Tishk International University
Science Faculty
IT Department

Programming II - IT-118

Vectors

1st Grade - Fall Semester

Lecture #4

Instructor: Hemin Ibrahim
Email: hemin.ibrahim@tiu.edu.iq

Overview

- ✓ Intro to vector
- ✓ Defining a new vector
- ✓ Vectors member functions
- ✓ Sequence Containers: vector
- ✓ Deque

Introduction to Vector

- In C++, arrays are used to store sequential data which are static in nature. Generally, arrays are non-dynamic/static, that is to say, they are of fixed size, however, C++ also allows us to store data in dynamic arrays which are known as vectors in C++.
- Vectors can resize itself automatically when an element is inserted or deleted depending on the need of the task to be executed. It is not same in an array where only a given number of values can be stored under a single variable name.
- Vector is a dynamic array container provided by the Standard Template Library (STL) in C++.

Defining a new vector

Syntax: `vector<of what>`

For example :

```
vector<int> - vector of integers.
vector<string> - vector of strings.
vector<int * > - vector of pointers to integers.
```

Defining a new vector

- Basic construction
  - `vector<T> A;`

Vector name

Base element type

- Example
  - `vector<int> A;        // 0 ints`
  - `vector<float> B;      // 0 floats`

Defining a new vector

#include <vector>

- Declaration: vector<type> vectorName(size);

Example: vector<int> a(3);

Example: vector<int> b; // Zero size

Example: vector<int> A(5);   // 5 ints

Example: vector<float> B(10);   // 10 floats

- Vector can be indexed as an array, using [ ]

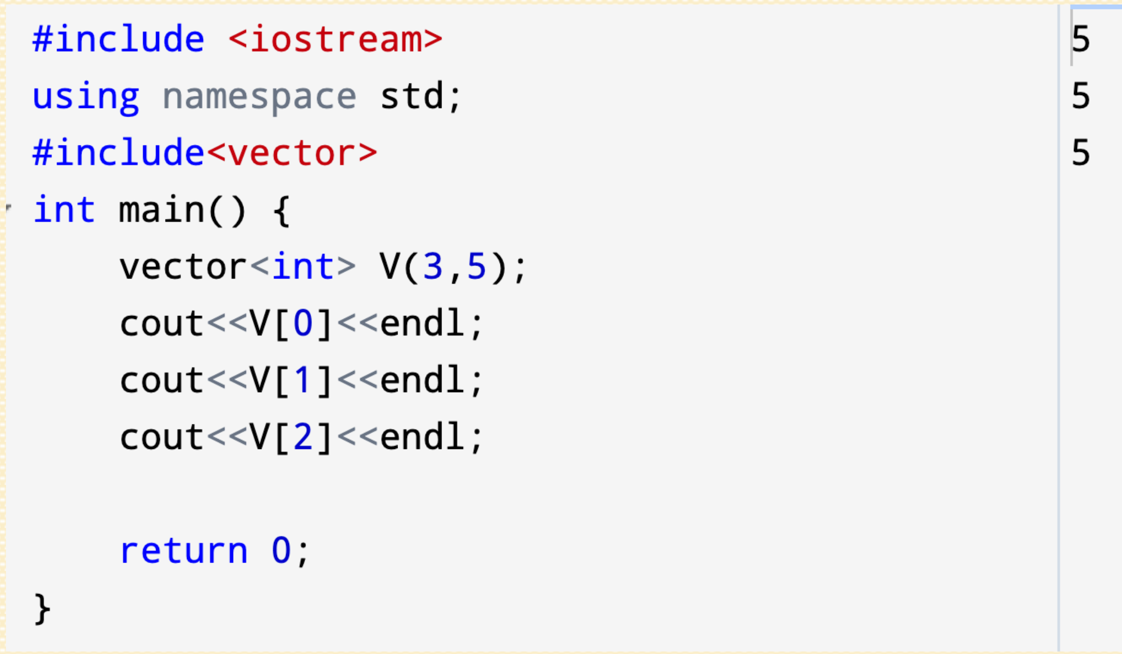
Defining a new vector

#include <vector>

- Declaration: vector<type> vectorName(size,Value);

Example: vector<int> a(3,5); // 3 5s

Example: vector<float> b(20,1.5); // 20 1.5s



```
#include <iostream>
using namespace std;
#include<vector>
int main() {
    vector<int> V(3,5);
    cout<<V[0]<<endl;
    cout<<V[1]<<endl;
    cout<<V[2]<<endl;

    return 0;
}
```

```
5
5
5
```

Defining a new vector

In the bellow example, a blank vector is being created. Vector is a dynamic array and, doesn’t needs size declaration.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
      vector<int> my_vector;
      vector<int> A = {1,2,6,-9,5};
      vector<int> C {1,2,6,-9,5};
}
```

Why Vector?!

- Provides an alternative to the built in array.
- A vector is self grown.
- Use it instead of the built in array!
- It is one of the most commonly used data structures in C++.
- Vector provides a dynamic, resizable array-like container with several built-in functionalities.

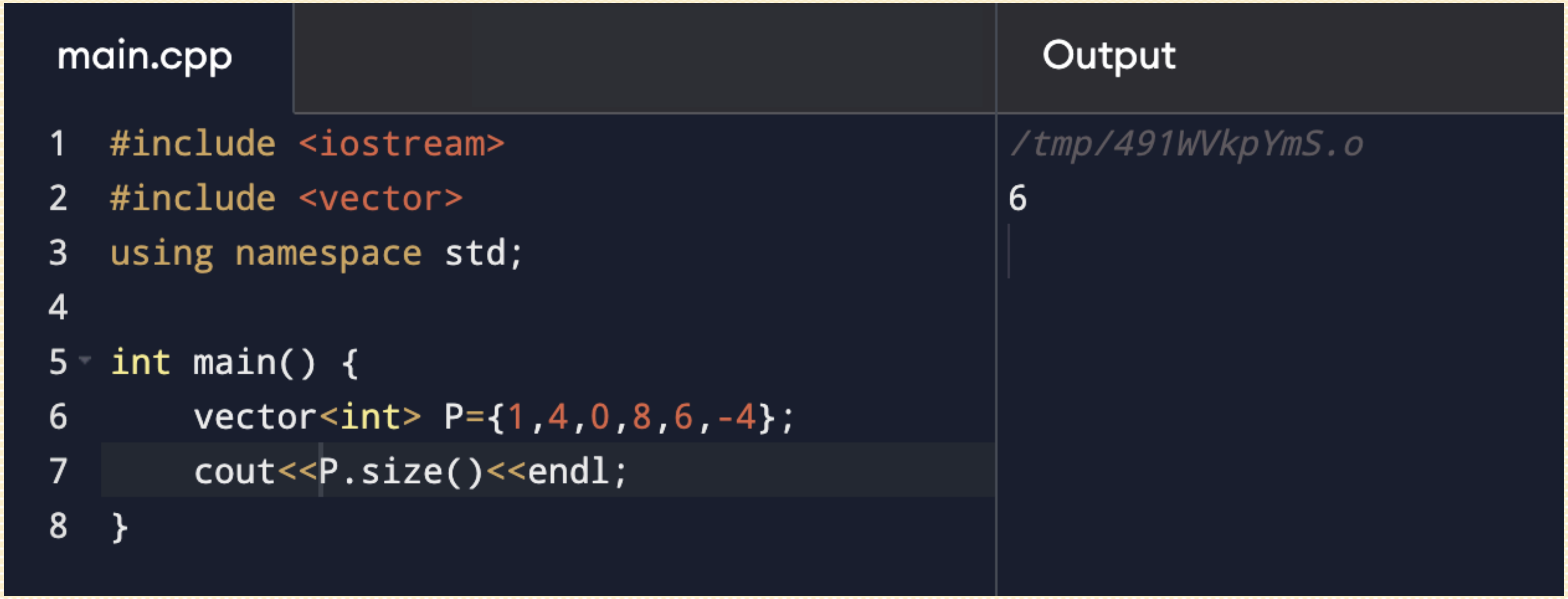
Vectors member functions

v{7,3,4,0,9,-4,-3,5,2,11}

- **v.size();** // return the number of elements in v , which is 10
- **v.push_back(14);** //appends 14 to the end of v{7,3,4,0,9,-4,-3,5,2,11,14};
- **v.at(4);** // retrieves the element at index 4, which is 9
- **v.resize(15)**   // change size of v to 15
- **v.pop_back()** // remove the last element in v {7,3,4,0,9,-4,-3,5,2};
- **v.front();** // return the first element in v, which is 7
- **v.clear();** // delete all the element in v
- **v.empty();** // return 1 if v is empty; else return 0;
- **v.back();** // return the last element in v, which is 11
- **v.begin();** //the beginning of the vector, which is 7
- **v.erase();** // erases a specific element
- **v.capacity();**// return the number of element that vector can hold before it will need to allocate more space

Vectors member functions: Size()

Returns the number of elements in the vector.



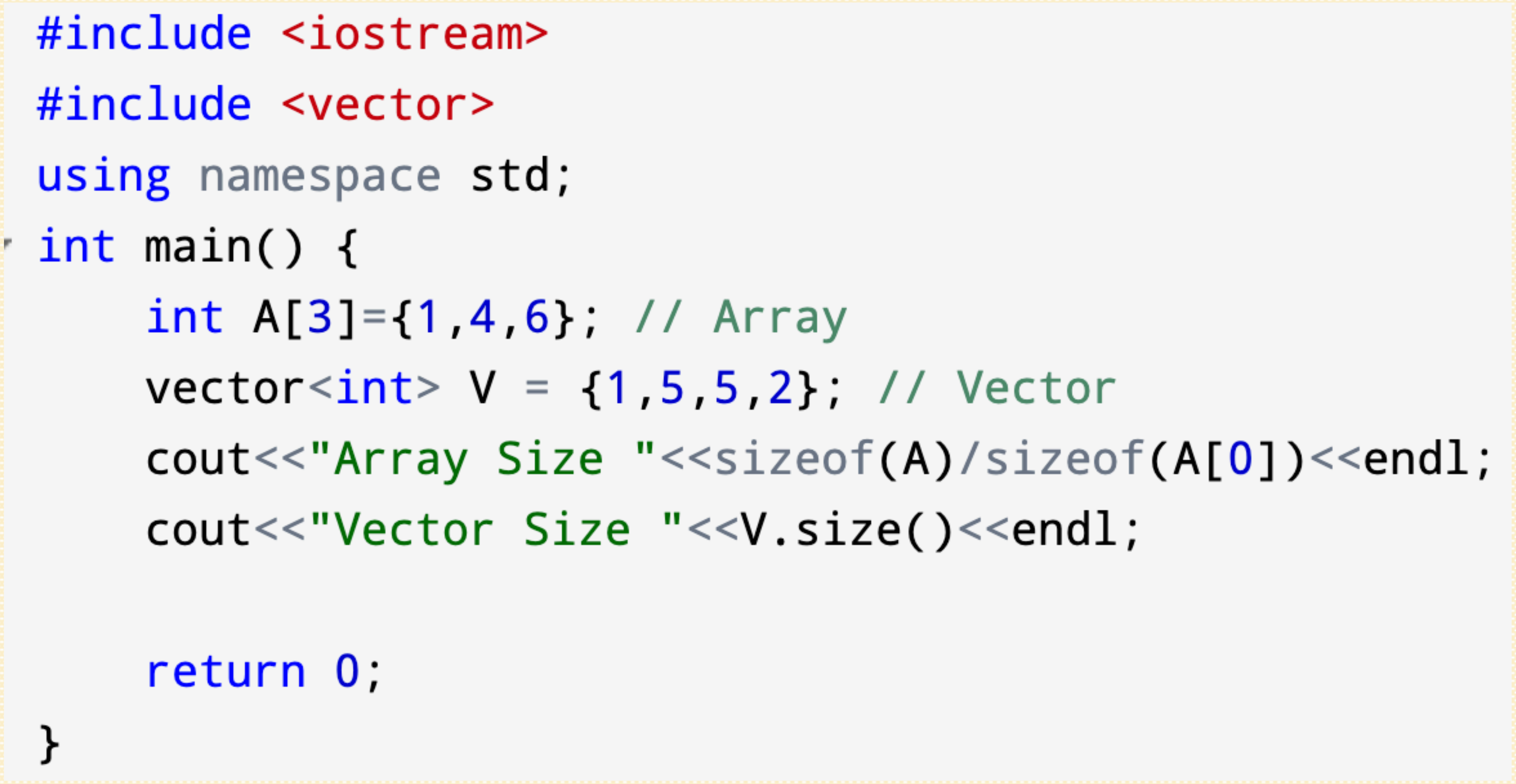
main.cpp

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

int main() {
    vector<int> P={1,4,0,8,6,-4};
    cout<<P.size()<<endl;
}
```

Output

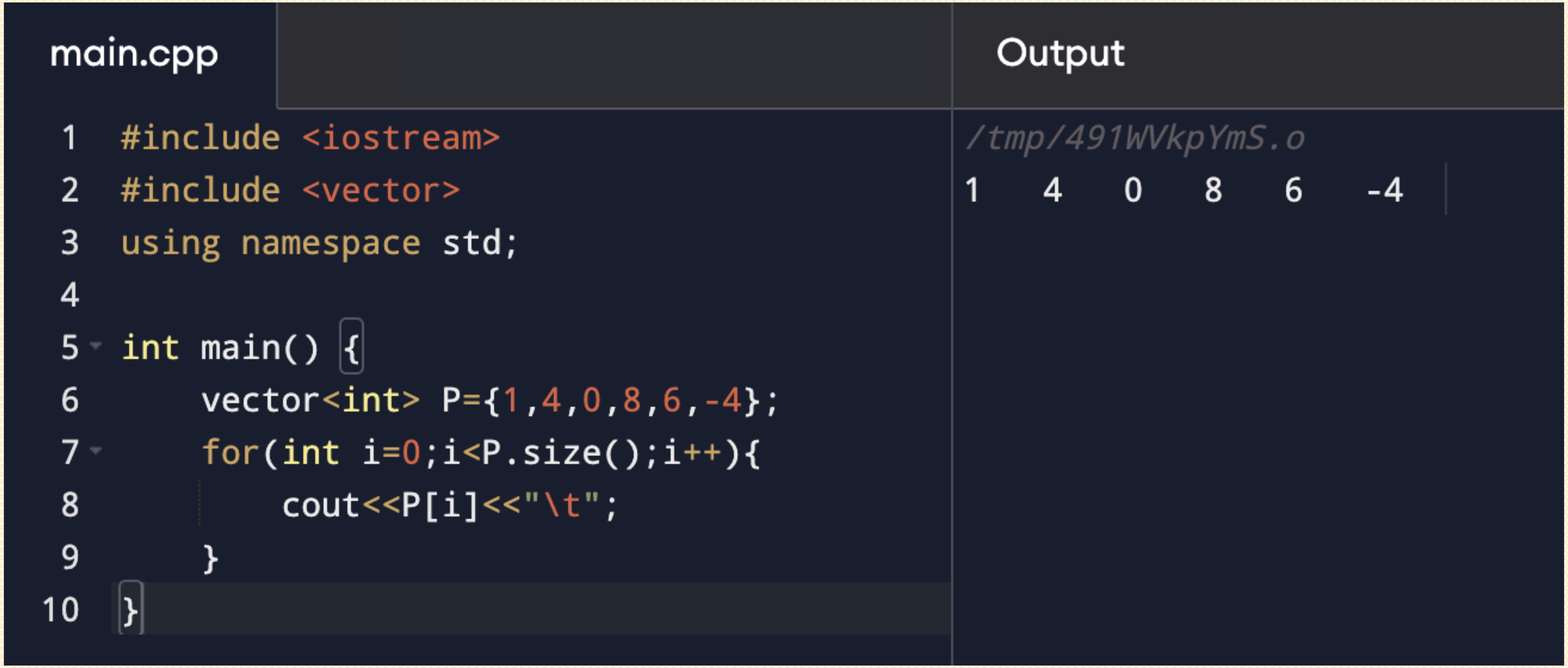
```
/tmp/491WVkpYmS.o
6
```

Vectors size vs array size



```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    int A[3]={1,4,6}; // Array
    vector<int> V = {1,5,5,2}; // Vector
    cout<<"Array Size "<<sizeof(A)/sizeof(A[0])<<endl;
    cout<<"Vector Size "<<V.size()<<endl;

    return 0;
}
```

Example of size()


main.cpp

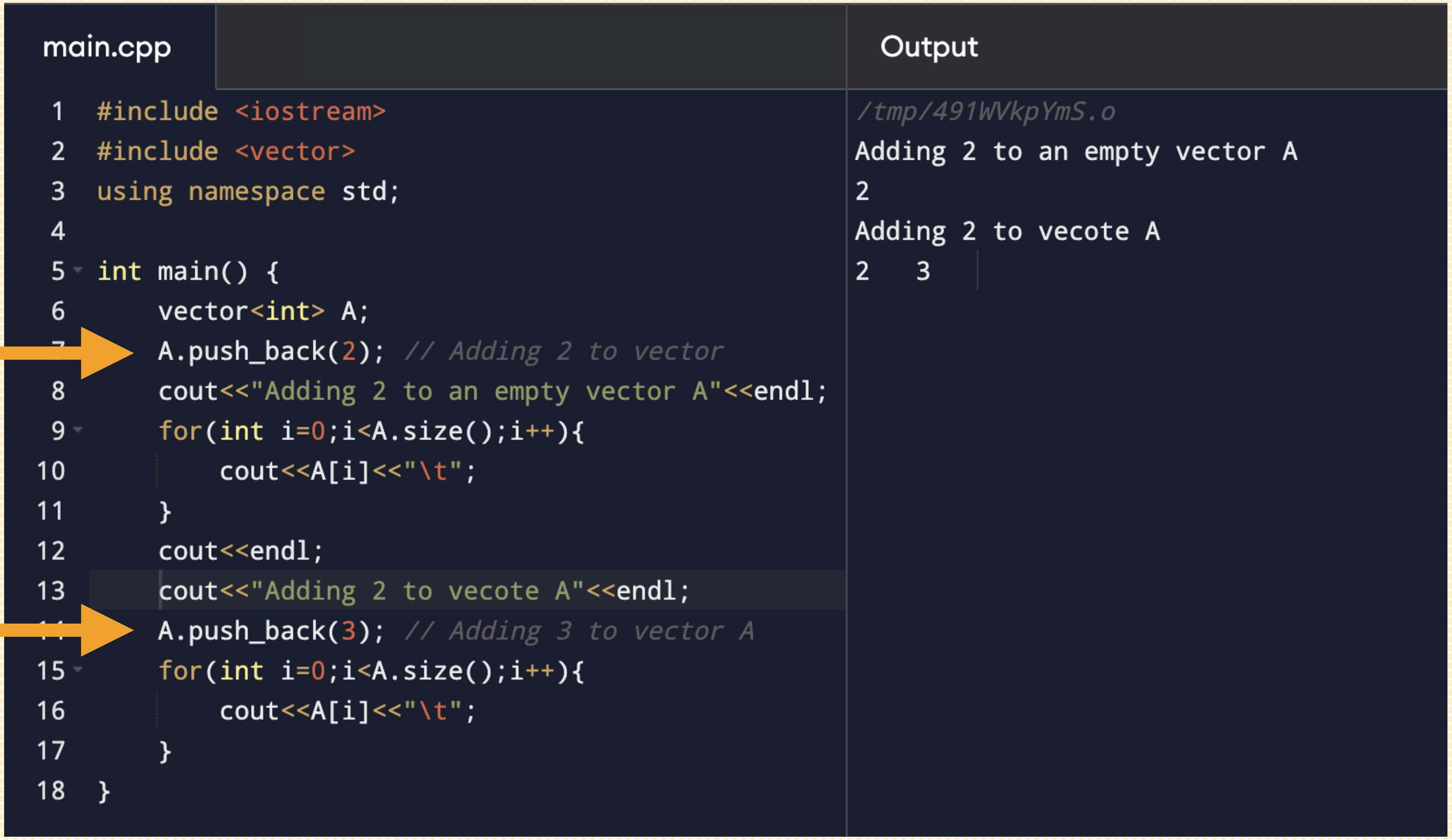
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> P={1,4,0,8,6,-4};
    for(int i=0;i<P.size();i++){
        cout<<P[i]<<"\t";
    }
}
```

Output

```
/tmp/491WVkpYmS.o
1	4	0	8	6	-4
```

Vectors member functions: Adding elements

push_back(element): Inserts an element with value x at the end of the controlled sequence.



main.cpp

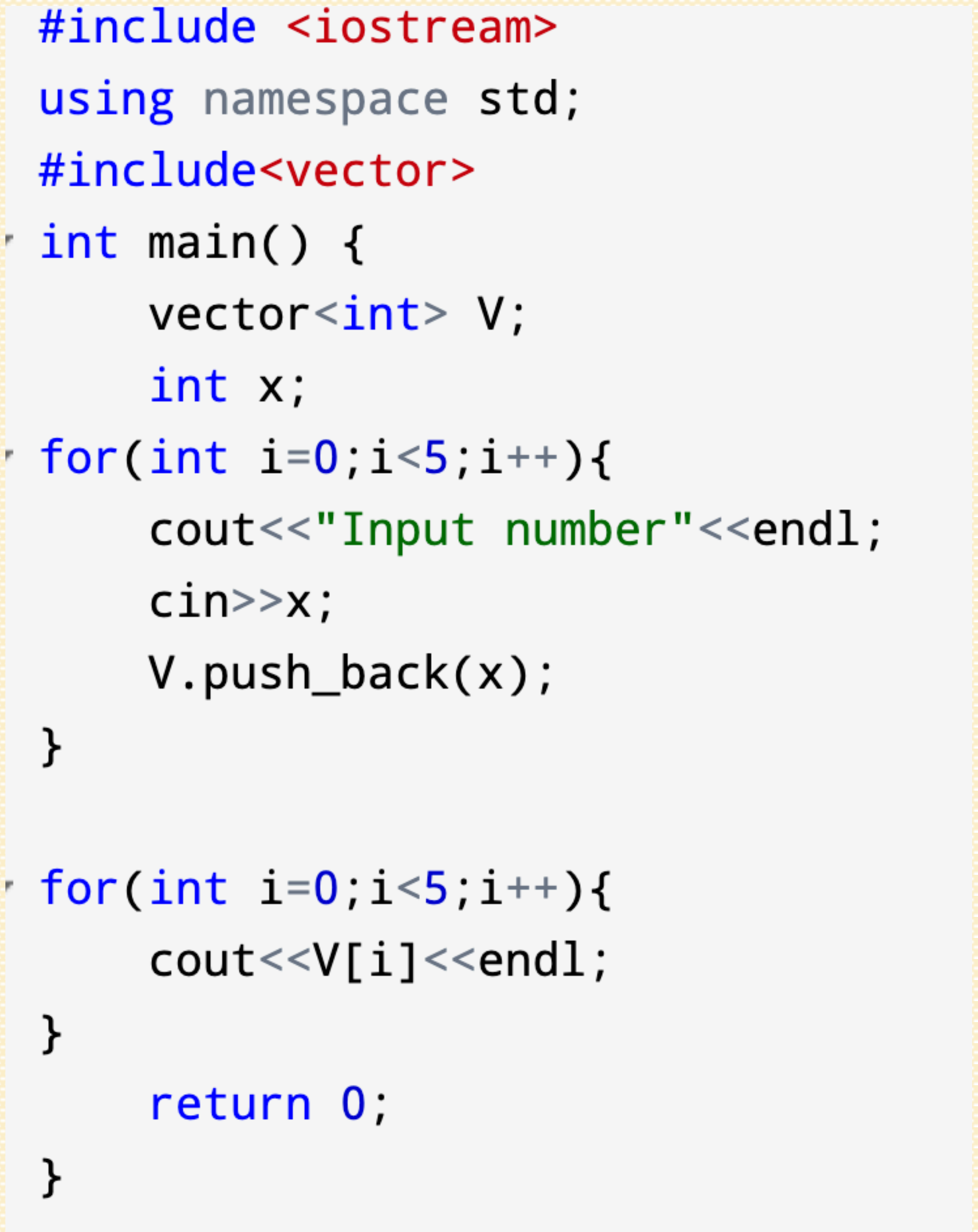
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> A;
    A.push_back(2); // Adding 2 to vector
    cout<<"Adding 2 to an empty vector A"<<endl;
    for(int i=0;i<A.size();i++){
        cout<<A[i]<<"\t";
    }
    cout<<endl;
    cout<<"Adding 2 to vecote A"<<endl;
    A.push_back(3); // Adding 3 to vector A
    for(int i=0;i<A.size();i++){
        cout<<A[i]<<"\t";
    }
}
```

Output

```
/tmp/491WVkpYmS.o
Adding 2 to an empty vector A
2
Adding 2 to vecote A
2    3
```

Adding numbers by For loop

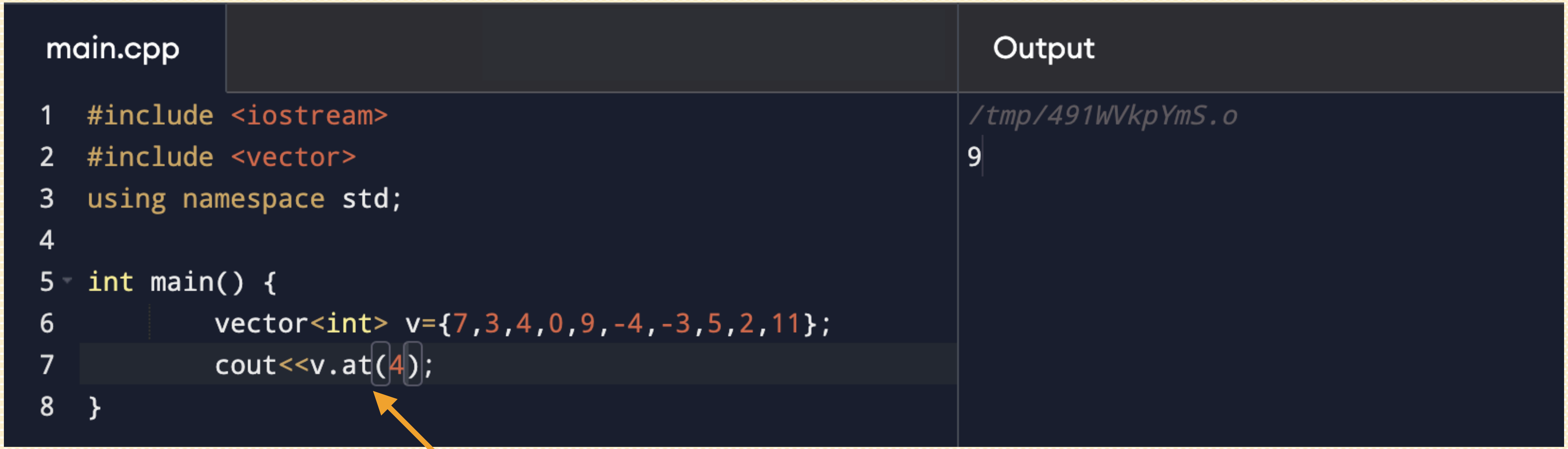
Use a for loop to iteratively input values into the vector.



```
#include <iostream>
using namespace std;
#include<vector>
int main() {
    vector<int> V;
    int x;
for(int i=0;i<5;i++){
    cout<<"Input number"<<endl;
    cin>>x;
    V.push_back(x);
}

for(int i=0;i<5;i++){
    cout<<V[i]<<endl;
}
    return 0;
}
```

Vectors member functions: at()

at(index): retrieves the element at index 4, which is 9



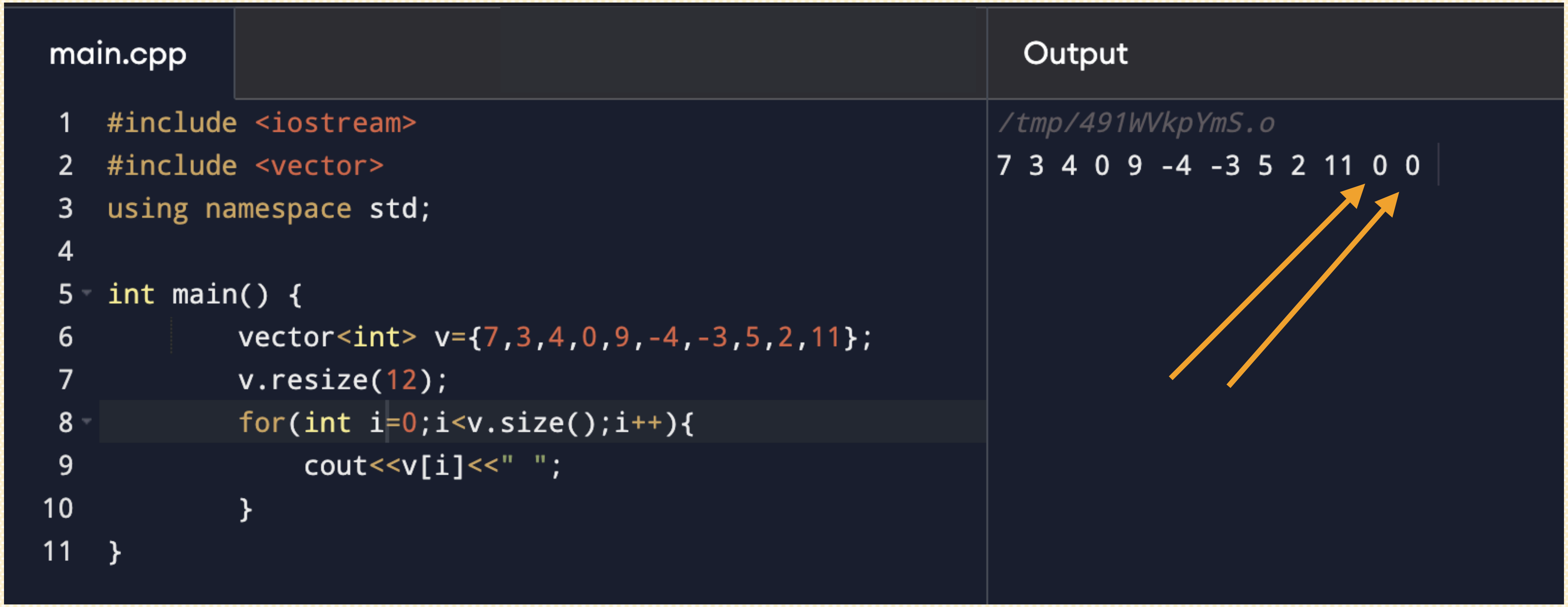
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
        vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
        cout<<v.at(4);
}
```

```
/tmp/491WVkpYmS.o
9
```

Same as v[4]

Vectors member functions: resize()

resize(newSize): Resizing a vector to a new size.


main.cpp

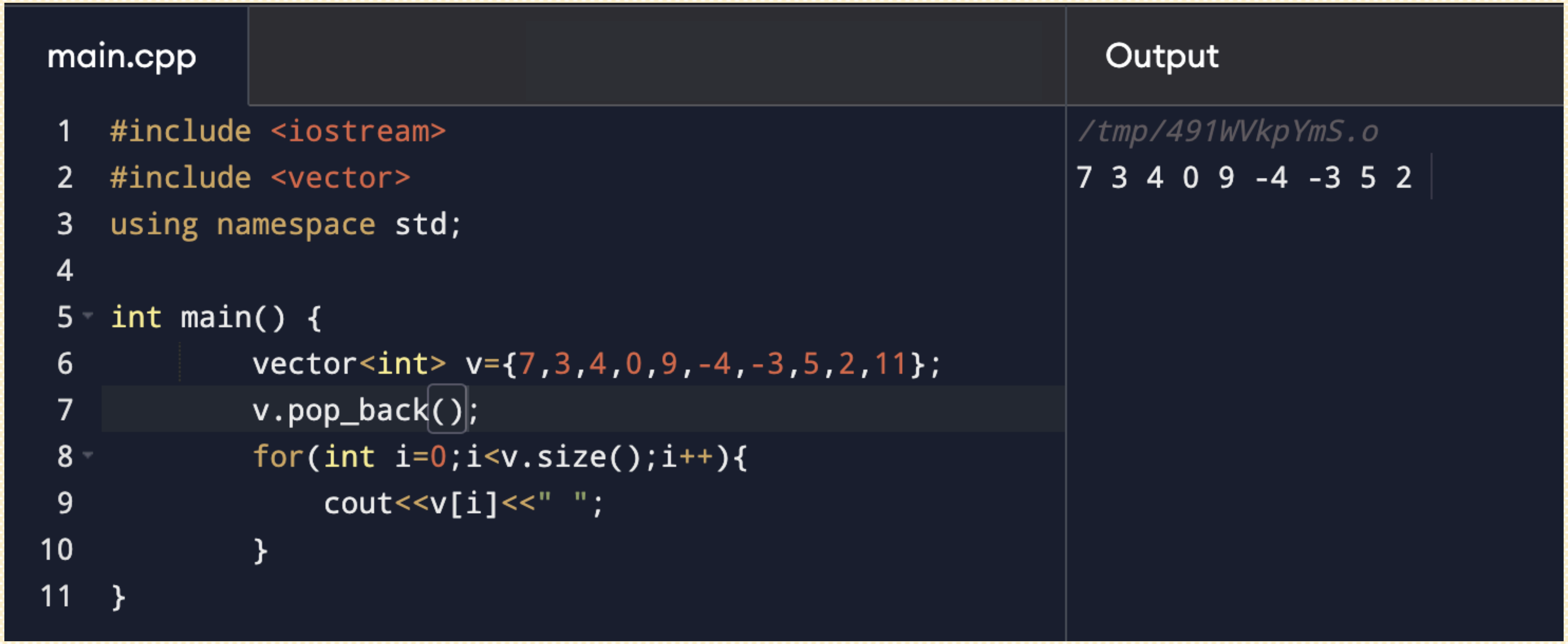
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
        vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
        v.resize(12);
        for(int i=0;i<v.size();i++){
            cout<<v[i]<<" ";
        }
}
```

Output

```
/tmp/491WVkpYmS.o
7 3 4 0 9 -4 -3 5 2 11 0 0
```

Vectors member functions: pop_back()

v.pop_back(): Remove the last element of the vector



main.cpp

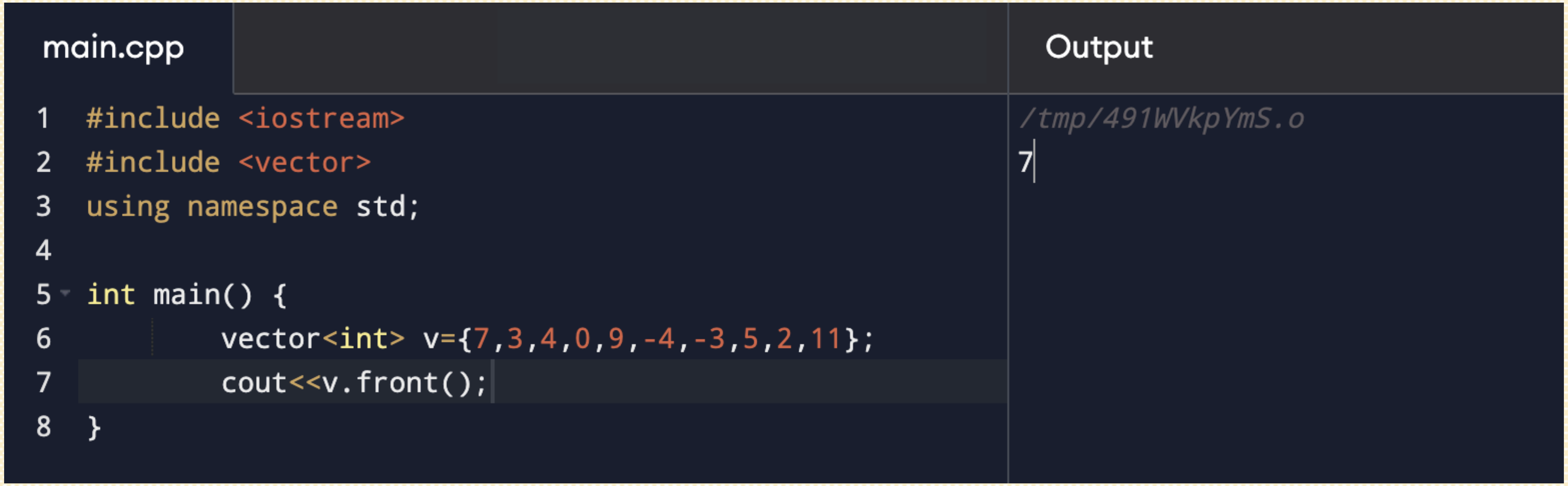
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
        vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
        v.pop_back();
        for(int i=0;i<v.size();i++){
            cout<<v[i]<<" ";
        }
}
```

Output

```
/tmp/491WVkpYmS.o
7 3 4 0 9 -4 -3 5 2
```

Vectors member functions: front()

v.front(): return the first element in v, which is 7



main.cpp

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

int main() {
        vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
        cout<<v.front();
}
```

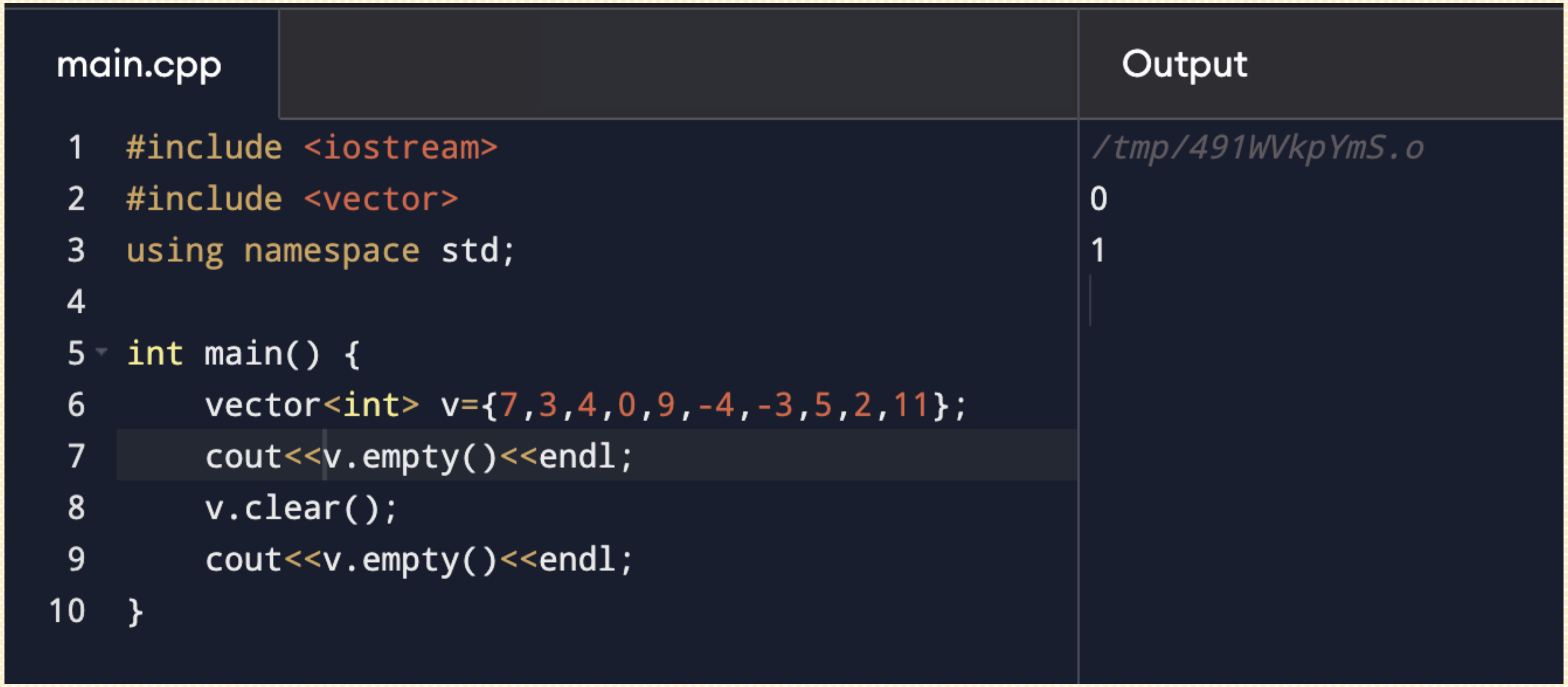
Output

```
/tmp/491WVkpYmS.o
7
```

Vectors member functions: clear() & empty()

v.clear(): Remove all elements inside the vector

v.empty(): Check the empty vector, return 1 if vector is empty; else return 0;



main.cpp

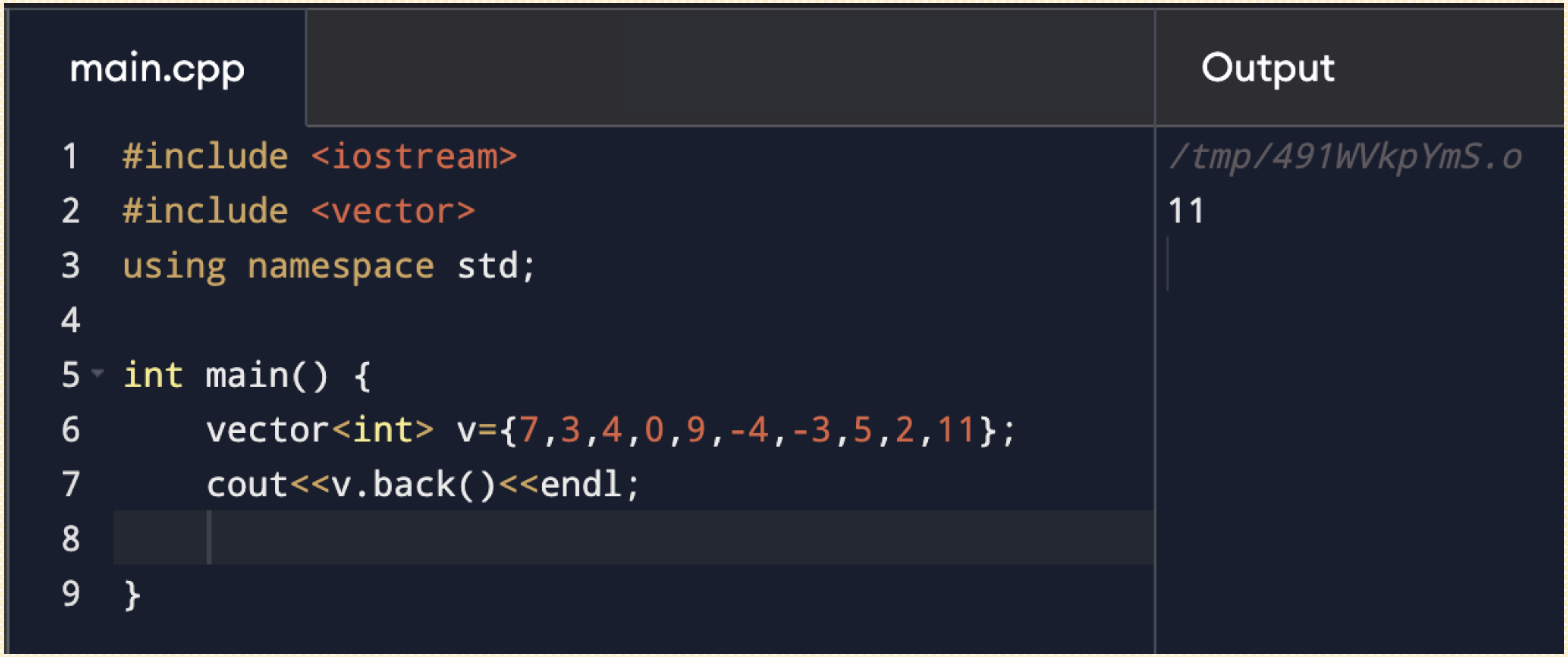
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
    cout<<v.empty()<<endl;
    v.clear();
    cout<<v.empty()<<endl;
}
```

Output

```
/tmp/491WVkpYmS.o
0
1
```

Vectors member functions: back()

v.back(): Return the last element in the vector


main.cpp

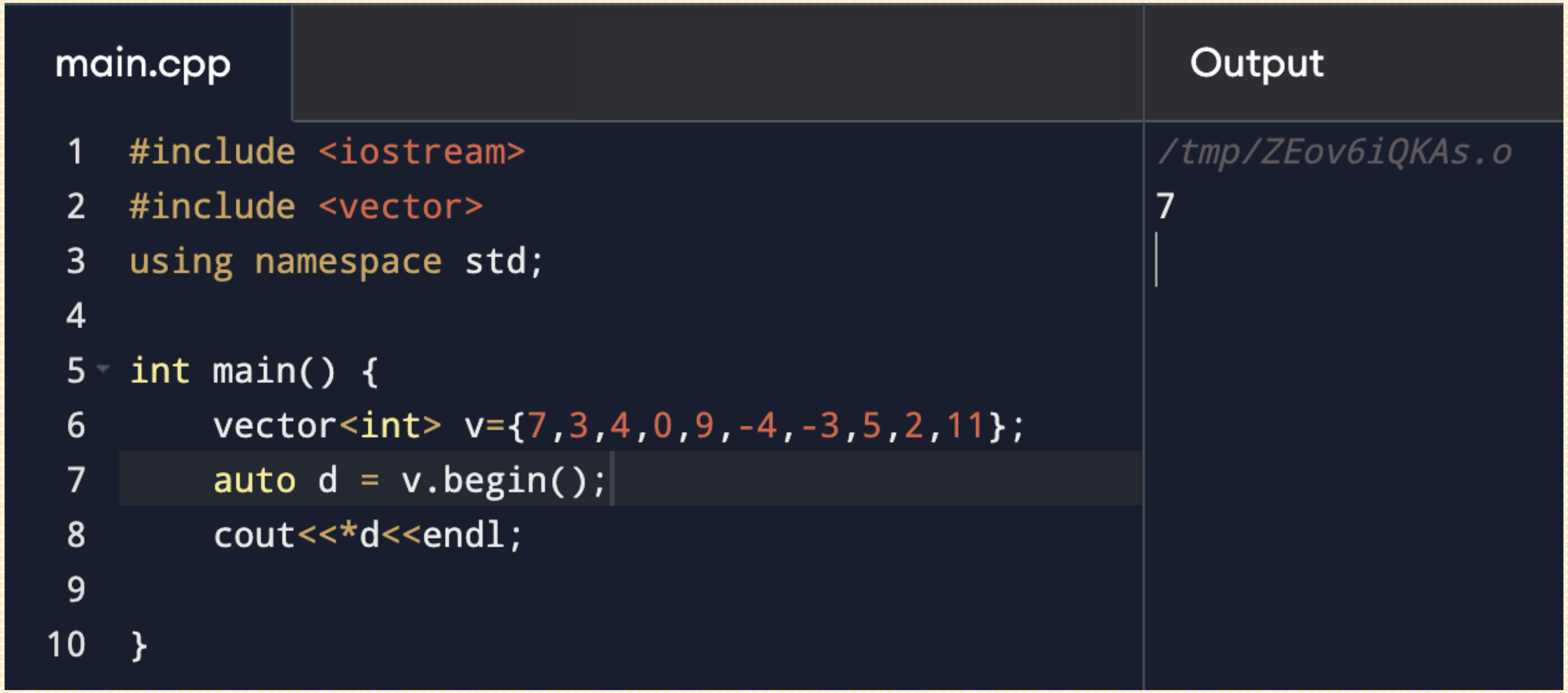
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
    cout<<v.back()<<endl;
    
}
```

Output

```
/tmp/491WVkpYmS.o
11
```

Vectors member functions: begin()

v.begin(): It returns an iterator pointing to the first element of the vector.



main.cpp

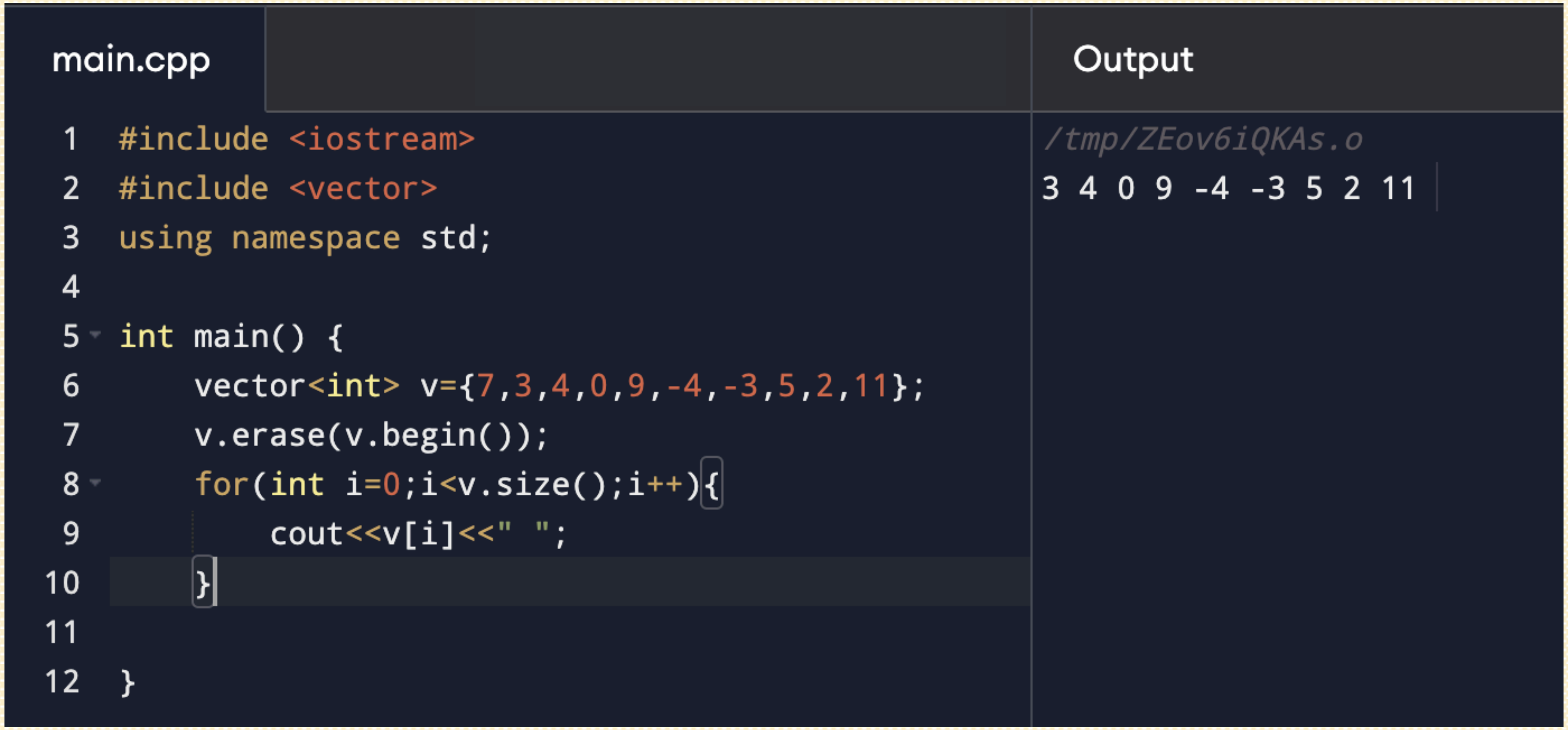
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
    auto d = v.begin();
    cout<<*d<<endl;

}
```

Output

```
/tmp/ZEov6iQKAs.o
7
```

Vectors member functions: erase()

v.erase(): It is used to remove elements from a vector at a specified position or within a range.



main.cpp

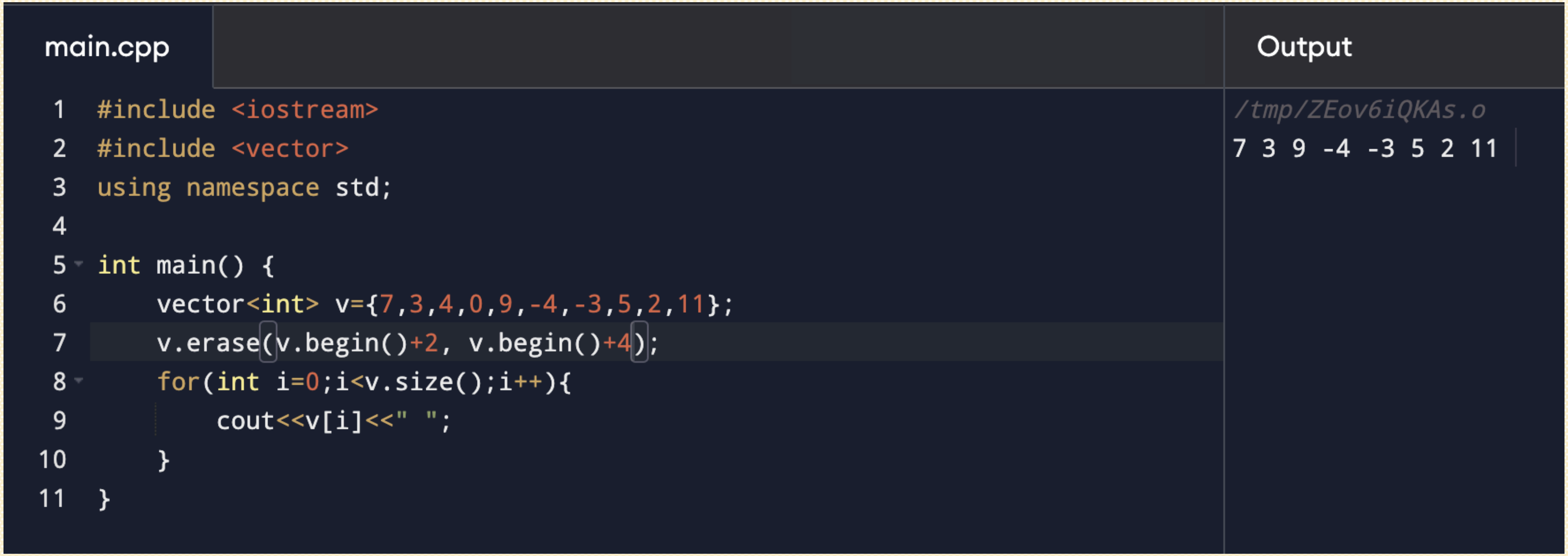
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
    v.erase(v.begin());
    for(int i=0;i<v.size();i++){
        cout<<v[i]<<" ";
    }

}
```

Output

```
/tmp/ZEov6iQKAs.o
3 4 0 9 -4 -3 5 2 11
```

Vectors member functions: erase()

Remove the range between index 2 until 4



main.cpp

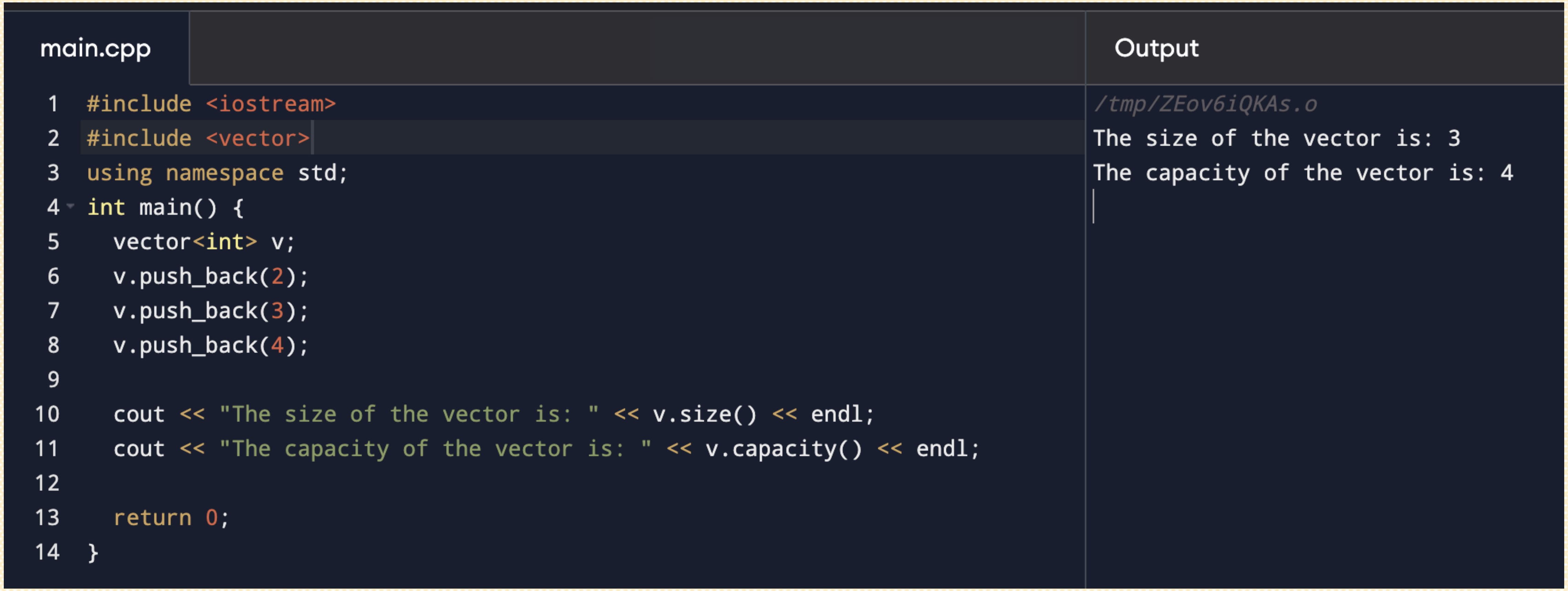
```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
    v.erase(v.begin()+2, v.begin()+4);
    for(int i=0;i<v.size();i++){
        cout<<v[i]<<" ";
    }
}
```

Output

```
/tmp/ZEov6iQKAs.o
7 3 9 -4 -3 5 2 11
```

Vectors member functions: capacity()

When you add an element to a vector, the size of the vector increases by 1. However, the capacity of the vector does not necessarily increase. The capacity of the vector will only increase if the current capacity is not enough to store the new element. In this case, the capacity of the vector does not increase because there is still enough space to store the new element.



main.cpp

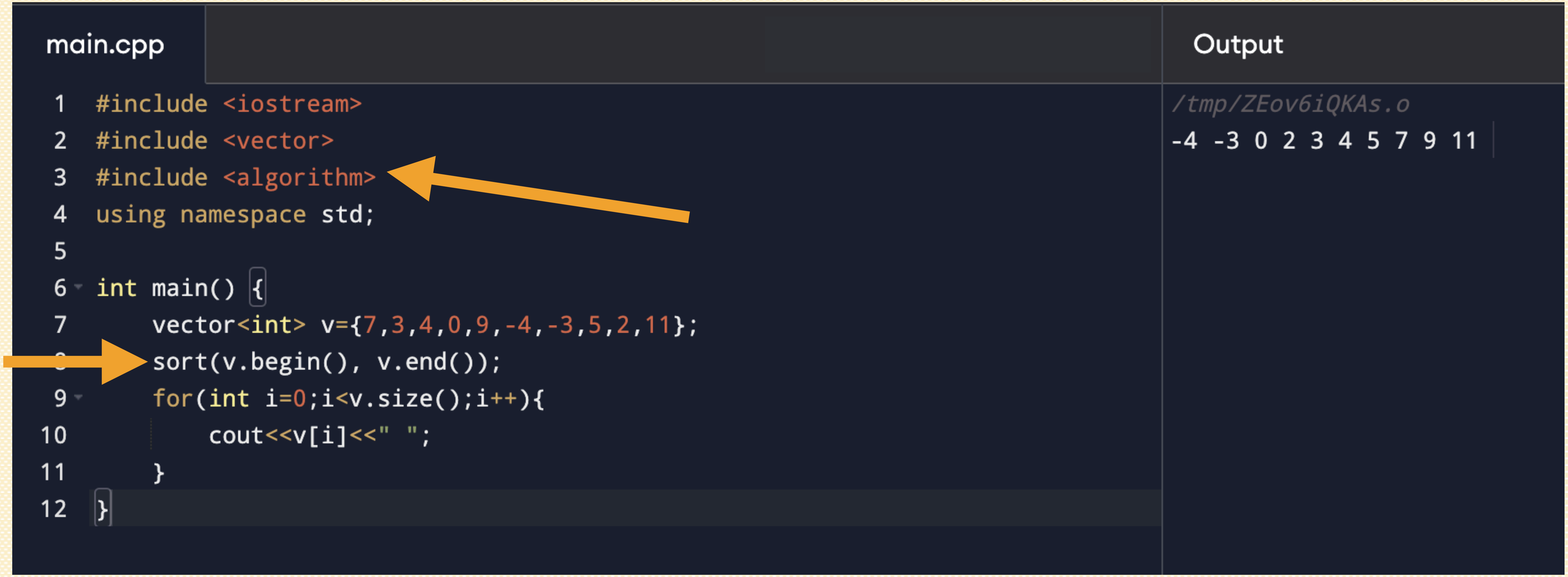
```cpp
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> v;
  v.push_back(2);
  v.push_back(3);
  v.push_back(4);

  cout << "The size of the vector is: " << v.size() << endl;
  cout << "The capacity of the vector is: " << v.capacity() << endl;

  return 0;
}
```

Output

```
/tmp/ZEov6iQKAs.o
The size of the vector is: 3
The capacity of the vector is: 4
```

STL Algorithms with Vectors:

The STL (Standard Template Library) provides a set of algorithms that work seamlessly with vectors such as sort(), find(), and accumulate().



main.cpp

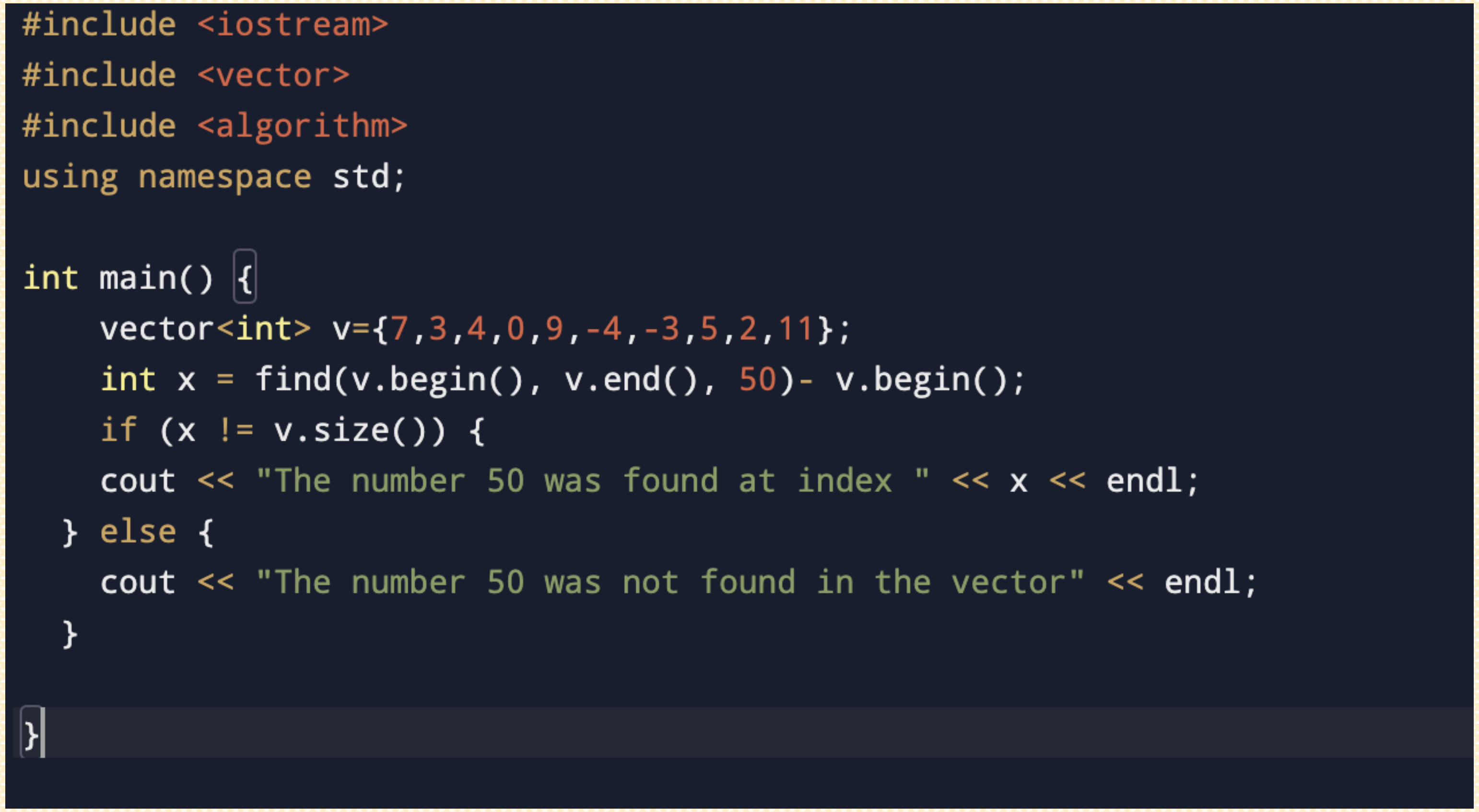
```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
    sort(v.begin(), v.end());
    for(int i=0;i<v.size();i++){
        cout<<v[i]<<" ";
    }
}
```

Output

```
/tmp/ZEov6iQKAs.o
-4 -3 0 2 3 4 5 7 9 11
```

STL Algorithms with Vectors:

The STL provides a set of algorithms that work seamlessly with vectors such as sort(), find(), and accumulate().

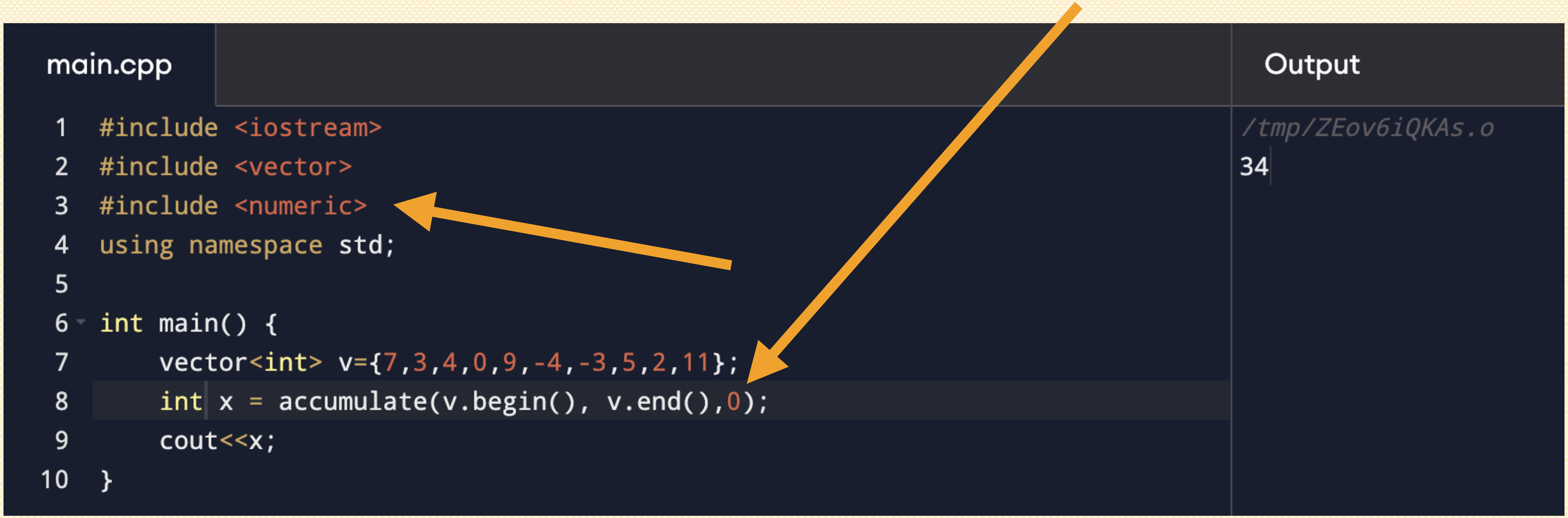


```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
    int x = find(v.begin(), v.end(), 50)- v.begin();
    if (x != v.size()) {
    cout << "The number 50 was found at index " << x << endl;
  } else {
    cout << "The number 50 was not found in the vector" << endl;
  }

}
```

STL Algorithms with Vectors:

accumulate(): It is used to perform a summation or accumulation operation on a range of elements

sum variable starting from the initial value of 0.


main.cpp

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

int main() {
    vector<int> v={7,3,4,0,9,-4,-3,5,2,11};
    int x = accumulate(v.begin(), v.end(),0);
    cout<<x;
}
```

Output

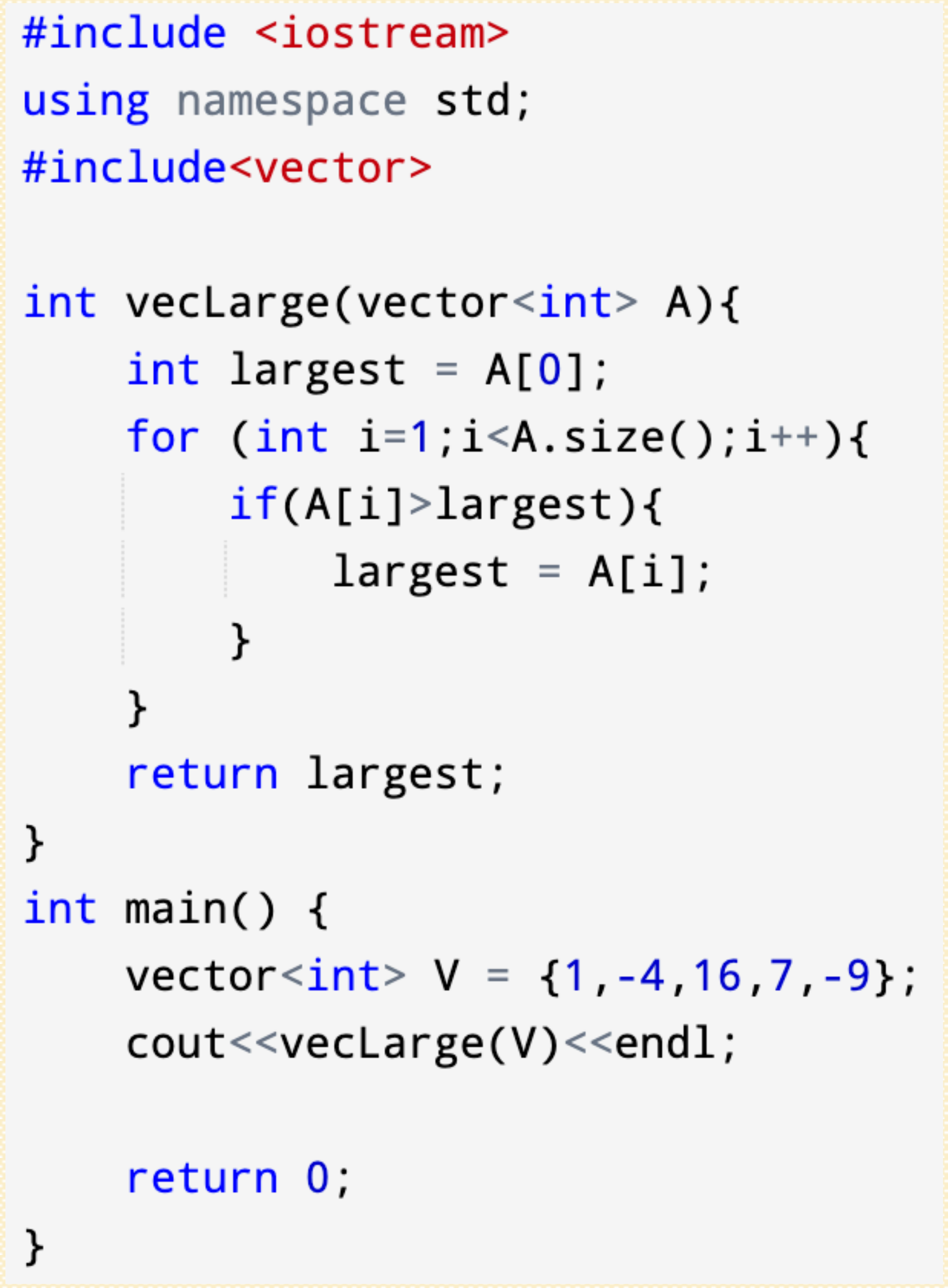
```
/tmp/ZEov6iQKAs.o
34
```

Vector and functions

- Passing vectors to functions enhances code organization, readability, and reusability.
- It helps for:
  - **Modularity**: It breaks down the program into smaller, manageable tasks. Each function can focus on a specific task related to the vector.
  - **Reusability**: Once the function has been written to work with a vector, it can be reused in different parts of the program or even in different programs.
  - **Readability**: By using vector-related operations in functions, the code can be easier to understand and maintain.

Vector and functions

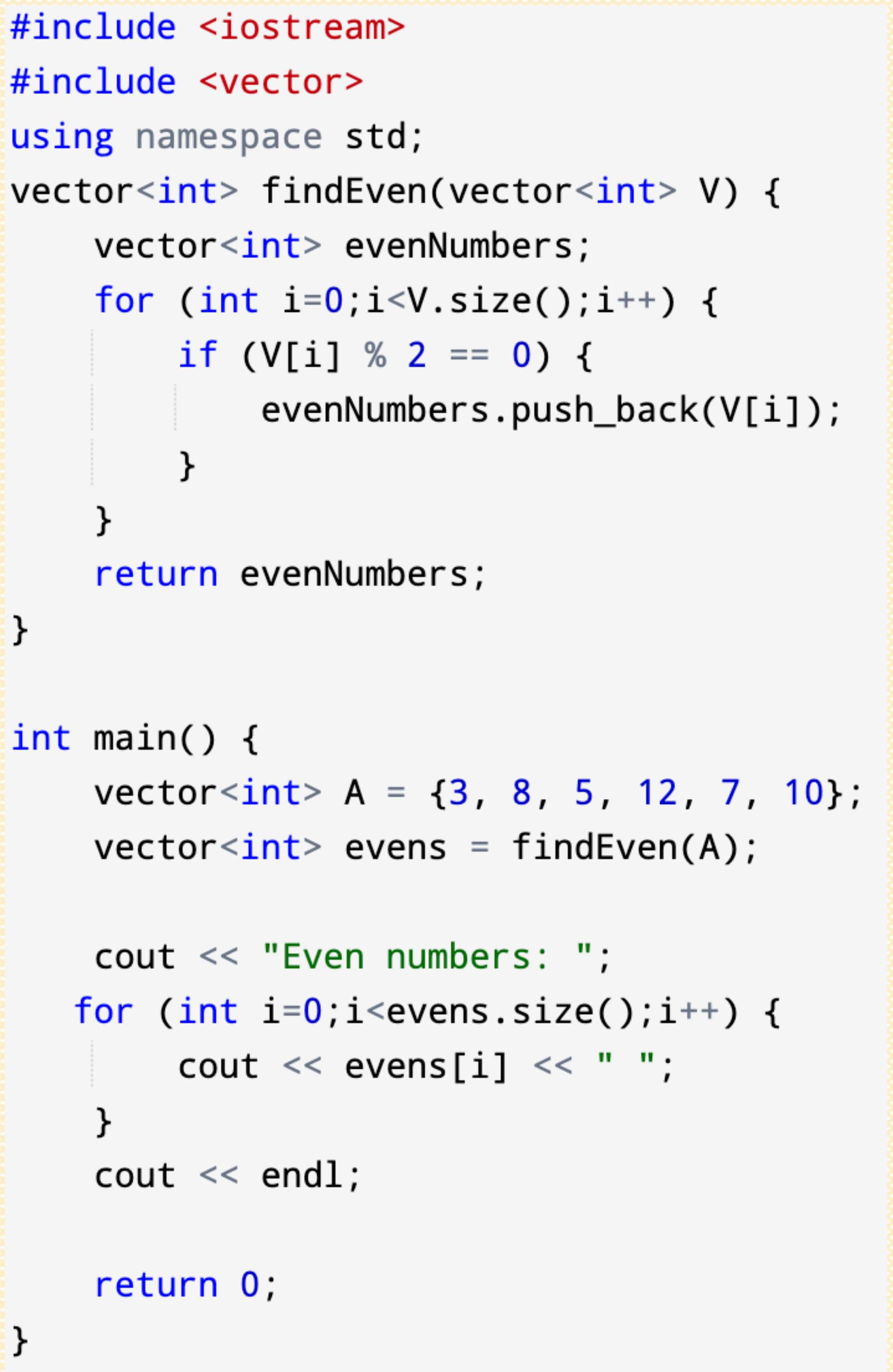
- By using function, write a C++ program that finds the largest number inside a vector.



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

int vecLarge(vector<int> A){
    int largest = A[0];
    for (int i=1;i<A.size();i++){
        if(A[i]>largest){
            largest = A[i];
        }
    }
    return largest;
}
int main() {
    vector<int> V = {1,-4,16,7,-9};
    cout<<vecLarge(V)<<endl;

    return 0;
}
```

Vector and functions

- By using function, write a C++ program that finds the even numbers inside a vector and return the even numbers inside a new vector, then print it.



```
#include <iostream>
#include <vector>
using namespace std;
vector<int> findEven(vector<int> V) {
    vector<int> evenNumbers;
    for (int i=0;i<V.size();i++) {
        if (V[i] % 2 == 0) {
            evenNumbers.push_back(V[i]);
        }
    }
    return evenNumbers;
}

int main() {
    vector<int> A = {3, 8, 5, 12, 7, 10};
    vector<int> evens = findEven(A);

    cout << "Even numbers: ";
   for (int i=0;i<evens.size();i++) {
        cout << evens[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Vector and functions

- By using function, write a C++ program that finds the average of the vector elements.

Sequence Containers: vector

- The implementation of a vector is based on arrays
- Vectors allow direct access to any element via indexes
- Insertion at the end is normally efficient.
  - *The vector simply grows*
- Insertion and deletion in the middle is expensive
  - *An entire portion of the vector needs to be moved*

Sequence Containers

- STL provides three sequence containers
  - ***vector**: based on arrays*
  - ***deque** (double-ended queue): based on arrays*
  - ***list**: based on linked lists*

Deque

- Double Ended Queue
- Functionality similar to vectors,
- but with efficient insertion and deletion of elements also at the beginning of the sequence, and not only at its end
- Sequence
  - *Elements in sequence containers are ordered in a strict linear sequence. Individual elements are accessed by their position in this sequence.*

Sequence Containers: deque

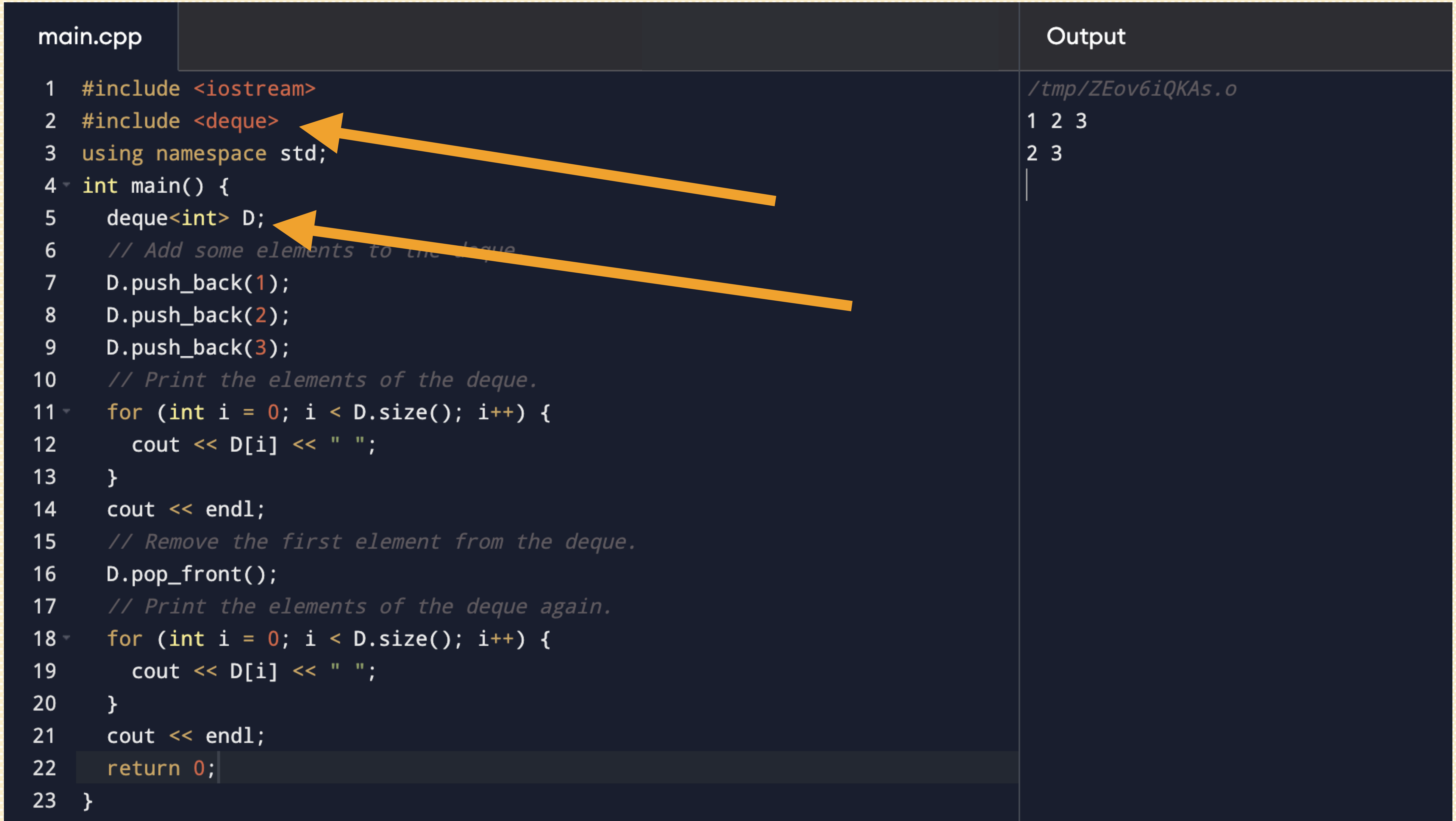
- deque stands for double-ended queue
- deque combines the benefits of vector and list
- It provides indexed access using indexes (which is not possible using lists)
- It also provides efficient insertion and deletion in the front (which is not efficient using vectors) and the end

deque (cont.)

- Additional storage for a deque is allocated using blocks of memory
  - *that are maintained as an array of pointers to those blocks*

- Same basic functions as vector, in addition to that
  - *deque supports push_front and pop_front for insertion and deletion at beginning of deque*

Deque Operations



main.cpp

```cpp
#include <iostream>
#include <deque>
using namespace std;
int main() {
  deque<int> D;
  // Add some elements to the deque
  D.push_back(1);
  D.push_back(2);
  D.push_back(3);
  // Print the elements of the deque.
  for (int i = 0; i < D.size(); i++) {
    cout << D[i] << " ";
  }
  cout << endl;
  // Remove the first element from the deque.
  D.pop_front();
  // Print the elements of the deque again.
  for (int i = 0; i < D.size(); i++) {
    cout << D[i] << " ";
  }
  cout << endl;
  return 0;
}
```

Output

```
/tmp/ZEov6iQKAs.o
1 2 3
2 3
```