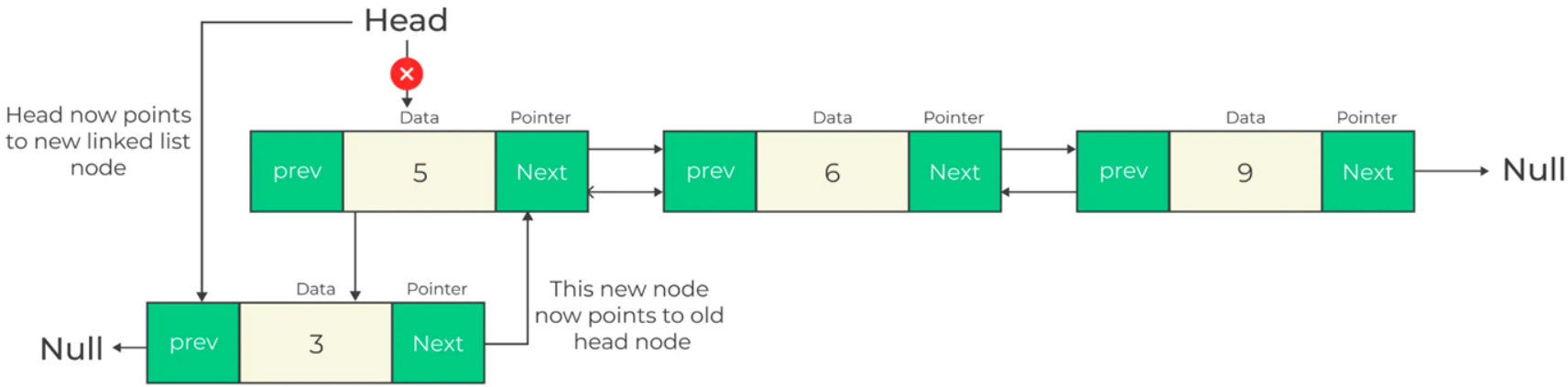
Singly Linked List *append* operation



Insertion at Beginning in Singly Linked list in C++



Insertion At Beginning in Doubly Linked List in C



```
5.  struct node {
6.      int data;
7.      node *next;
8.  };
9.
10. class linked_list {
11. private:
12.     node *head,*tail;
13. public:
14.     linked_list() {
15.         head = nullptr;
16.         tail = nullptr;
17.     }
18.
19.     void push_back(int n) {
20.         node *new_node = new node;
21.         new_node->data = n;
22.         new_node->next = nullptr;
23.
24.         if(head == nullptr) {
25.             head = new_node;
26.             tail = new_node;
27.         }
28.         else {
29.             tail->next = new_node;
30.             tail = tail->next;
31.         }
32.     }
33.
```

```
    void push_front(int n) {
        node *new_node = new node;
        new_node->data = n;
        new_node->next = nullptr;

        if(head == nullptr) {
            head = new_node;
            tail = new_node;
        }
        else {
            new_node->next = head;
            head = new_node;
        }
    }

    void print() {
        auto pointer = head;
        while (pointer) {
            cout << pointer->data << " ";
            pointer = pointer->next;
        }
        cout << "\n";
    }

};

int main()
{
    linked_list a;
    a.push_back(1);
    a.push_front(2);
    a.push_front(3);
    a.print();
    return 0;
}
```

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

    def __repr__(self):
        return self.data

class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def __repr__(self):
        node = self.head
        nodes = []
        while node is not None:
            nodes.append(node.data)
            node = node.next
        nodes.append("None")
        return " -> ".join(nodes)

    def push_back(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = self.tail = new_node
        else:
            self.tail.next = new_node
            self.tail = new_node
```

```
    def push_front(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = self.tail = new_node
        else:
            new_node.next = self.head
            self.head = new_node

llist = LinkedList()
llist.push_back('1')
llist.push_front('2')
llist.push_front('3')
print(llist)
```

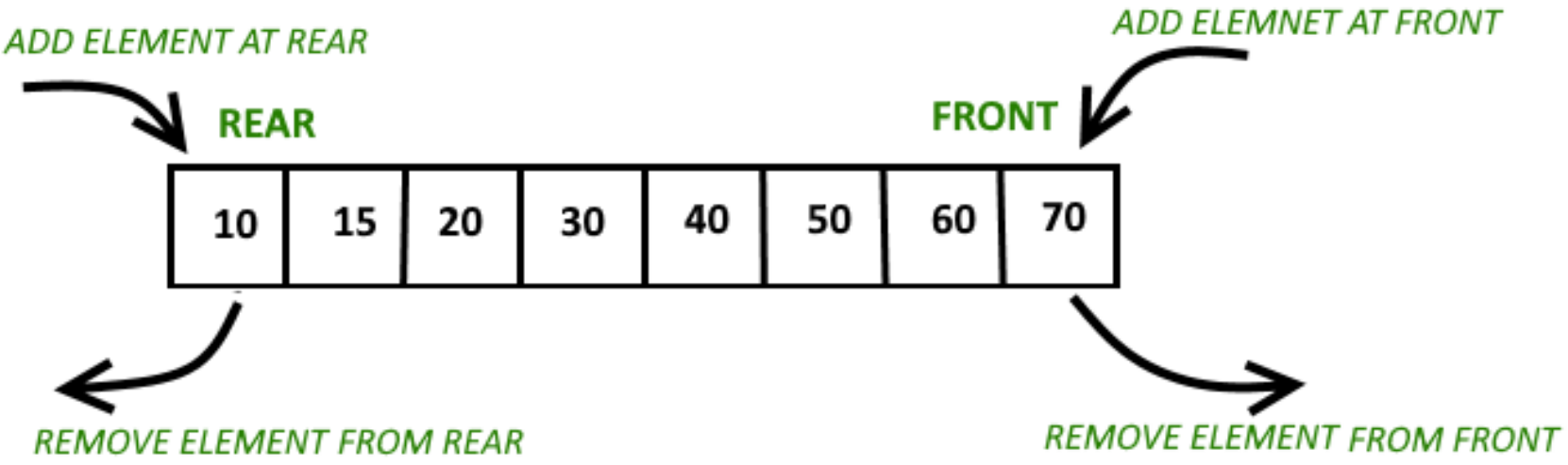
Success #stdin #stdout 0.03s 9568KB

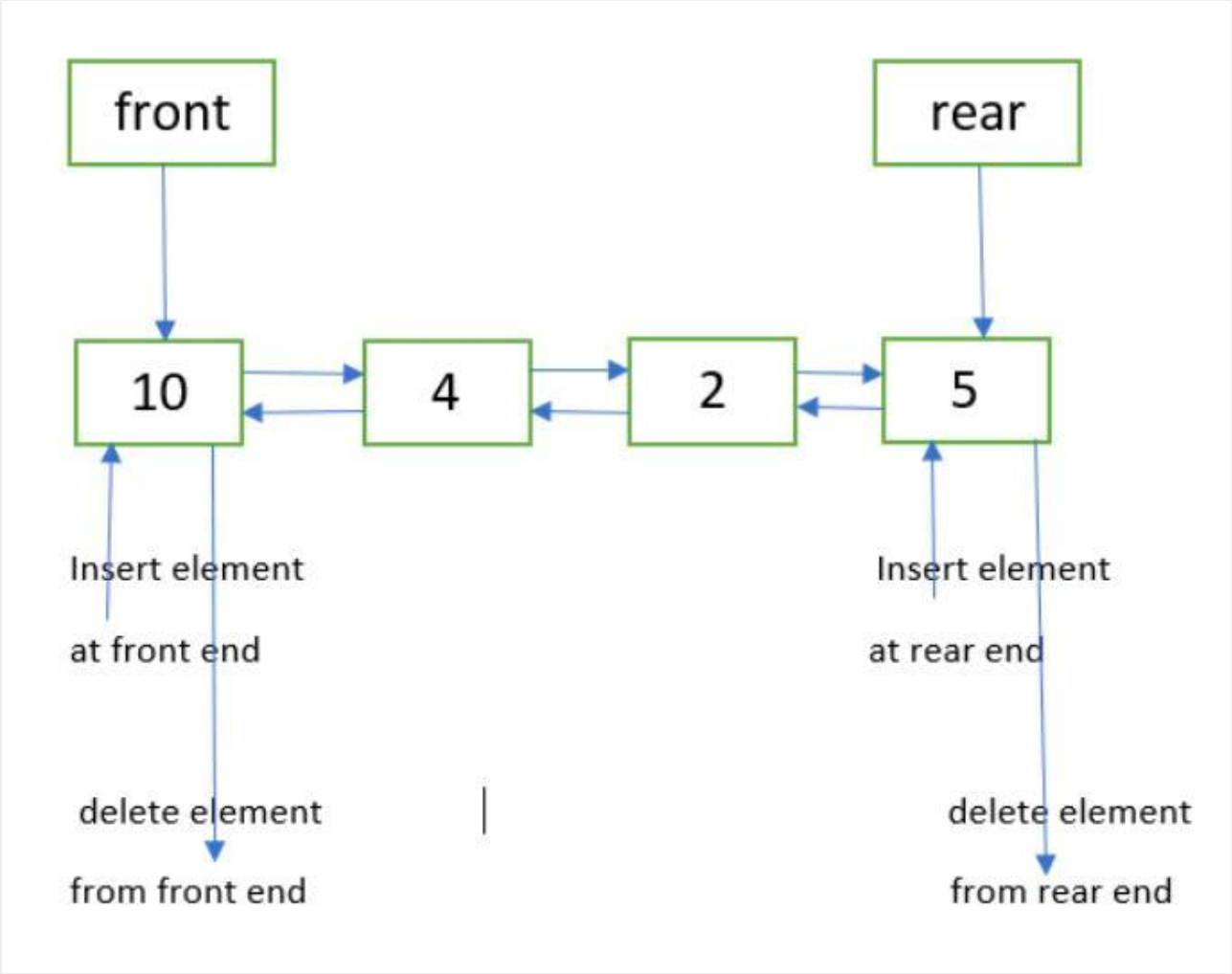
stdin

```
Standard input is empty
```

stdout

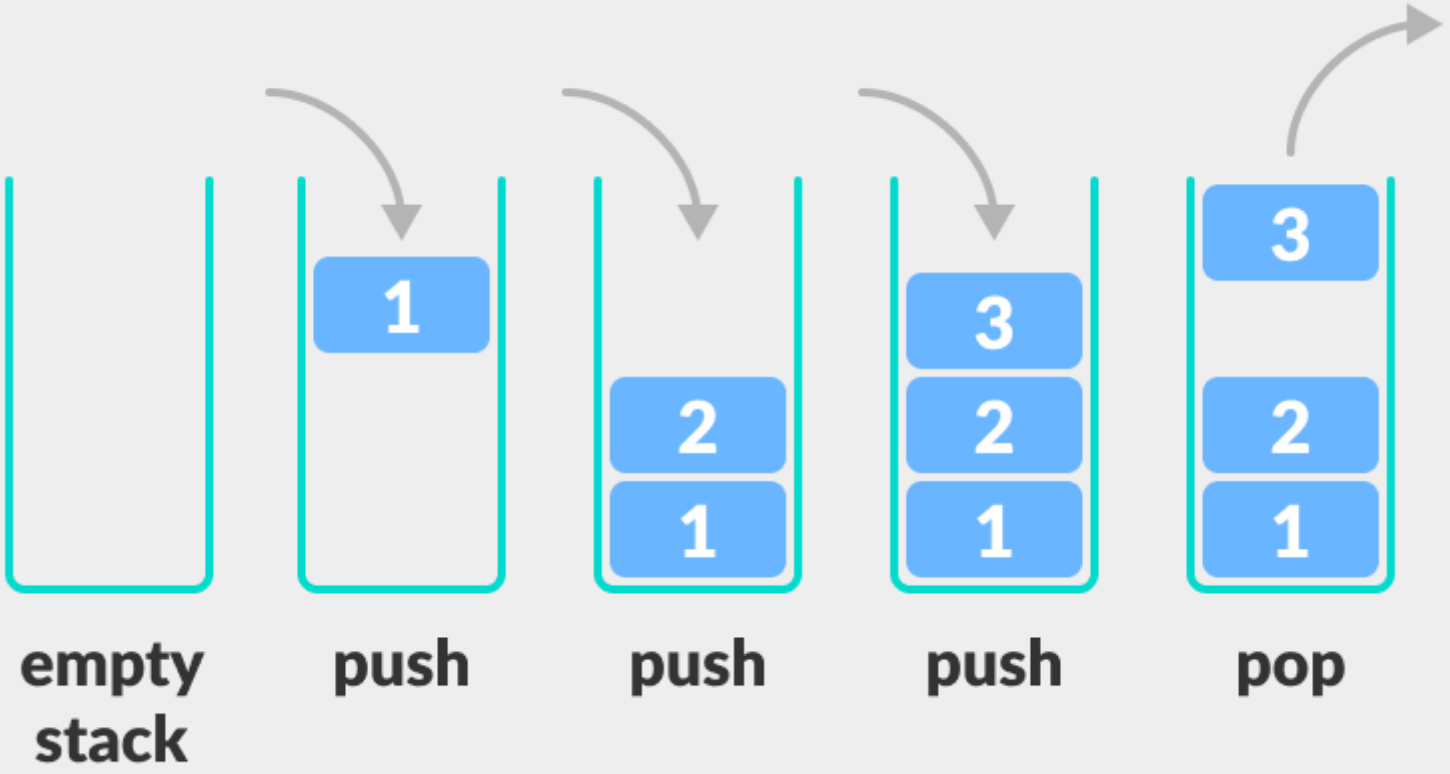
```
3 -> 2 -> 1 -> None
```


ADD ELEMENT AT REAR
ADD ELEMNET AT FRONT
REAR
FRONT
10
15
20
30
40
50
60
70
REMOVE ELEMENT FROM REAR
REMOVE ELEMENT FROM FRONT


front
rear
10
4
2
5
Insert element
at front end
Insert element
at rear end
delete element
from front end
delete element
from rear end

```
>>> from collections import deque
>>> d = deque('ghi')                 # make a new deque with three items
>>> for elem in d:                   # iterate over the deque's elements
...     print(elem.upper())
G
H
I

>>> d.append('j')                    # add a new entry to the right side
>>> d.appendleft('f')                # add a new entry to the left side
>>> d                                # show the representation of the deque
deque(['f', 'g', 'h', 'i', 'j'])

>>> d.pop()                          # return and remove the rightmost item
'j'
>>> d.popleft()                      # return and remove the leftmost item
'f'
>>> list(d)                          # list the contents of the deque
['g', 'h', 'i']
>>> d[0]                             # peek at leftmost item
'g'
>>> d[-1]                            # peek at rightmost item
'i'
```

```cpp
#include <deque>
#include <iostream>

int main()
{
    // Create a deque containing integers
    std::deque<int> d = {7, 5, 16, 8};

    // Add an integer to the beginning and end of the deque
    d.push_front(13);
    d.push_back(25);

    // Iterate and print values of deque
    for (int n : d)
        std::cout << n << ' ';
    std::cout << '\n';
}
```


1
2
1
3
2
1
3
2
1
empty stack
push
push
push
pop

std::stack

Defined in header <stack>

```
template<
    class T,
    class Container = std::deque<T>
> class stack;
```

```python
from collections import deque
stack_deque = deque()

stack_deque.append(1)
stack_deque.append(2)

print(stack_deque)

print(stack_deque.pop())
print(stack_deque.pop())

print(stack_deque)

stack_list = []
stack_list.append(1)
stack_list.append(2)

print(stack_list)

print(stack_list.pop())
print(stack_list.pop())

print(stack_list)
```

Success #stdin #stdout 0.03s 9644KB

comments (0)

stdin

copy

```
Standard input is empty
```

stdout

copy

```
deque([1, 2])
2
1
deque([])
[1, 2]
2
1
[]
```

Queue Data Structure

std::queue

Defined in header <queue>

```
template<
    class T,
    class Container = std::deque<T>
> class queue;
```

```python
from collections import deque
queue_deque = deque()

queue_deque.append(1)
queue_deque.append(2)

print(queue_deque)

print(queue_deque.popleft())
print(queue_deque.popleft())

print(queue_deque)
```

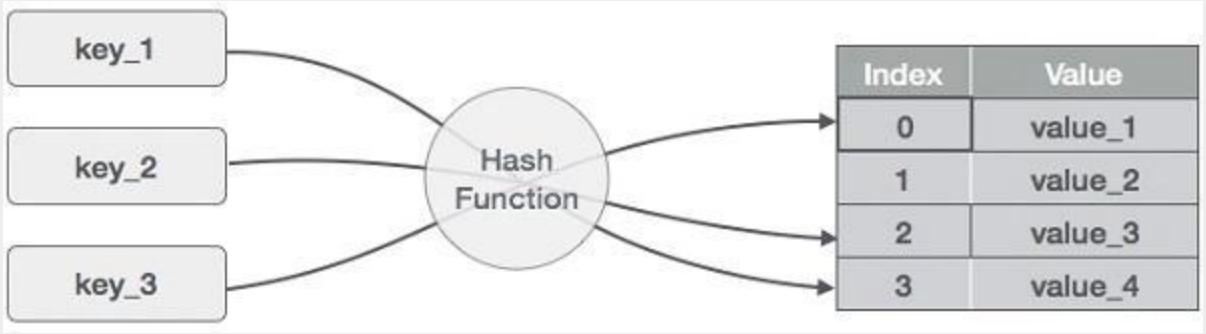
Success #stdin #stdout 0.03s 9484KB

stdin

```
Standard input is empty
```

stdout

```
deque([1, 2])
1
2
deque([])
```


key_1
key_2
key_3
Hash
Function
Index
Value
0
value_1
1
value_2
2
value_3
3
value_4

```python
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127
print(tel)
print(tel['jack'])
del tel['sape']
tel['irv'] = 4127
print(tel)
print('guido' in tel)
print('jack' not in tel)
```

Success #stdin #stdout 0.05s 9516KB

stdin

```
Standard input is empty
```

stdout

```
{'jack': 4098, 'sape': 4139, 'guido': 4127}
4098
{'jack': 4098, 'guido': 4127, 'irv': 4127}
True
False
```

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    unordered_map<string, int> tel = {
        {"jack", 4098},
        {"sape", 4139}
    };
    tel["guido"] = 4127;
    for (auto [key, value] : tel) {
        cout << key << " " << value << "\n";
    }
    cout << tel["jack"] << "\n";
    tel.erase("sape");
    tel["irv"] = 4127;
    for (auto [key, value] : tel) {
        cout << key << " " << value << "\n";
    }
    cout << tel.count("guido") << " " << tel.count("jack");
}
```

Success #stdin #stdout 0.01s 5516KB

stdin

```
Standard input is empty
```

stdout

```
guido 4127
jack 4098
sape 4139
4098
guido 4127
irv 4127
jack 4098
1 1
```

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    set<int> s = {1, 2, 3, 4};
    s.erase(2);
    s.insert(5);
    s.insert(3);
    for (auto elem: s) {
        cout << elem << " ";
    }
    cout << "\n";
    cout << s.count(0) << " " << s.count(3) << "\n";
}
```

Success #stdin #stdout 0.01s 5356KB

stdin

```
Standard input is empty
```

stdout

```
1 3 4 5
0 1
```

```
{'jack': 4098, 'sape': 4139, 'guido': 4127}
4098
{'jack': 4098, 'guido': 4127, 'irv': 4127}
True
False
```

```
1.  a = set(['apple', 'apple', 'orange', 'banana'])
2.  print(a)
```

Success #stdin #stdout 0.03s 9520KB

stdin

```
Standard input is empty
```

stdout

```
{'orange', 'apple', 'banana'}
```

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    unordered_set<int> s = {1, 2, 3, 4};
    s.erase(2);
    s.insert(5);
    s.insert(3);
    for (auto elem: s) {
        cout << elem << " ";
    }
    cout << "\n";
    cout << s.count(0) << " " << s.count(3) << "\n";
}
```

Success #stdin #stdout 0.01s 5400KB

stdin

```
Standard input is empty
```

stdout

```
5 4 3 1
0 1
```

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    set<int> s = {1, 2, 3, 4};
    s.erase(2);
    s.insert(5);
    s.insert(3);
    for (auto elem: s) {
        cout << elem << " ";
    }
    cout << "\n";
    cout << s.count(0) << " " << s.count(3) << "\n";
}
```

Success #stdin #stdout 0.01s 5356KB

stdin

```
Standard input is empty
```

stdout

```
1 3 4 5
0 1
```

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    map<string, int> tel = {
        {"jack", 4098},
        {"sape", 4139}
    };
    tel["guido"] = 4127;
    for (auto [key, value] : tel) {
        cout << key << " " << value << "\n";
    }
    cout << tel["jack"] << "\n";
    tel.erase("sape");
    tel["irv"] = 4127;
    for (auto [key, value] : tel) {
        cout << key << " " << value << "\n";
    }
    cout << tel.count("guido") << " " << tel.count("jack");
}
```

Success #stdin #stdout 0.01s 5436KB

stdin

```
Standard input is empty
```

stdout

```
guido 4127
jack 4098
sape 4139
4098
guido 4127
irv 4127
jack 4098
1 1
```

```cpp
#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
    for (int i = 0; i < 5; i++) {
        cout << rand() << " ";
    }

    return 0;
}
```

Success #stdin #stdout 0s 5356KB

stdin

```
Standard input is empty
```

stdout

```
1804289383 846930886 1681692777 1714636915 1957747793
```

```cpp
#include <random>
#include <iostream>

using namespace std;

int main()
{
    random_device dev;
    mt19937 random_generator(dev());
    uniform_int_distribution<mt19937::result_type> gen_1_5(1, 5);
    cout << gen_1_5(random_generator) << endl;
}
```

Success #stdin #stdout 0.01s 5472KB

stdin

```
Standard input is empty
```

stdout

```
3
```

```python
import random

print(random.uniform(2.5, 10.0)) #  Random float:  2.5 <= x <= 10.0
```

Success #stdin #stdout 0.03s 9720KB

stdin

```
Standard input is empty
```

stdout

```
9.21043731582682
```

```cpp
#include <algorithm>
#include <iostream>
#include <iterator>
#include <random>
#include <vector>

using namespace std;

int main()
{
    vector<int> a{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    random_shuffle(a.begin(), a.end());

    for (auto elem: a) {
        cout << elem << " ";
    }
    cout << "\n";
}
```

Success #stdin #stdout 0.01s 5516KB

stdin

```
Standard input is empty
```

stdout

```
5 4 8 9 1 6 3 2 7 10
```

```python
from random import shuffle
x = [i for i in range(10)]
shuffle(x)
print(x)
```

Success #stdin #stdout 0.03s 9740KB

stdin

```
Standard input is empty
```

stdout

```
[8, 3, 9, 1, 0, 4, 5, 6, 2, 7]
```

```cpp
#include <algorithm>
#include <iostream>
#include <iterator>
#include <random>
#include <vector>

using namespace std;

int main()
{
    vector<int> a{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    random_device rd;
    mt19937 generator(rd());

    shuffle(a.begin(), a.end(), generator);

    for (auto elem: a) {
        cout << elem << " ";
    }
    cout << "\n";
}
```

Success #stdin #stdout 0.01s 5452KB

stdin

```
Standard input is empty
```

stdout

```
6 2 10 8 3 9 4 5 7 1
```

```cpp
#include <algorithm>
#include <iostream>
#include <vector>

int main()
{
    const std::vector<int> data{1, 2, 4, 5, 5, 6};

    for (int i = 0; i < 7; ++i)
    {
        // Search first element that is greater than i
        auto upper = std::upper_bound(data.begin(), data.end(), i);

        std::cout << i << " < ";
        if (upper != data.end()) {
            std::cout << *upper << " at index " << std::distance(data.begin(), upper);
        }
        else {
            std::cout << "not found";
        }
        std::cout << '\n';
    }
}
```

Success #stdin #stdout 0.01s 5468KB

comments (0)

stdin copy

```
Standard input is empty
```

stdout copy

```
0 < 1 at index 0
1 < 2 at index 1
2 < 4 at index 2
3 < 4 at index 2
4 < 5 at index 3
5 < 6 at index 5
6 < not found
```

```python
from bisect import bisect_left

a = [1, 2, 4, 5, 5, 6]
for i in range(0, 7):
    pos = bisect_left(a, i, 0, 7)
    print(pos)
```

Success #stdin #stdout 0.03s 9584KB

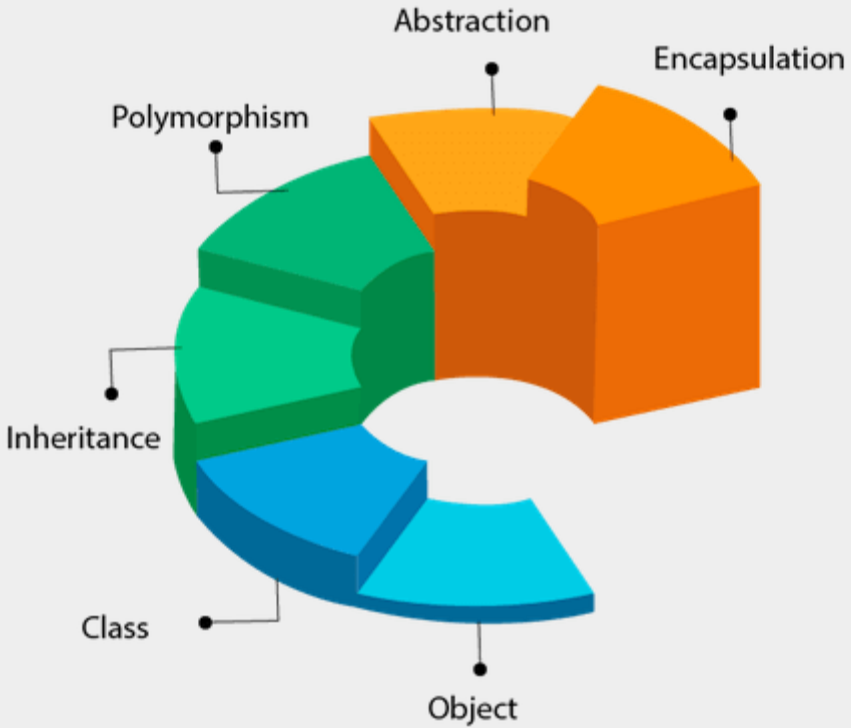
stdin

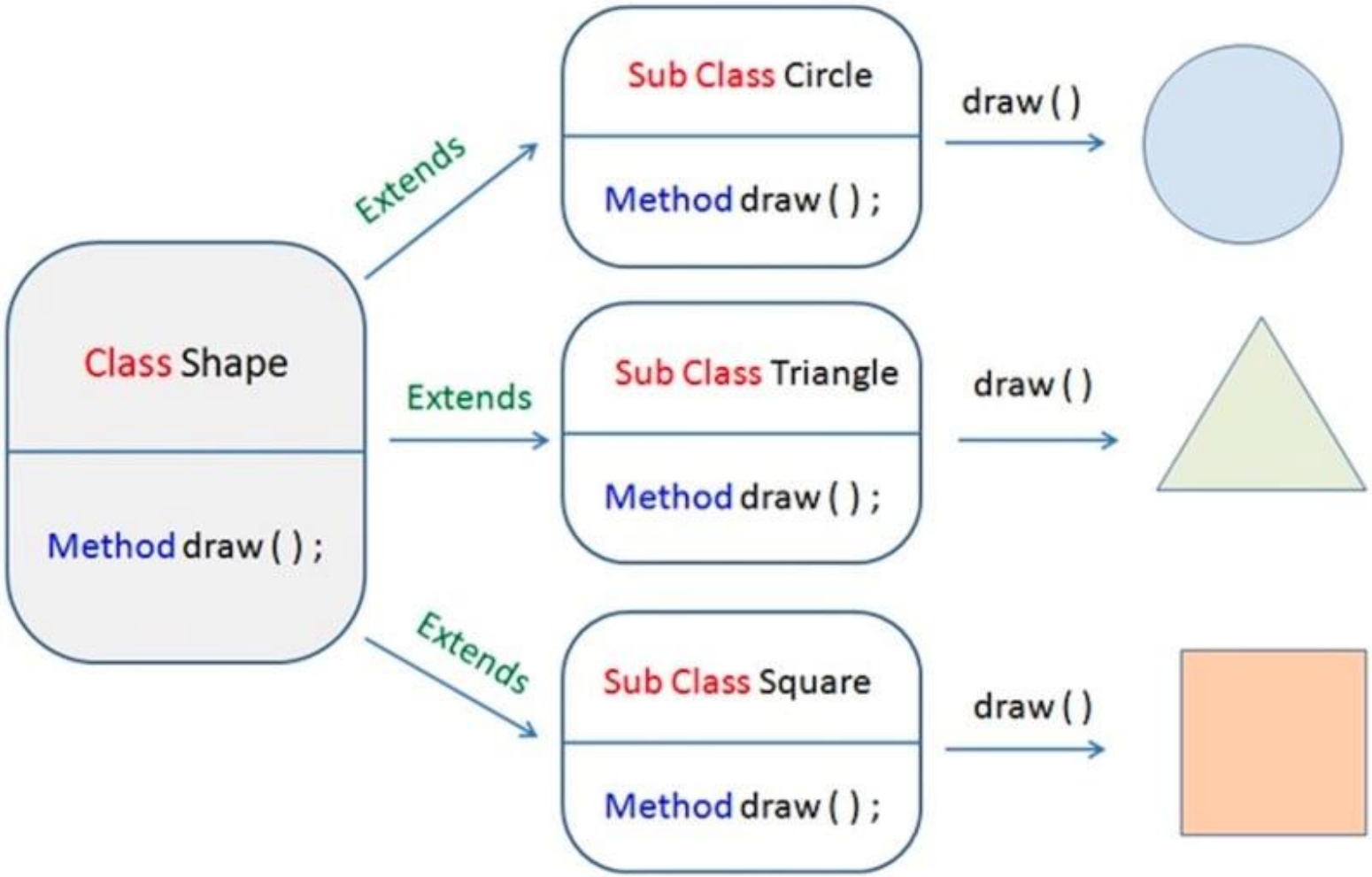
```
Standard input is empty
```

stdout

```
0
0
1
2
2
3
5
```

OOPs (Object-Oriented Programming System)




Class Shape
Method draw ( ) ;
Extends
Sub Class Circle
Method draw ( ) ;
draw ( )
Extends
Sub Class Triangle
Method draw ( ) ;
draw ( )
Extends
Sub Class Square
Method draw ( ) ;
draw ( )

Encapsulation

Car

model

speed

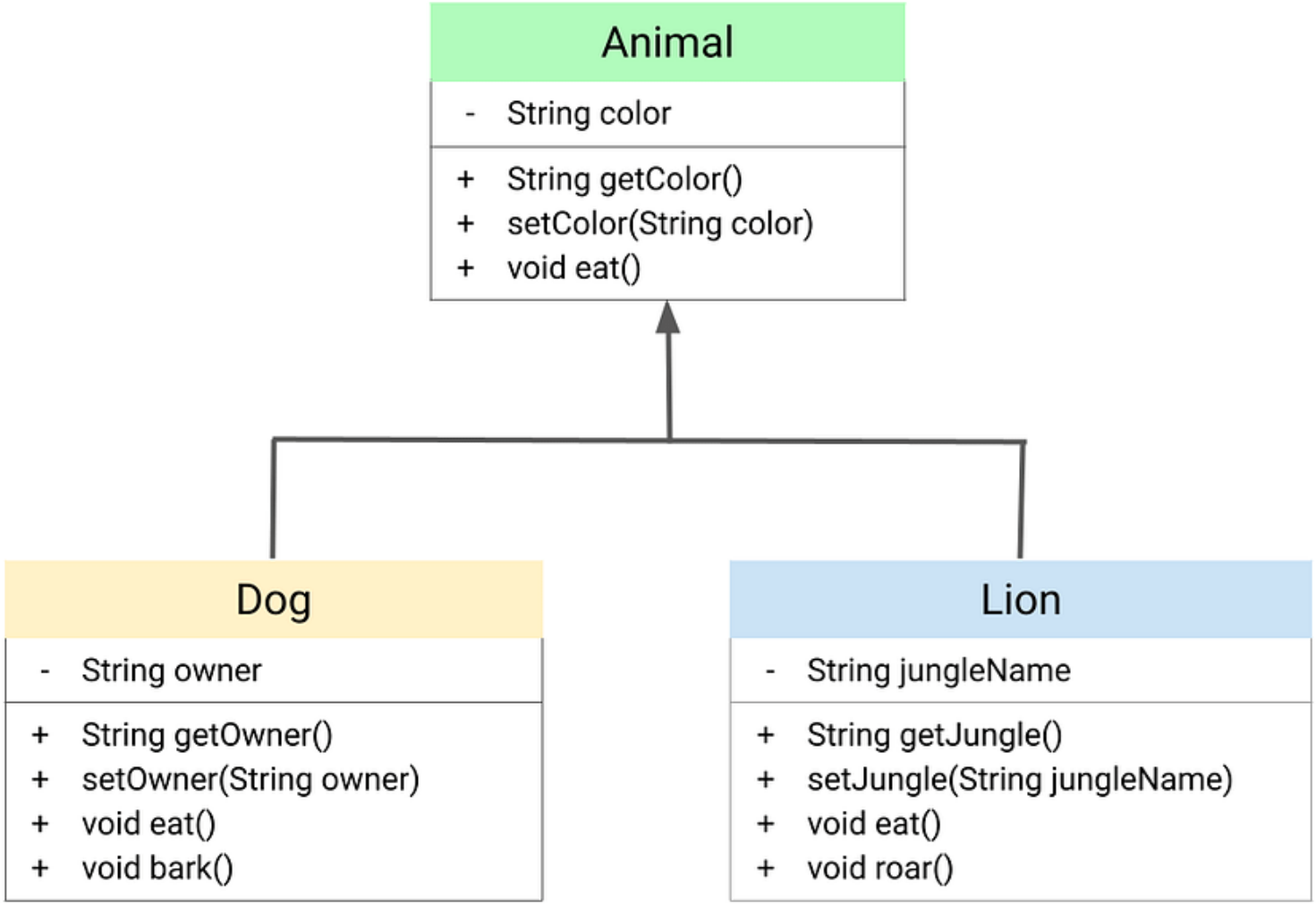
engine

speedLimit

drive()

stop()

setSpeed(number)


Animal
- String color
+ String getColor()
+ setColor(String color)
+ void eat()
Dog
- String owner
+ String getOwner()
+ setOwner(String owner)
+ void eat()
+ void bark()
Lion
- String jungleName
+ String getJungle()
+ setJungle(String jungleName)
+ void eat()
+ void roar()

```cpp
#include <iostream>
using namespace std;

struct Point{
    double x, y;
    Point(double x, double y) {
        this->x = x;
        this->y = y;
    }
};

int main() {
    auto pt = Point(100, 100);
    cout << pt.x << " " << pt.y << "\n";
    return 0;
}
```

Success #stdin #stdout 0.01s 5388KB

stdin

```
Standard input is empty
```

stdout

```
100 100
```

```
1. class Dog:
2.     def __init__(self, name):
3.         self.name = name
4.         self.tricks = []
5.
```

```cpp
#include <iostream>
using namespace std;

class Point{
private:
    double x, y;
public:
    Point(double x, double y) {
        this->x = x;
        this->y = y;
    }
    void print() const {
        cout << this->x << " " << this->y << "\n";
    }
};

int main() {
    auto pt = Point(100, 100);
    pt.print();
    return 0;
}
```

Success #stdin #stdout 0.01s 5388KB

stdin

```
Standard input is empty
```

stdout

```
100 100
```

```python
class Dog:
    def __init__(self, name):
        self.name = name
        self.tricks = []

    def add_trick(self, trick):
        self.tricks.append(trick)

Apollo = Dog('Apollo')
Marchello = Dog('Marchello')
Apollo.add_trick('sit')
Apollo.add_trick('roll')
print(Apollo.tricks)
print(Marchello.tricks)
print(Apollo.__class__.__name__)
```

Success #stdin #stdout 0.03s 9580KB

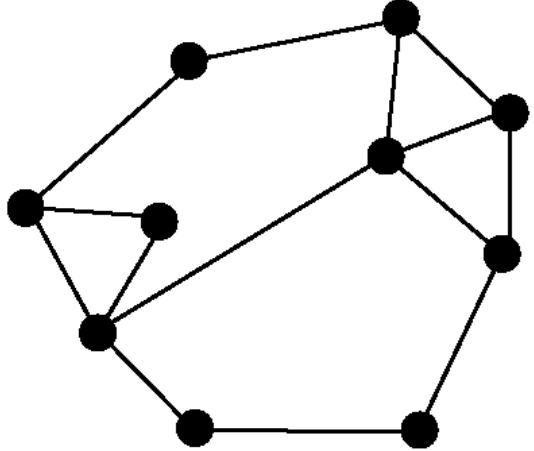
stdin

```
Standard input is empty
```

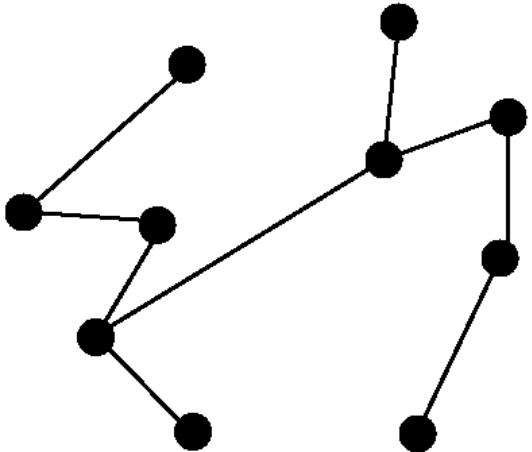
stdout

```
['sit', 'roll']
[]
Dog
```

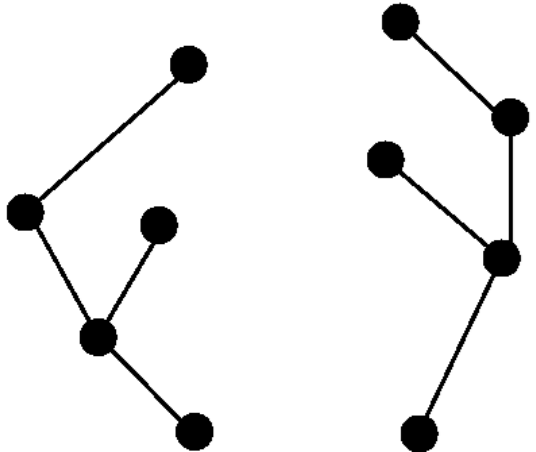
Graph
(with cycles)

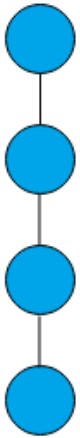


Tree
(no cycles, connected)

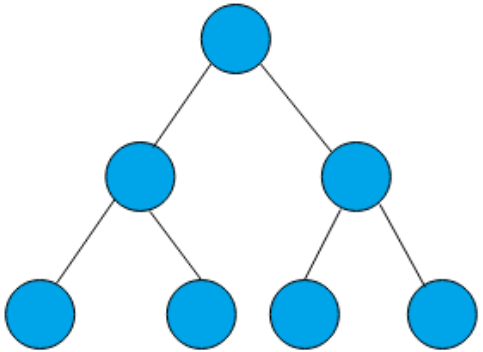


Forest
(no cycles, not connected)

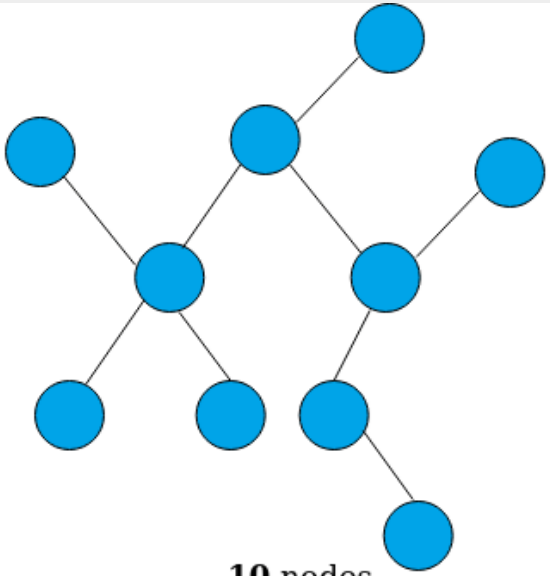




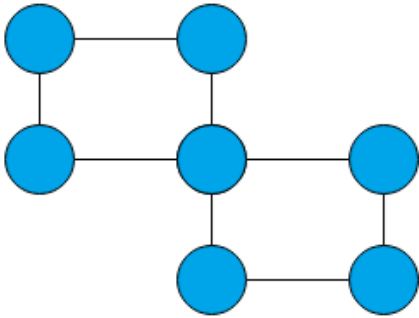
4 nodes
3 edges



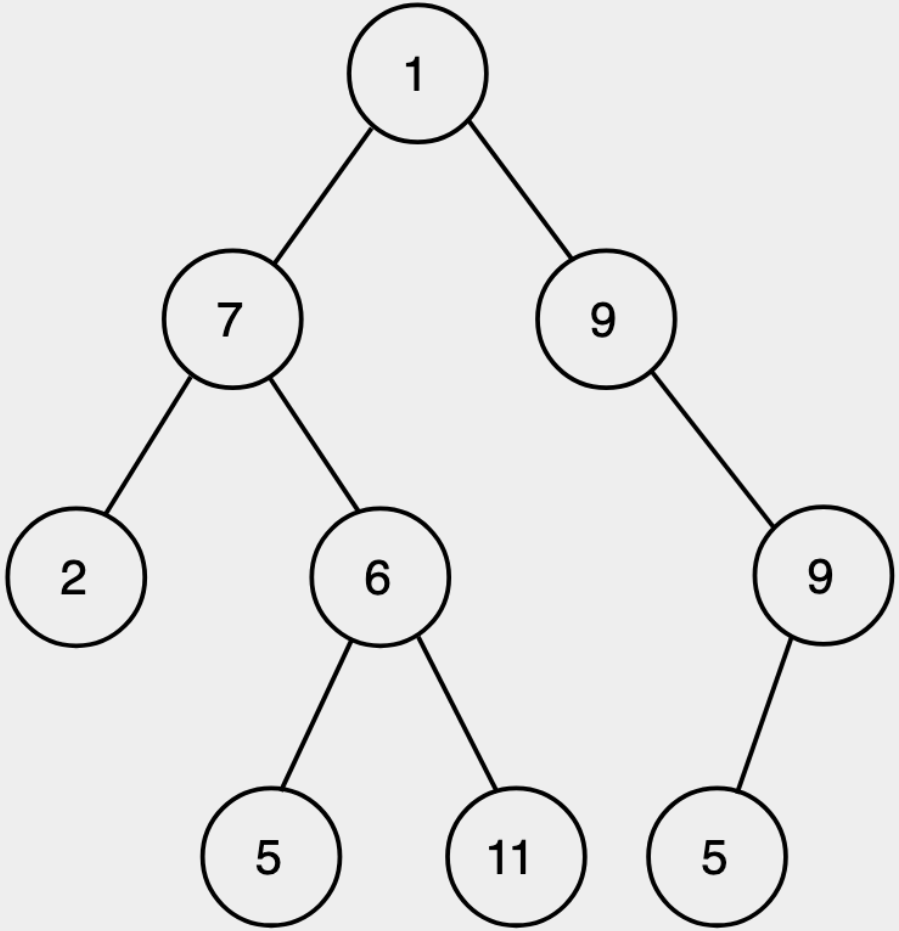
7 nodes
6 edges



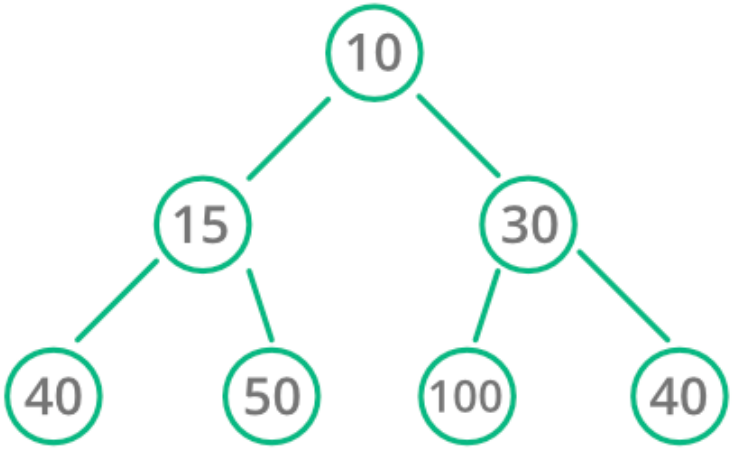
10 nodes
9 edges



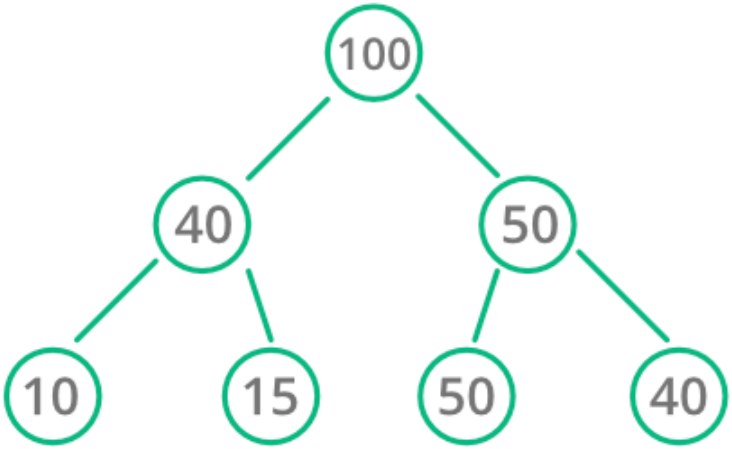
7 nodes
8 edges


1
7
9
2
6
9
5
11
5

Heap Data Structure



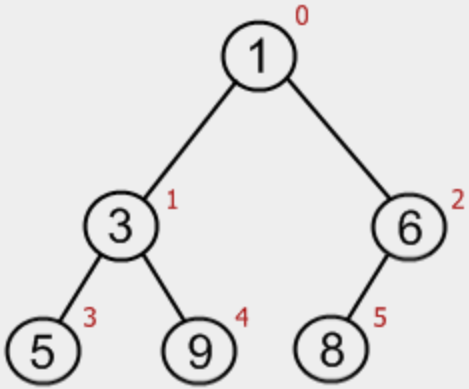
Min Heap



Max Heap

Bisection

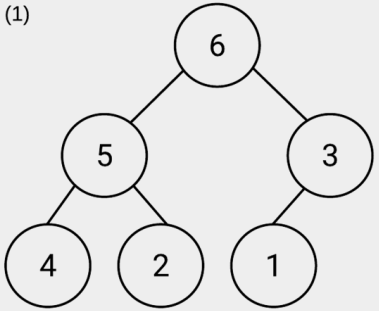
- Top of heap -> position 0
- index of vertex = n -> sons = [2 * n + 1, 2 * n + 2]



1	3	6	5	9	8
0	1	2	3	4	5

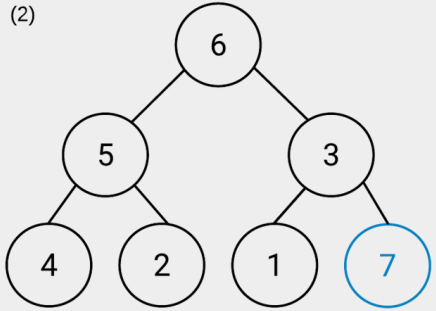
Inserting 7 into this heap

(1)



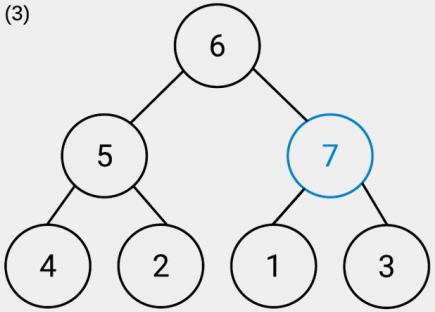
Starting with this max heap

(2)



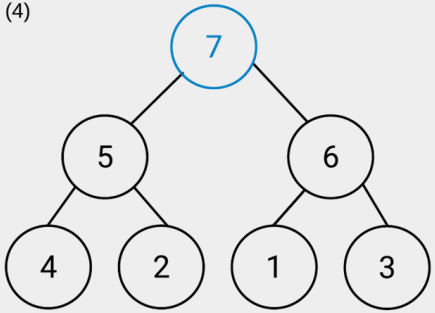
Step 1: 7 is inserted at the bottom most, right most position

(3)

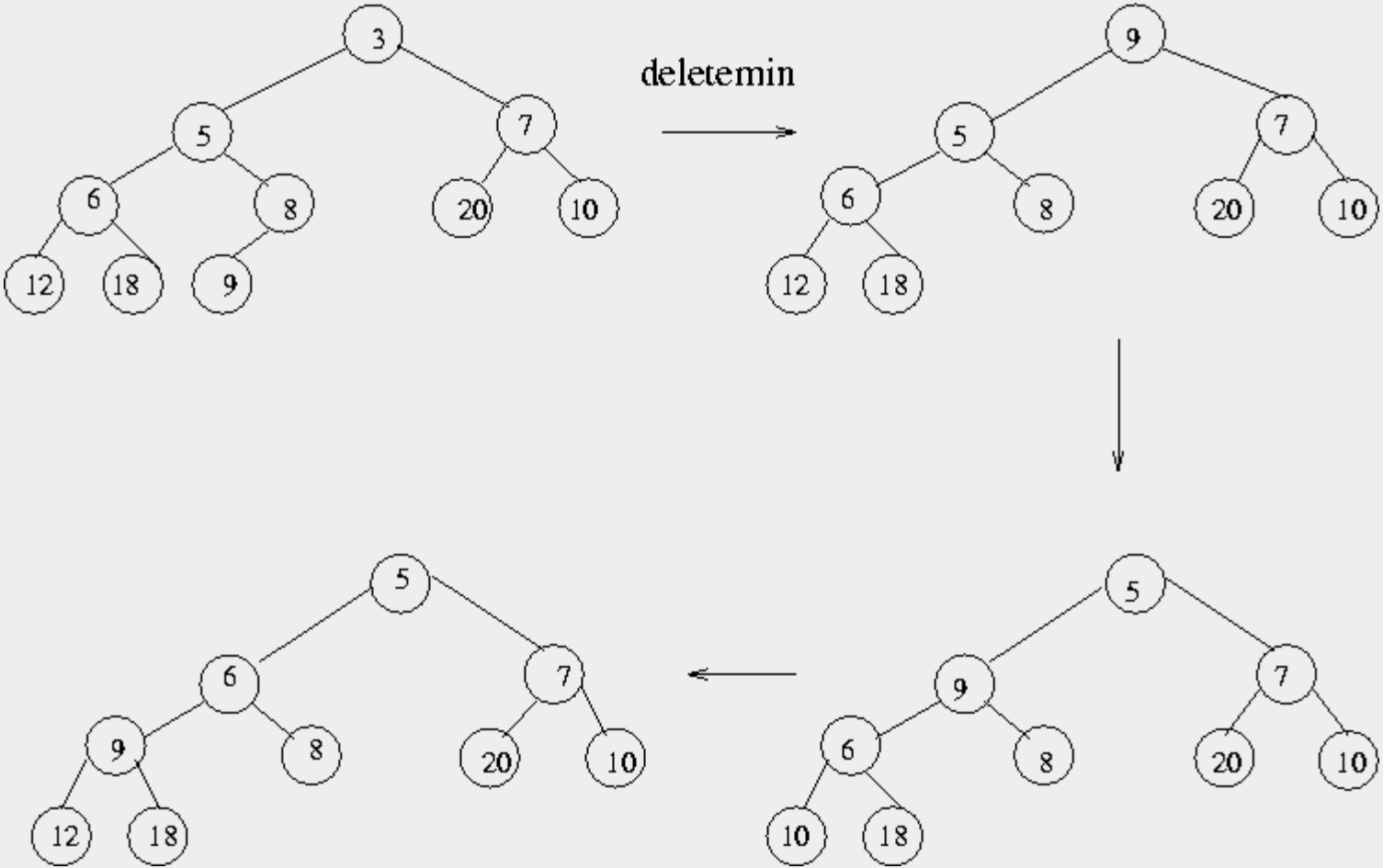


Step 2: Because 7 is bigger than its parent, the 3 node, it gets swapped

(4)



Step 3: Once again, 7 is bigger than its parent, the 6 node, so it gets swapped


3
5
7
6
8
20
10
12
18
9
deletemin
9
5
7
6
8
20
10
12
18
5
9
7
6
8
20
10
10
18
5
6
7
9
8
20
10
12
18

Proof

- 2^i = number of vertices in row i with n
- 1 + 2 + .. + 2^(i - 1) = 2^i - 1 = number of vertices in all rows before i
- n - (2^i - 1) - number of vertices in row i before vertex n
- 2^(i + 1) - 1 - n - after n in row i
- (2^(i + 1) - 1 - n)+2 * (n - (2^i - 1)) + n, after n in row i + sons of vertices before n + n vertices
- 2^(i + 1) - (2 * 2^i) - 1 + 2 - n + 2 * n + n = 2 * n + 1

Defined in header <queue>

```
template<
    class T,
    class Container = std::vector<T>,
    class Compare = std::less<typename Container::value_type>
> class priority_queue;
```

A `priority` queue is a container adaptor that provides constant time lookup of the largest (by default) element, at the expense of logarithmic insertion and extraction.

A user-provided `Compare` can be supplied to change the ordering, e.g. using `std::greater<T>` would cause the smallest element to appear as the `top()`.

Working with a `priority_queue` is similar to managing a heap in some random access container, with the benefit of no being able to accidentally invalidate the heap.

```python
from heapq import heappush, heappop

h = []
heappush(h, (5, 'write code'))
heappush(h, (7, 'release product'))
heappush(h, (1, 'write spec'))
heappush(h, (3, 'create tests'))
print(heappop(h))
```

Success #stdin #stdout 0.03s 9584KB

stdin

```
Standard input is empty
```

stdout

```
(1, 'write spec')
```

```cpp
class MinHeap {
private:
    vector<int> heap; // Vector to store heap elements

    // Helper functions for working with node indices
    int parent(int i) { return (i - 1) / 2; }
    int leftChild(int i) { return 2 * i + 1; }
    int rightChild(int i) { return 2 * i + 2; }

    // Function to maintain the heap property (upward)
    void heapifyUp(int i) {
        while (i > 0 && heap[parent(i)] > heap[i]) {
            swap(heap[i], heap[parent(i)]);
            i = parent(i);
        }
    }

    // Function to maintain the heap property (downward)
    void heapifyDown(int i) {
        int minIndex = i;
        int left = leftChild(i);
        int right = rightChild(i);

        if (left < heap.size() && heap[left] < heap[minIndex]) {
            minIndex = left;
        }

        if (right < heap.size() && heap[right] < heap[minIndex]) {
            minIndex = right;
        }

        if (i != minIndex) {
            swap(heap[i], heap[minIndex]);
            heapifyDown(minIndex);
        }
    }

```

```
public:
    // Insert an element into the heap
    void insert(int value) {
        heap.push_back(value);
        int lastIndex = heap.size() - 1;
        heapifyUp(lastIndex);
    }

    // Extract the element with the highest priority (minimum element in the case of a min heap)
    int extractMin() {
        if (heap.empty()) {
            throw runtime_error("Heap is empty");
        }

        int minValue = heap[0];
        heap[0] = heap.back();
        heap.pop_back();
        heapifyDown(0);

        return minValue;
    }

    // Get the minimum element without removing it
    int getMin() {
        if (heap.empty()) {
            throw runtime_error("Heap is empty");
        }

        return heap[0];
    }
};
```

```cpp
int main() {
    MinHeap minHeap;

    minHeap.insert(4);
    minHeap.insert(2);
    minHeap.insert(7);
    minHeap.insert(1);
    minHeap.insert(9);

    cout << "Minimum element: " << minHeap.getMin() << endl;

    minHeap.extractMin();
    cout << "Minimum element after extraction: " << minHeap.getMin() << endl;

    return 0;
}
```

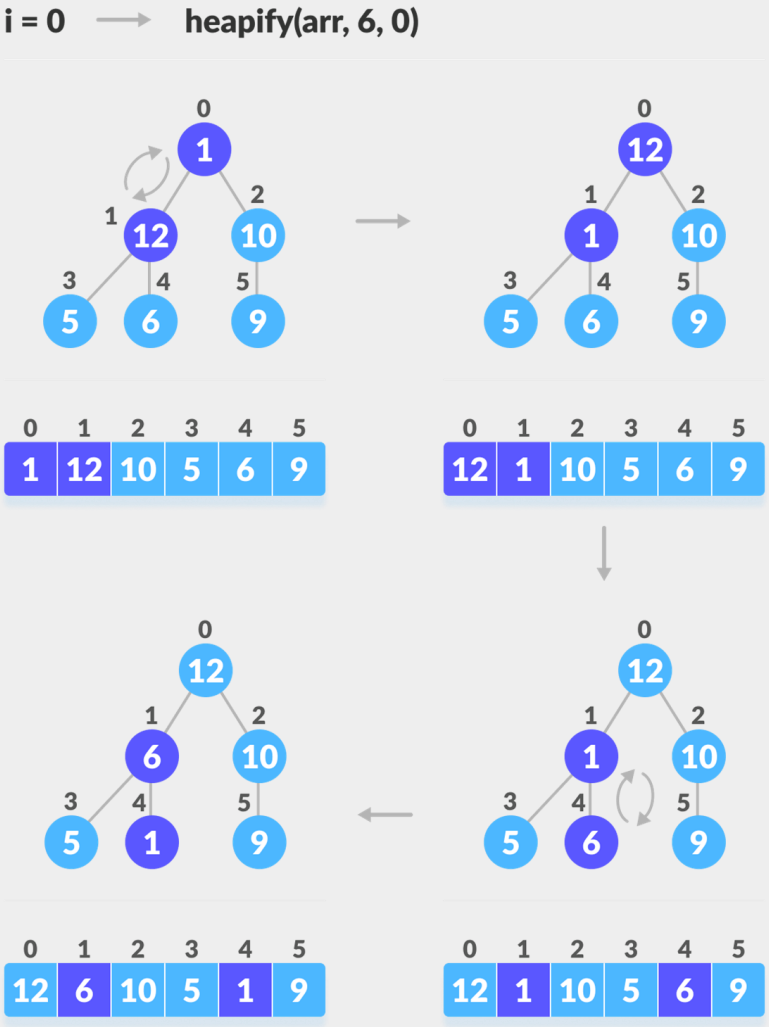
```python
class MinHeap:
    def __init__(self):
        self.heap = []

    # Helper functions for working with node indices
    def parent(self, i):
        return (i - 1) // 2

    def leftChild(self, i):
        return 2 * i + 1

    def rightChild(self, i):
        return 2 * i + 2

    # Function to maintain the heap property (upward)
    def heapifyUp(self, i):
        while i > 0 and self.heap[self.parent(i)] > self.heap[i]:
            self.heap[i], self.heap[self.parent(i)] = self.heap[self.parent(i)], self.heap[i]
            i = self.parent(i)

    # Function to maintain the heap property (downward)
    def heapifyDown(self, i):
        minIndex = i
        left = self.leftChild(i)
        right = self.rightChild(i)

        if left < len(self.heap) and self.heap[left] < self.heap[minIndex]:
            minIndex = left

        if right < len(self.heap) and self.heap[right] < self.heap[minIndex]:
            minIndex = right

        if i != minIndex:
            self.heap[i], self.heap[minIndex] = self.heap[minIndex], self.heap[i]
            self.heapifyDown(minIndex)

```

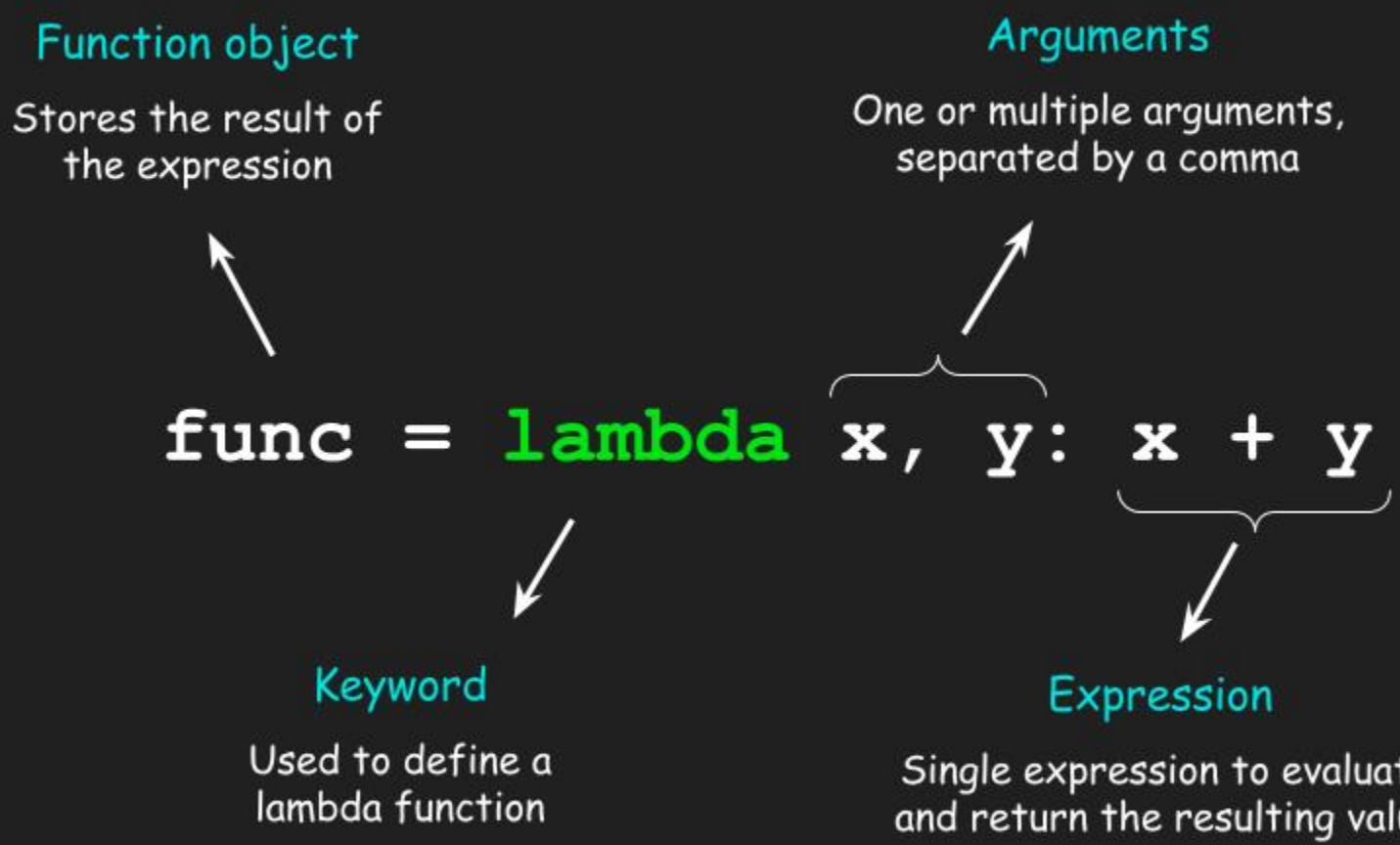
```
    # Insert an element into the heap
    def insert(self, value):
        self.heap.append(value)
        lastIndex = len(self.heap) - 1
        self.heapifyUp(lastIndex)

    # Extract the element with the highest priority (minimum element in the case of a min heap)
    def extractMin(self):
        if not self.heap:
            raise Exception("Heap is empty")

        minValue = self.heap[0]
        self.heap[0] = self.heap[-1]
        self.heap.pop()
        self.heapifyDown(0)

        return minValue

    # Get the minimum element without removing it
    def getMin(self):
        if not self.heap:
            raise Exception("Heap is empty")

        return self.heap[0]

```

```
63.  # Example usage
64.  min_heap = MinHeap()
65.
66.  min_heap.insert(4)
67.  min_heap.insert(2)
68.  min_heap.insert(7)
69.  min_heap.insert(1)
70.  min_heap.insert(9)
71.
72.  print("Minimum element:", min_heap.getMin())
73.
74.  min_heap.extractMin()
75.  print("Minimum element after extraction:", min_heap.getMin())
```

Asymptotic

- O(log(n))


i = 0
heapify(arr, 6, 0)
0 1 2 3 4 5
1 12 10 5 6 9
0 1 2 3 4 5
12 1 10 5 6 9
0 1 2 3 4 5
12 1 10 5 6 9
0 1 2 3 4 5
12 6 10 5 1 9


Function object
Stores the result of the expression
Arguments
One or multiple arguments, separated by a comma
func = lambda x, y: x + y
Keyword
Used to define a lambda function
Expression
Single expression to evaluate and return the resulting value

```cpp
bool comp(int a, int b) {
    return a < b;
}

int main() {
    vector<int> s_lam = {1, 3, 2, 4, 8, 6, 7};
    vector<int> s_comp = {1, 3, 2, 4, 8, 6, 7};
    vector<int> s = {1, 3, 2, 4, 8, 6, 7};

    std::sort(s.begin(), s.end());

    std::sort(
        s_comp.begin(),
        s_comp.end(),
        comp
    );

    std::sort(
        s_lam.begin(),
        s_lam.end(),
        [](int a, int b) {
            return a < b;
        }
    );

```

```
29.     for (auto elem: s) {
30.         cout << elem << " ";
31.     }
32.     cout << "\n";
33.     for (auto elem: s_comp) {
34.         cout << elem << " ";
35.     }
36.     cout << "\n";
37.     for (auto elem: s_lam) {
38.         cout << elem << " ";
39.     }
40.     cout << "\n";
41.     return 0;
42. }
```

Success #stdin #stdout 0.01s 5536KB

stdin

```
Standard input is empty
```

stdout

```
1 2 3 4 6 7 8
1 2 3 4 6 7 8
1 2 3 4 6 7 8
```

```python
import functools

def comparator(a, b):
    if a < b:
        return -1
    elif a > b:
        return 1
    else:
        return 0

arr = [5, 2, 8, 1, 9]
sorted_arr = sorted(arr)

sorted_arr_comparator = sorted(arr, key=functools.cmp_to_key(comparator))

sorted_arr_lambda = sorted(arr, key=lambda x: x)

print(sorted_arr)
print(sorted_arr_comparator)
print(sorted_arr_lambda)

```

Success #stdin #stdout 0.03s 9584KB

stdin

```
Standard input is empty
```

stdout

```
[1, 2, 5, 8, 9]
[1, 2, 5, 8, 9]
[1, 2, 5, 8, 9]
```

```
1. a = [1, 1, 2, 3, 4]
2. for elem in a:
3.     if elem % 2:
4.         a.remove(elem)
5. print(a)
6.
```

Success #stdin #stdout 0.04s 9576KB

stdin

```
Standard input is empty
```

stdout

```
[1, 2, 4]
```

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    set<int> a = {1, 3, 4, 5};
    for (auto elem: a) {
        if (elem % 2) {
            a.erase(elem);
        }
    }
    for (auto elem: a) {
        cout << elem << " ";
    }
    return 0;
}
```

Runtime error #stdin #stdout 0.01s 5504KB

stdin

```
Standard input is empty
```

stdout

```
Standard output is empty
```

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    set<int> a = {1, 3, 4, 5};
    lower_bound(a.begin(), a.end(), 2); // O(n)
    a.lower_bound(2); // O(log(n))
    return 0;
}
```

Success #stdin #stdout 0.01s 5316KB

stdin

```
Standard input is empty
```

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int,null_type,less<int>,rb_tree_tag,tree_order_statistics_node_update> order_set;

order_set X;

signed main() {
    X.insert(-1);
    X.insert(2);
    X.insert(4);
    X.insert(5);
    cout << X.order_of_key(2) << " " << X.order_of_key(3) << "\n";
    cout << *X.find_by_order(2) << " " << *X.find_by_order(3) << "\n";
}
```

Success #stdin #stdout 0.01s 5536KB

comments (0)

stdin

copy

```
Standard input is empty
```

stdout

copy

```
1 2
4 5
```

Permutation operations

Defined in header <algorithm>

is_permutation (C++11)	determines if a sequence is a permutation of another sequence (function template)
ranges::is_permutation (C++20)	determines if a sequence is a permutation of another sequence (niebloid)
next_permutation	generates the next greater lexicographic permutation of a range of elements (function template)
ranges::next_permutation (C++20)	generates the next greater lexicographic permutation of a range of elements (niebloid)
prev_permutation	generates the next smaller lexicographic permutation of a range of elements (function template)
ranges::prev_permutation (C++20)	generates the next smaller lexicographic permutation of a range of elements (niebloid)

All codes

types with limits(c++) - https://ideone.com/VabNzS
float types with limits(c++) - https://ideone.com/dwubzv
stack(python) - https://ideone.com/k5OfYT
queue(python) - https://ideone.com/Lp8ien
dict(python) - https://ideone.com/lv0VJ5
unordered_map(c++) - https://ideone.com/6xoOfb
set(c++) - https://ideone.com/Yz4aQT
map(c++) - https://ideone.com/67vrY5
good_random(c++) - https://ideone.com/SvV2Fz
random(c++) - https://ideone.com/ysaXW4
random_shuffle(c++) - https://ideone.com/SJxrPx
random_shuffle(python) - https://ideone.com/HmfE1D
upper_bound(c++) - https://ideone.com/i5IpPP
bin_search(python) - https://ideone.com/51QMvS
struct(c++) - https://ideone.com/D8ytqj
class(c++) - https://ideone.com/3oDX4c
struct(python) - https://ideone.com/RA2fii
heap(python) https://ideone.com/Xk8RMl
sort with lambda(c++) - https://ideone.com/R77wav
sort with lambda(python) - https://ideone.com/M5Qlz0
list(c++) - https://ideone.com/uF82jh
array(c++) - https://ideone.com/VVABUo
pointers(c++) - https://ideone.com/5Vh5nG
list(python) - https://ideone.com/sCyllG
unordered_set(c++) - https://ideone.com/F9NOFI
set(python) - https://ideone.com/OdjF1I
gnu_pbds(c++) - https://ideone.com/YbwvqP

That's All Folks!