C++

Prepared by: Mohamed Ayman

facebook.com/sw.eng.MohamedAyman

sw.eng.MohamedAyman@gmail.com

wuzzuf.net/me/engMohamedAyman

codeforces.com/profile/Mohamed_Ayman

Array

Outline

1) C++ Array Definition
2) Declaring Arrays
3) Initializing Arrays
4) Accessing Array Elements
5) Multi-dimensional Arrays
6) Dynamic Arrays
7) Static Arrays vs. Dynamic Arrays

C++ Array Definition

- C++ provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

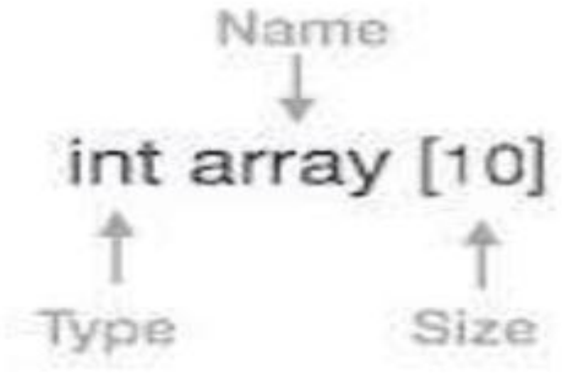
- Instead of declaring individual variables, such as x0, x1, ..., and x99, you declare one array variable such as numbers and use x[0], x[1], and ..., x[99] to represent individual variables. A specific element in an array is accessed by an index.

- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Declaring Arrays

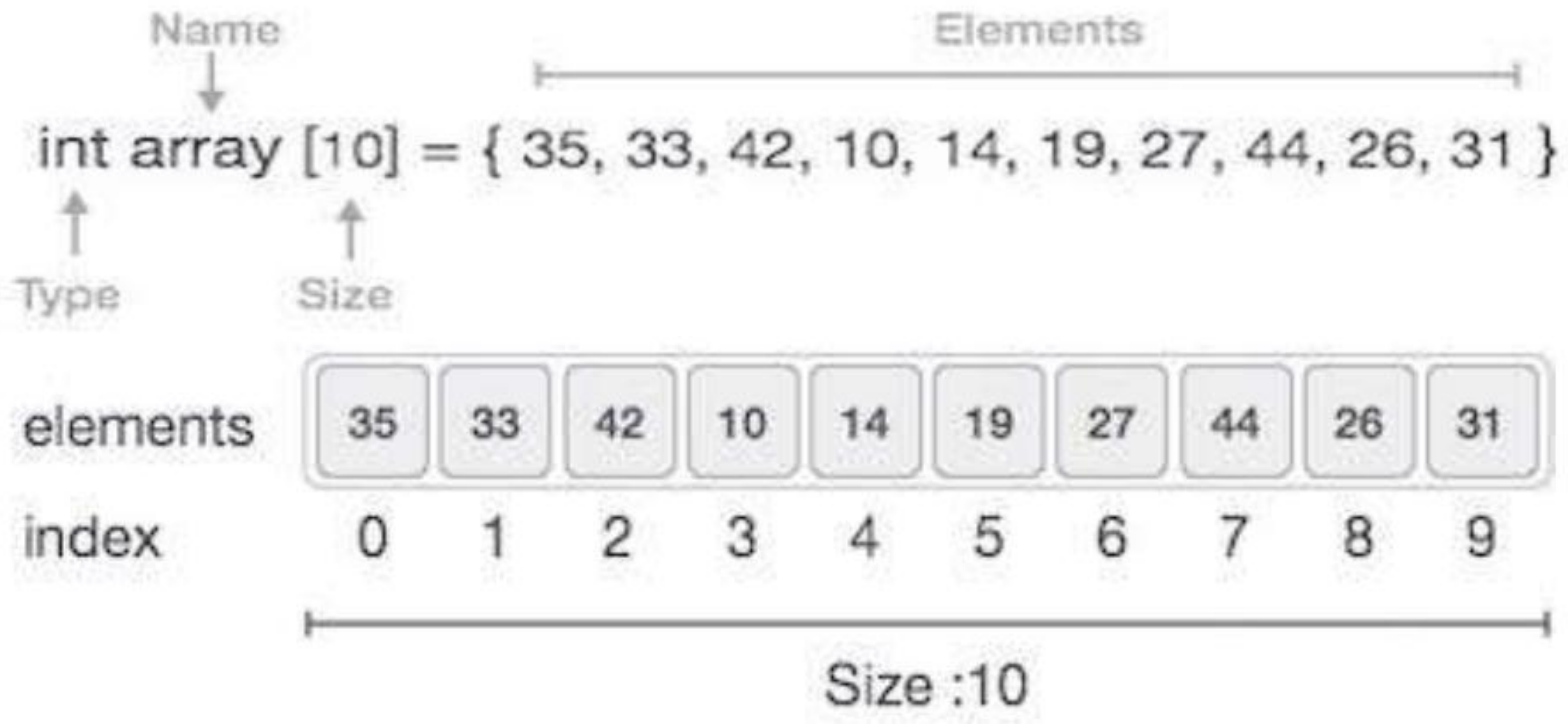
- To declare an array in C++, the programmer specifies the type of the elements and the number of elements required by an array.

- This is called a single-dimension array. The arraySize must be an integer constant greater than zero and type can be any valid C++ data type. For example, to declare a 10-element array called balance of type int, use this statement:



Initializing Arrays

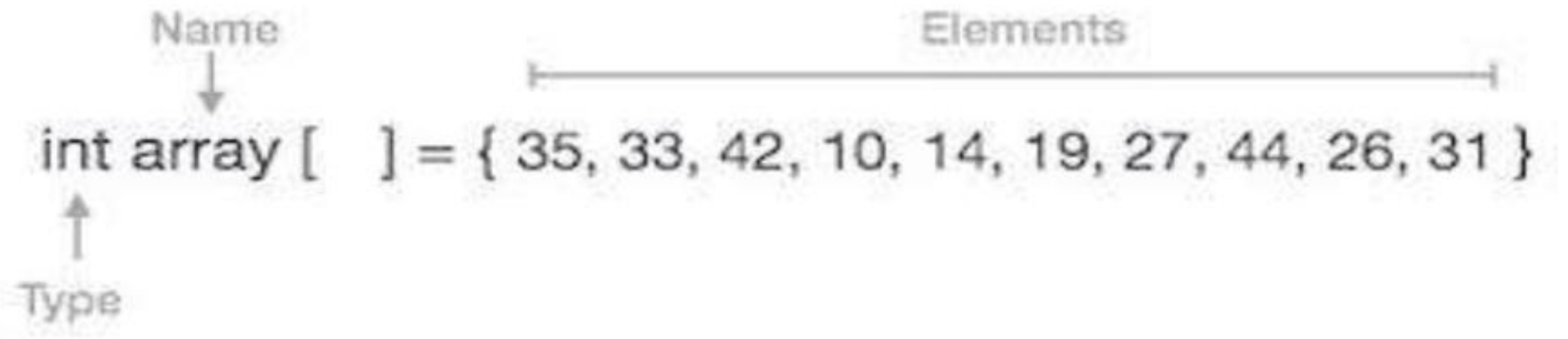
- You can initialize C++ array elements either one by one or using a single statement as follows:



Initializing Arrays

- The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array.

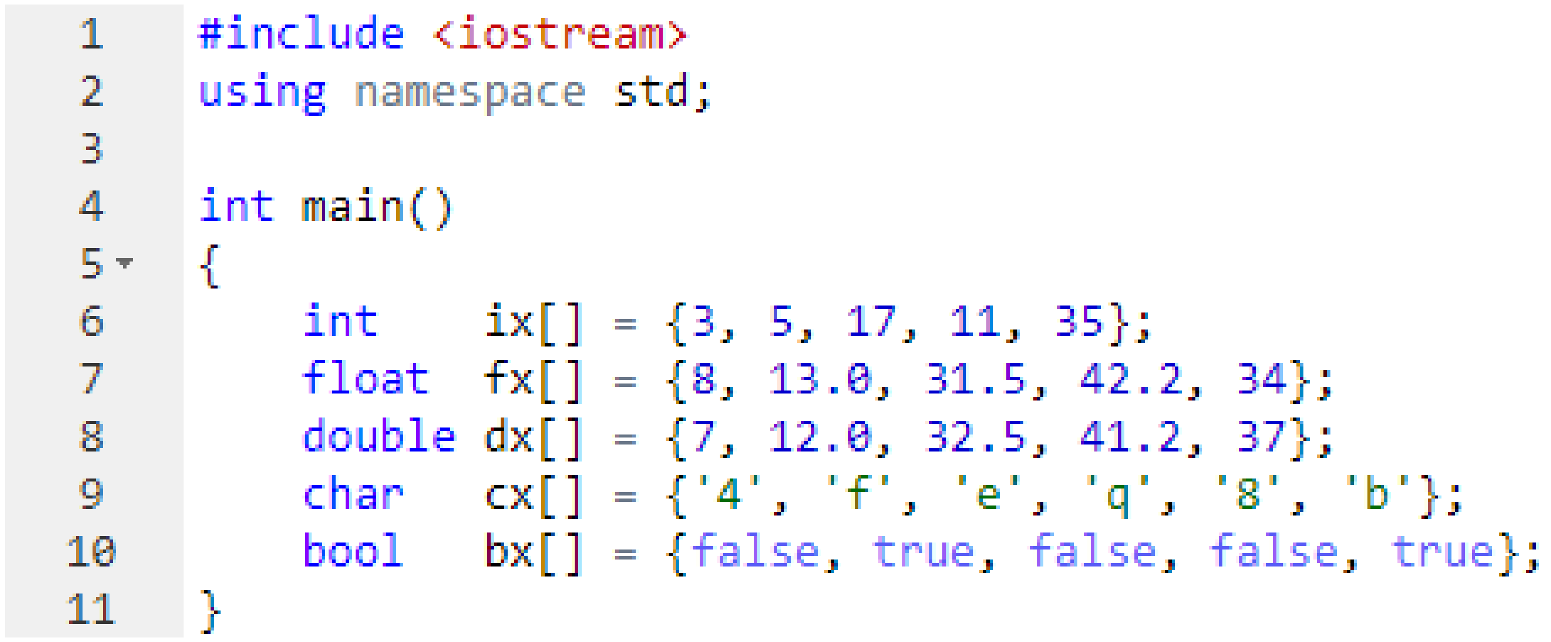
- If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write:



```
int array [   ] = { 35, 33, 42, 10, 14, 19, 27, 44, 26, 31 }
```

Initializing Arrays Example

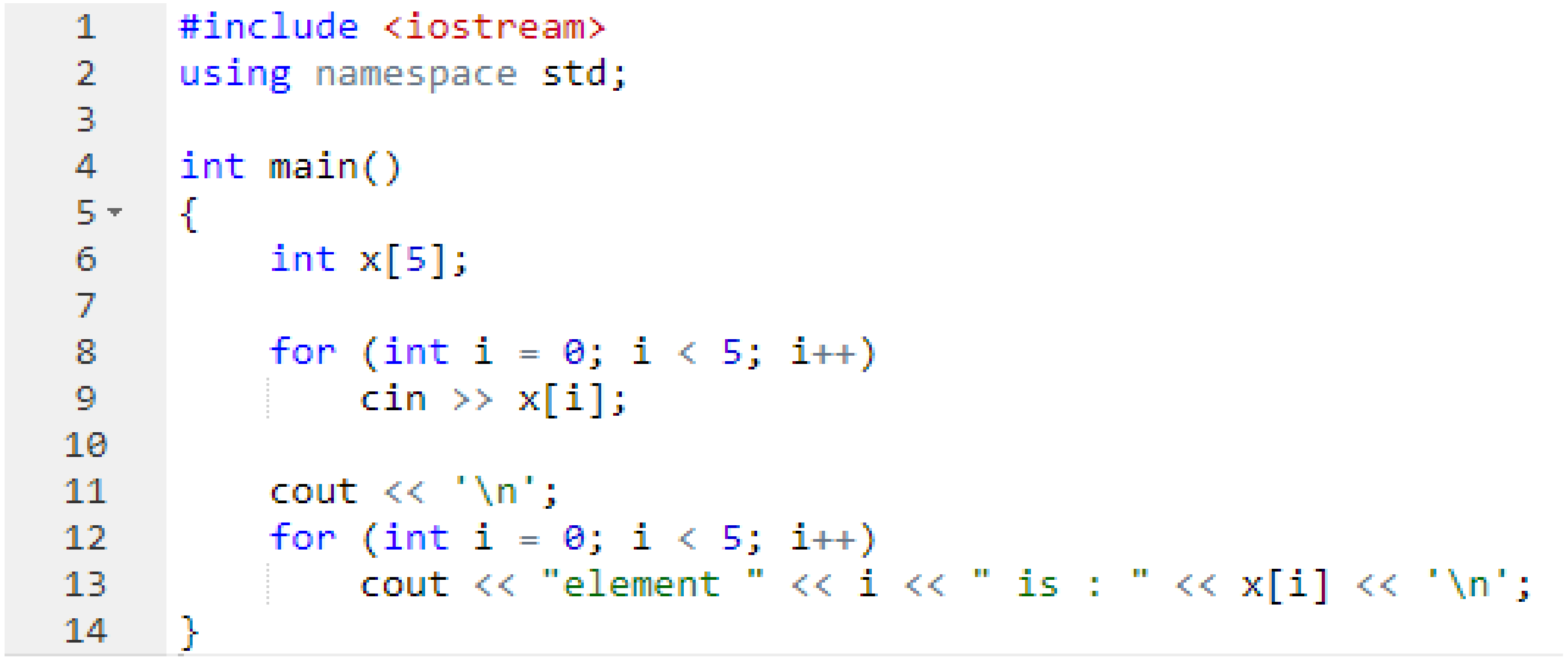
- Source code: https://repl.it/repls/StridentDarkcyanPseudodynerusquadrisectus



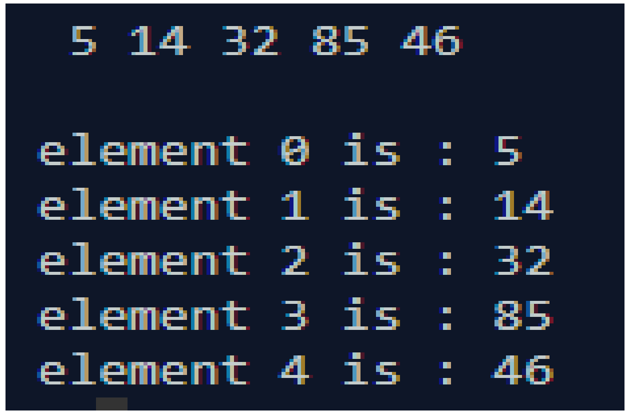
```cpp
#include <iostream>
using namespace std;

int main()
{
    int    ix[] = {3, 5, 17, 11, 35};
    float  fx[] = {8, 13.0, 31.5, 42.2, 34};
    double dx[] = {7, 12.0, 32.5, 41.2, 37};
    char   cx[] = {'4', 'f', 'e', 'q', '8', 'b'};
    bool   bx[] = {false, true, false, false, true};
}
```

Initializing Arrays Example

- Source code: https://repl.it/repls/SteelCreepyAlpaca



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

int main()
{
    int x[5];

    for (int i = 0; i < 5; i++)
        cin >> x[i];

    cout << '\n';
    for (int i = 0; i < 5; i++)
        cout << "element " << i << " is : " << x[i] << '\n';
}
```

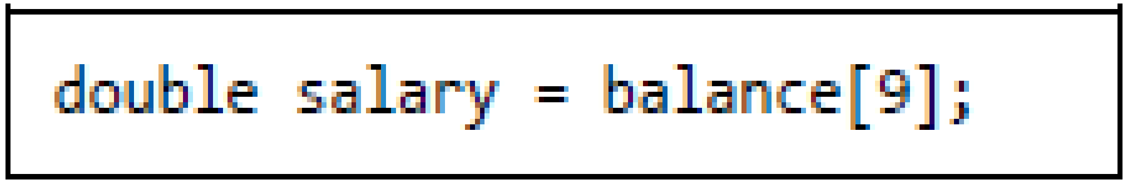


```
 5 14 32 85 46

element 0 is : 5
element 1 is : 14
element 2 is : 32
element 3 is : 85
element 4 is : 46
```

Accessing Array Elements

- An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array.

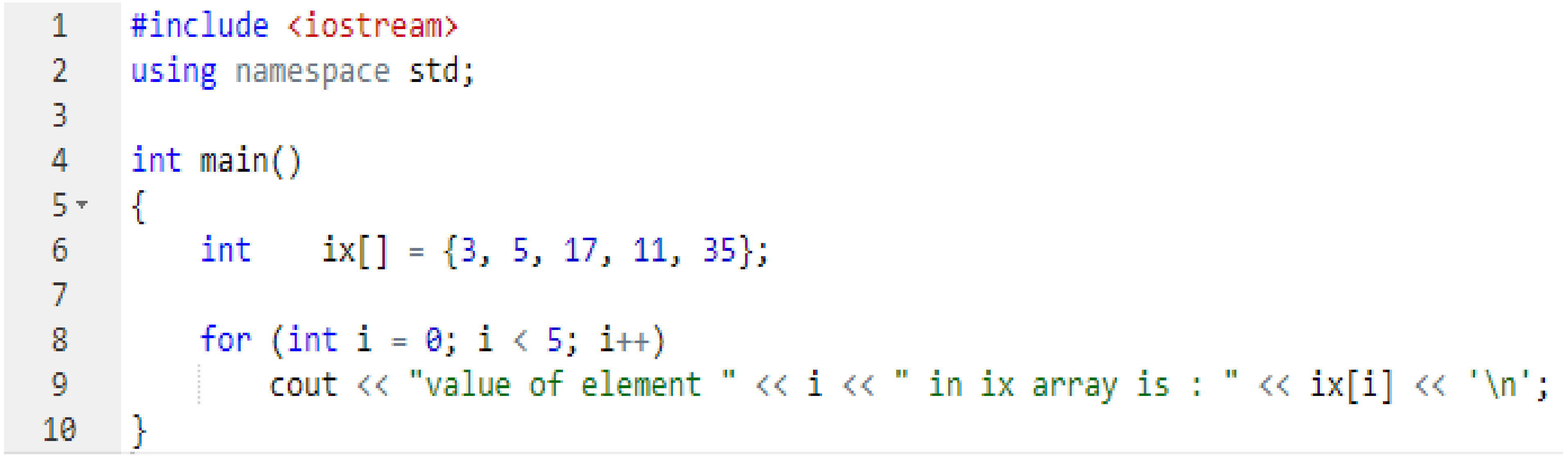
- The above statement will take 10th element from the array and assign the value to salary variable. Following is an example, which will use all the abovementioned three concepts viz. declaration, assignment and accessing arrays:



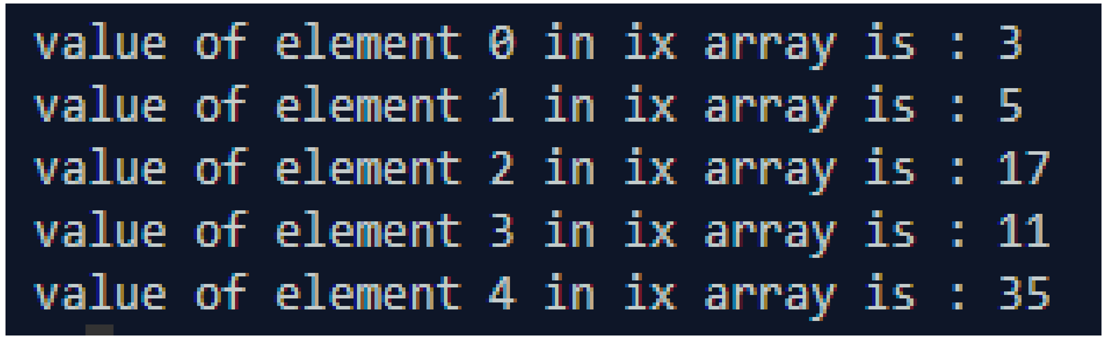
```
double salary = balance[9];
```

Accessing Array Elements Example

- Source code: https://repl.it/repls/PaleOvercookedMole



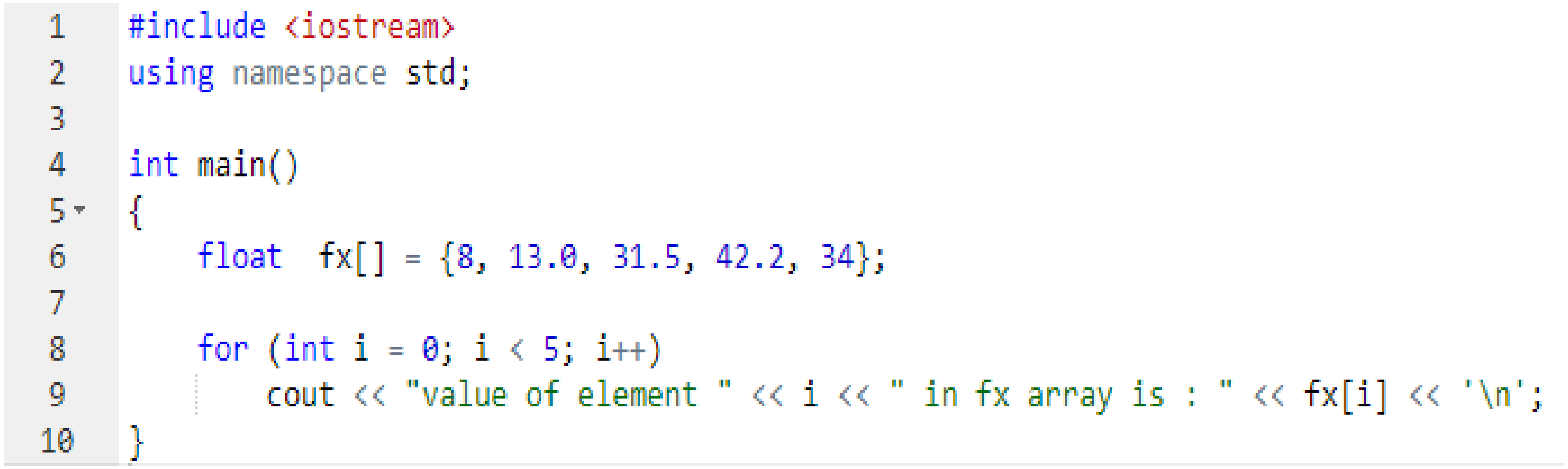
```cpp
#include <iostream>
using namespace std;

int main()
{
    int    ix[] = {3, 5, 17, 11, 35};

    for (int i = 0; i < 5; i++)
        cout << "value of element " << i << " in ix array is : " << ix[i] << '\n';
}
```



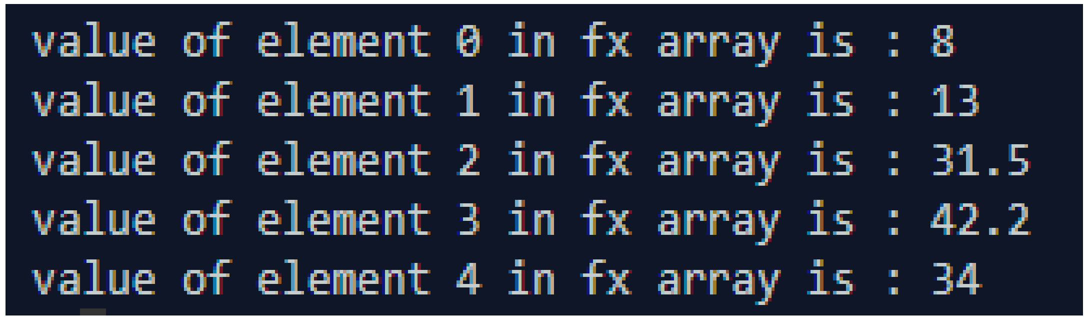
```
value of element 0 in ix array is : 3
value of element 1 in ix array is : 5
value of element 2 in ix array is : 17
value of element 3 in ix array is : 11
value of element 4 in ix array is : 35
```

Accessing Array Elements Example

- Source code: https://repl.it/repls/MediumforestgreenWorthlessCardinal



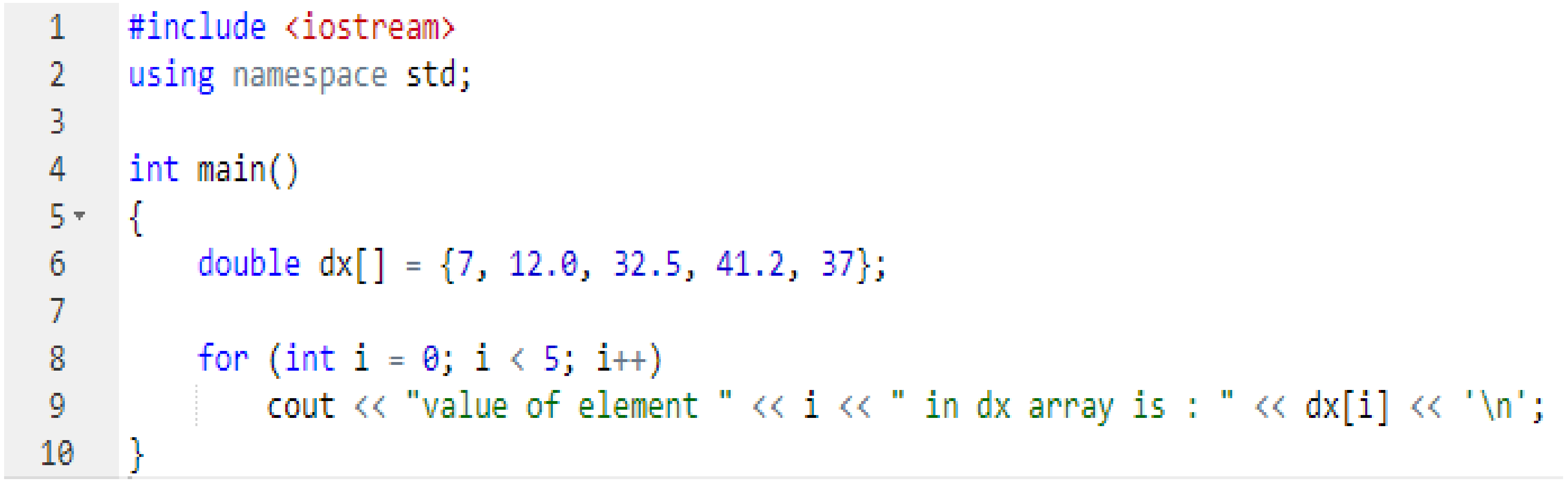
```cpp
#include <iostream>
using namespace std;

int main()
{
    float  fx[] = {8, 13.0, 31.5, 42.2, 34};

    for (int i = 0; i < 5; i++)
        cout << "value of element " << i << " in fx array is : " << fx[i] << '\n';
}
```



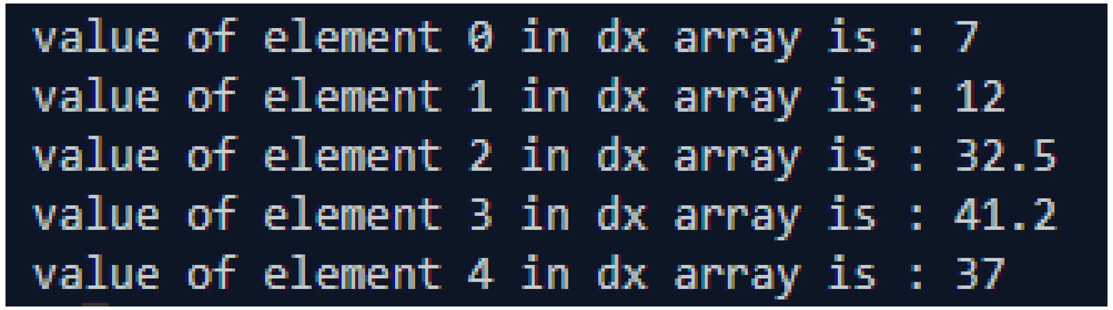
```
value of element 0 in fx array is : 8
value of element 1 in fx array is : 13
value of element 2 in fx array is : 31.5
value of element 3 in fx array is : 42.2
value of element 4 in fx array is : 34
```

Accessing Array Elements Example

- Source code: https://repl.it/repls/AwfulSurprisedFennecfox



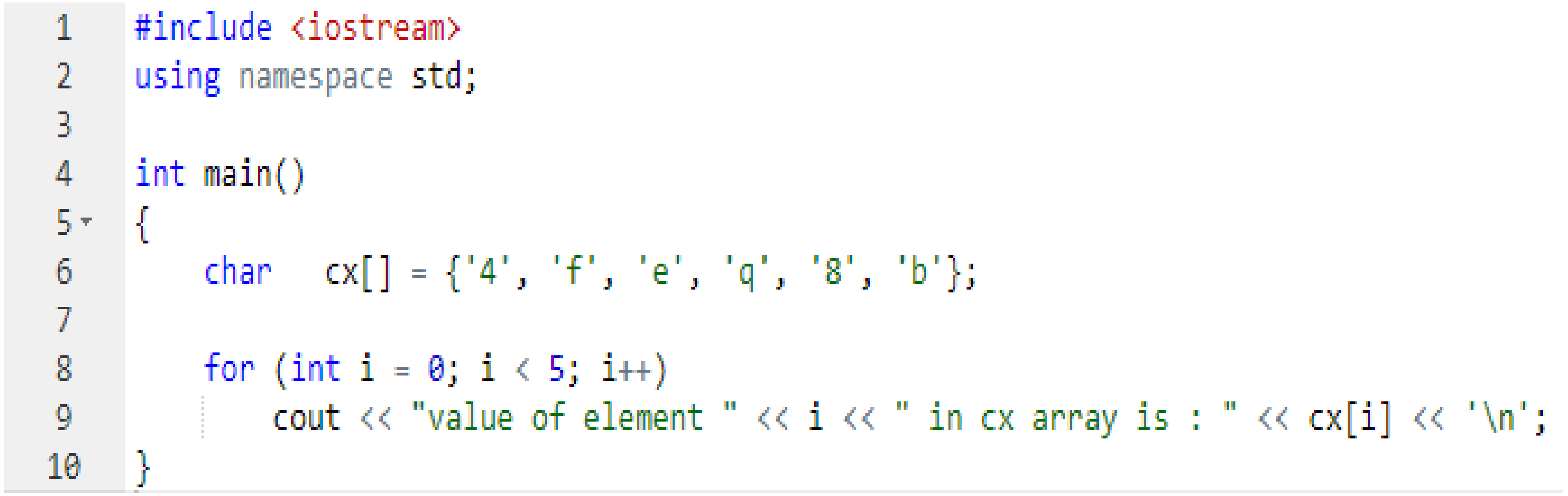
```cpp
#include <iostream>
using namespace std;

int main()
{
    double dx[] = {7, 12.0, 32.5, 41.2, 37};

    for (int i = 0; i < 5; i++)
        cout << "value of element " << i << " in dx array is : " << dx[i] << '\n';
}
```



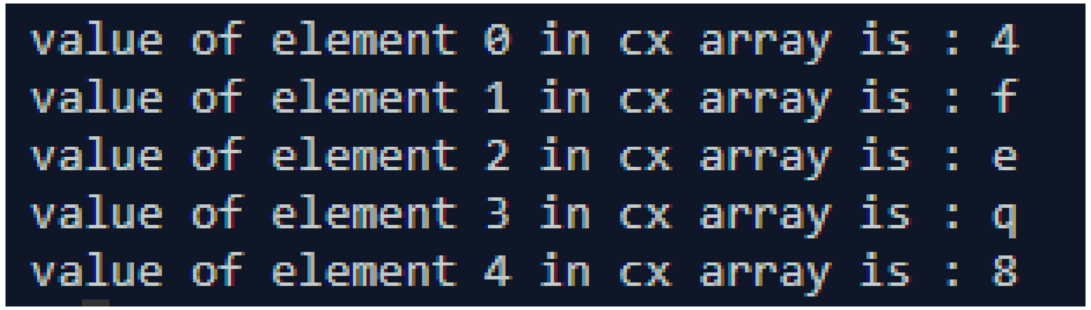
```
value of element 0 in dx array is : 7
value of element 1 in dx array is : 12
value of element 2 in dx array is : 32.5
value of element 3 in dx array is : 41.2
value of element 4 in dx array is : 37
```

Accessing Array Elements Example

- Source code: https://repl.it/repls/AustereFuchsiaQueensnake



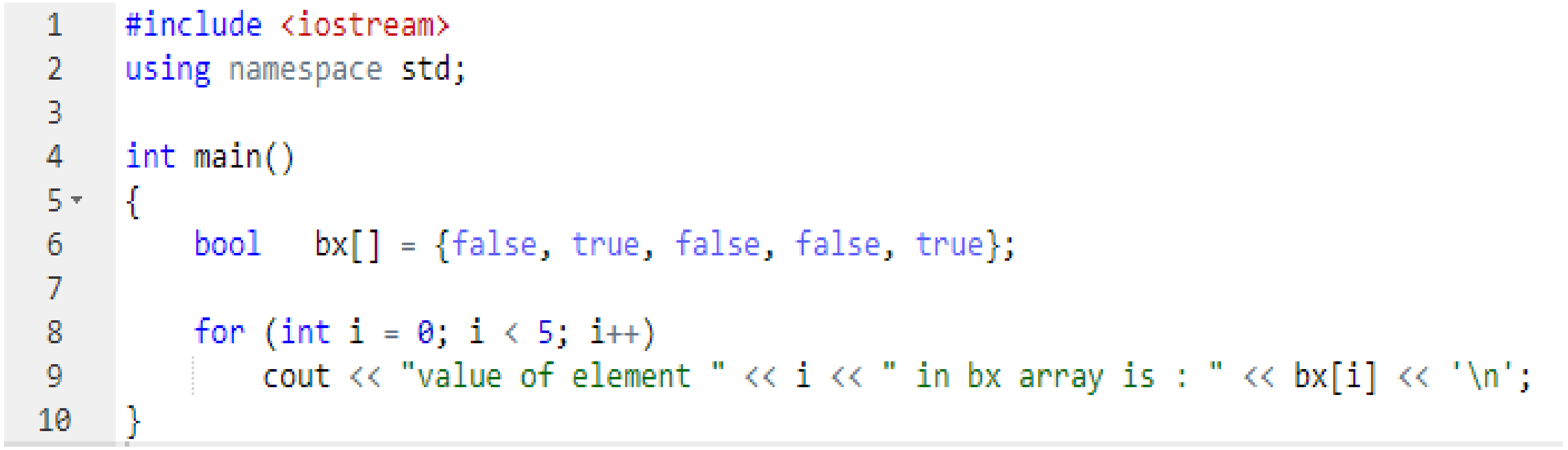
```cpp
#include <iostream>
using namespace std;

int main()
{
    char   cx[] = {'4', 'f', 'e', 'q', '8', 'b'};

    for (int i = 0; i < 5; i++)
        cout << "value of element " << i << " in cx array is : " << cx[i] << '\n';
}
```



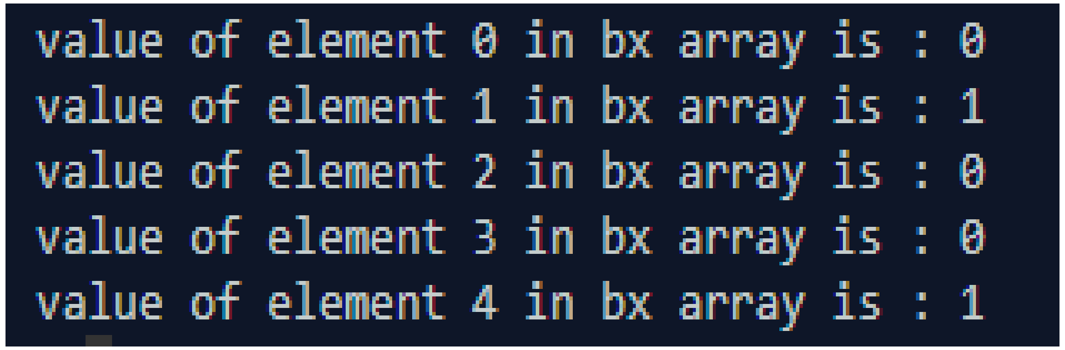
```
value of element 0 in cx array is : 4
value of element 1 in cx array is : f
value of element 2 in cx array is : e
value of element 3 in cx array is : q
value of element 4 in cx array is : 8
```

Accessing Array Elements Example

- Source code: https://repl.it/repls/AggravatingGoldenrodArgentinehornedfrog



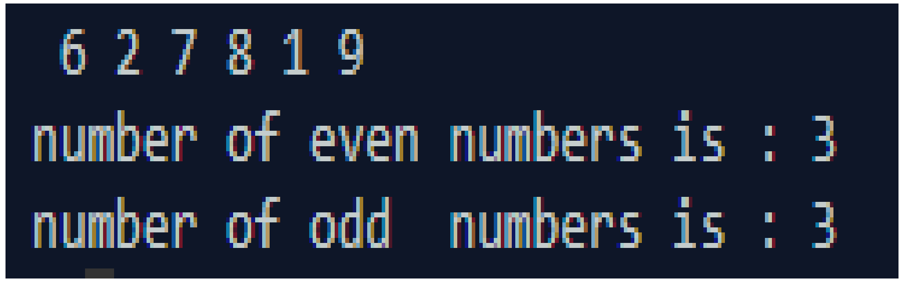
```cpp
#include <iostream>
using namespace std;

int main()
{
    bool   bx[] = {false, true, false, false, true};

    for (int i = 0; i < 5; i++)
        cout << "value of element " << i << " in bx array is : " << bx[i] << '\n';
}
```



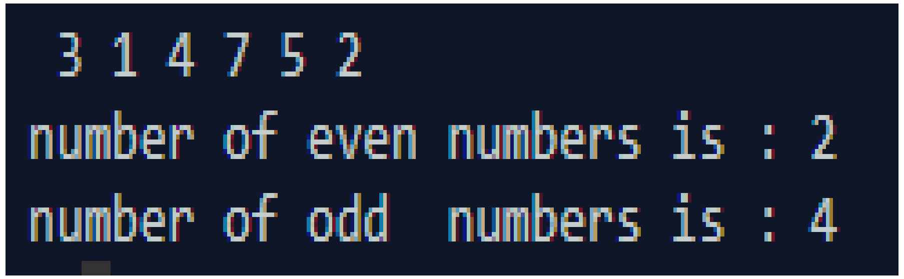
```
value of element 0 in bx array is : 0
value of element 1 in bx array is : 1
value of element 2 in bx array is : 0
value of element 3 in bx array is : 0
value of element 4 in bx array is : 1
```

Practice

- Take as an input 6 numbers as an array then count number of even and odd numbers in this array
- Test Cases:



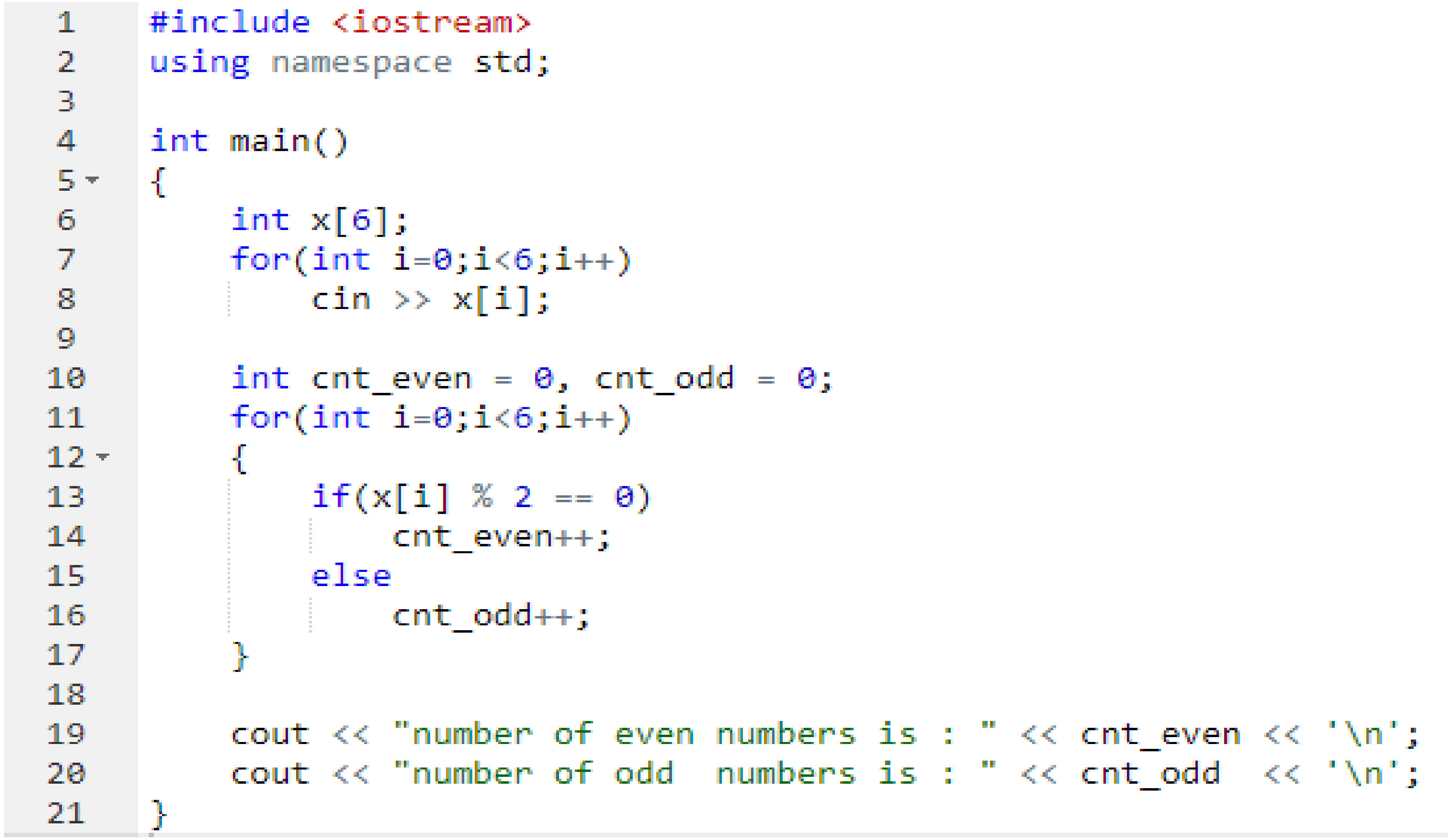
```
 6 2 7 8 1 9
number of even numbers is : 3
number of odd  numbers is : 3
```



```
 3 1 4 7 5 2
number of even numbers is : 2
number of odd  numbers is : 4
```

Practice Solution

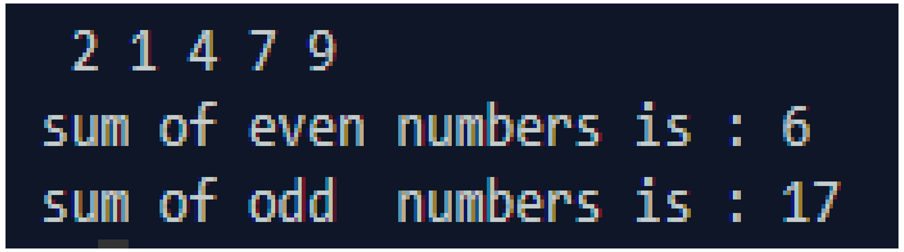
- Source code: https://repl.it/repls/UnequaledSteelSolitaire



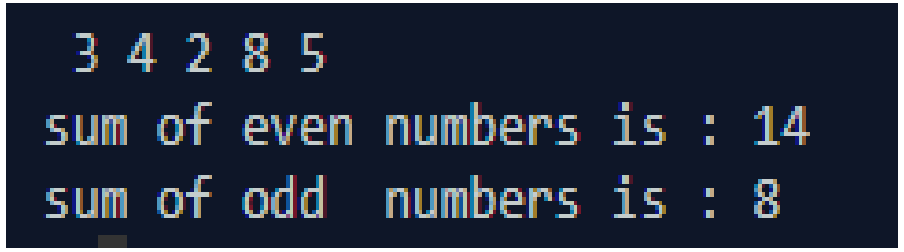
```cpp
#include <iostream>
using namespace std;

int main()
{
    int x[6];
    for(int i=0;i<6;i++)
        cin >> x[i];

    int cnt_even = 0, cnt_odd = 0;
    for(int i=0;i<6;i++)
    {
        if(x[i] % 2 == 0)
            cnt_even++;
        else
            cnt_odd++;
    }

    cout << "number of even numbers is : " << cnt_even << '\n';
    cout << "number of odd  numbers is : " << cnt_odd  << '\n';
}
```

Practice

- Take as an input 5 numbers as an array then calculate sum of even and odd numbers in this array
- Test Cases:



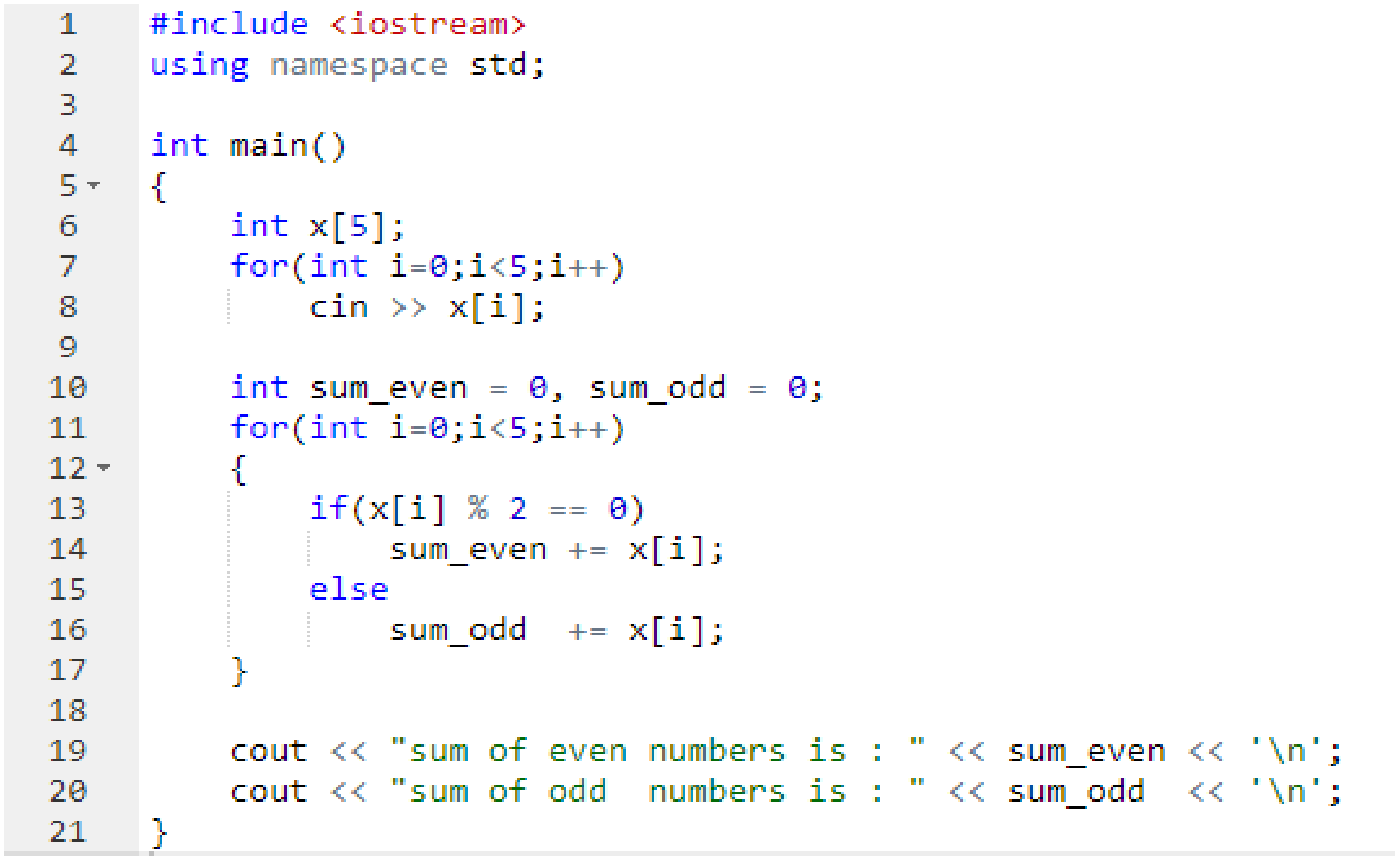
```
 2 1 4 7 9
sum of even numbers is : 6
sum of odd  numbers is : 17
```



```
 3 4 2 8 5
sum of even numbers is : 14
sum of odd  numbers is : 8
```

Practice Solution

- Source code: https://repl.it/repls/HeftyHardCommabutterfly



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

int main()
{
    int x[5];
    for(int i=0;i<5;i++)
        cin >> x[i];

    int sum_even = 0, sum_odd = 0;
    for(int i=0;i<5;i++)
    {
        if(x[i] % 2 == 0)
            sum_even += x[i];
        else
            sum_odd  += x[i];
    }

    cout << "sum of even numbers is : " << sum_even << '\n';
    cout << "sum of odd  numbers is : " << sum_odd  << '\n';
}
```

Multi-dimensional Arrays

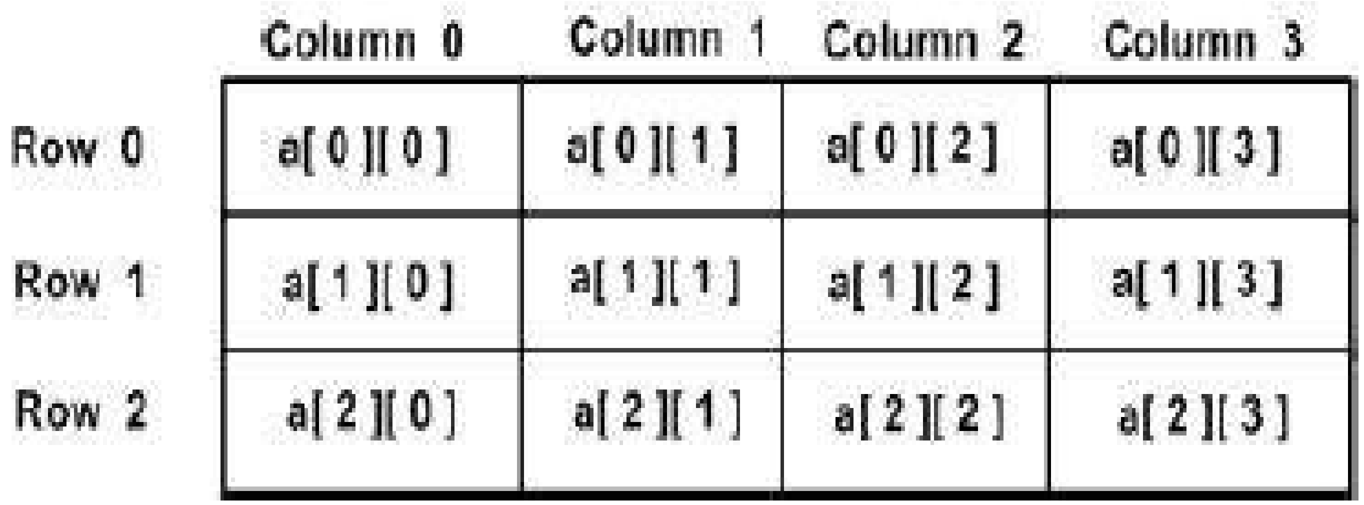
- C++ allows multidimensional arrays. Here is the general form of a multidimensional array declaration:
- Two-Dimensional Arrays
  - The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y you would write something as follows:

```
type arrayName [ x ][ y ];
```

  - Where type can be any valid C++ data type and arrayName will be a valid C++ identifier.

Multi-dimensional Arrays

- Thus, every element in array a is identified by an element name of the form a[i][j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Initializing Two-Dimensional Arrays

- Multi-dimensioned arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row have 4 columns.

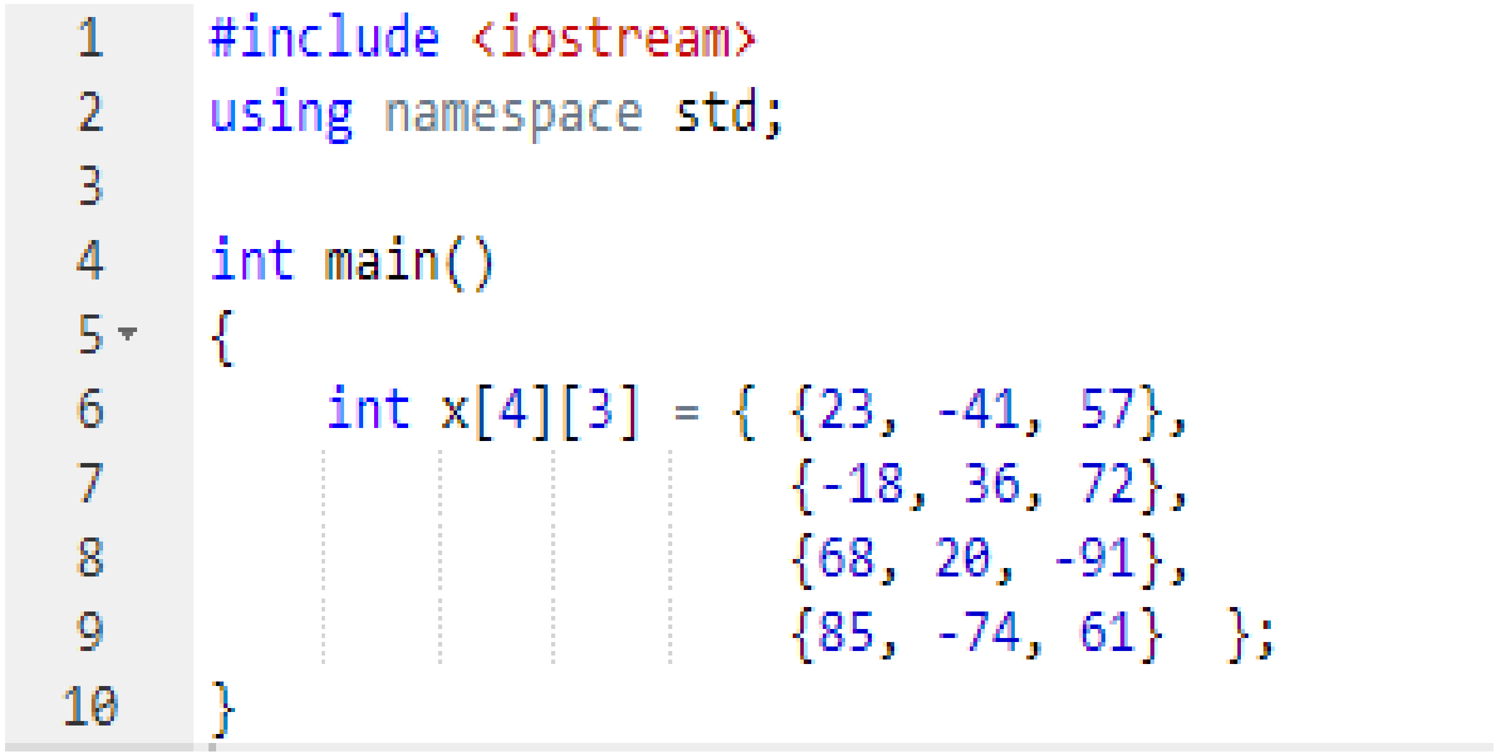
```
int a[3][4] = {
 {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
 {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
 {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

- The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to previous example.

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

Initializing Two-Dimensional Arrays Example

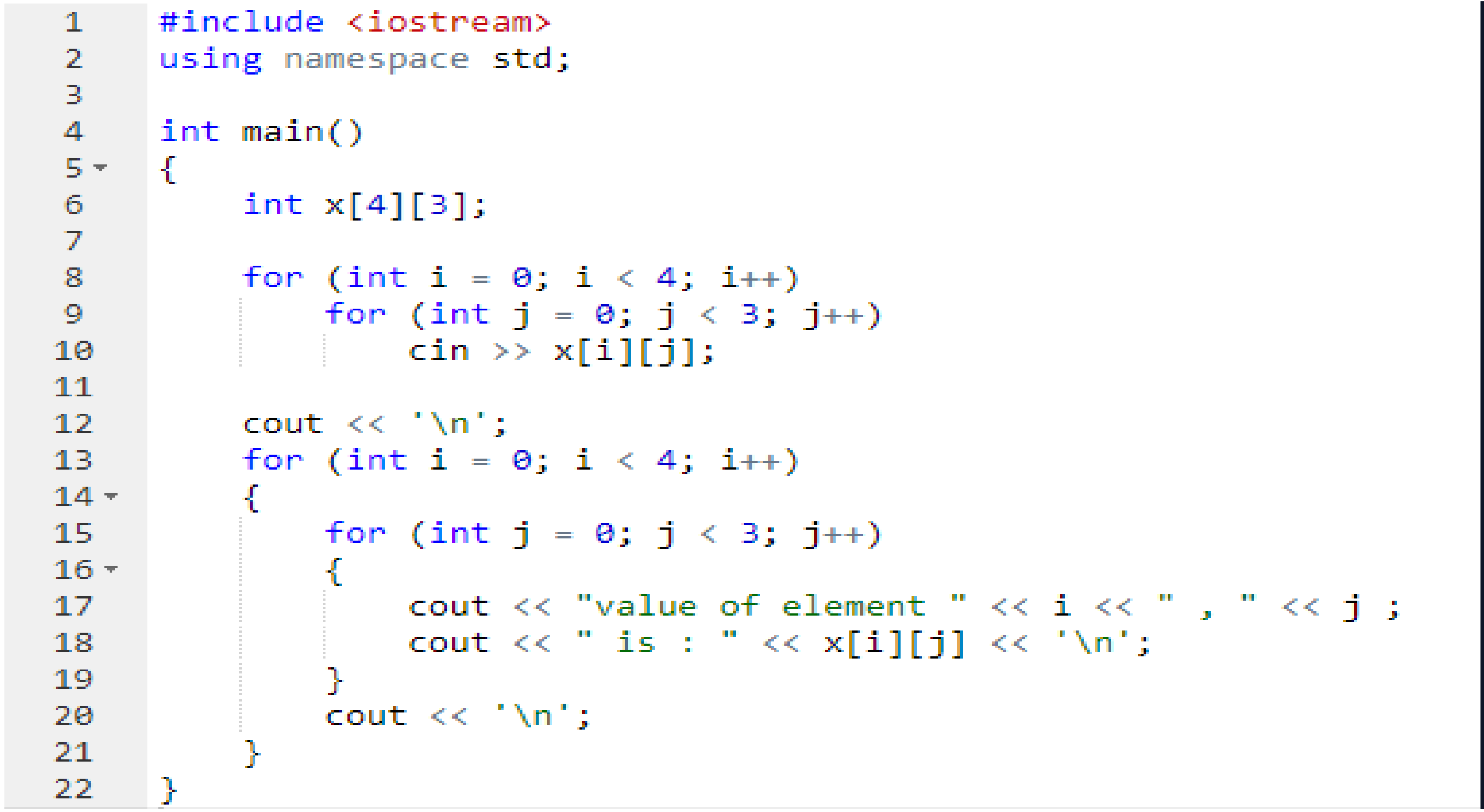
- Source code: https://repl.it/repls/CrowdedScaredZebradove



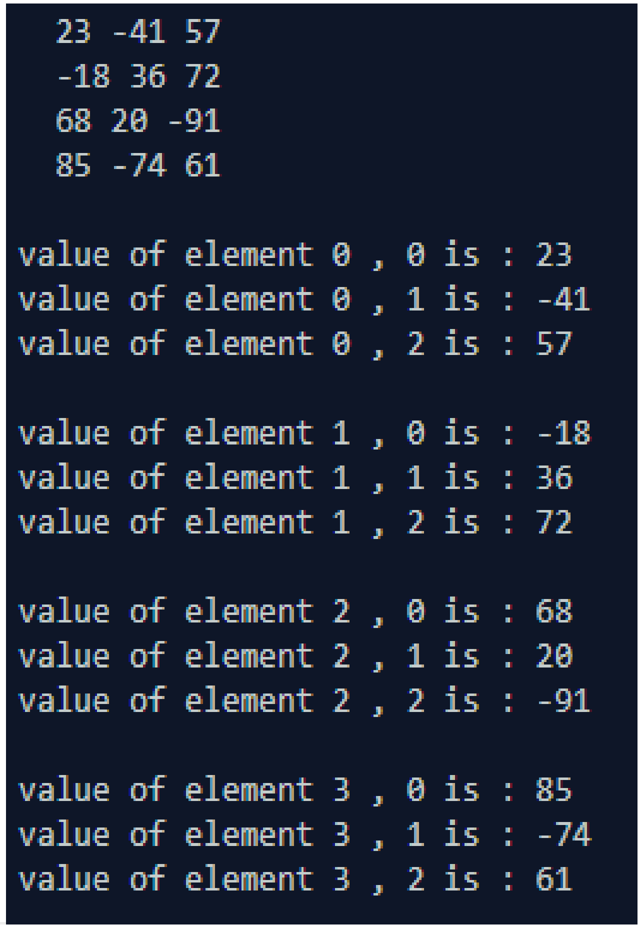
```
#include <iostream>
using namespace std;

int main()
{
    int x[4][3] = { {23, -41, 57},
                    {-18, 36, 72},
                    {68, 20, -91},
                    {85, -74, 61}  };
}
```

Initializing Two-Dimensional Arrays Example

- Source code: https://repl.it/repls/QuerulousSkeletalPikeperch



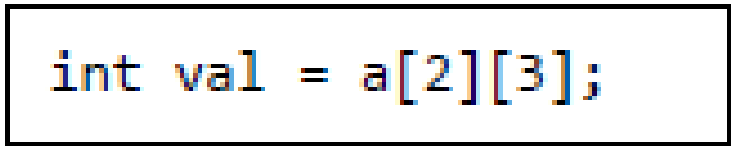
```cpp
#include <iostream>
using namespace std;

int main()
{
    int x[4][3];

    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 3; j++)
            cin >> x[i][j];

    cout << '\n';
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << "value of element " << i << " , " << j ;
            cout << " is : " << x[i][j] << '\n';
        }
        cout << '\n';
    }
}
```



```
  23 -41 57
  -18 36 72
  68 20 -91
  85 -74 61

value of element 0 , 0 is : 23
value of element 0 , 1 is : -41
value of element 0 , 2 is : 57

value of element 1 , 0 is : -18
value of element 1 , 1 is : 36
value of element 1 , 2 is : 72

value of element 2 , 0 is : 68
value of element 2 , 1 is : 20
value of element 2 , 2 is : -91

value of element 3 , 0 is : 85
value of element 3 , 1 is : -74
value of element 3 , 2 is : 61
```

Accessing Two-Dimensional Array Elements

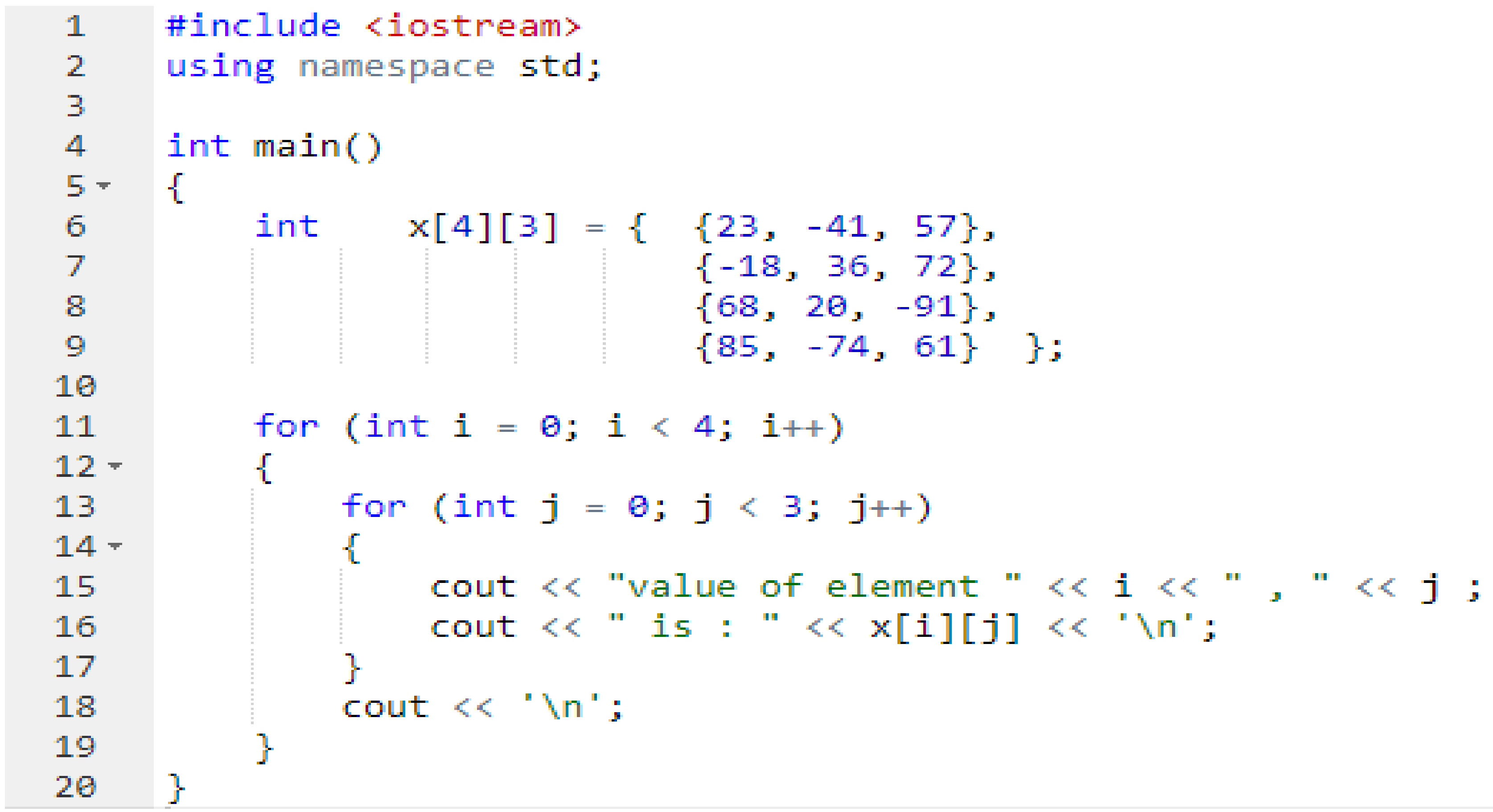
- An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array. For example:



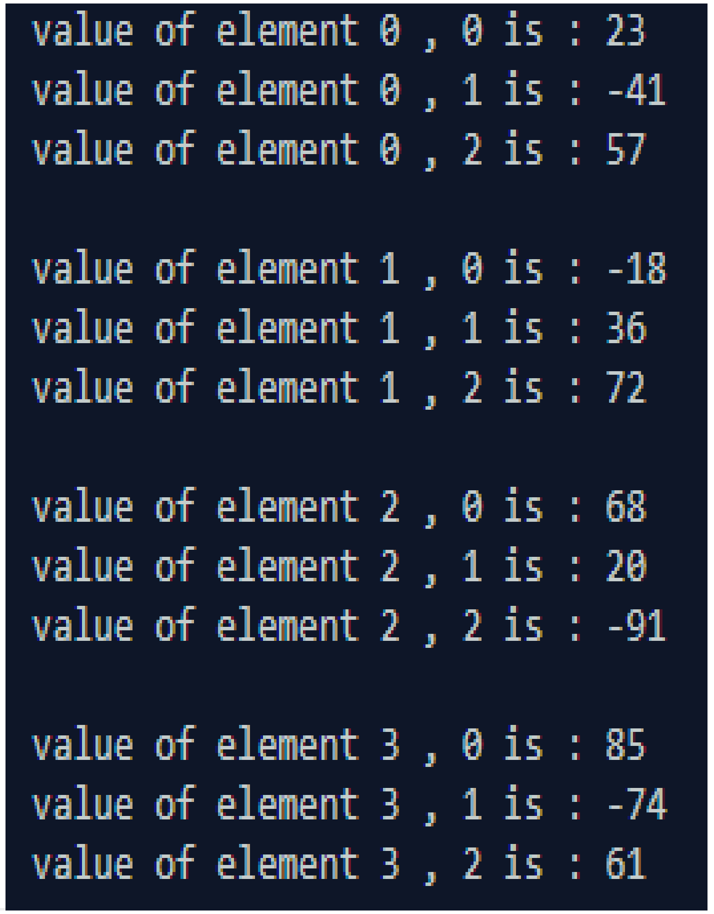
```
int val = a[2][3];
```

Accessing Two-Dimensional Array Elements Example

- Source code: https://repl.it/repls/OffbeatFirebrickMassospondylus



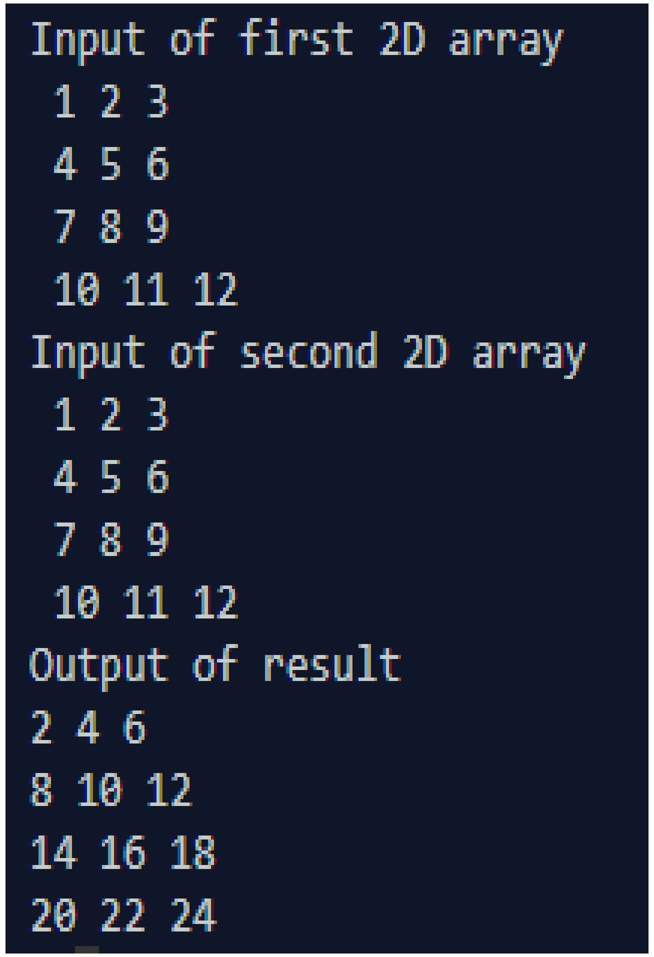
```cpp
#include <iostream>
using namespace std;

int main()
{
    int    x[4][3] = {  {23, -41, 57},
                        {-18, 36, 72},
                        {68, 20, -91},
                        {85, -74, 61}  };

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            cout << "value of element " << i << " , " << j ;
            cout << " is : " << x[i][j] << '\n';
        }
        cout << '\n';
    }
}
```



```
value of element 0 , 0 is : 23
value of element 0 , 1 is : -41
value of element 0 , 2 is : 57

value of element 1 , 0 is : -18
value of element 1 , 1 is : 36
value of element 1 , 2 is : 72

value of element 2 , 0 is : 68
value of element 2 , 1 is : 20
value of element 2 , 2 is : -91

value of element 3 , 0 is : 85
value of element 3 , 1 is : -74
value of element 3 , 2 is : 61
```

Practice

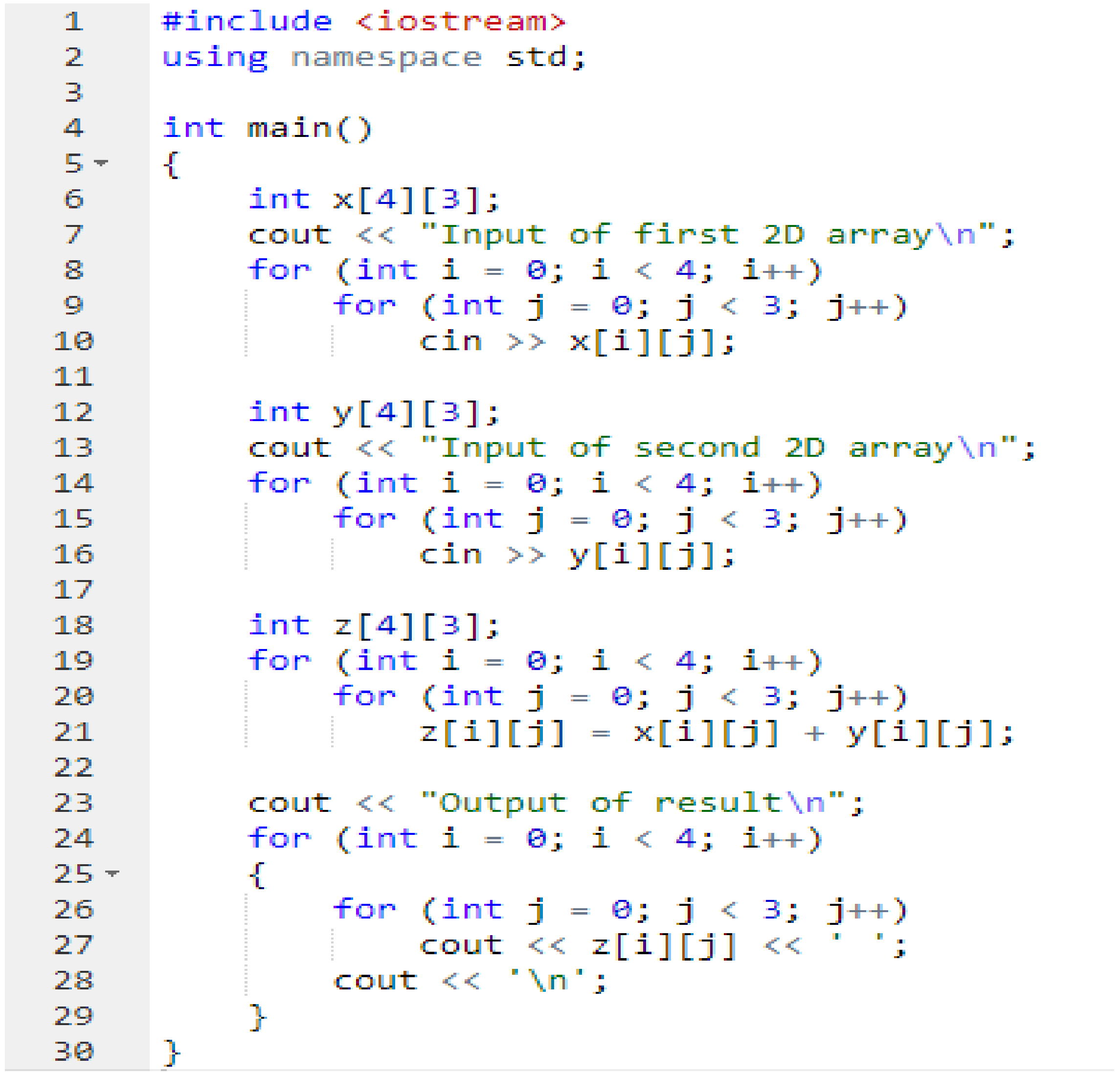
- Take two matrices as an input 2D array in shape 4*3 for each one then calculate the addition of them
- Test Cases:



```
Input of first 2D array
 1 2 3
 4 5 6
 7 8 9
 10 11 12
Input of second 2D array
 1 2 3
 4 5 6
 7 8 9
 10 11 12
Output of result
2 4 6
8 10 12
14 16 18
20 22 24
```

Practice Solution

- Source code: https://repl.it/repls/AntiquewhiteLimpingWhooper



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

int main()
{
    int x[4][3];
    cout << "Input of first 2D array\n";
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 3; j++)
            cin >> x[i][j];

    int y[4][3];
    cout << "Input of second 2D array\n";
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 3; j++)
            cin >> y[i][j];

    int z[4][3];
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 3; j++)
            z[i][j] = x[i][j] + y[i][j];

    cout << "Output of result\n";
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << z[i][j] << ' ';
        cout << '\n';
    }
}
```

Dynamic Memory

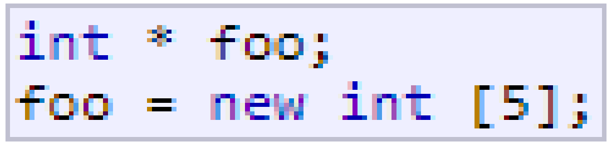
- In the programs seen in previous chapters, all memory needs were determined before program execution by defining the variables needed. But there may be cases where the memory needs of a program can only be determined during runtime.
- For example, when the memory needed depends on user input. On these cases, programs need to dynamically allocate memory, for which the C++ language integrates the operators new and delete.

Dynamic Arrays

- Operators new and new[]
  - Dynamic memory is allocated using operator new. new is followed by a data type specifier and, if a sequence of more than one element is required, the number of these within brackets []. It returns a pointer to the beginning of the new block of memory allocated. Its syntax is:

    pointer = new type

    pointer = new type [number_of_elements]



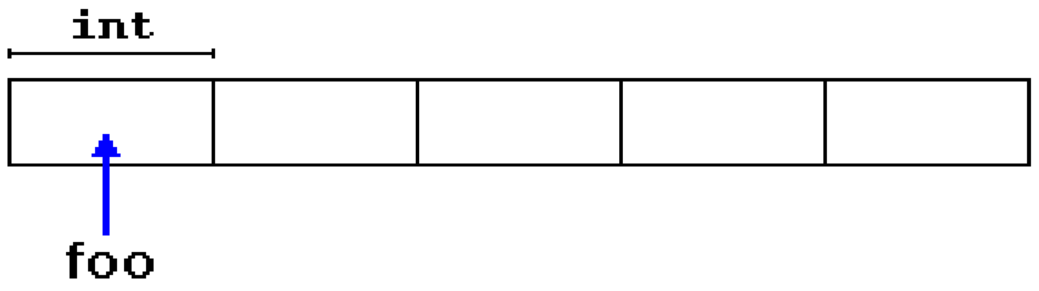
```
int * foo;
foo = new int [5];
```

Dynamic Arrays

- The first expression is used to allocate memory to contain one single element of type type. The second one is used to allocate a block (an array) of elements of type type, where number_of_elements is an integer value representing the amount of these. For example:

```
int * foo;
foo = new int [5];
```

- In this case, the system dynamically allocates space for five elements of type int and returns a pointer to the first element of the sequence, which is assigned to foo (a pointer). Therefore, foo now points to a valid block of memory with space for five elements of type int.



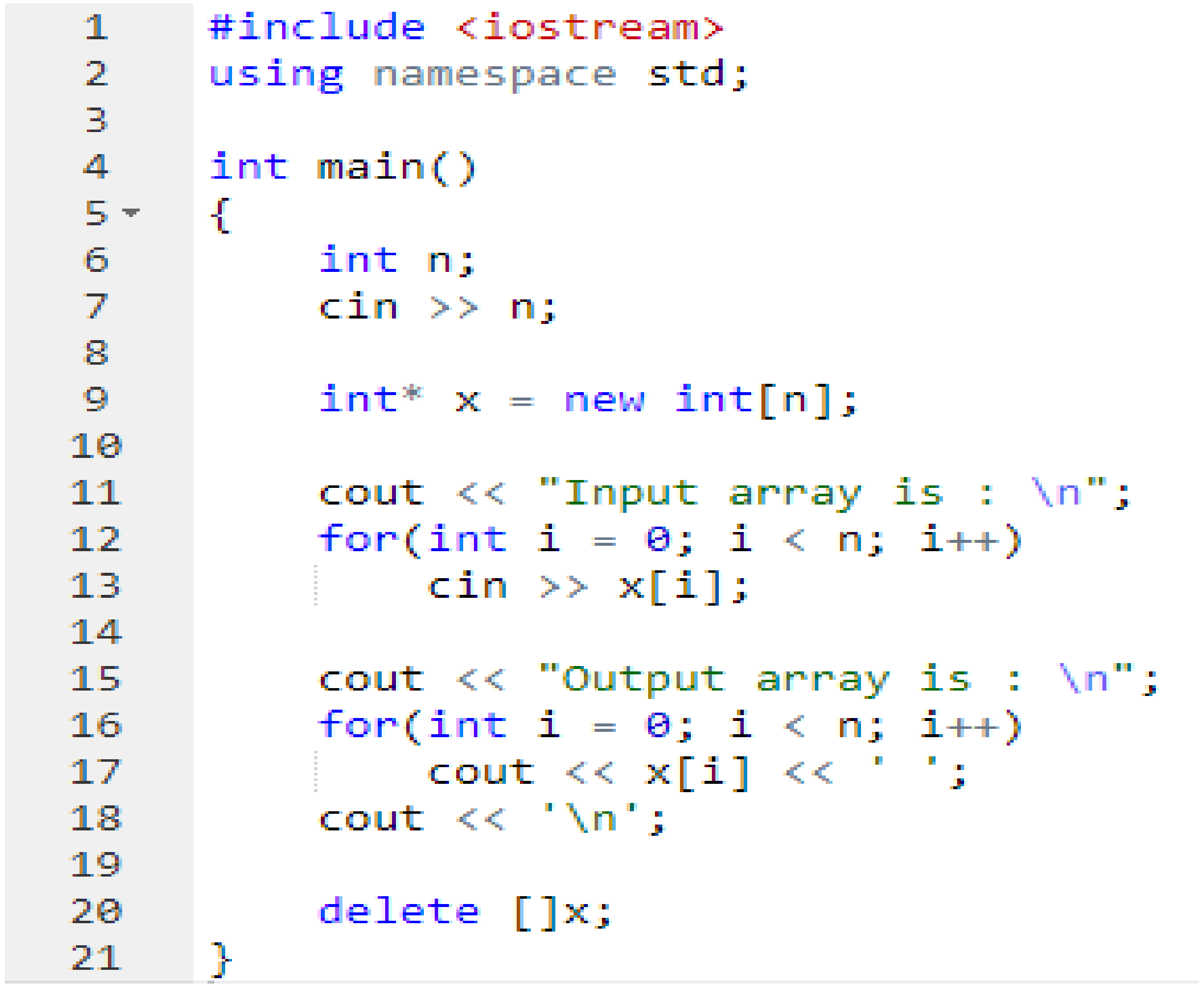
Dynamic Arrays

- Operators delete and delete[]
  - In most cases, memory allocated dynamically is only needed during specific periods of time within a program; once it is no longer needed, it can be freed so that the memory becomes available again for other requests of dynamic memory. This is the purpose of operator delete, whose syntax is:

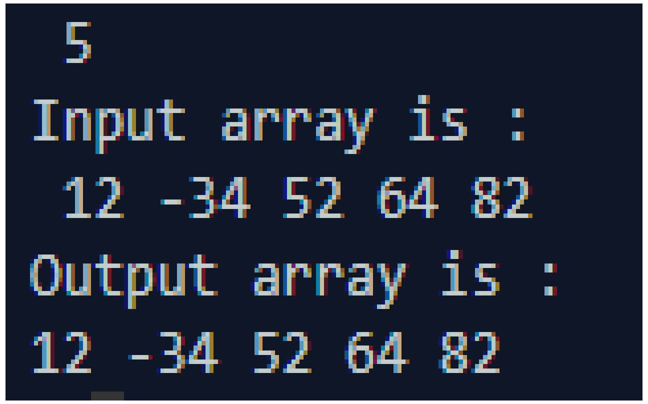
```
delete pointer;
delete[] pointer;
```

Dynamic Arrays Example

- Source code: https://repl.it/repls/HardtofindOpaqueThrasher



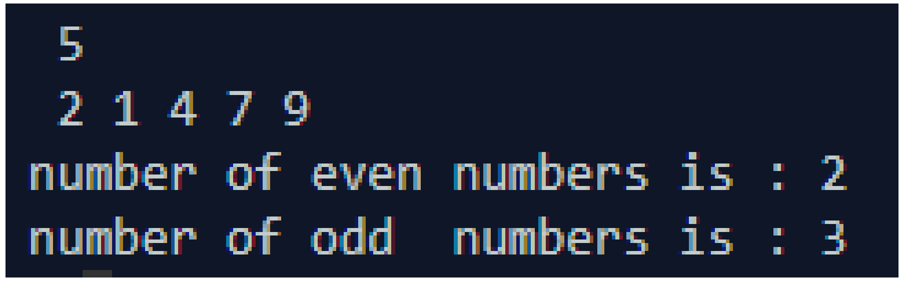
```cpp
#include <iostream>
using namespace std;

int main()
{
    int n;
    cin >> n;

    int* x = new int[n];

    cout << "Input array is : \n";
    for(int i = 0; i < n; i++)
        cin >> x[i];

    cout << "Output array is : \n";
    for(int i = 0; i < n; i++)
        cout << x[i] << ' ';
    cout << '\n';

    delete []x;
}
```



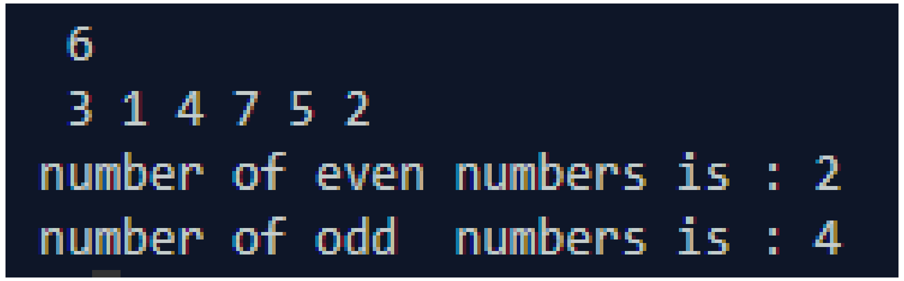
```
 5
Input array is :
 12 -34 52 64 82
Output array is :
12 -34 52 64 82
```

Practice

- Take as an input n and read n numbers as an array then count number of even and odd numbers in this array, such that n >= 1 and integer
- Test Cases:



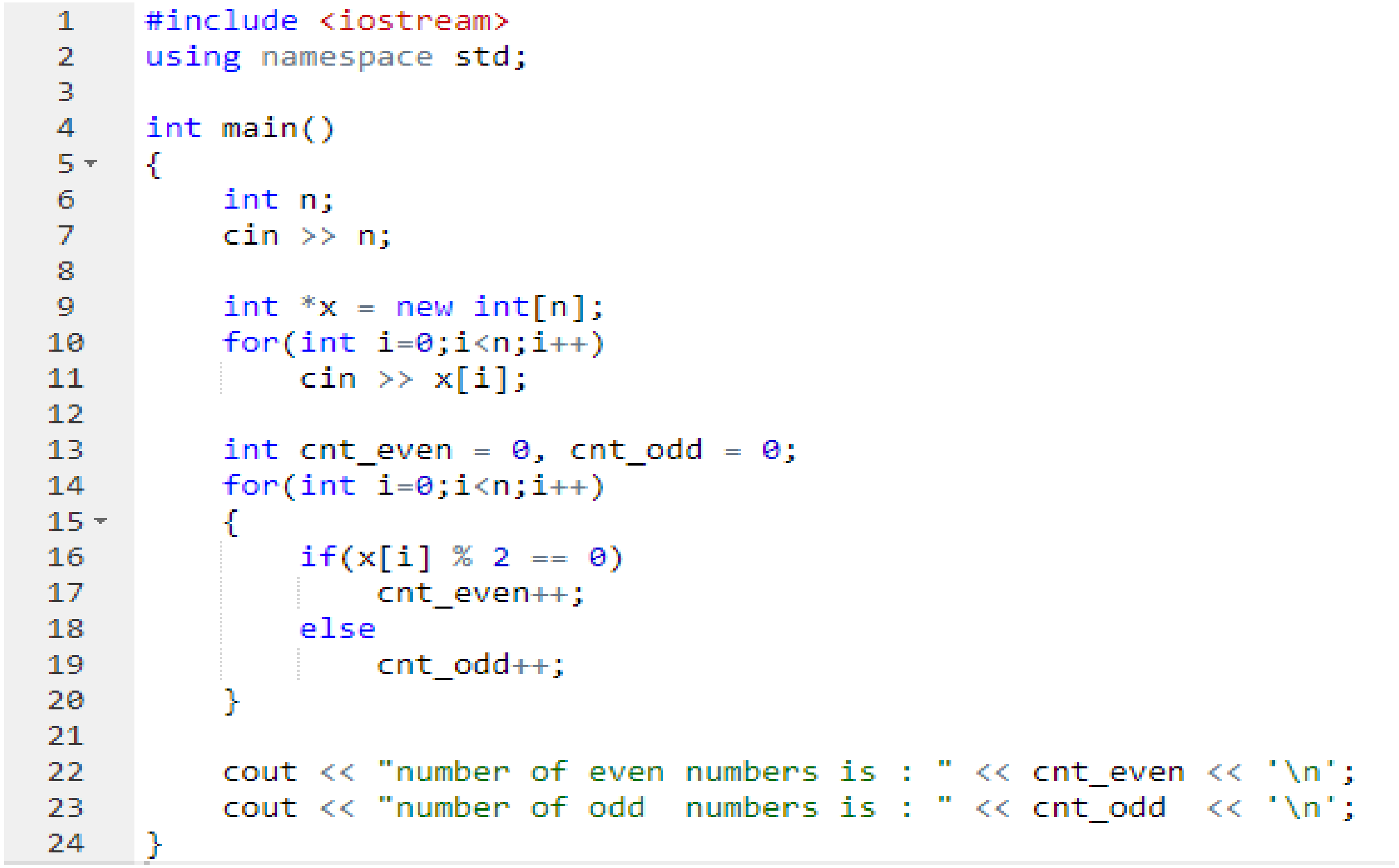
```
 5
 2 1 4 7 9
number of even numbers is : 2
number of odd  numbers is : 3
```



```
 6
 3 1 4 7 5 2
number of even numbers is : 2
number of odd  numbers is : 4
```

Practice Solution

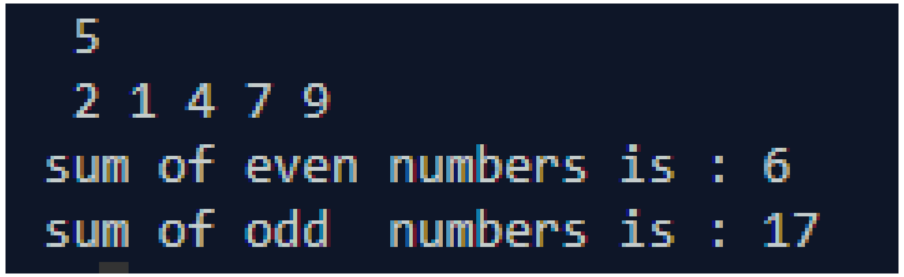
- Source code: https://repl.it/repls/GoldSmartServal



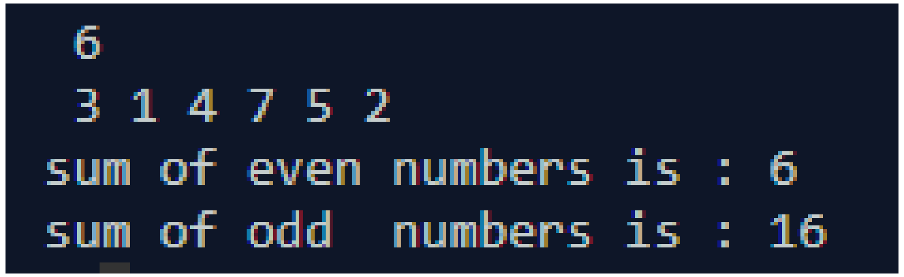
```cpp
#include <iostream>
using namespace std;

int main()
{
    int n;
    cin >> n;

    int *x = new int[n];
    for(int i=0;i<n;i++)
        cin >> x[i];

    int cnt_even = 0, cnt_odd = 0;
    for(int i=0;i<n;i++)
    {
        if(x[i] % 2 == 0)
            cnt_even++;
        else
            cnt_odd++;
    }

    cout << "number of even numbers is : " << cnt_even << '\n';
    cout << "number of odd  numbers is : " << cnt_odd  << '\n';
}
```

Practice

- Take as an input n and read n numbers as an array then calculate sum of even and odd numbers in this array, such that n >= 1 and integer
- Test Cases:



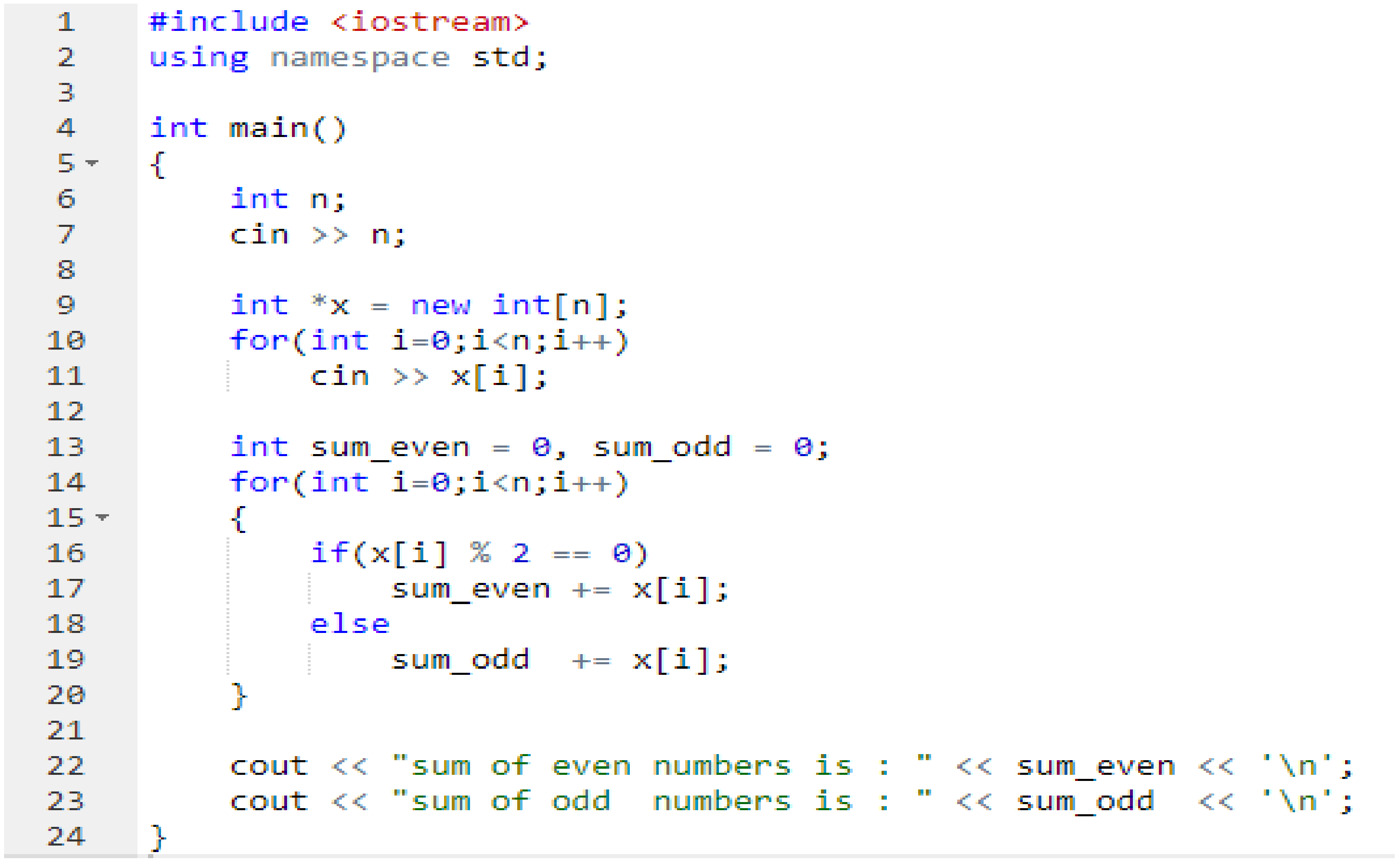
```
 5
 2 1 4 7 9
sum of even numbers is : 6
sum of odd  numbers is : 17
```



```
 6
 3 1 4 7 5 2
sum of even numbers is : 6
sum of odd  numbers is : 16
```

Practice Solution

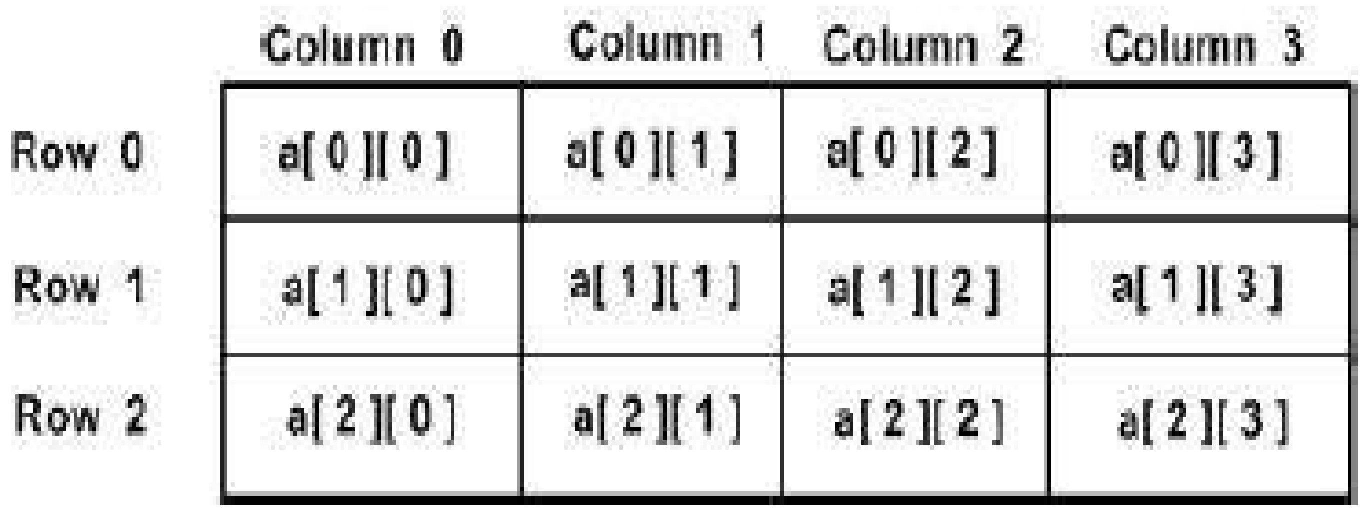
- Source code: https://repl.it/repls/BareFamousImperatorangel



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

int main()
{
    int n;
    cin >> n;

    int *x = new int[n];
    for(int i=0;i<n;i++)
        cin >> x[i];

    int sum_even = 0, sum_odd = 0;
    for(int i=0;i<n;i++)
    {
        if(x[i] % 2 == 0)
            sum_even += x[i];
        else
            sum_odd  += x[i];
    }

    cout << "sum of even numbers is : " << sum_even << '\n';
    cout << "sum of odd  numbers is : " << sum_odd  << '\n';
}
```

Multi-dimensional Arrays

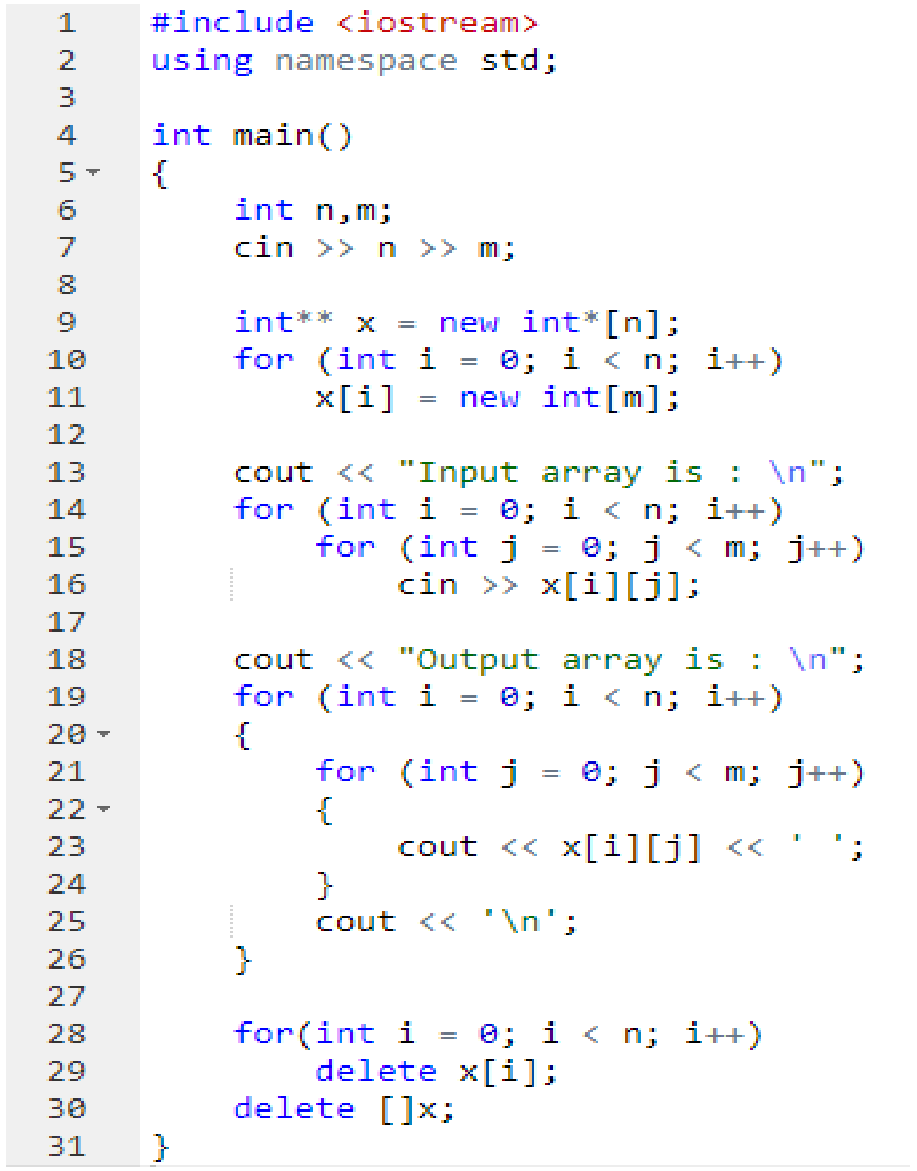
- Thus, every element in array a is identified by an element name of the form a[i][j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.



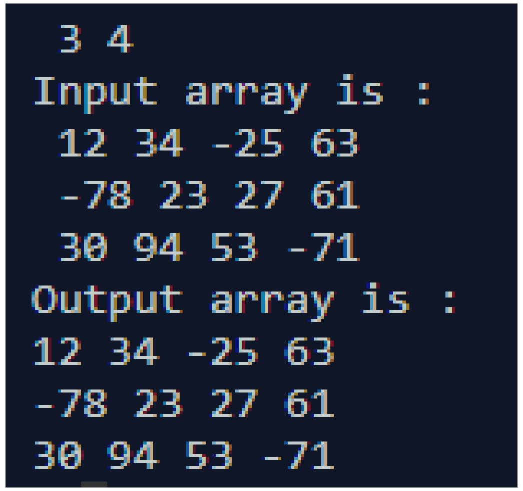
	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Dynamic Arrays Example

- Source code: https://repl.it/repls/ImpressionableWelldocumentedDachshund



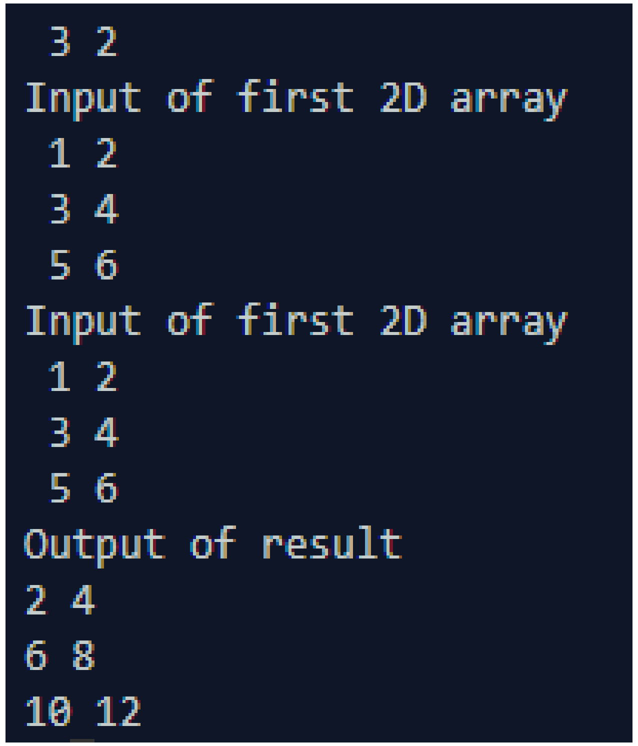
```cpp
#include <iostream>
using namespace std;

int main()
{
    int n,m;
    cin >> n >> m;

    int** x = new int*[n];
    for (int i = 0; i < n; i++)
        x[i] = new int[m];

    cout << "Input array is : \n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> x[i][j];

    cout << "Output array is : \n";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cout << x[i][j] << ' ';
        }
        cout << '\n';
    }

    for(int i = 0; i < n; i++)
        delete x[i];
    delete []x;
}
```



```
 3 4
Input array is :
 12 34 -25 63
 -78 23 27 61
 30 94 53 -71
Output array is :
12 34 -25 63
-78 23 27 61
30 94 53 -71
```

Practice

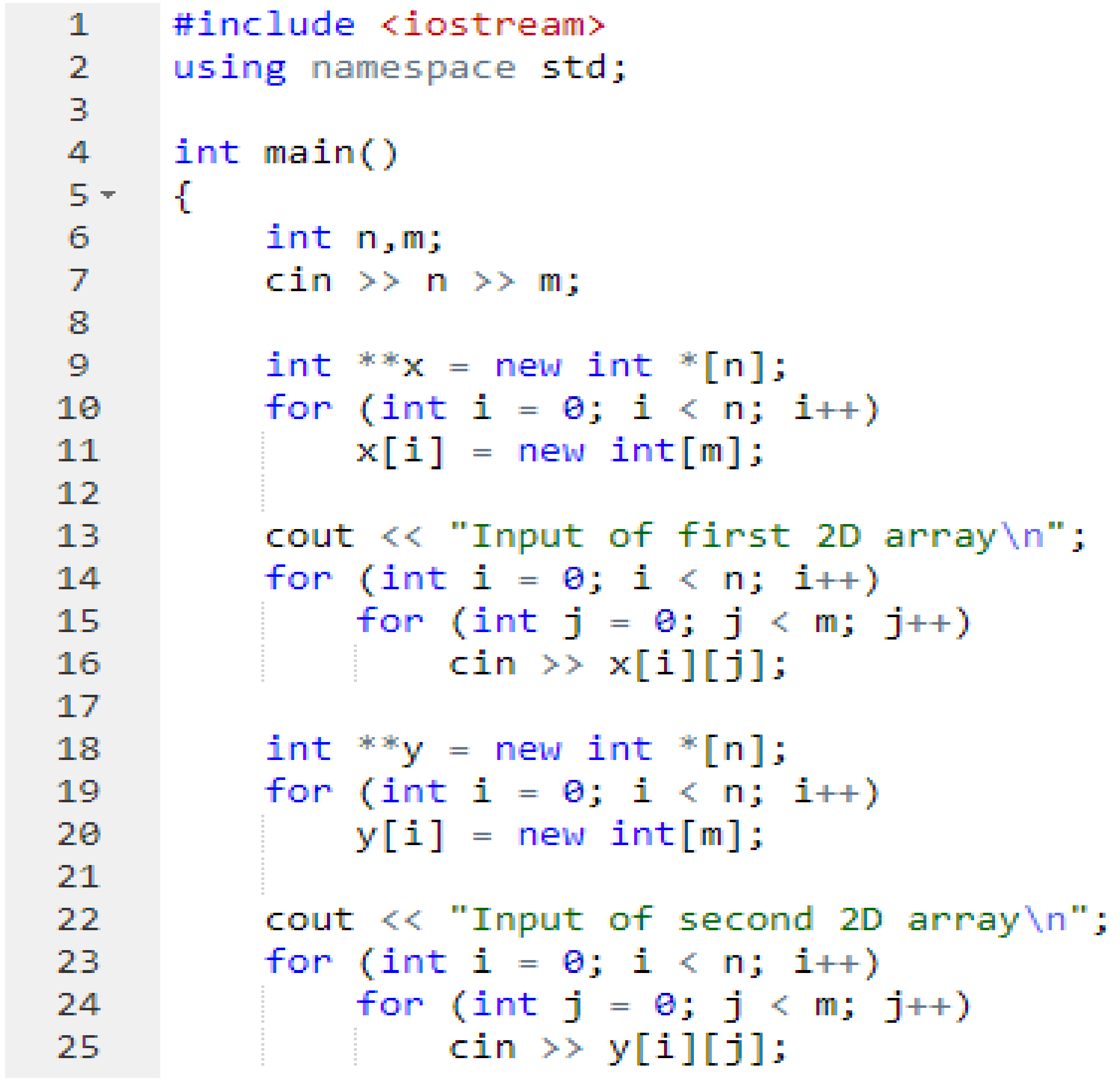
- Take as an input n, m then take as an input two matrices as an input 2D array in shape n*m for each one then calculate the addition of them
- Test Cases:



```
 3 2
Input of first 2D array
 1 2
 3 4
 5 6
Input of first 2D array
 1 2
 3 4
 5 6
Output of result
2 4
6 8
10 12
```

Practice Solution

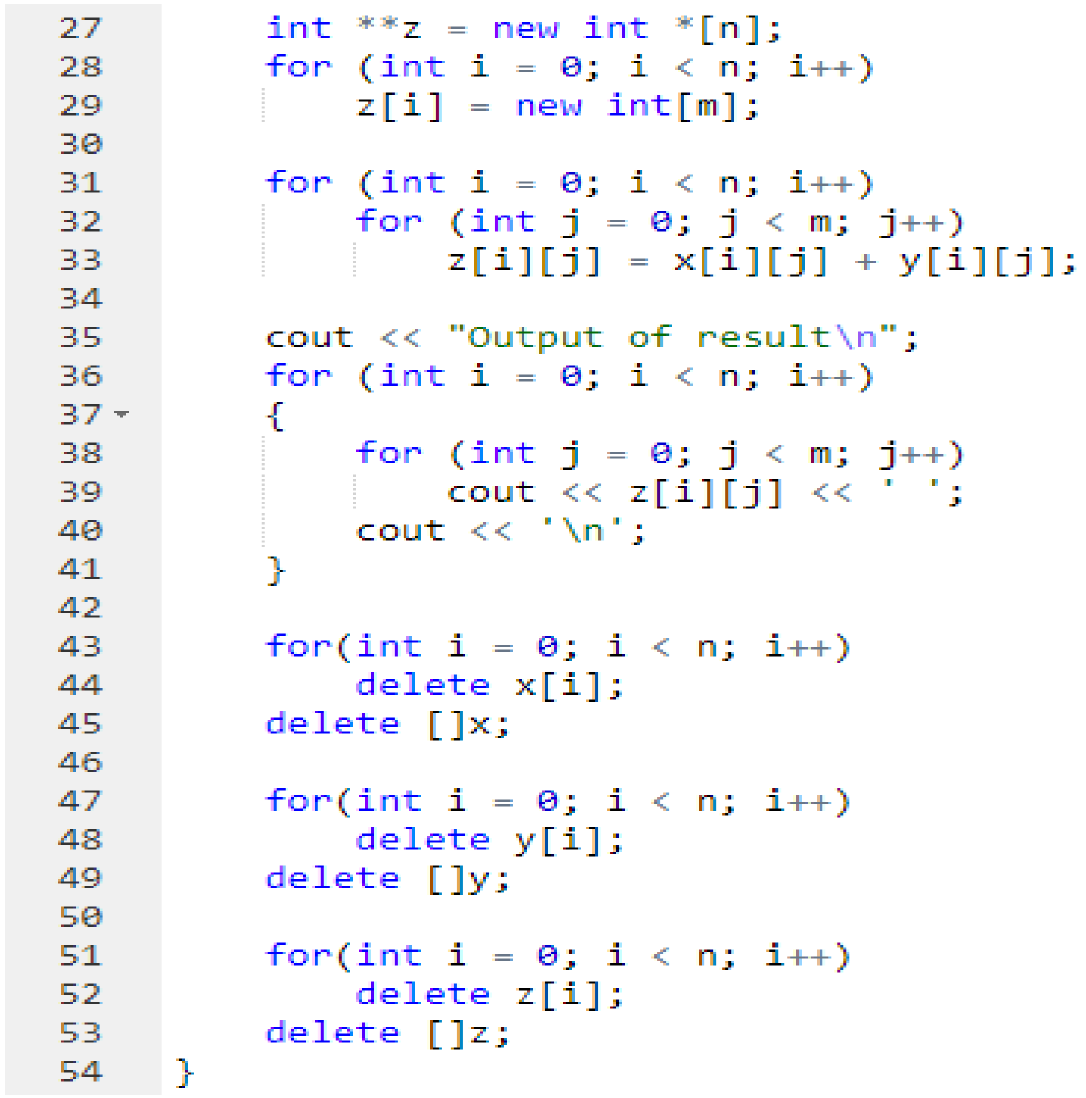
- Source code: https://repl.it/repls/AlertAngryHarrierhawk



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

int main()
{
    int n,m;
    cin >> n >> m;

    int **x = new int *[n];
    for (int i = 0; i < n; i++)
        x[i] = new int[m];

    cout << "Input of first 2D array\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> x[i][j];

    int **y = new int *[n];
    for (int i = 0; i < n; i++)
        y[i] = new int[m];

    cout << "Input of second 2D array\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> y[i][j];
```

Practice Solution

- Source code: https://repl.it/repls/AlertAngryHarrierhawk



```cpp
    int **z = new int *[n];
    for (int i = 0; i < n; i++)
        z[i] = new int[m];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            z[i][j] = x[i][j] + y[i][j];

    cout << "Output of result\n";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
            cout << z[i][j] << ' ';
        cout << '\n';
    }

    for(int i = 0; i < n; i++)
        delete x[i];
    delete []x;

    for(int i = 0; i < n; i++)
        delete y[i];
    delete []y;

    for(int i = 0; i < n; i++)
        delete z[i];
    delete []z;
}
```

Static Arrays vs. Dynamic Arrays

- Static Arrays:
  - Local arrays are created on the stack and have automatic storage duration, you don't need to manually manage memory, but they get destroyed when the function they're in ends. They necessarily have a fixed size.
  - Static arrays are allocated memory at compile time and the memory is allocated on the stack.

```
int foo[10];
```

- Dynamic Arrays:
  - Arrays created with operator new[ ] have dynamic storage duration and are stored on the heap (technically the "free store"). They can have any size, but you need to allocate and free them yourself since they're not part of the stack frame.
  - Dynamic arrays are allocated memory at the runtime and the memory is allocated from heap.

```
int* foo = new int[10];
```

Questions ?

References

Online Courses YouTube playlists:	http://bit.ly/2kAPL5K
C++ Documentation	http://bit.ly/1flmcHO
CPP For School	http://bit.ly/2kifMdj
C++ Language Tutorial	http://bit.ly/1kyBMdz
C++ Language Tutorial	http://bit.ly/2rzE4hQ
C++ Tutorial Point	http://bit.ly/2BGFeO0
Fundamentals of C++ Programming	http://bit.ly/2rJHhyI
Teach Yourself C++ in 21 Days	http://bit.ly/1JXhDtL
A Complete Guide to Programming in C++	http://bit.ly/2dVkGY9