Lecture 15

M M Imran

Vector

- Vectors are used to store elements of similar data types.
- However, unlike arrays, the size of a vector can grow dynamically.
- That is, we can change the size of the vector during the execution of a program as per our requirements.

Vector Declaration

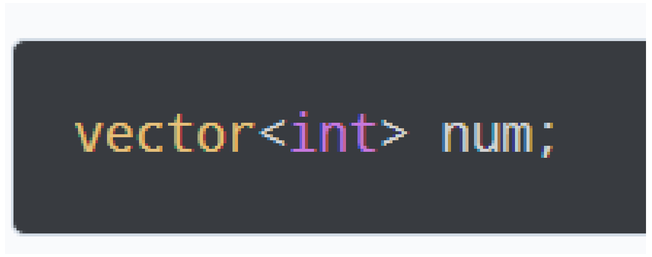
- Vectors are part of the C++ Standard Template Library. To use vectors, we need to include the vector header file in our program.

  #include <vector>

- Once we include the header file, here's how we can declare a vector

  std::vector<T> vector_name;

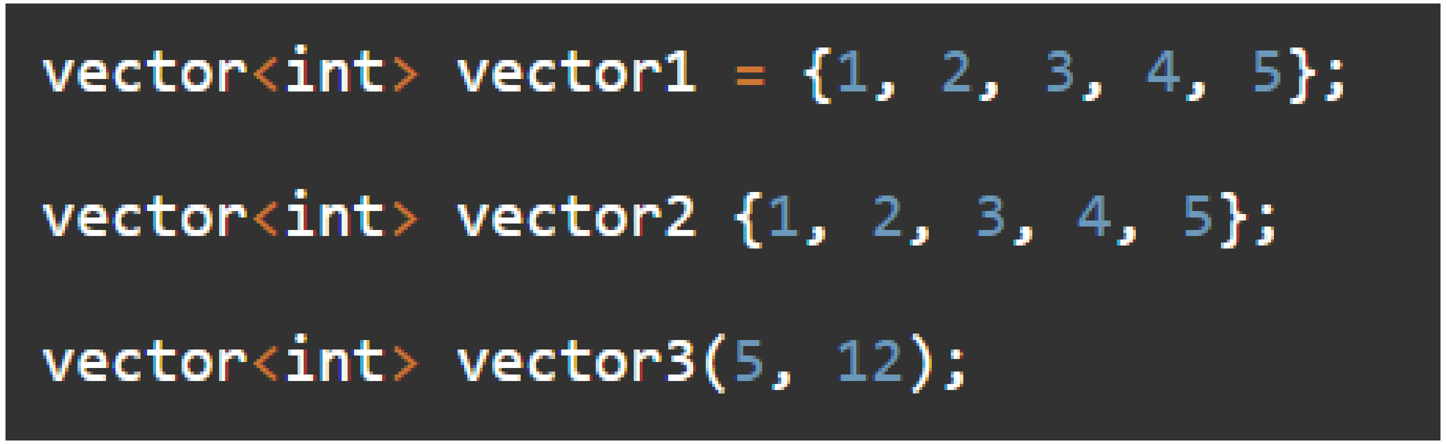
- The type parameter <T> specifies the type of the vector. It can be any primitive data type such as int, char, float, etc.



```
vector<int> num;
```

Vector Initialization

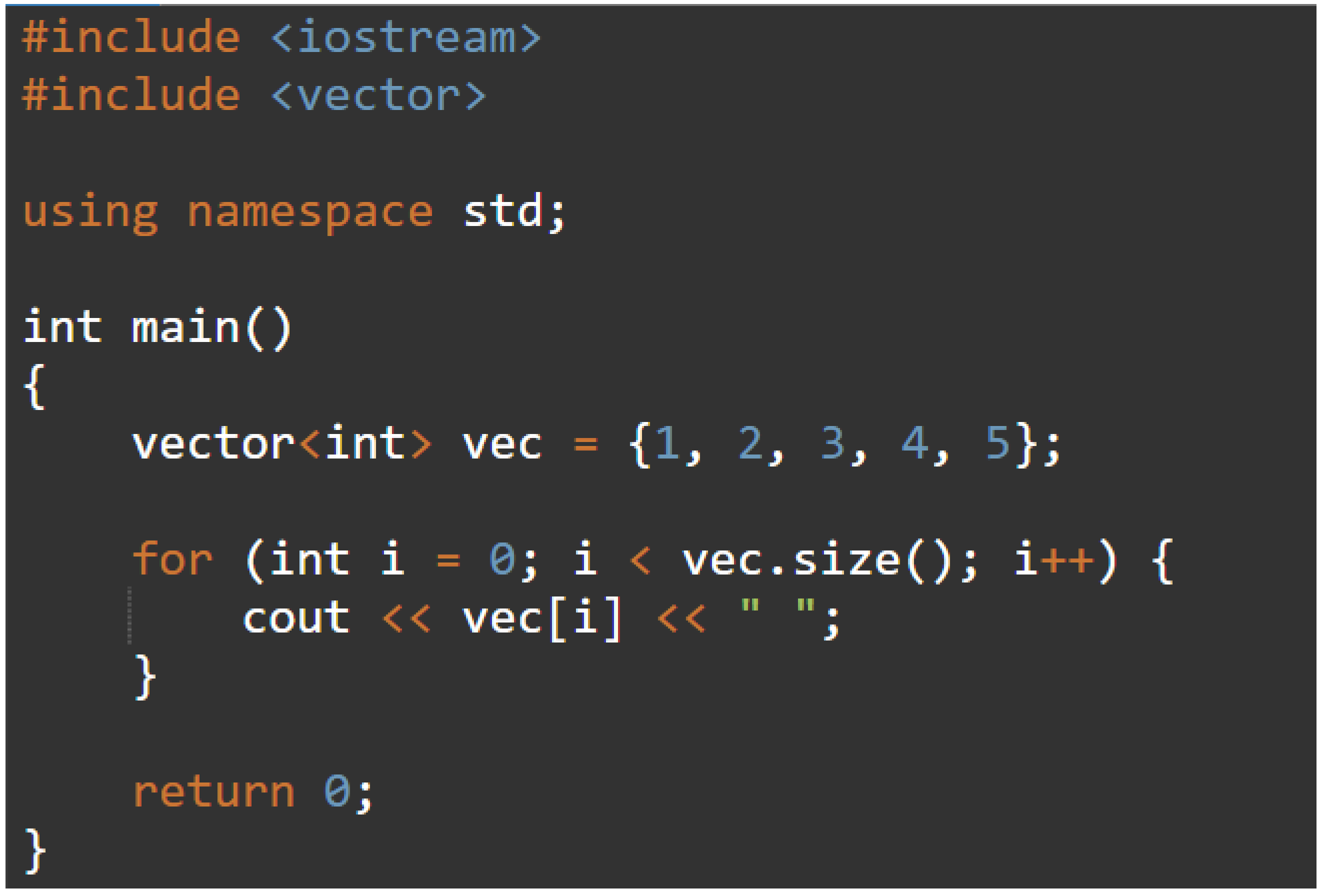
There are different ways to initialize a vector



```cpp
vector<int> vector1 = {1, 2, 3, 4, 5};

vector<int> vector2 {1, 2, 3, 4, 5};

vector<int> vector3(5, 12);
```

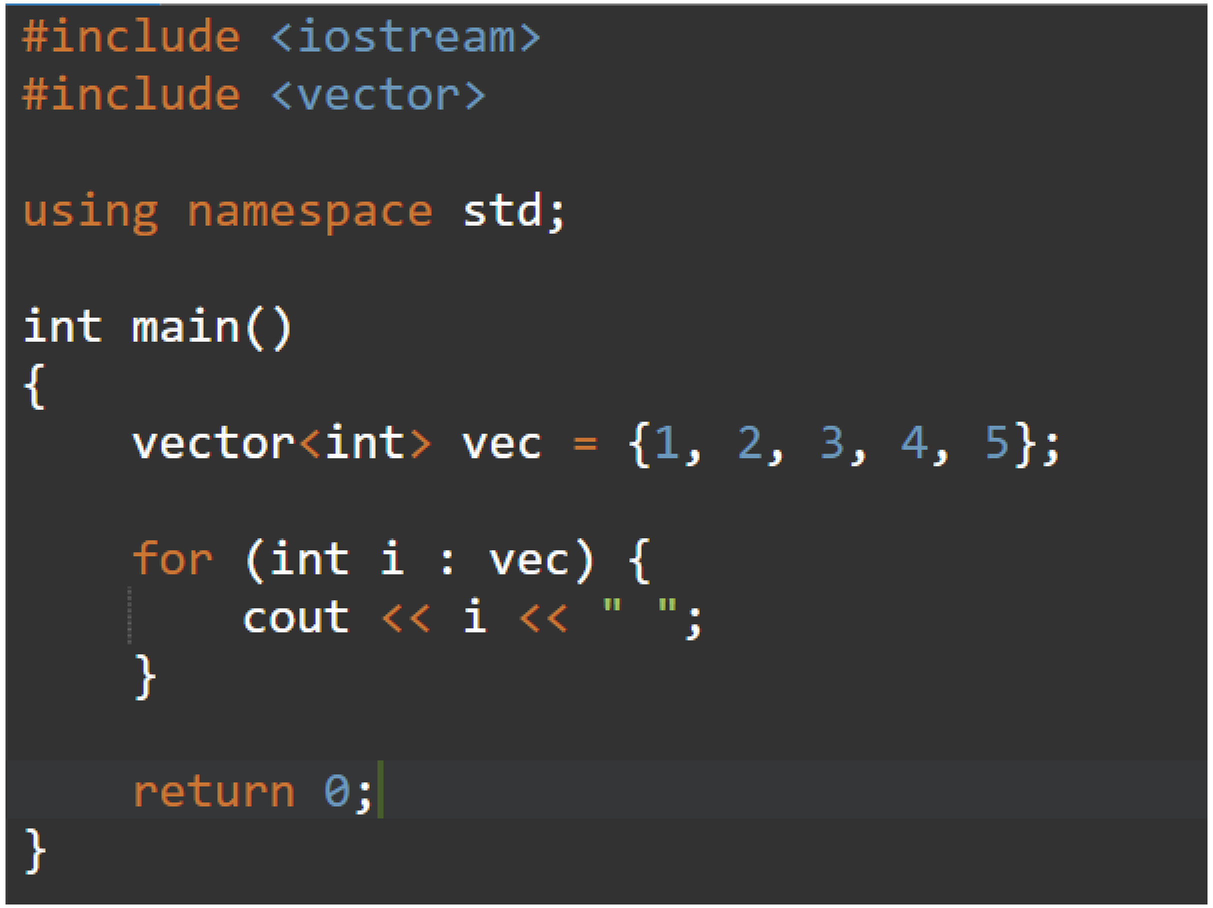
The first one more preferred style.

Iterating Through Vector - 1



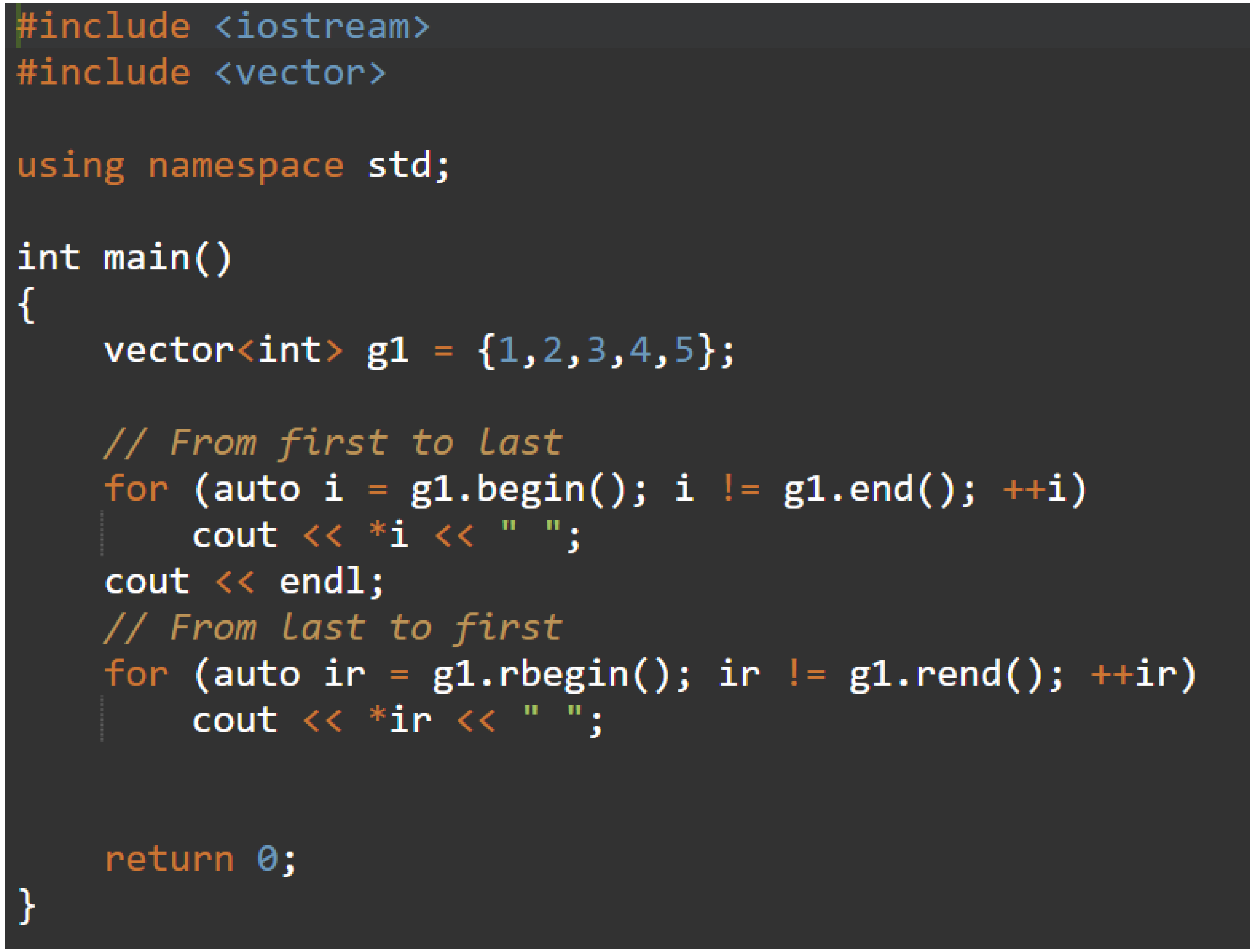
```cpp
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> vec = {1, 2, 3, 4, 5};

    for (int i = 0; i < vec.size(); i++) {
        cout << vec[i] << " ";
    }

    return 0;
}
```

Iterating Through Vector - 2

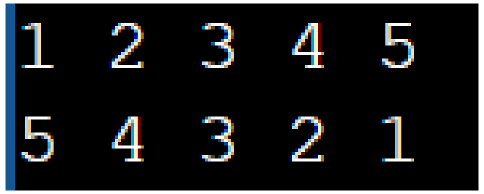


```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> vec = {1, 2, 3, 4, 5};

    for (int i : vec) {
        cout << i << " ";
    }

    return 0;
}
```

Iterating Through Vector - 3

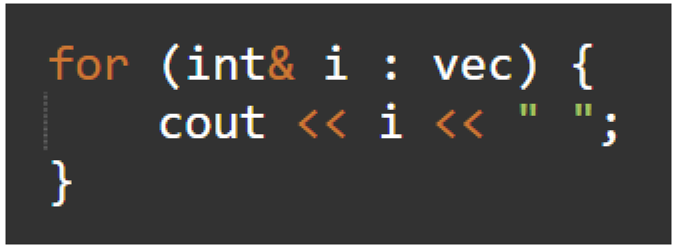


```cpp
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    vector<int> g1 = {1,2,3,4,5};

    // From first to last
    for (auto i = g1.begin(); i != g1.end(); ++i)
        cout << *i << " ";
    cout << endl;
    // From last to first
    for (auto ir = g1.rbegin(); ir != g1.rend(); ++ir)
        cout << *ir << " ";


    return 0;
}
```

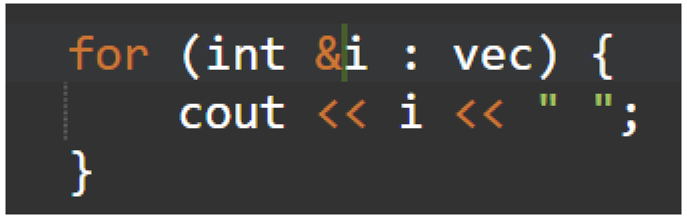


```
1 2 3 4 5
5 4 3 2 1
```

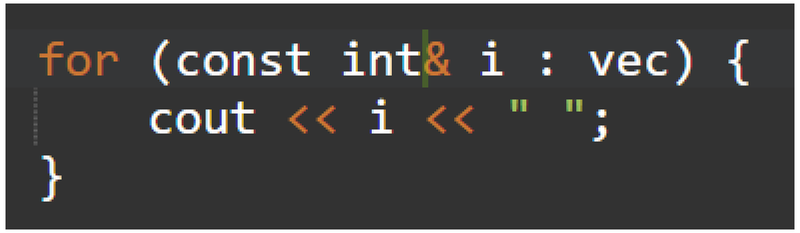
Iterating Through Vector – Other Variations



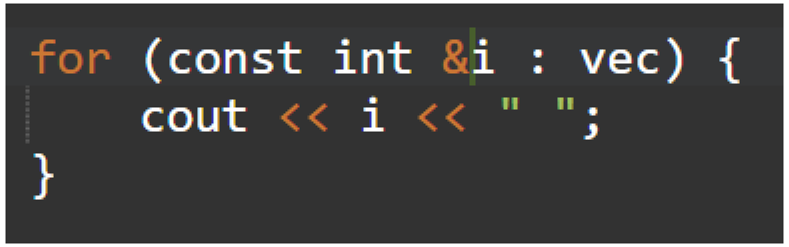
```
for (int& i : vec) {
    cout << i << " ";
}
```



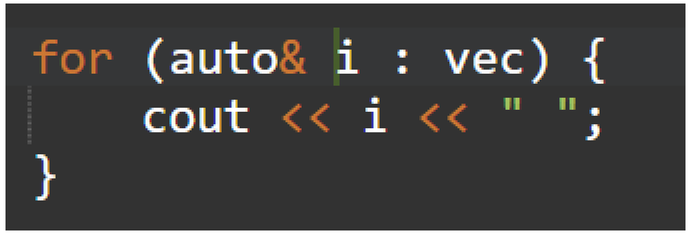
```
for (int &i : vec) {
    cout << i << " ";
}
```



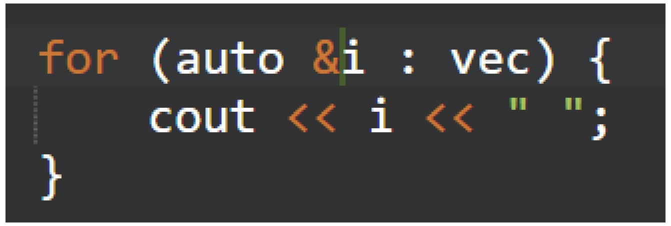
```
for (const int& i : vec) {
    cout << i << " ";
}
```



```
for (const int &i : vec) {
    cout << i << " ";
}
```



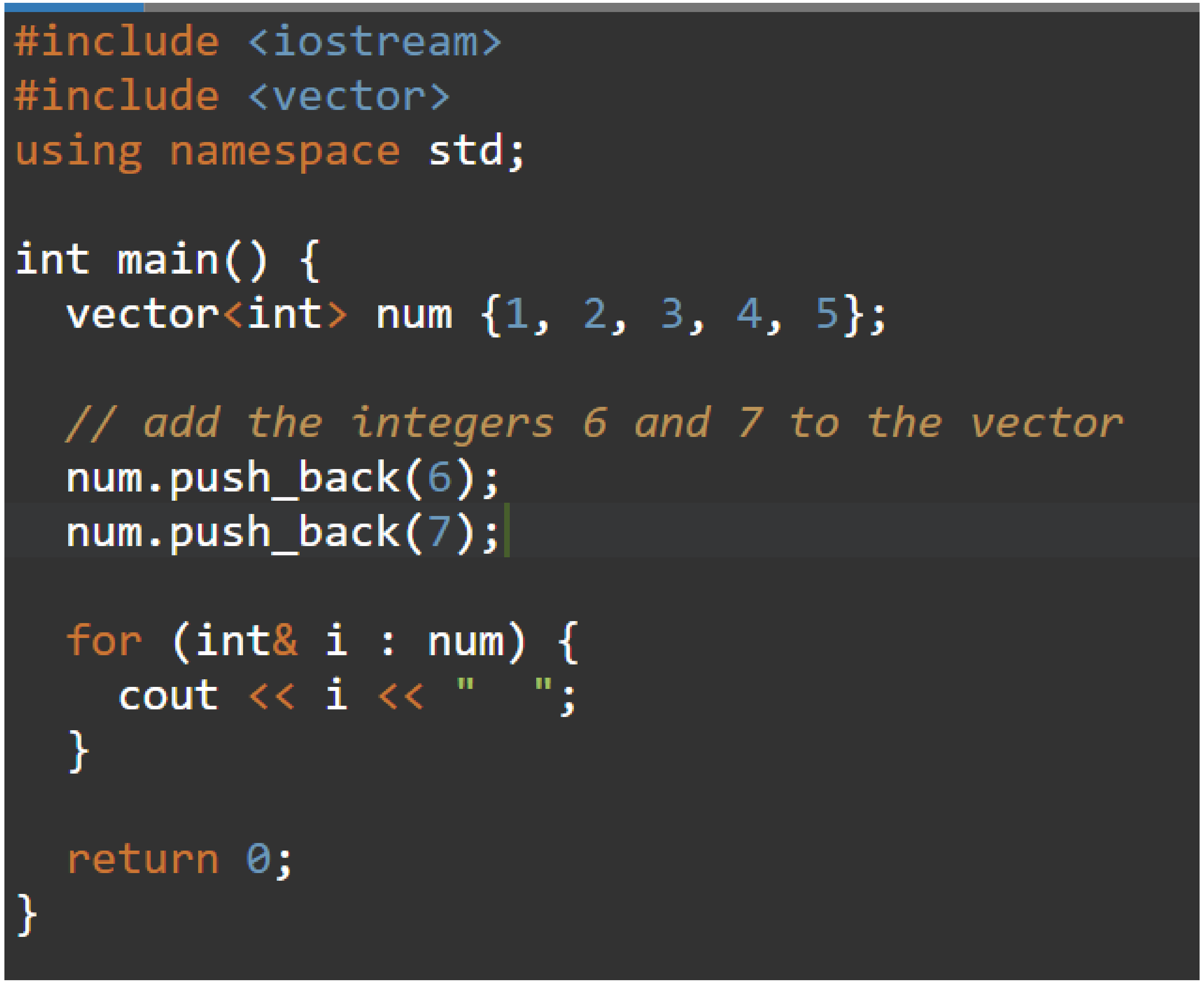
```
for (auto& i : vec) {
    cout << i << " ";
}
```



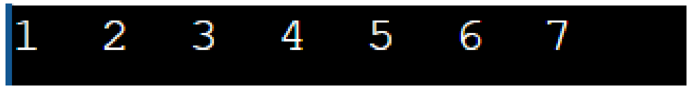
```
for (auto &i : vec) {
    cout << i << " ";
}
```

Add Elements to a Vector

- To add a single element into a vector, we use the **push_back()** function.
- It inserts an element into the end of the vector.



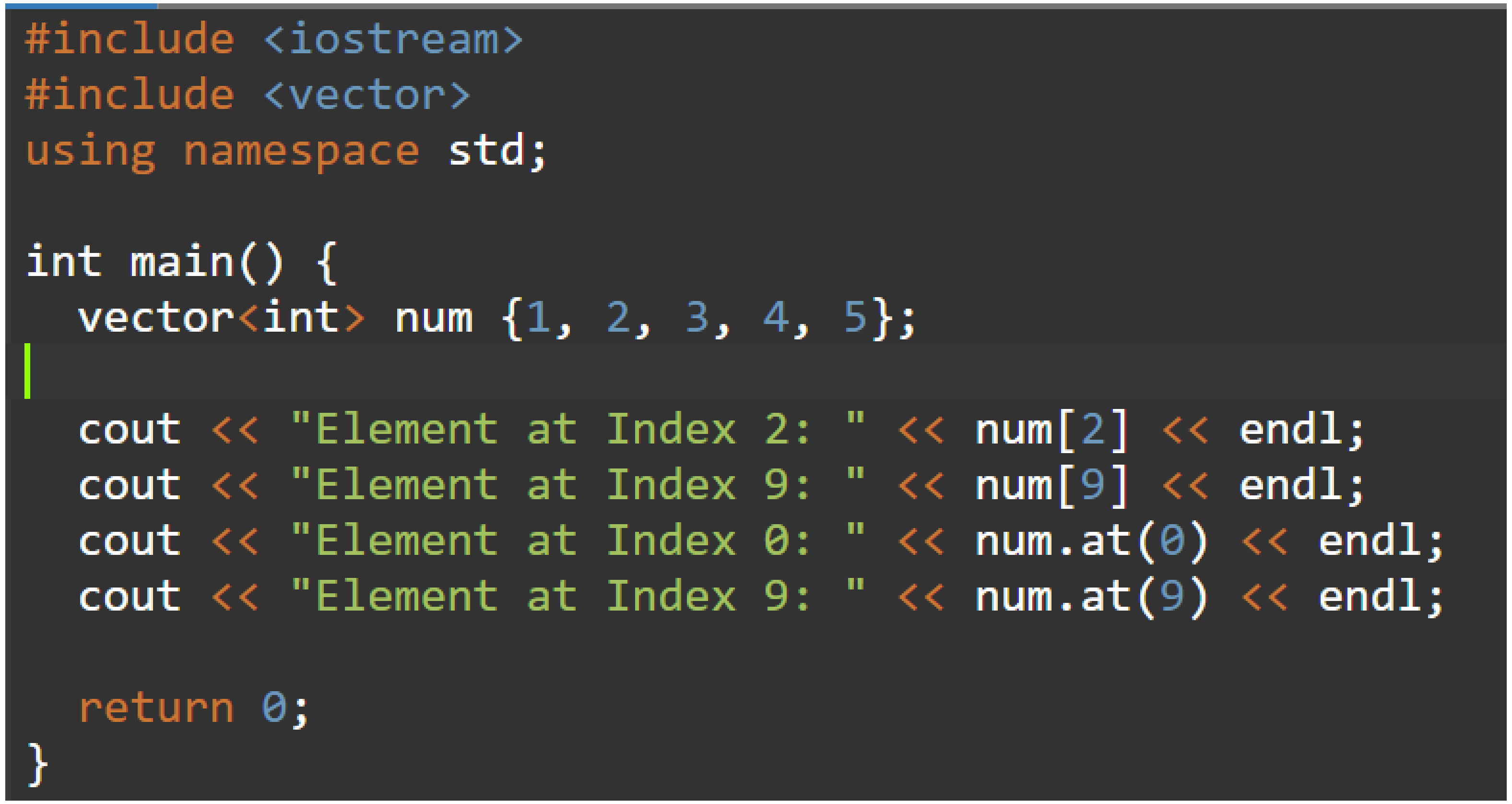
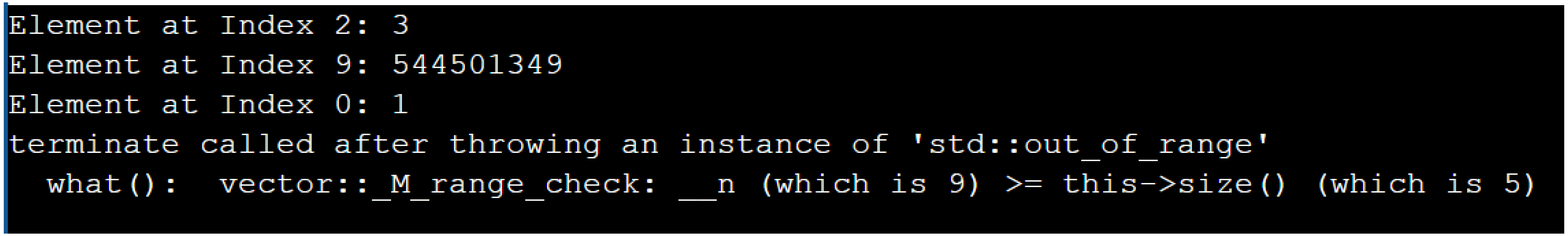
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> num {1, 2, 3, 4, 5};

  // add the integers 6 and 7 to the vector
  num.push_back(6);
  num.push_back(7);

  for (int& i : num) {
    cout << i << "  ";
  }

  return 0;
}
```



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

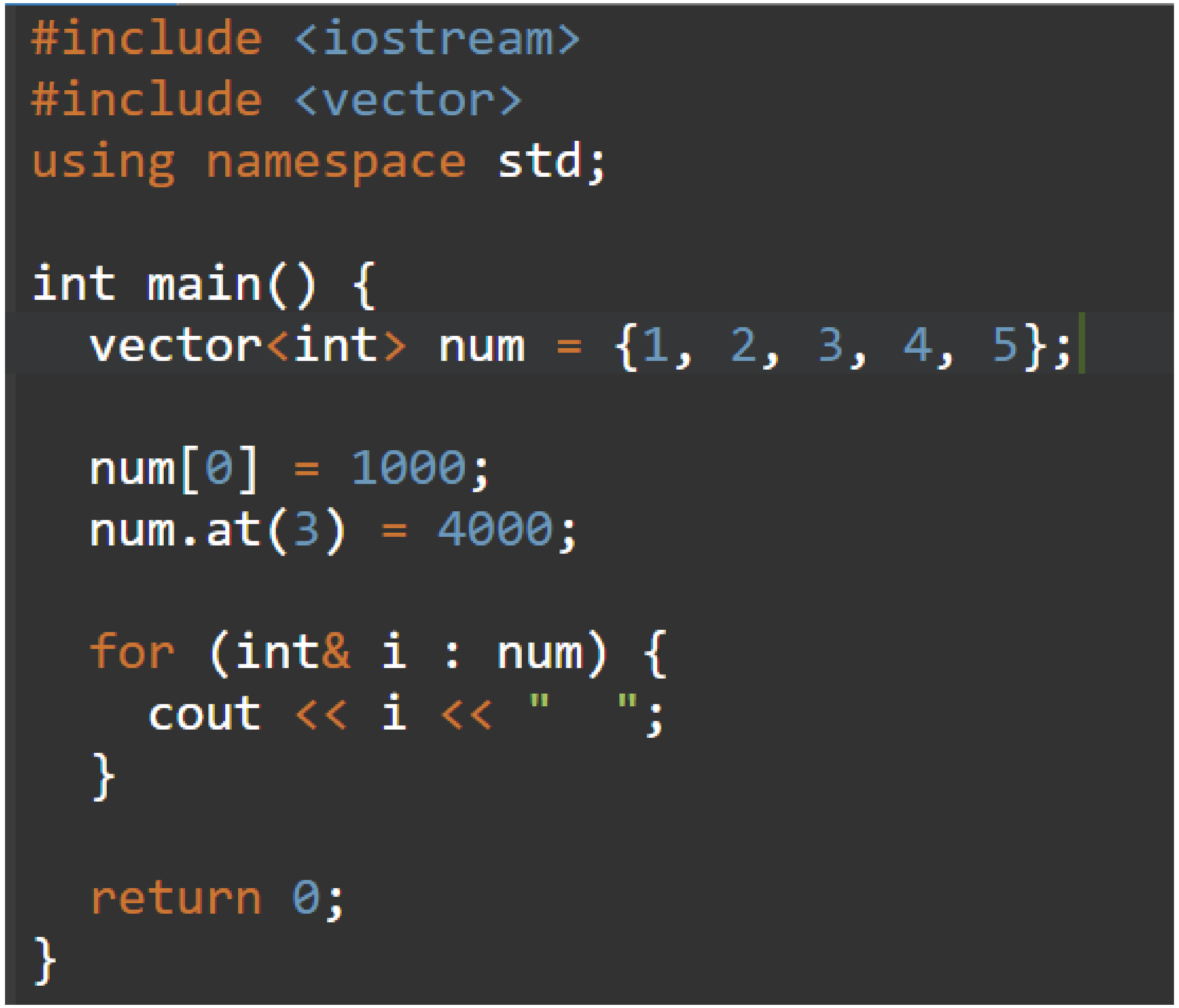
Access Elements of a Vector

- We use the index number to access the vector elements.
- Here, we use the at() function to access the element from the specified index.
- Like an array, we can also use the square brackets [] to access vector elements.
- However, the at() function is preferred over [] because at() throws an exception whenever the vector is out of bound, while [] gives a garbage value.

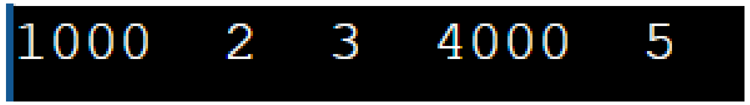

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> num {1, 2, 3, 4, 5};

  cout << "Element at Index 2: " << num[2] << endl;
  cout << "Element at Index 9: " << num[9] << endl;
  cout << "Element at Index 0: " << num.at(0) << endl;
  cout << "Element at Index 9: " << num.at(9) << endl;

  return 0;
}
```

```
Element at Index 2: 3
Element at Index 9: 544501349
Element at Index 0: 1
terminate called after throwing an instance of 'std::out_of_range'
  what():  vector::_M_range_check: __n (which is 9) >= this->size() (which is 5)
```

Change Vector Element

- We can change an element of the vector using the same at() function.
- Or, we can use [] notation but at() notation is preferred.



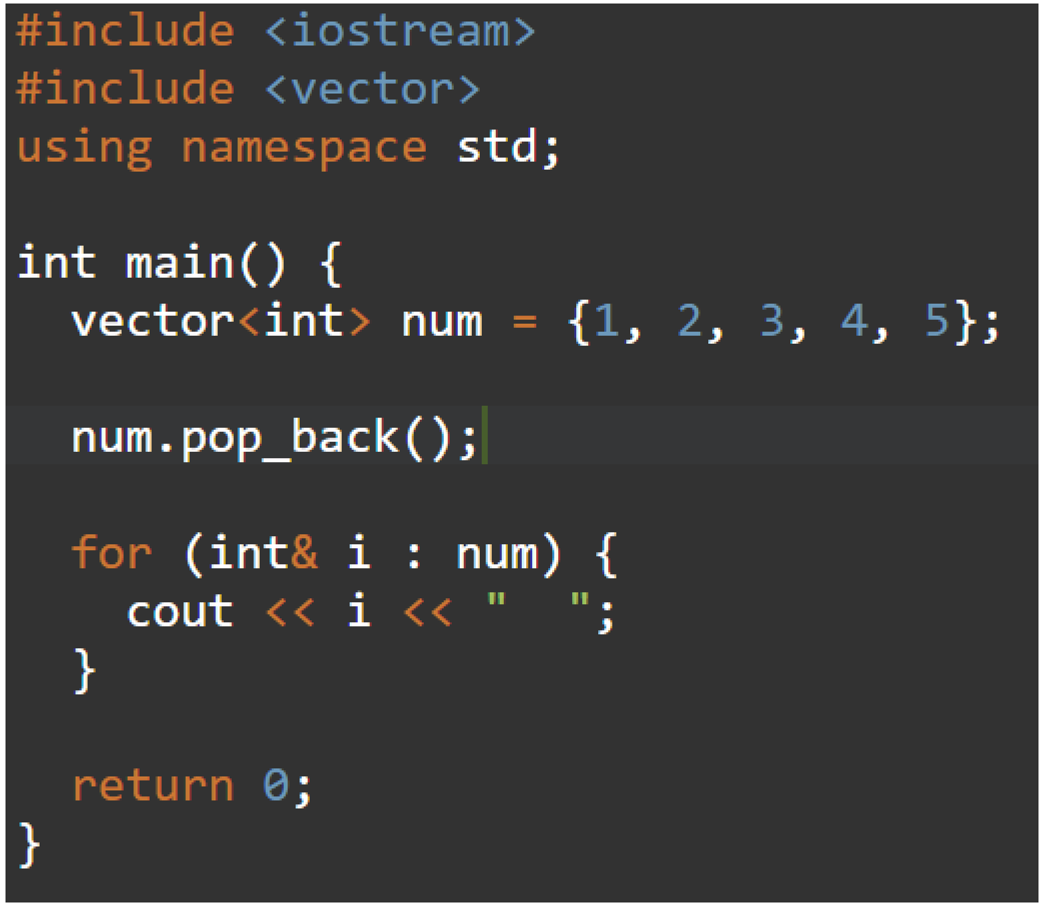
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> num = {1, 2, 3, 4, 5};

  num[0] = 1000;
  num.at(3) = 4000;

  for (int& i : num) {
    cout << i << "  ";
  }

  return 0;
}
```



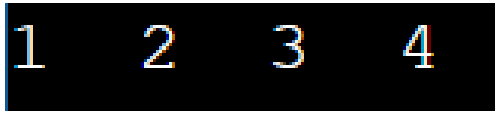
```
1000  2  3  4000  5
```

Delete Vectors Elements

- To delete a single element from a vector, we use the pop_back() function

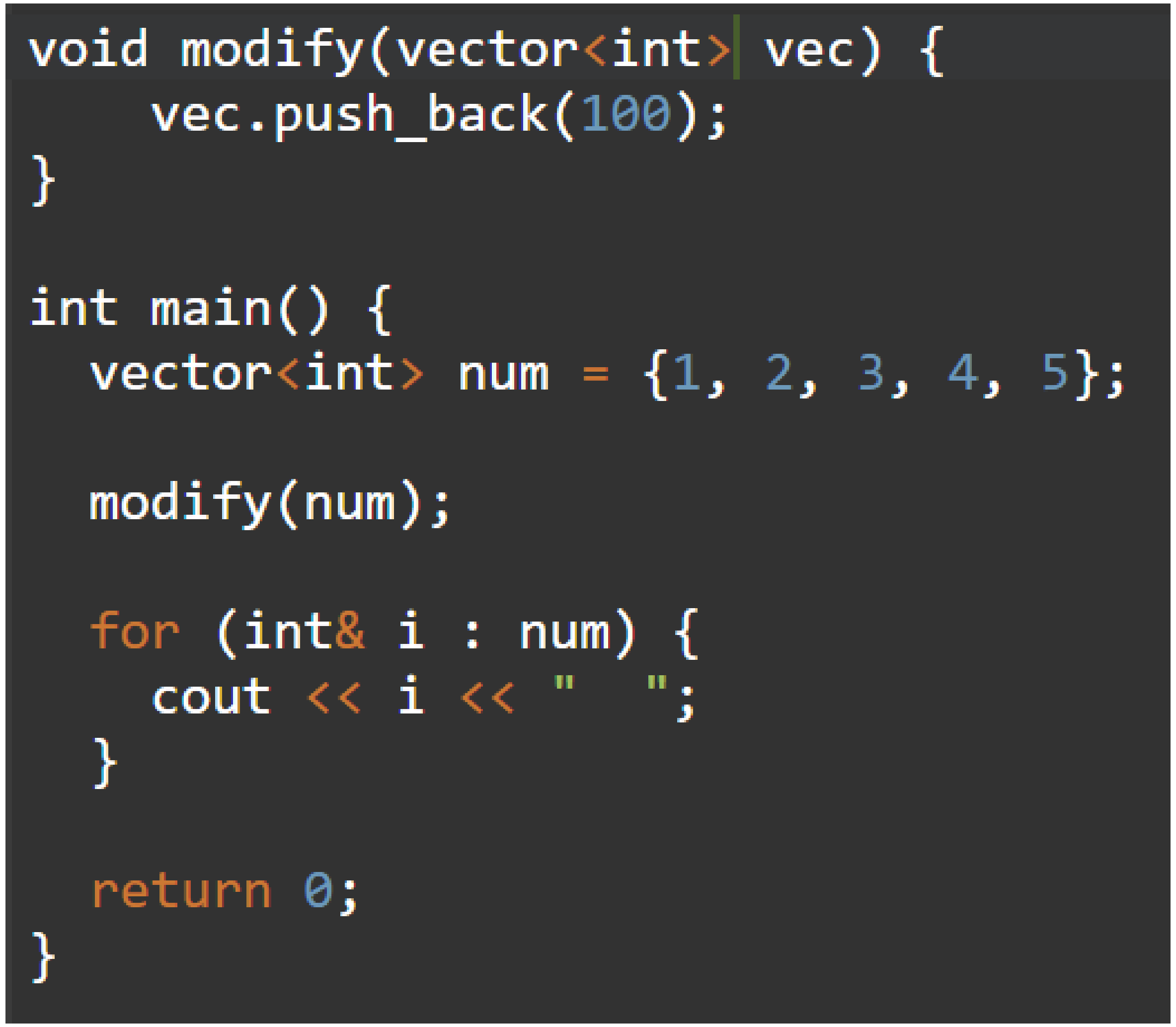


```
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> num = {1, 2, 3, 4, 5};

  num.pop_back();

  for (int& i : num) {
    cout << i << "  ";
  }

  return 0;
}
```

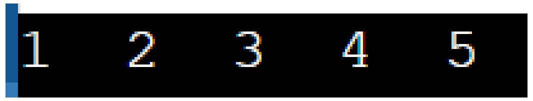


```
1   2   3   4
```

Passing Vector as Value



```
void modify(vector<int> vec) {
    vec.push_back(100);
}

int main() {
  vector<int> num = {1, 2, 3, 4, 5};

  modify(num);

  for (int& i : num) {
    cout << i << "  ";
  }

  return 0;
}
```

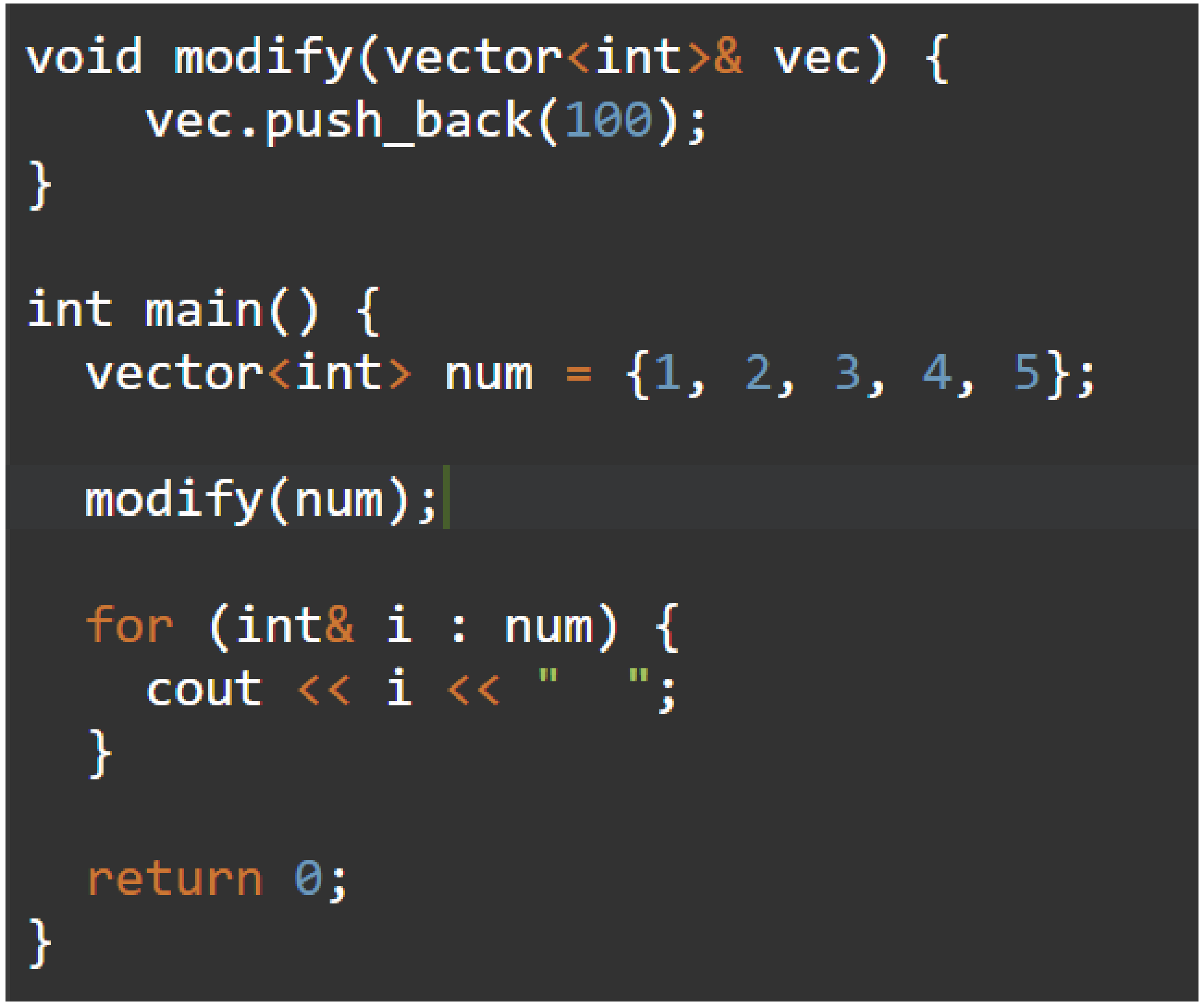


```
1   2   3   4   5
```

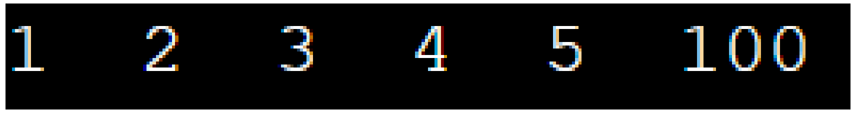
Passing by value keeps the original vector unchanged and doesn’t modify the original values of the vector.

However, the above style of passing might also take a lot of time in cases of large vectors.

Passing Vector as Reference



```
void modify(vector<int>& vec) {
    vec.push_back(100);
}

int main() {
  vector<int> num = {1, 2, 3, 4, 5};

  modify(num);

  for (int& i : num) {
    cout << i << "  ";
  }

  return 0;
}
```



```
1   2   3   4   5   100
```

Passing by reference saves a lot of time and makes the implementation of the code faster.

Other Functions

There are many member functions associated with vector.

Here is the official link: https://cplusplus.com/reference/vector/vector/

Back To Sorting

- Selection Sort
- Insertion Sort

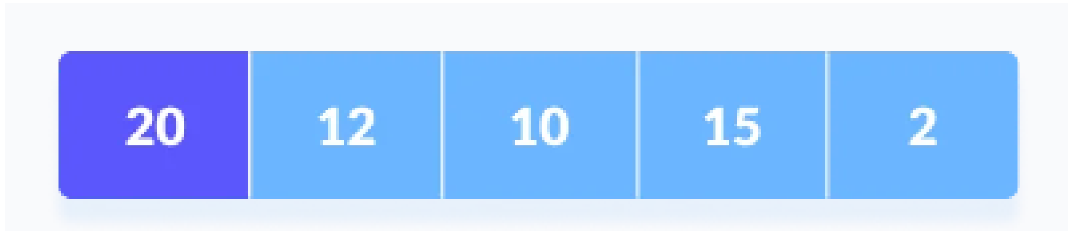
Selection Sort

Selection sort is a sorting algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

Steps

- Let’s consider this array: [20, 12, 10, 15, 2]
- We want to sort this array in ascending order

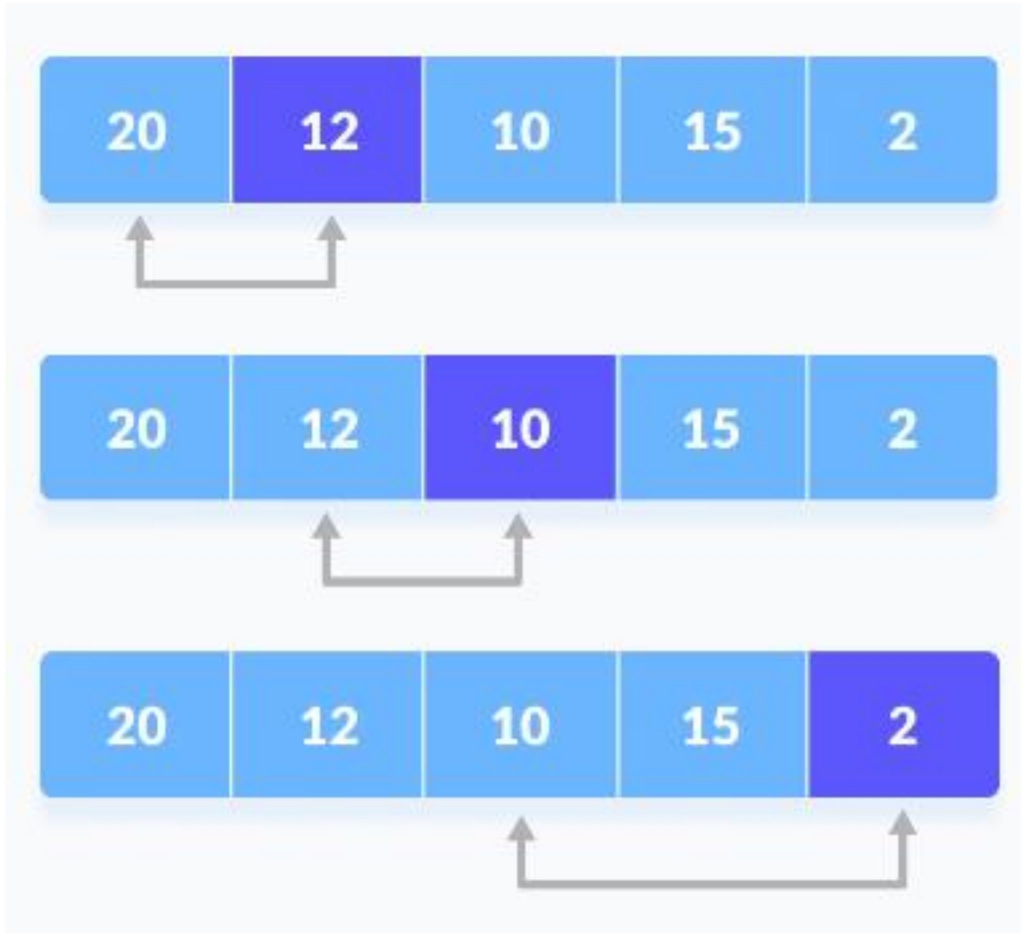
1. Set the first element as minimum



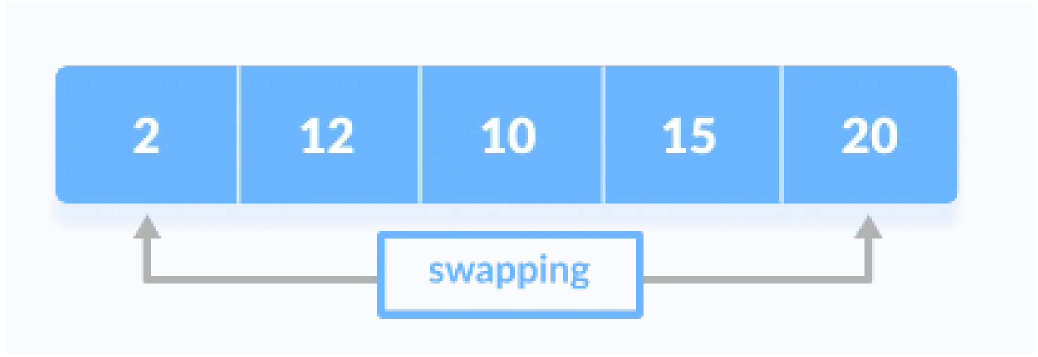
2. Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum.

Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing.

The process goes on until the last element.



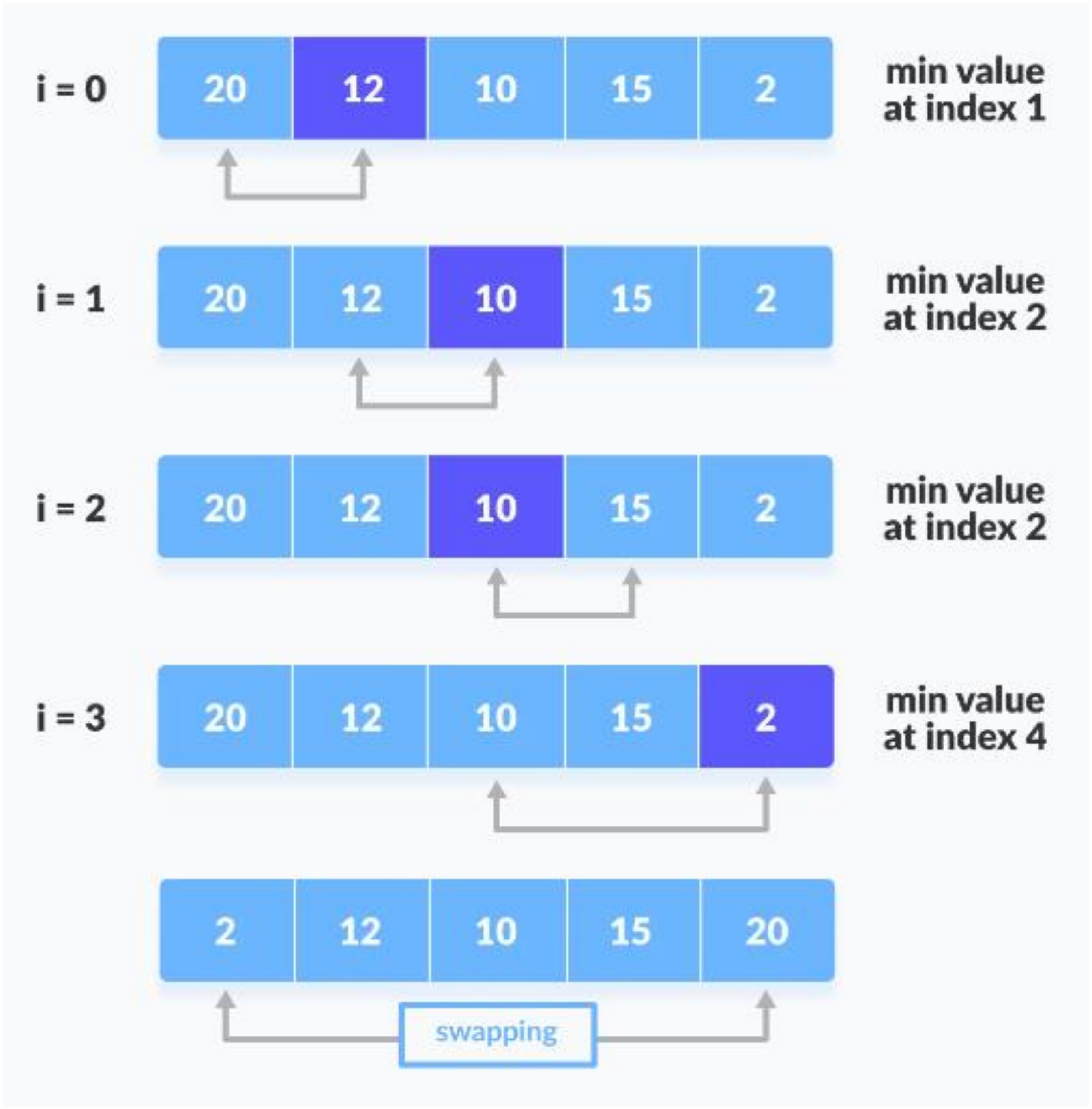
3. After each iteration, minimum is placed in the front of the unsorted list.



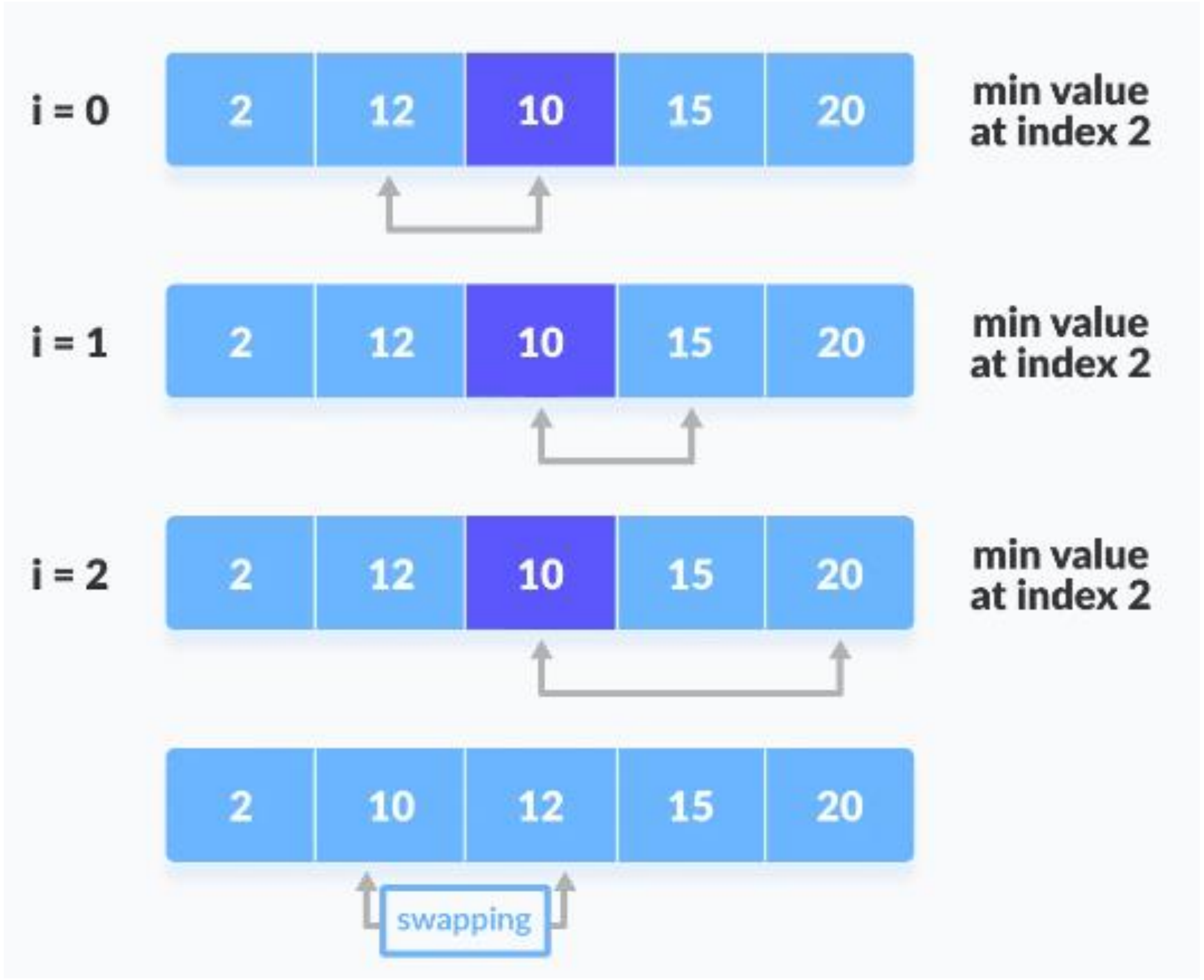
4. For each iteration, indexing starts from the first unsorted element.

Step 1 to 3 are repeated until all the elements are placed at their correct positions.

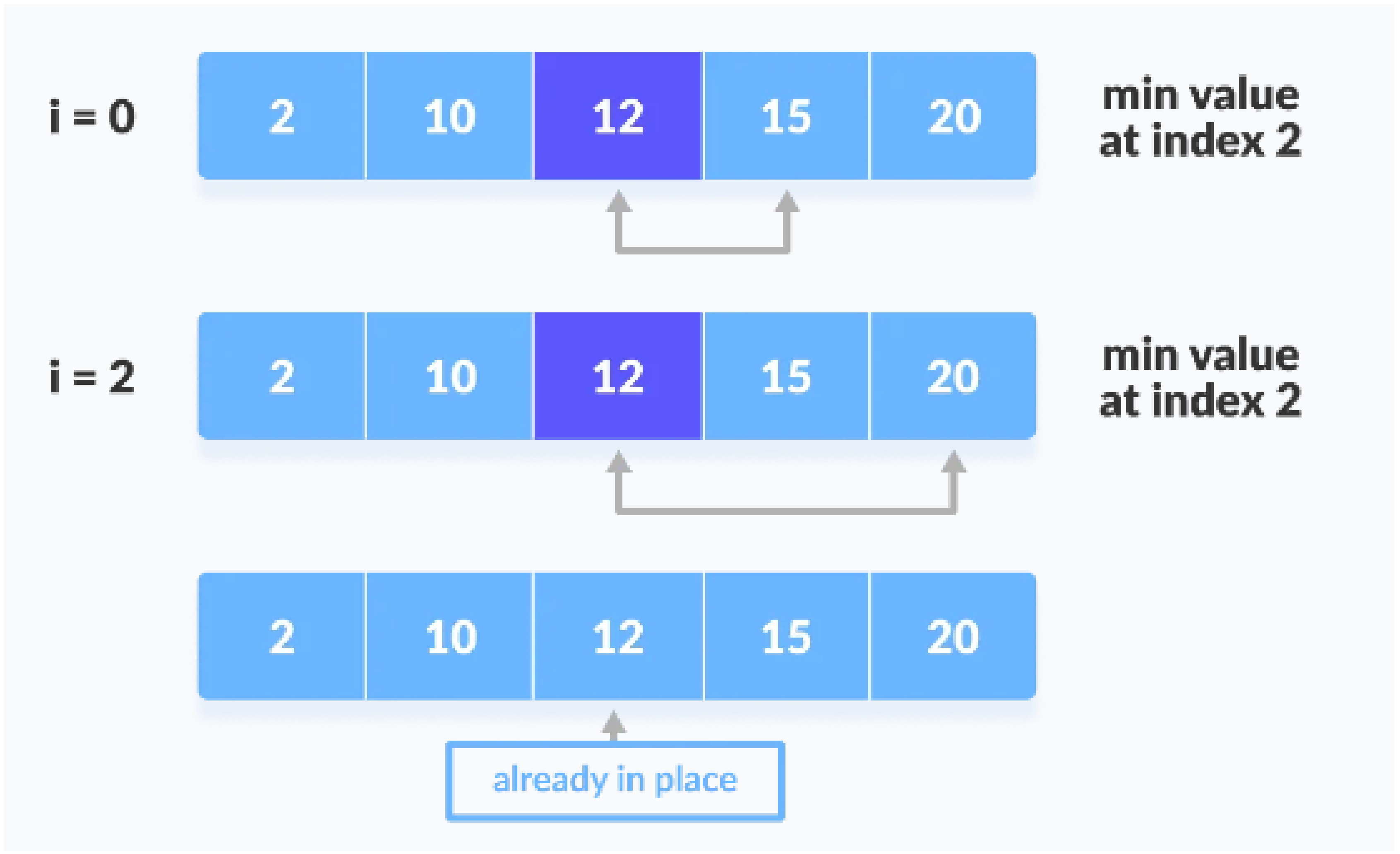
Iteration 1


i = 0
20 12 10 15 2
min value at index 1
i = 1
20 12 10 15 2
min value at index 2
i = 2
20 12 10 15 2
min value at index 2
i = 3
20 12 10 15 2
min value at index 4
2 12 10 15 20
swapping

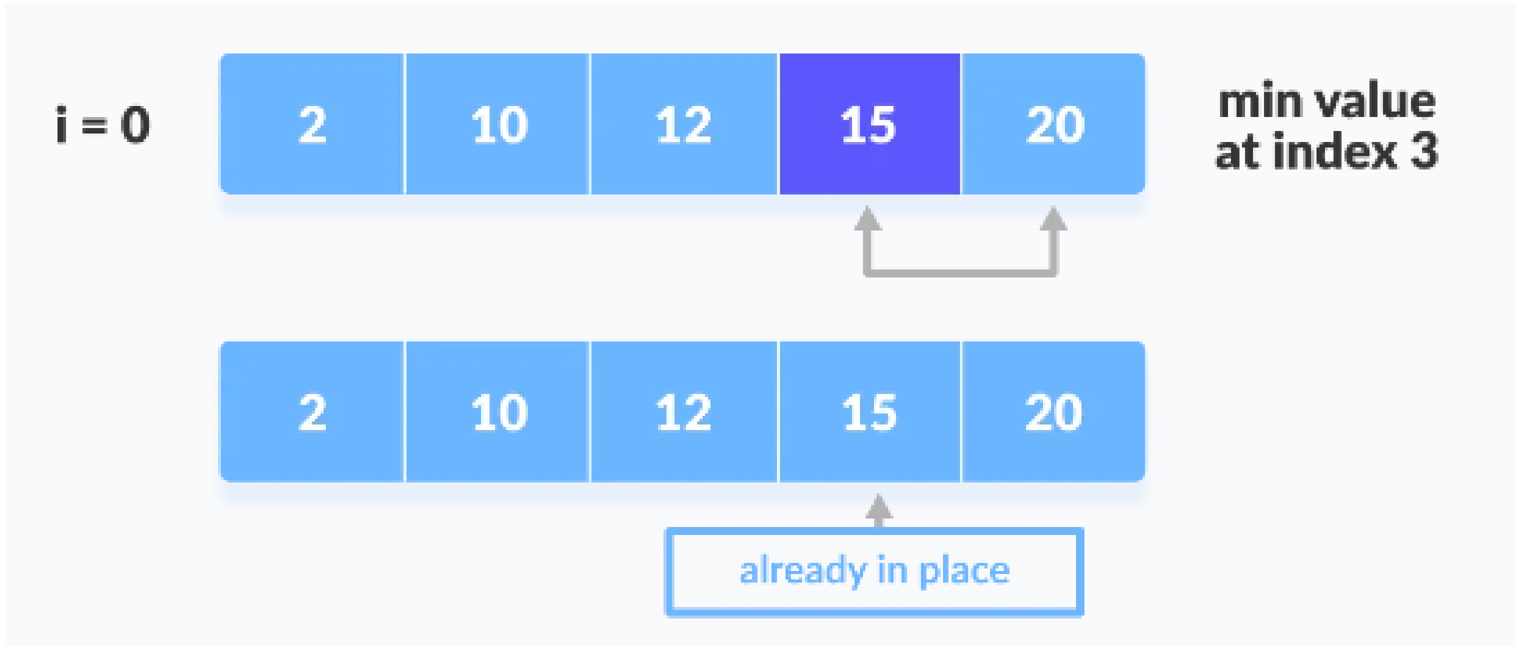
Iteration 2


i = 0
2
12
10
15
20
min value at index 2
i = 1
2
12
10
15
20
min value at index 2
i = 2
2
12
10
15
20
min value at index 2
2
10
12
15
20
swapping

Iteration 3

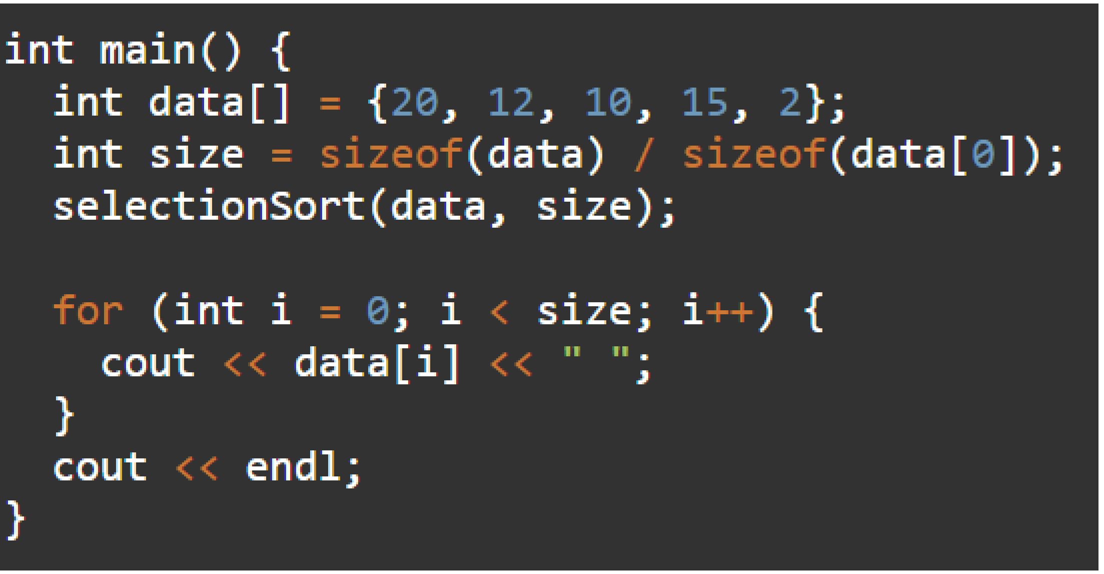

i = 0
2
10
12
15
20
min value at index 2
i = 2
2
10
12
15
20
min value at index 2
2
10
12
15
20
already in place

Iteration 4

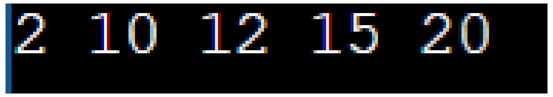

i = 0
2
10
12
15
20
min value at index 3
2
10
12
15
20
already in place

Selection Sort Pseudocode

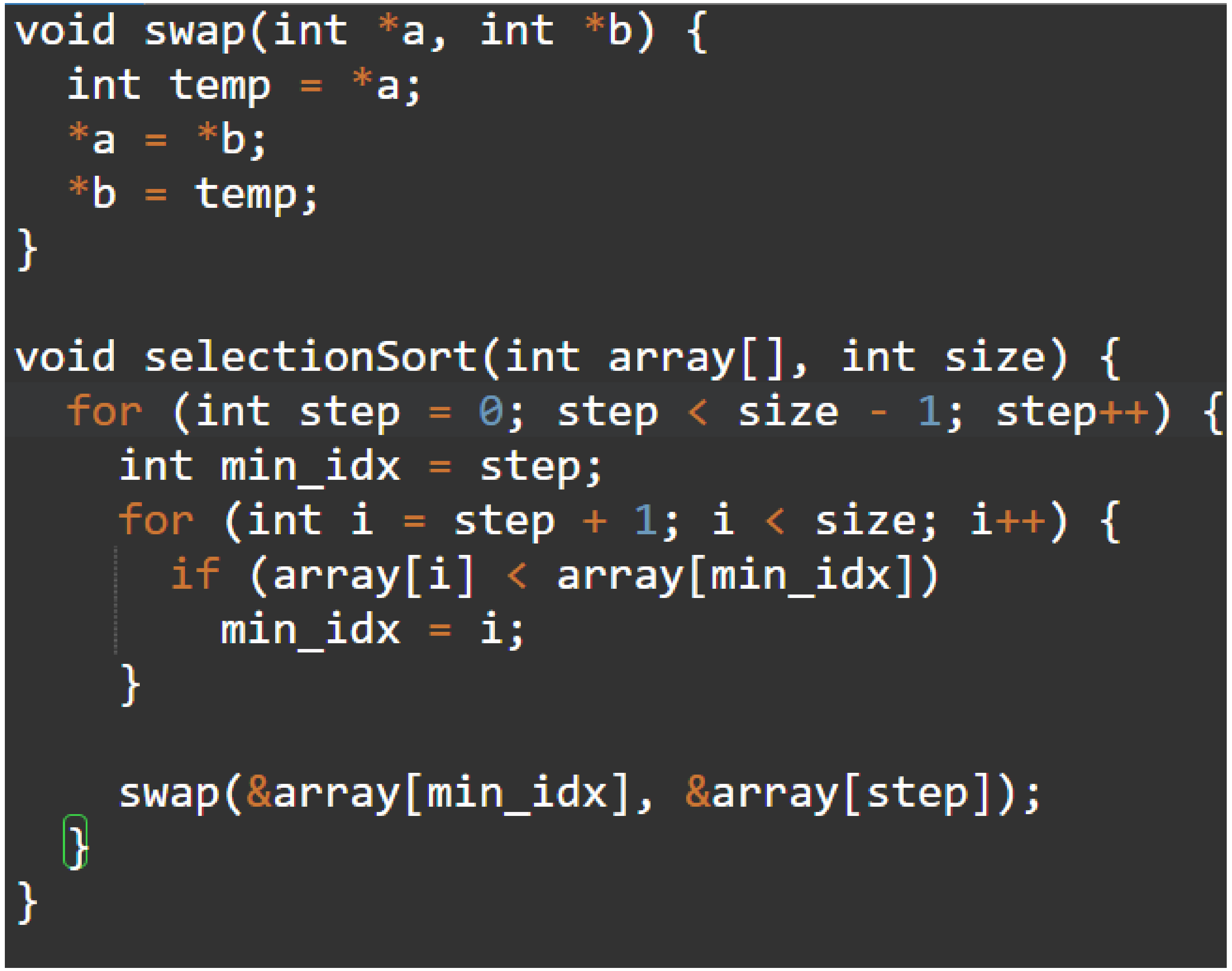
```
selectionSort(array, size)
    repeat (size - 1) times
    set the first unsorted element as the minimum
    for each of the unsorted elements
        if element < currentMinimum
            set element as new minimum
    swap minimum with first unsorted position
```



```
int main() {
  int data[] = {20, 12, 10, 15, 2};
  int size = sizeof(data) / sizeof(data[0]);
  selectionSort(data, size);

  for (int i = 0; i < size; i++) {
    cout << data[i] << " ";
  }
  cout << endl;
}
```

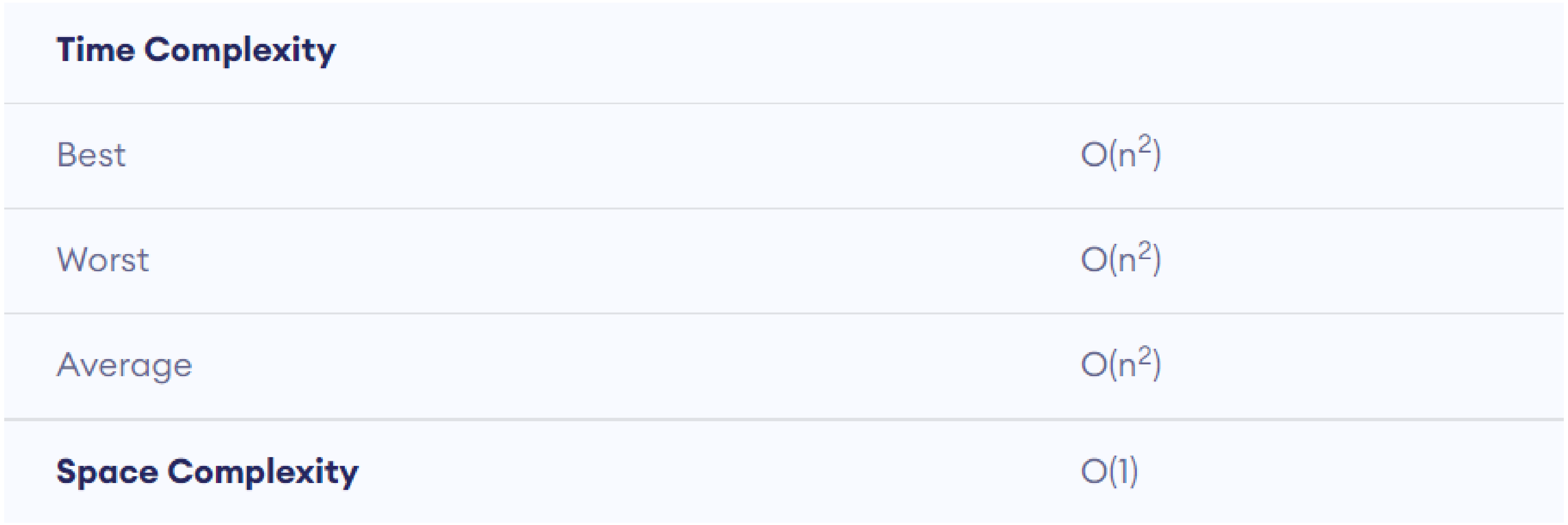


```
2 10 12 15 20
```



```
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

void selectionSort(int array[], int size) {
  for (int step = 0; step < size - 1; step++) {
    int min_idx = step;
    for (int i = step + 1; i < size; i++) {
      if (array[i] < array[min_idx])
        min_idx = i;
    }

    swap(&array[min_idx], &array[step]);
  }
}
```

Selection Sort Complexity



Time Complexity	
Best	$O(n^2)$
Worst	$O(n^2)$
Average	$O(n^2)$
Space Complexity	$O(1)$

Practice

- Declare an integer vector called **result** with random values
- Pass the vector to a function called **sort**
- Write **Selection Sort** inside sort function
- Apply sorting to the vector
- Display the sorted vector