Object Oriented Programming (CS1143)

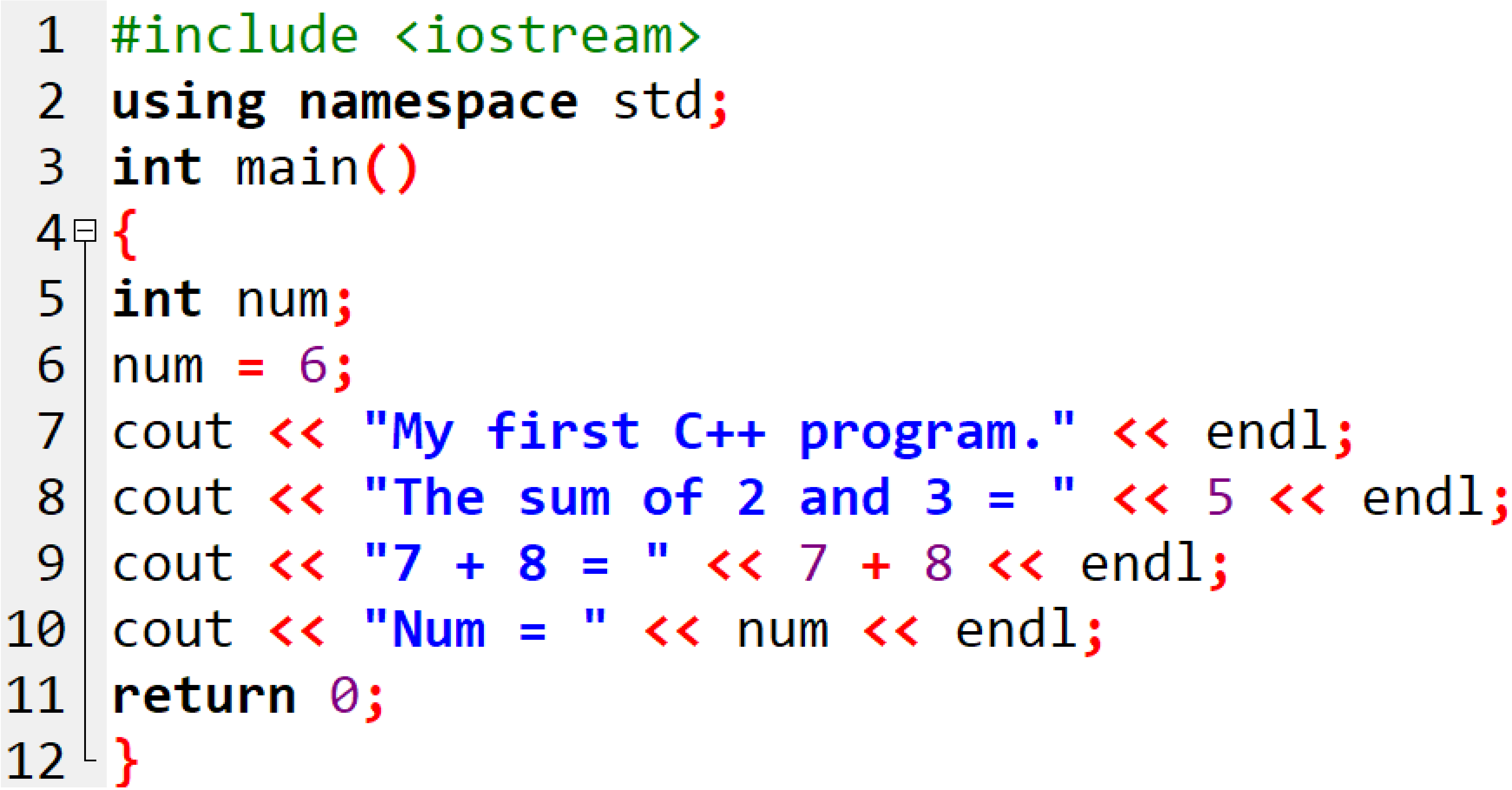
Department of Computer Science

Capital University of Science and Technology (CUST)

Structured Programming (Revision)

Week 1

Example:



```
#include <iostream>
using namespace std;
int main()
{
int num;
num = 6;
cout << "My first C++ program." << endl;
cout << "The sum of 2 and 3 = " << 5 << endl;
cout << "7 + 8 = " << 7 + 8 << endl;
cout << "Num = " << num << endl;
return 0;
}
```

#Include

- **# Include**
  - The **#include** directive tells the compiler to include some already existing C++ code in your program
  - The included file is then linked with the program
  - There are two forms of **#include** statements:

```
#include <iostream> //for pre-defined files

#include "my_lib.h"   //for user-defined files
```

Input and Output Statements

- cin >> variable-name;

Meaning: read the value of the variable called <variable-name> from the user

Example:

```
cin >> a;
cin >> b >> c;
```

Input and Output Statements Cont..

- cout << variable-name;

  Meaning: print the value of variable <name> to the user
- cout << “any message”;

  Meaning: print the message within quotes to the user
- cout << endl;

  Meaning: print a new line

Example:

```
  cout << a;
  cout << b << c;
  cout << “This is my character: “<<endl << my-character<< endl;
```

Comments

- Part of good programming is the inclusion of comments in the program
- There are two type of Comments
  - Single-line comments
    - // Welcome to C++ Programming
  - Multiple-line comments
    - Enclosed between /* and */

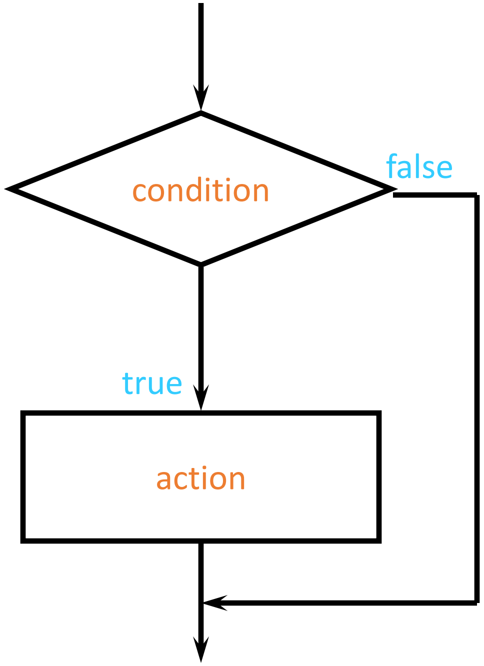
A Choice Statement

- Syntax

```
    if(condition)
          action
```

- if the `condition` is true then execute the `action`.

- `action` is either a single statement or a group of statements within braces.



Another Choice Statement

- Syntax

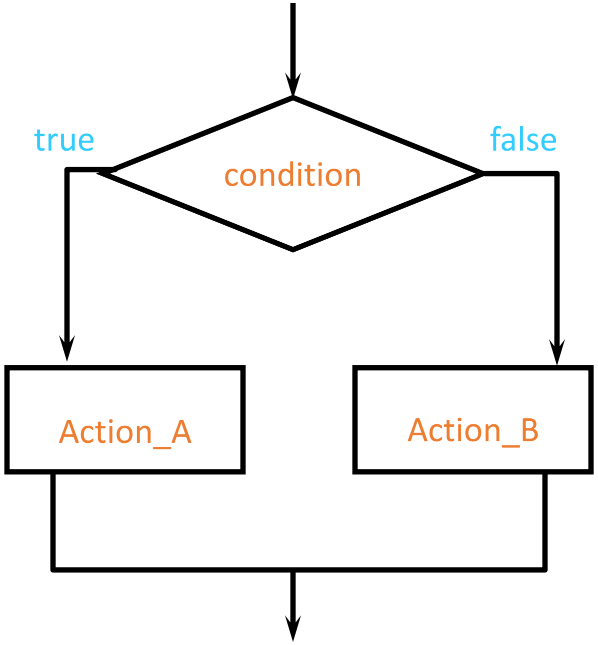
```
if (condition)
       Action_A
else
       Action_B
```

- if the condition is true

    execute Action_A

  else

    execute Action_B

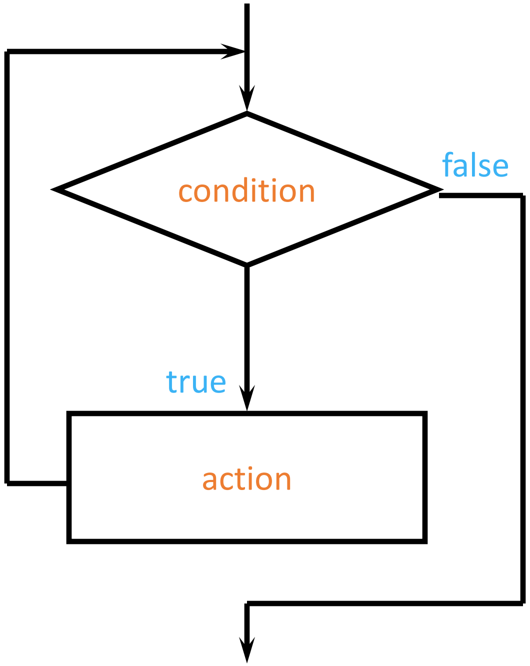


A Loop Statement

- Syntax

```
    while (condition)
        action
```

- How it works:
    - if condition is true then execute action
    - repeat this process until condition evaluates to false
- action is either a single statement or a group of statements within braces.

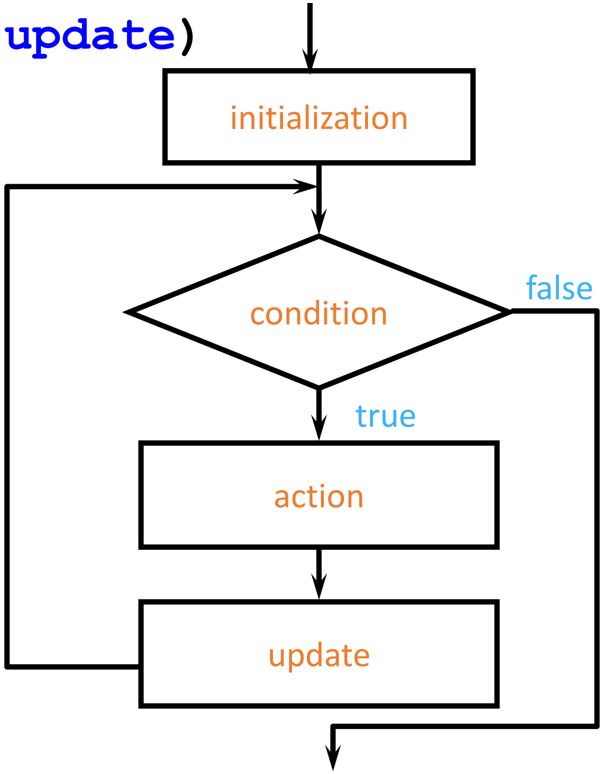


Another Loop Statement

- Syntax

```
for (initialization; condition; update)
      action
```

- How it works:
  - execute **initialization** statement
  - while **condition** is true
    - execute **action**
    - execute **update**

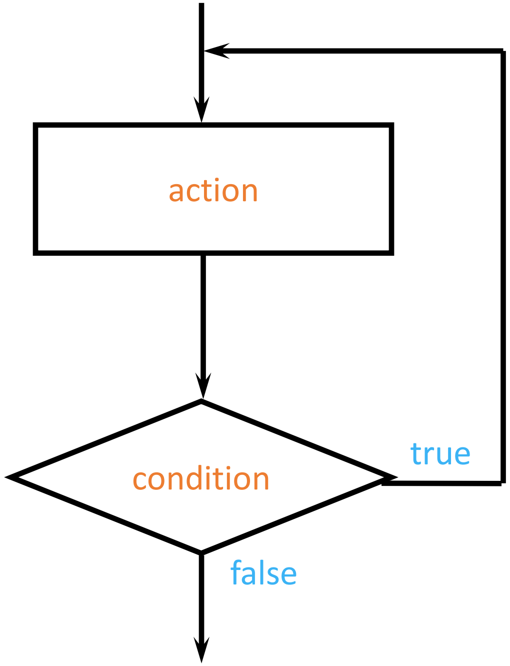


Yet Another Loop Statement

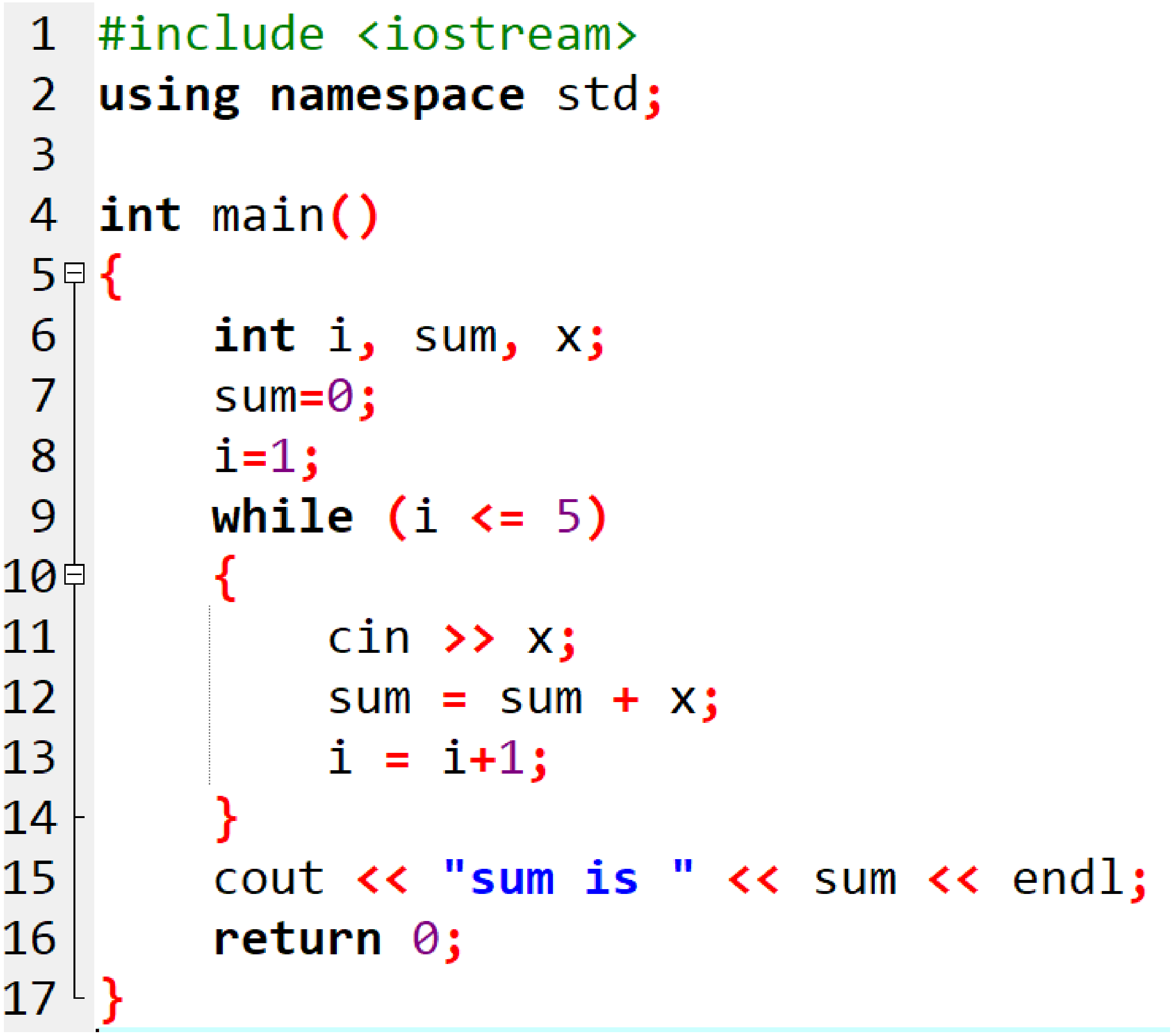
- Syntax

```
    do action
   while  (condition)
```

- How it works:
  - execute action
  - if condition is true then execute action again
  - repeat this process until condition evaluates to false.
- action is either a single statement or a group of statements within braces.

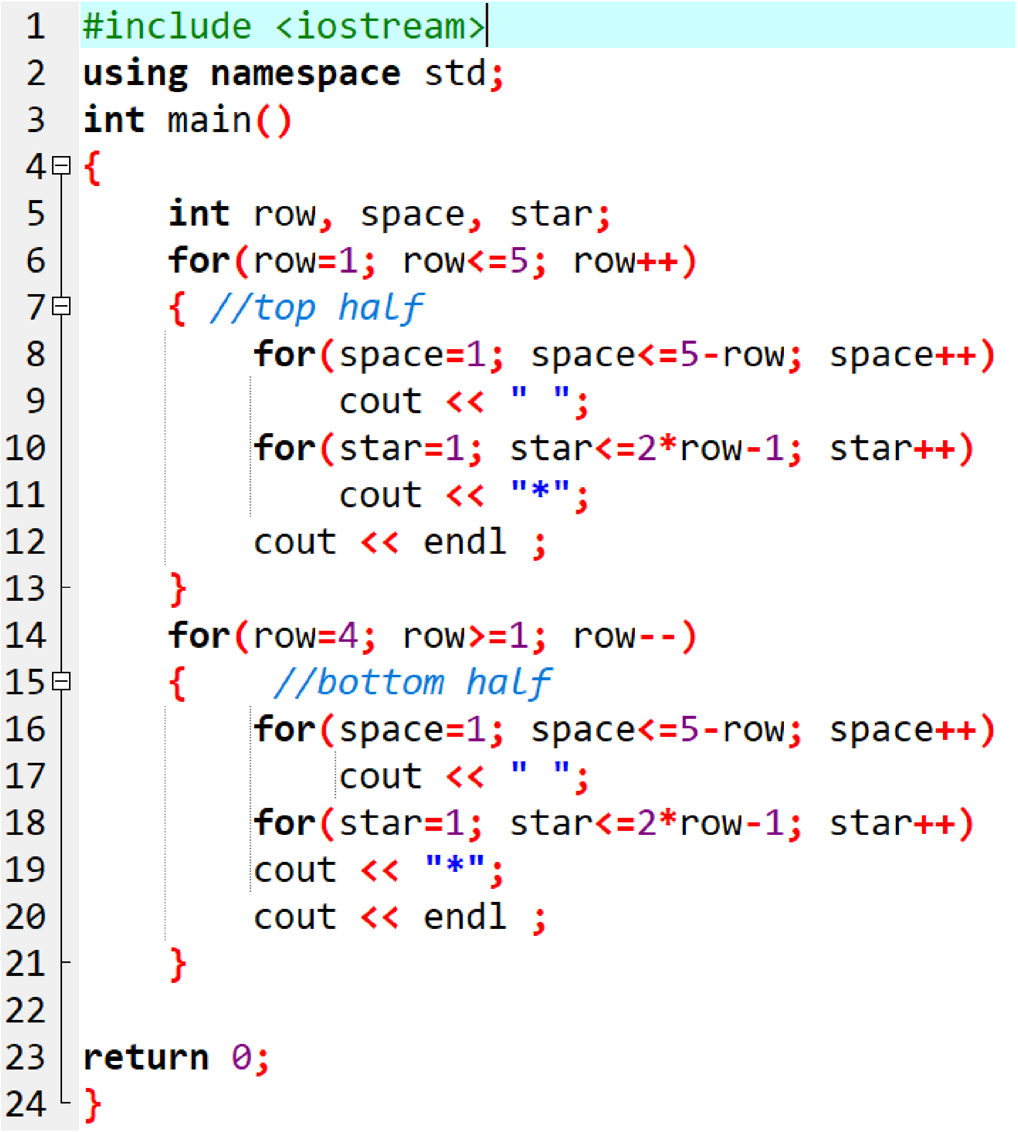


Example: Get 5 numbers from the user and add them



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

int main()
{
    int i, sum, x;
    sum=0;
    i=1;
    while (i <= 5)
    {
        cin >> x;
        sum = sum + x;
        i = i+1;
    }
    cout << "sum is " << sum << endl;
    return 0;
}
```

Example: Diamond Pattern



```cpp
#include <iostream>
using namespace std;
int main()
{
    int row, space, star;
    for(row=1; row<=5; row++)
    { //top half
        for(space=1; space<=5-row; space++)
            cout << " ";
        for(star=1; star<=2*row-1; star++)
            cout << "*";
        cout << endl ;
    }
    for(row=4; row>=1; row--)
    {    //bottom half
        for(space=1; space<=5-row; space++)
            cout << " ";
        for(star=1; star<=2*row-1; star++)
        cout << "*";
        cout << endl ;
    }

return 0;
}
```

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Arrays

- Array is a data structure that represents a collection of variables of the same data type
- Once an array is created, its size is fixed
- The following will create an array of size 5

```
    int c[5];
 We can directly initialize the same array as
  int c[]={1,2,3,4,5};
```

Using the Shorthand Notation

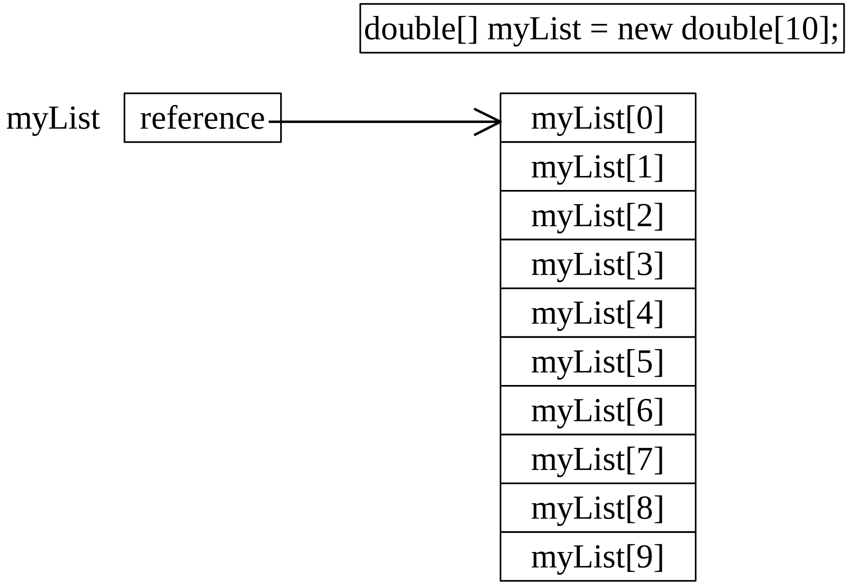
```
double[] myList = {1, 9, 3, 5};
```

This shorthand notation is equivalent to the following statements:

```
    double[] myList = new double[4];
    myList[0] = 1;
    myList[1] = 9;
    myList[2] = 3;
    myList[3] = 5;
```

Arrays Cont..

- Example



```
double[] myList = new double[10];
```

myList

reference

myList[0]

myList[1]

myList[2]

myList[3]

myList[4]

myList[5]

myList[6]

myList[7]

myList[8]

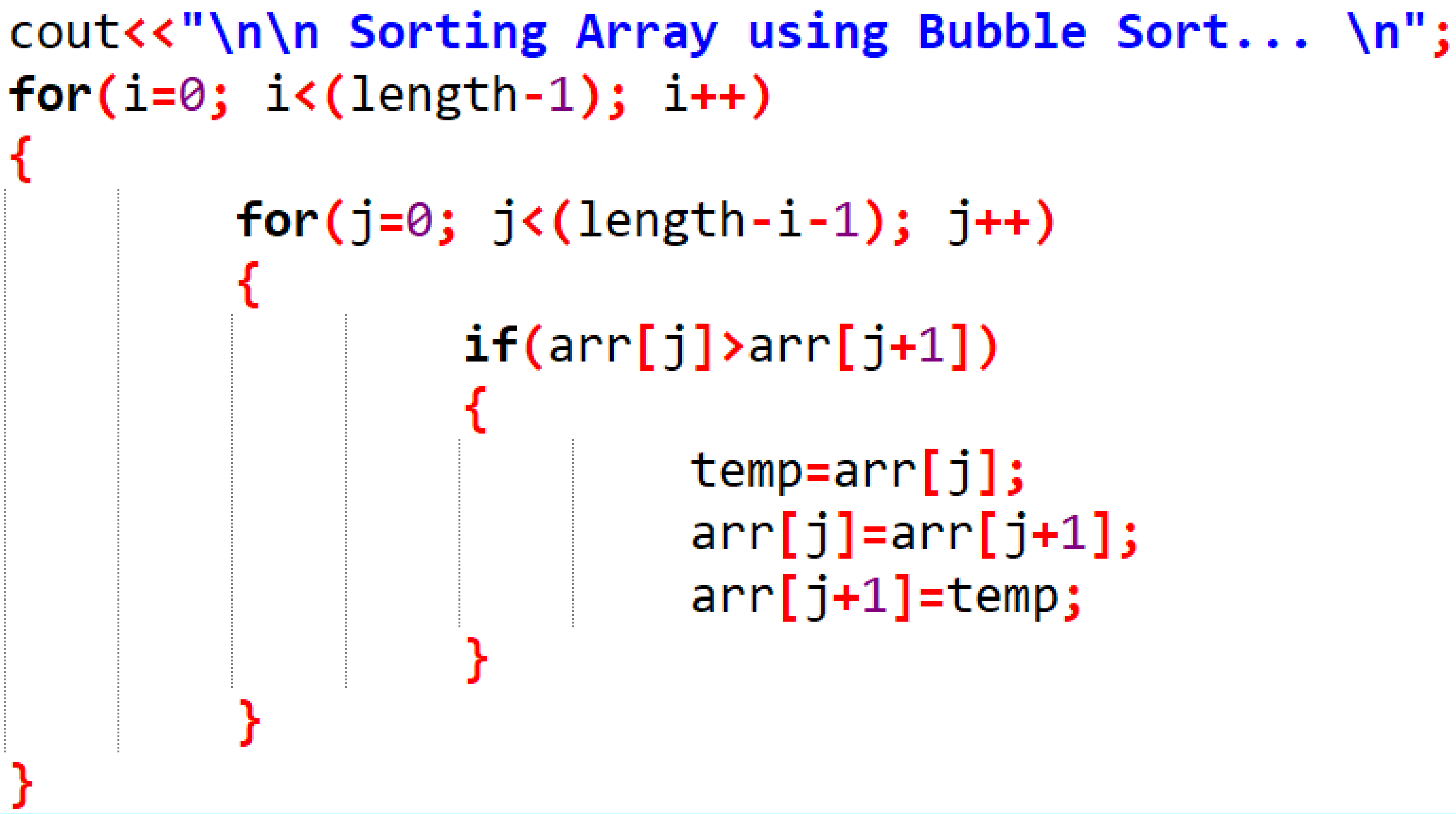
myList[9]

Initializing Arrays

- Using a loop:

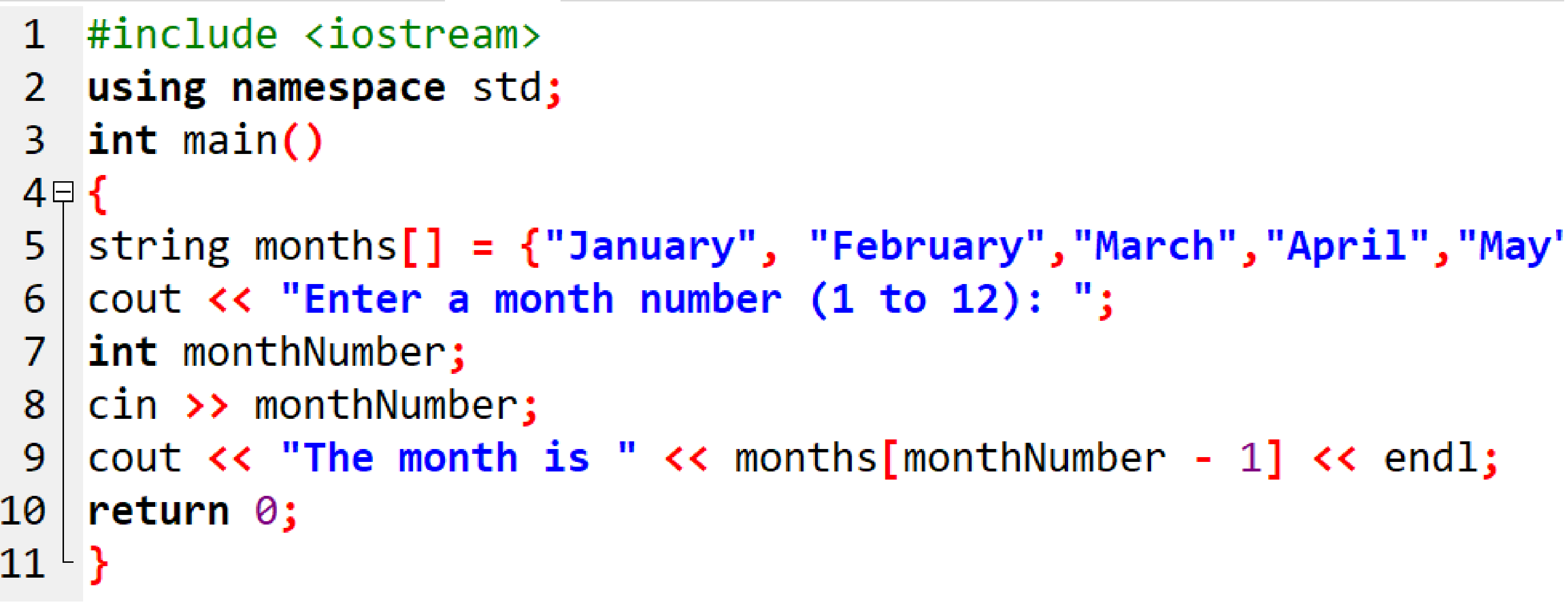
```
for (int i = 0; i < 5; i++)
        myList[i] = i;
```

Example Sorting

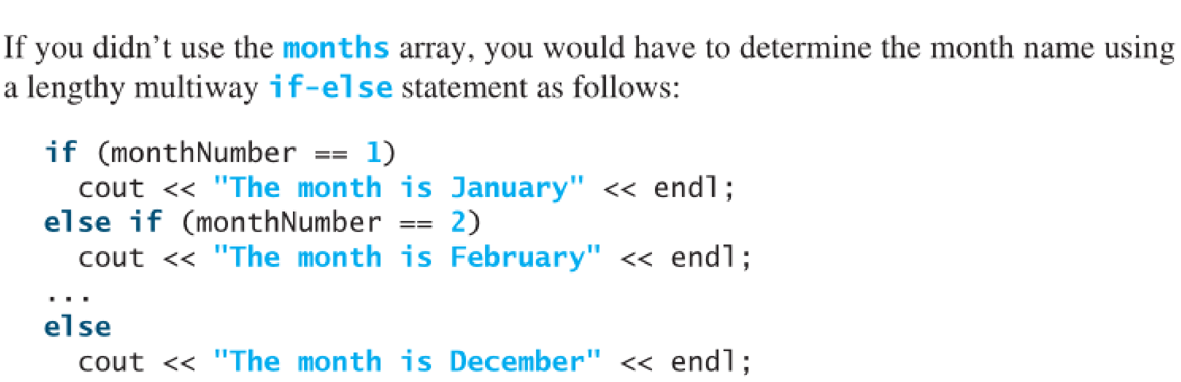


```
cout<<"\n\n Sorting Array using Bubble Sort... \n";
for(i=0; i<(length-1); i++)
{
    for(j=0; j<(length-i-1); j++)
    {
        if(arr[j]>arr[j+1])
        {
            temp=arr[j];
            arr[j]=arr[j+1];
            arr[j+1]=temp;
        }
    }
}
```

String array



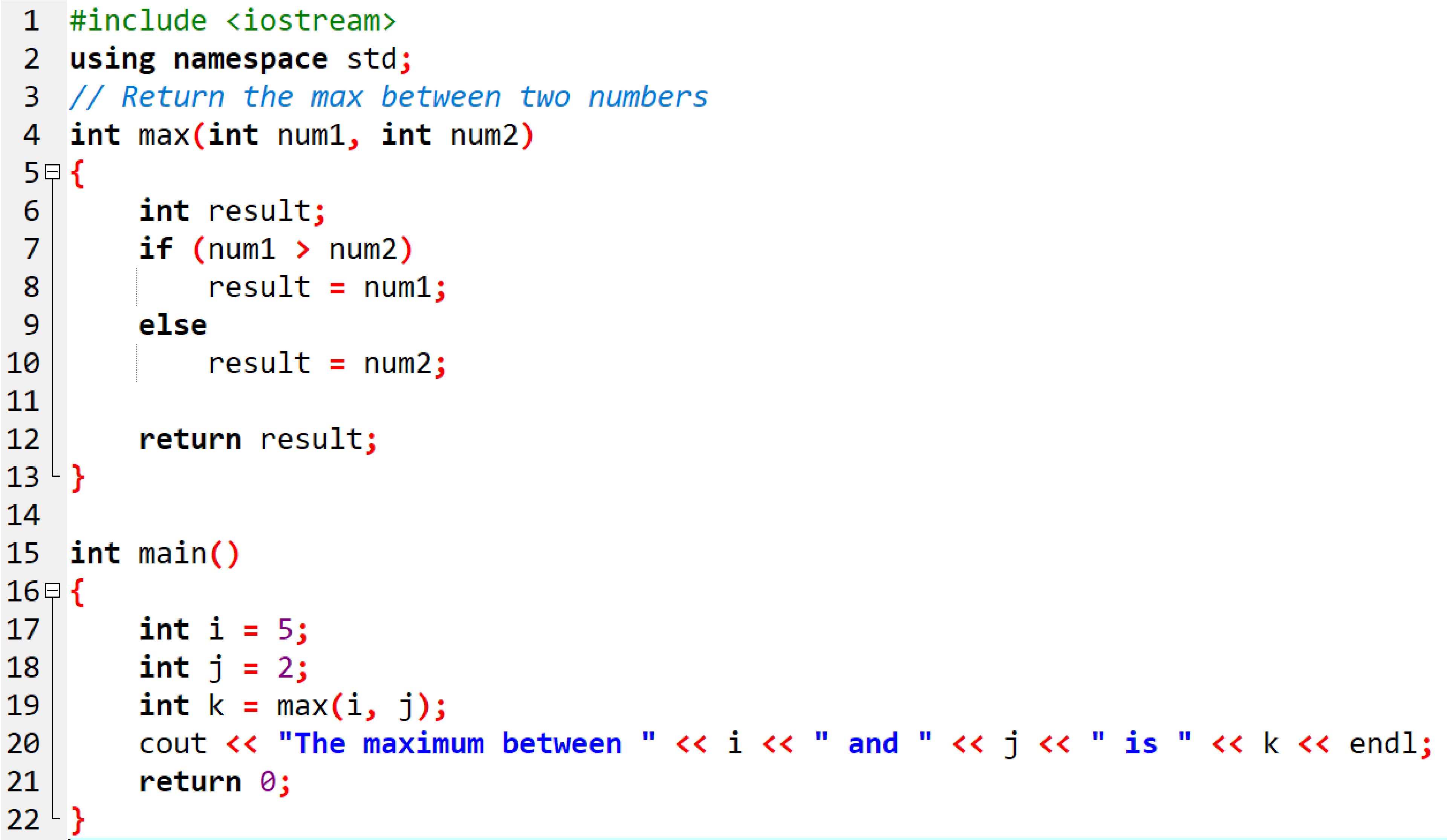
```
#include <iostream>
using namespace std;
int main()
{
string months[] = {"January", "February","March","April","May'
cout << "Enter a month number (1 to 12): ";
int monthNumber;
cin >> monthNumber;
cout << "The month is " << months[monthNumber - 1] << endl;
return 0;
}
```



If you didn't use the **months** array, you would have to determine the month name using a lengthy multiway **if-else** statement as follows:

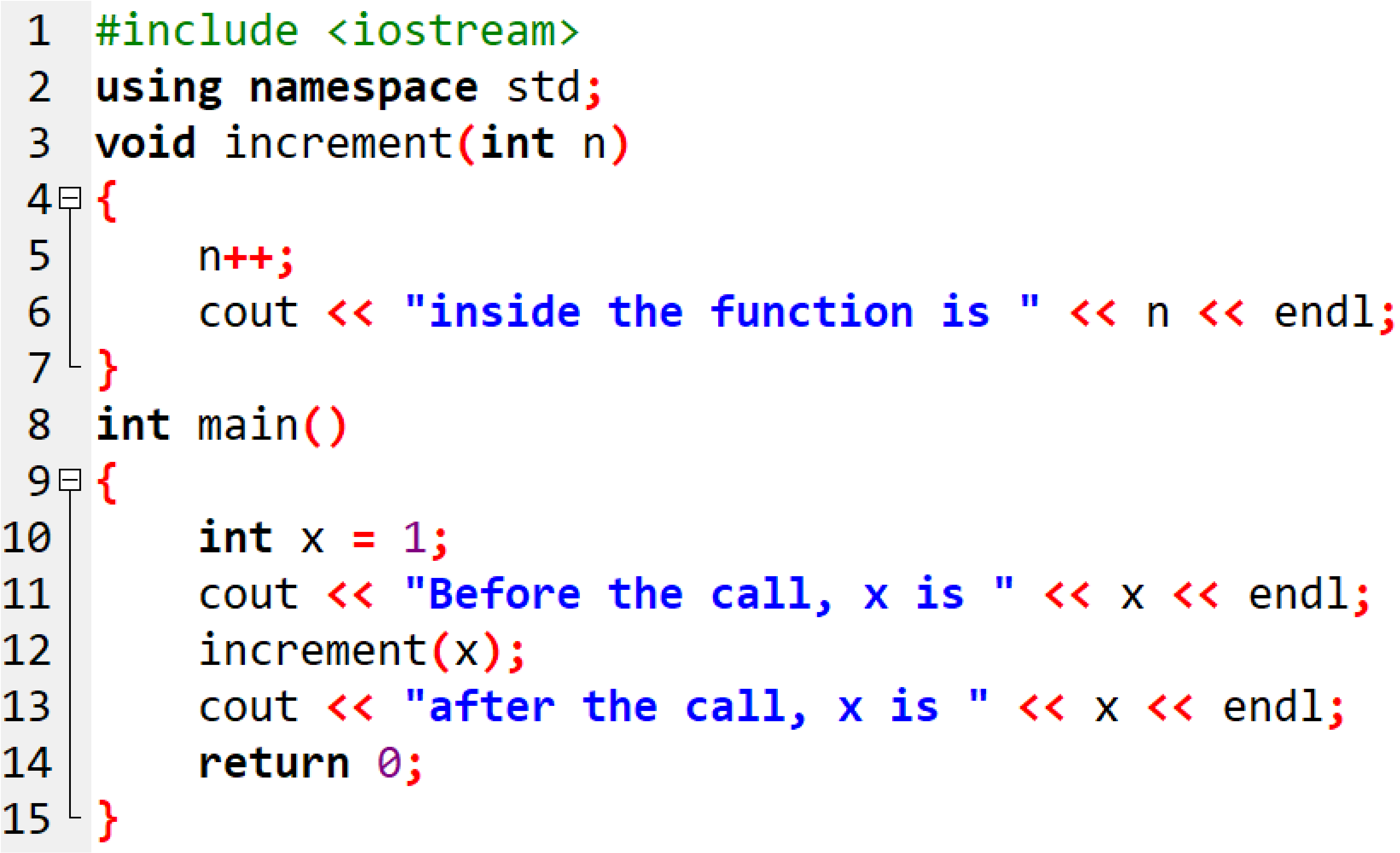
```
if (monthNumber == 1)
  cout << "The month is January" << endl;
else if (monthNumber == 2)
  cout << "The month is February" << endl;
...
else
  cout << "The month is December" << endl;
```

Functions



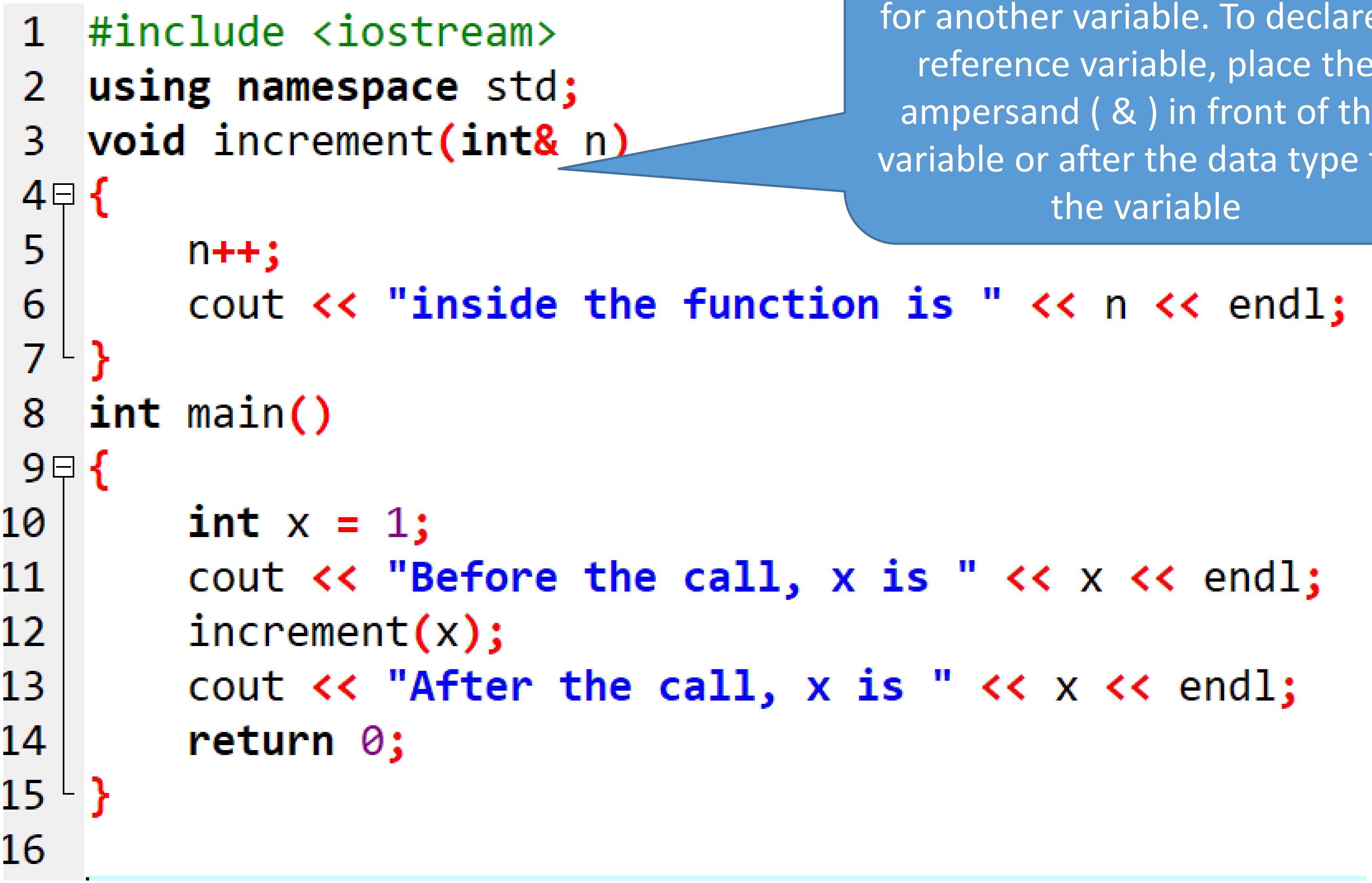
```cpp
#include <iostream>
using namespace std;
// Return the max between two numbers
int max(int num1, int num2)
{
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}

int main()
{
    int i = 5;
    int j = 2;
    int k = max(i, j);
    cout << "The maximum between " << i << " and " << j << " is " << k << endl;
    return 0;
}
```

Pass by Value Example



```cpp
#include <iostream>
using namespace std;
void increment(int n)
{
    n++;
    cout << "inside the function is " << n << endl;
}
int main()
{
    int x = 1;
    cout << "Before the call, x is " << x << endl;
    increment(x);
    cout << "after the call, x is " << x << endl;
    return 0;
}
```

Pass by Reference Example

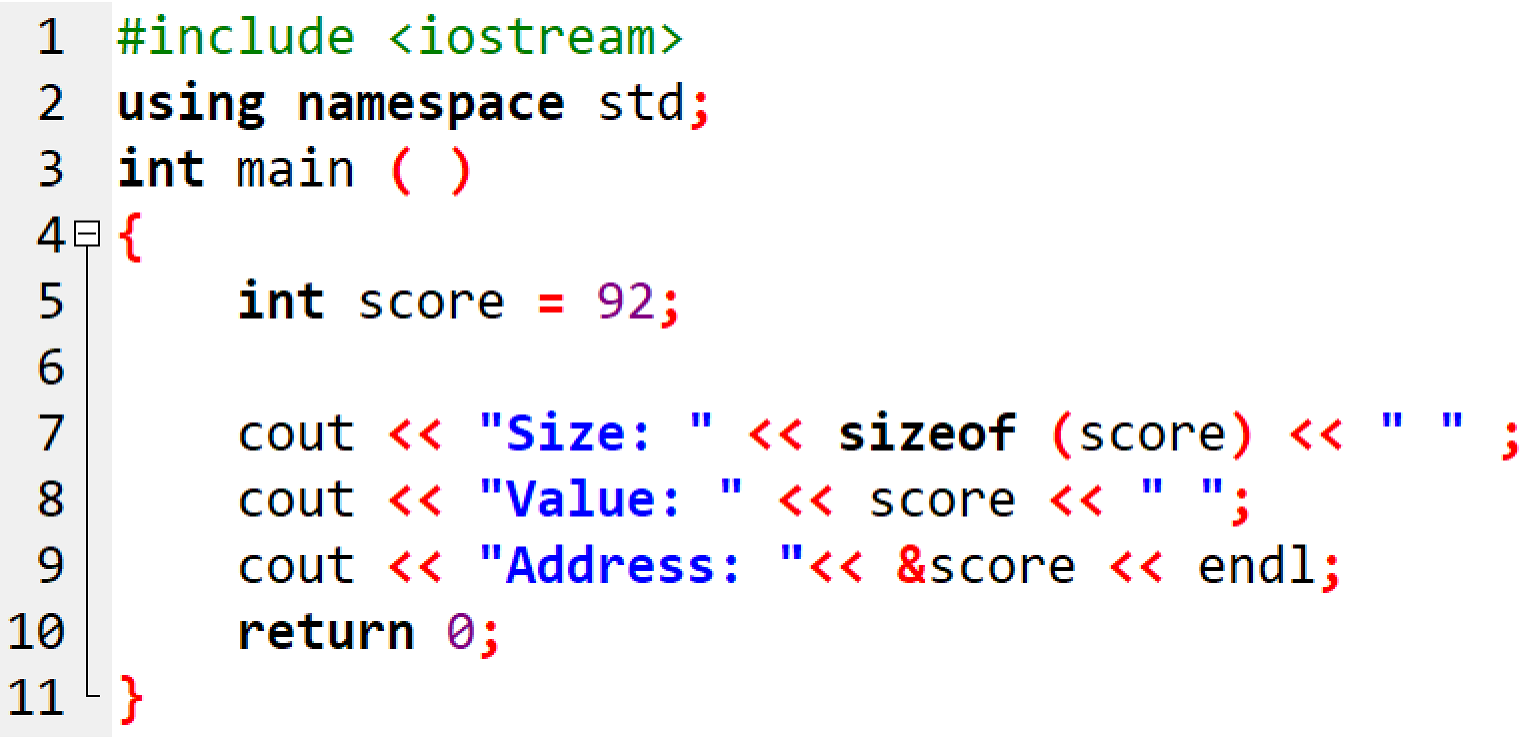


```
#include <iostream>
using namespace std;
void increment(int& n)
{
    n++;
    cout << "inside the function is " << n << endl;
}
int main()
{
    int x = 1;
    cout << "Before the call, x is " << x << endl;
    increment(x);
    cout << "After the call, x is " << x << endl;
    return 0;
}

```

A reference variable is an alias for another variable. To declare a reference variable, place the ampersand (&) in front of the variable or after the data type for the variable

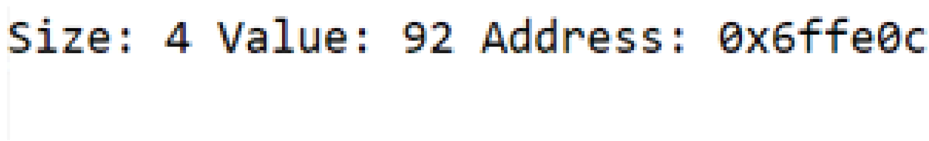
Pointers

Using & to get address of a variable



```
#include <iostream>
using namespace std;
int main ( )
{
	int score = 92;

	cout << "Size: " << sizeof (score) << " " ;
	cout << "Value: " << score << " ";
	cout << "Address: "<< &score << endl;
	return 0;
}
```

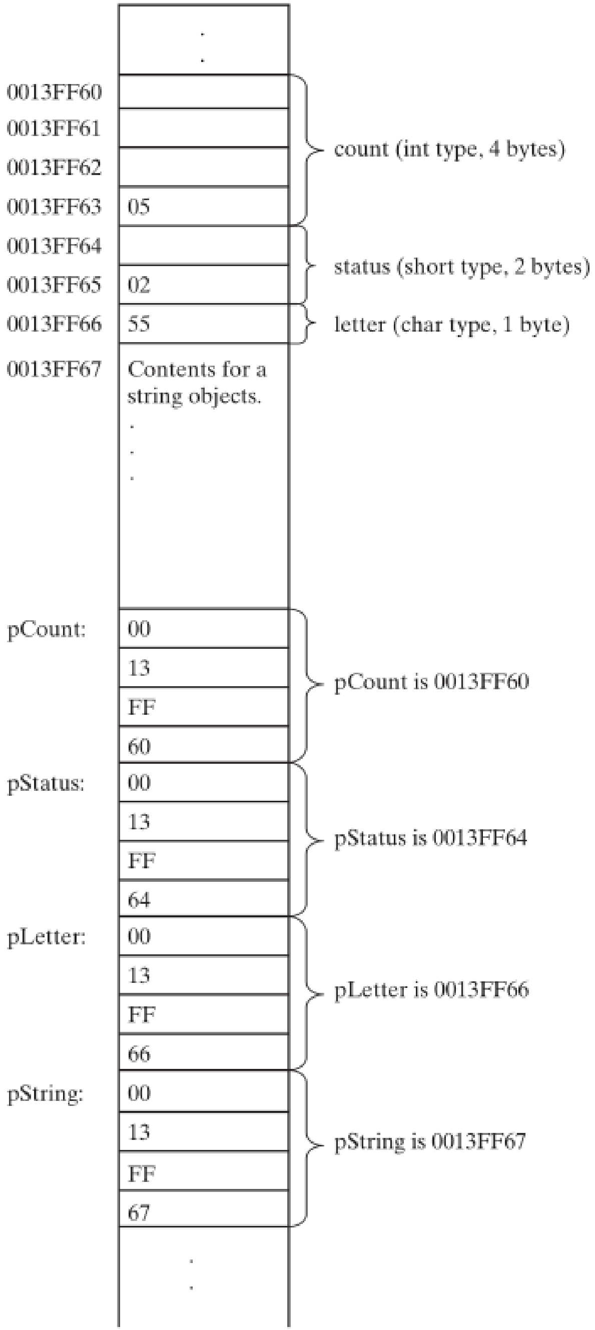
OUTPUT



```
Size: 4 Value: 92 Address: 0x6ffe0c
```

Pointers

- Pointer variables, simply called pointers, are declared to hold memory addresses as their values.
- Although we can have different pointer types, what is stored in a pointer variable is a 4-byte address.
- In other words, the size of a pointer variable is fixed in C++, which means that a pointer variable can have an address from 0x00000000 to 0xFFFFFFFF.



0013FF60

0013FF61

0013FF62

0013FF63 05

0013FF64

0013FF65 02

0013FF66 55

0013FF67 Contents for a string objects.

count (int type, 4 bytes)

status (short type, 2 bytes)

letter (char type, 1 byte)

pCount: 00 13 FF 60

pCount is 0013FF60

pStatus: 00 13 FF 64

pStatus is 0013FF64

pLetter: 00 13 FF 66

pLetter is 0013FF66

pString: 00 13 FF 67

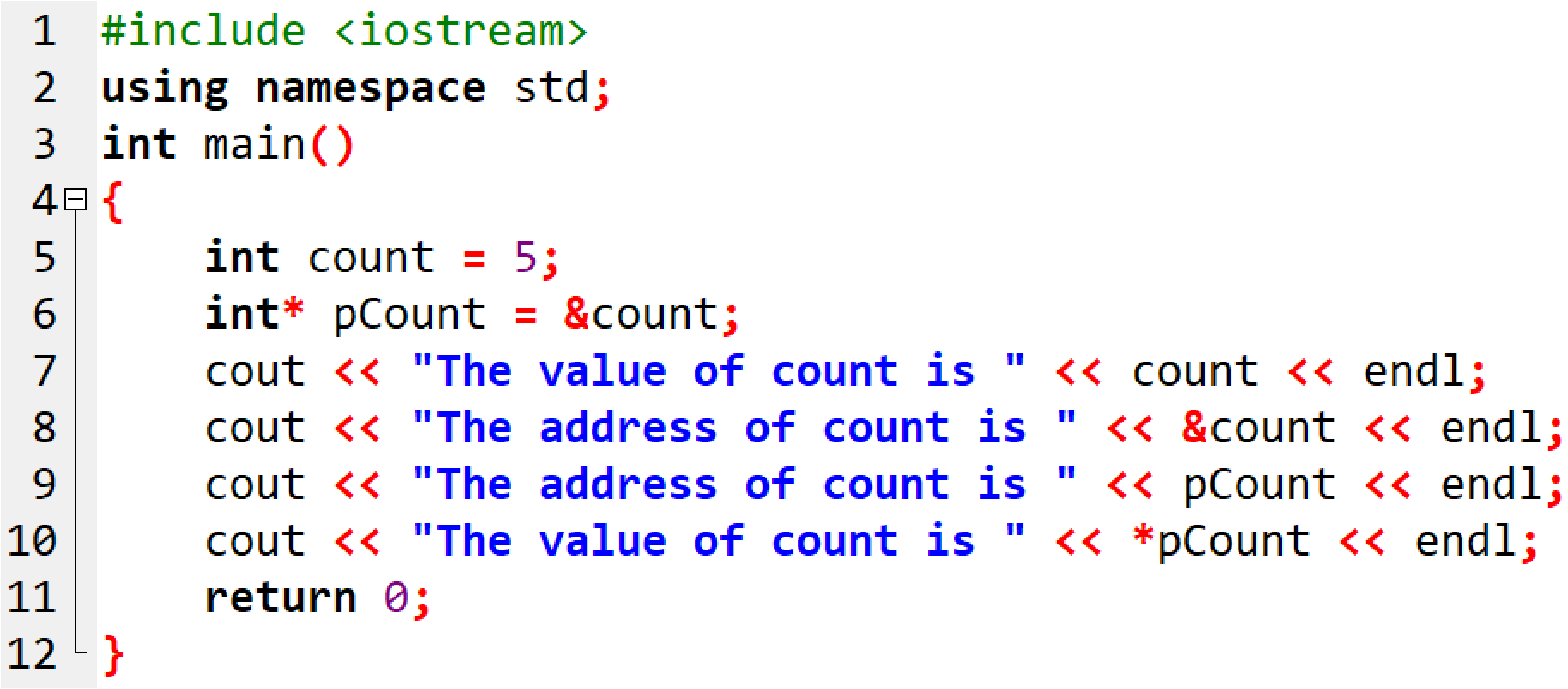
pString is 0013FF67

```
int count = 5;
short status = 2;
char letter = 'A';
string s = "ABC";

int* pCount = &count;
short* pStatus = &status;
char* pLetter = &letter;
string* pString = &s;

pCount = &count;
```

&: address operator
&count means the address of count

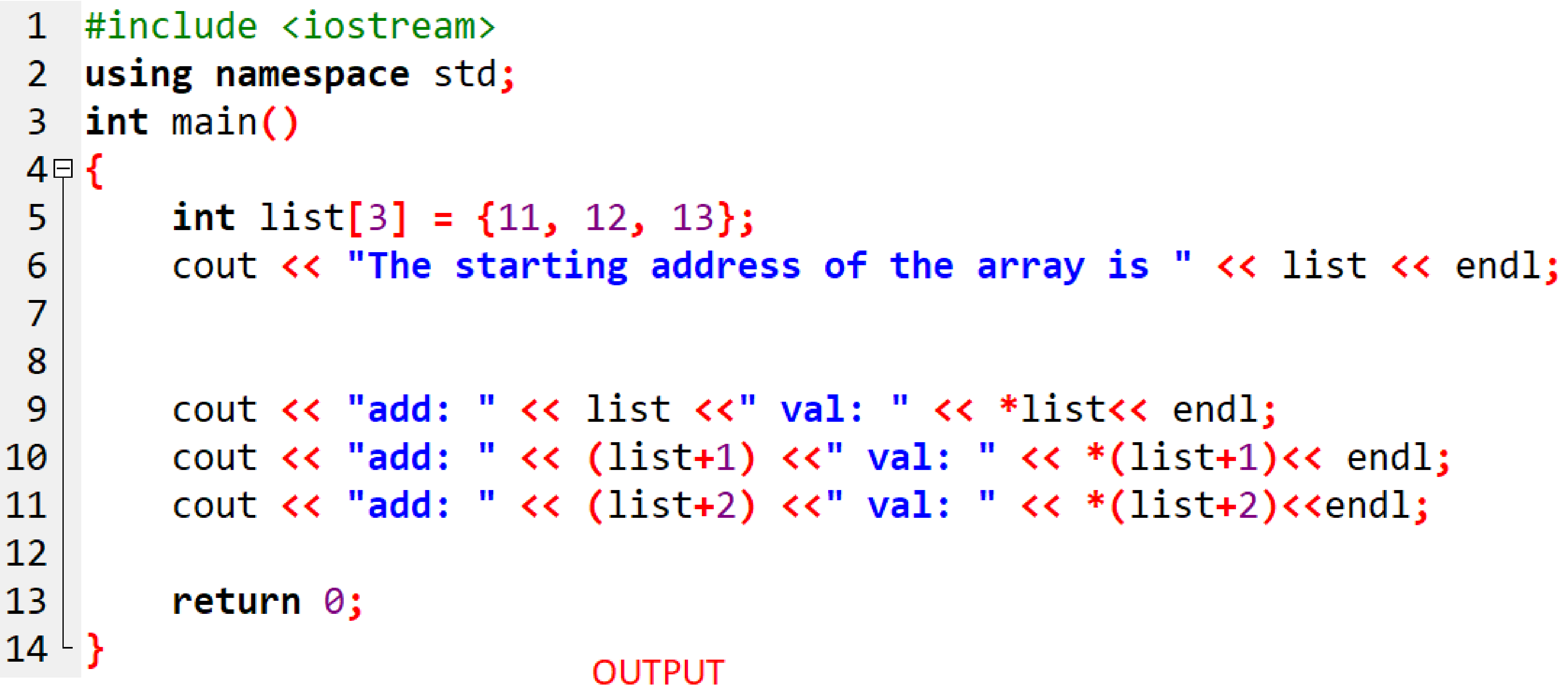
*: dereferencing operator
*pCount means the value pointed by pCount is assigned to v.

Example: Pointers



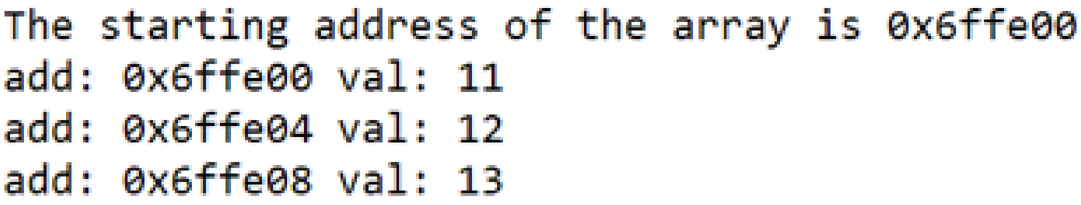
```
#include <iostream>
using namespace std;
int main()
{
    int count = 5;
    int* pCount = &count;
    cout << "The value of count is " << count << endl;
    cout << "The address of count is " << &count << endl;
    cout << "The address of count is " << pCount << endl;
    cout << "The value of count is " << *pCount << endl;
    return 0;
}
```

Example: Pointers and Arrays



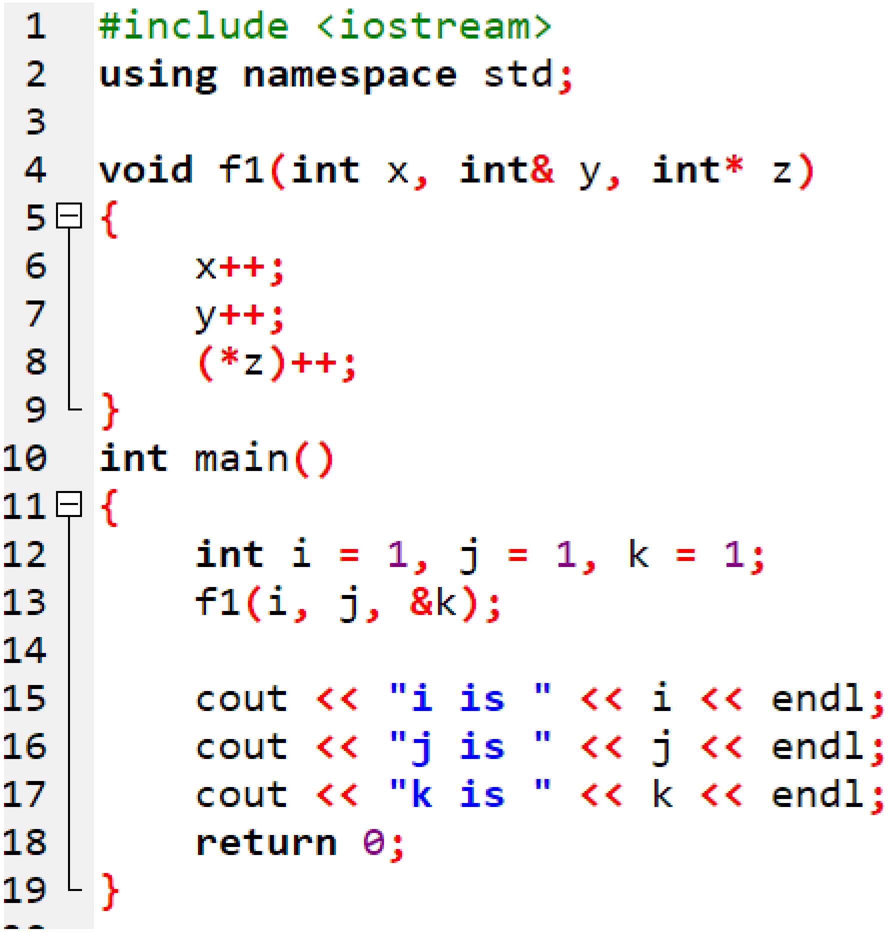
```
#include <iostream>
using namespace std;
int main()
{
    int list[3] = {11, 12, 13};
    cout << "The starting address of the array is " << list << endl;


    cout << "add: " << list <<" val: " << *list<< endl;
    cout << "add: " << (list+1) <<" val: " << *(list+1)<< endl;
    cout << "add: " << (list+2) <<" val: " << *(list+2)<<endl;

    return 0;
}
```

OUTPUT



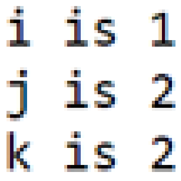
```
The starting address of the array is 0x6ffe00
add: 0x6ffe00 val: 11
add: 0x6ffe04 val: 12
add: 0x6ffe08 val: 13
```

Example: Pointers as Function Parameters



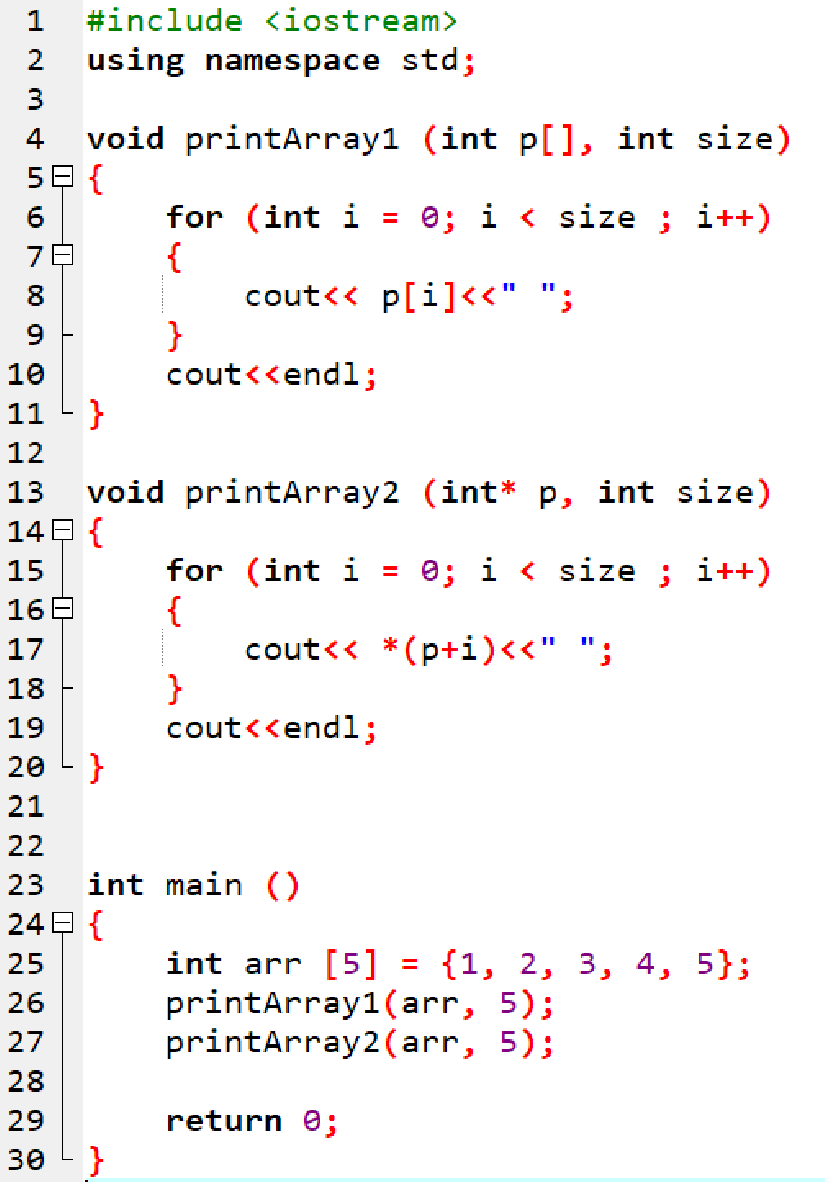
```
#include <iostream>
using namespace std;

void f1(int x, int& y, int* z)
{
    x++;
    y++;
    (*z)++;
}
int main()
{
    int i = 1, j = 1, k = 1;
    f1(i, j, &k);

    cout << "i is " << i << endl;
    cout << "j is " << j << endl;
    cout << "k is " << k << endl;
    return 0;
}
```

OUTPUT



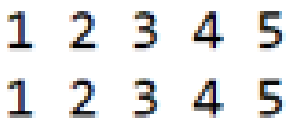
```
i is 1
j is 2
k is 2
```

Example: Passing Arrays to Functions



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

void printArray1 (int p[], int size)
{
    for (int i = 0; i < size ; i++)
    {
        cout<< p[i]<<" ";
    }
    cout<<endl;
}

void printArray2 (int* p, int size)
{
    for (int i = 0; i < size ; i++)
    {
        cout<< *(p+i)<<" ";
    }
    cout<<endl;
}


int main ()
{
    int arr [5] = {1, 2, 3, 4, 5};
    printArray1(arr, 5);
    printArray2(arr, 5);

    return 0;
}
```

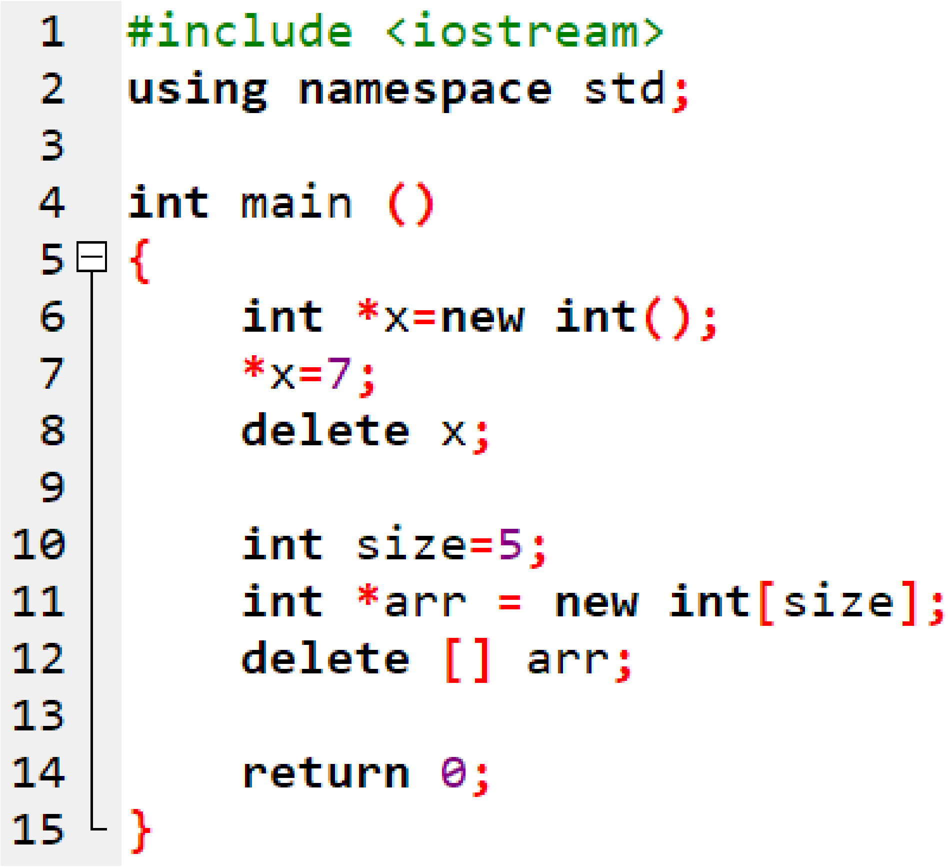
OUTPUT



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

Are arrays passed by value or by reference?

Dynamic Memory allocation: new and delete



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

int main ()
{
    int *x=new int();
    *x=7;
    delete x;

    int size=5;
    int *arr = new int[size];
    delete [] arr;

    return 0;
}
```

That is all for Week 1