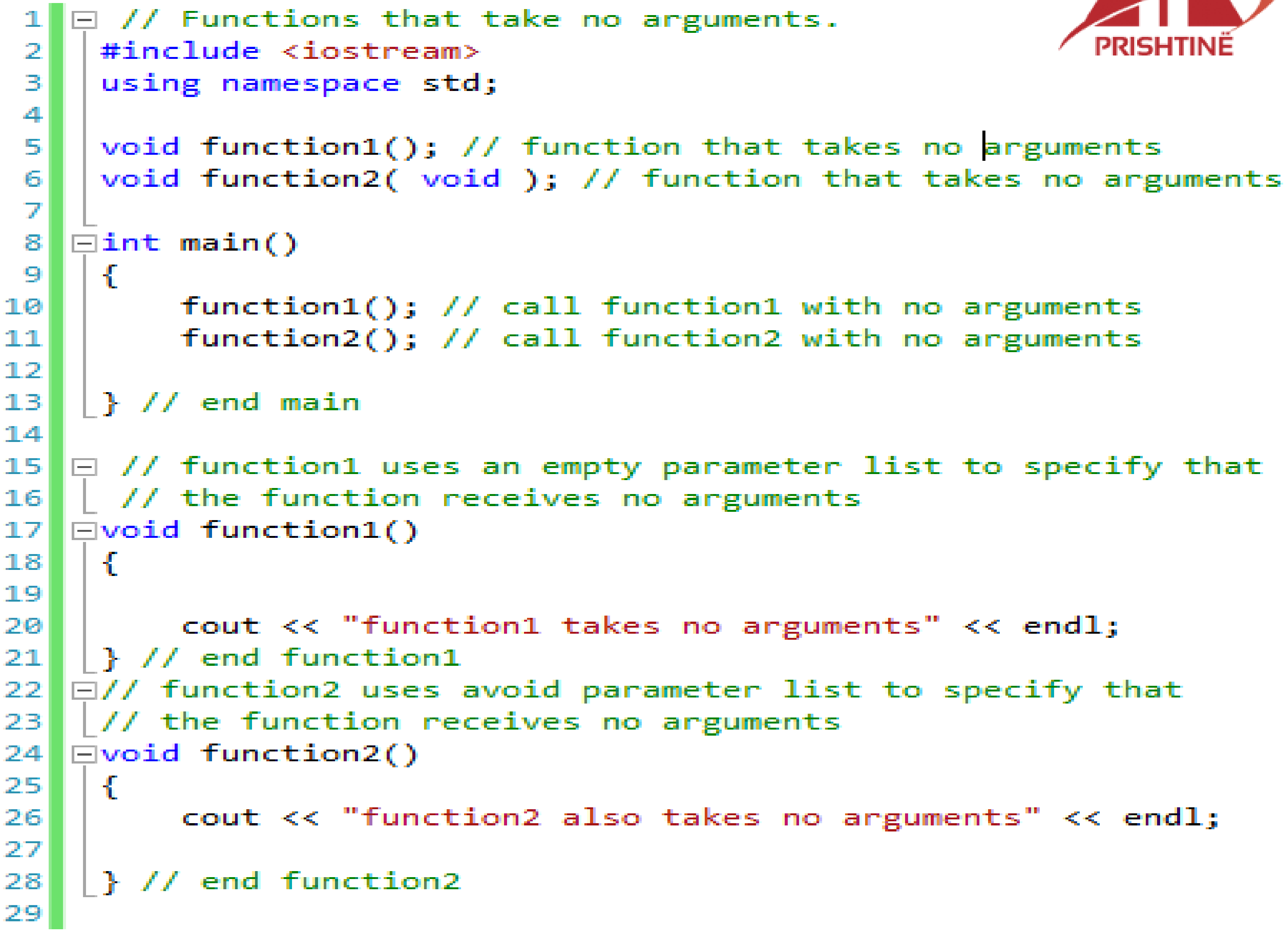
# PROGRAMI I ORIENTUAR NE OBJEKTE

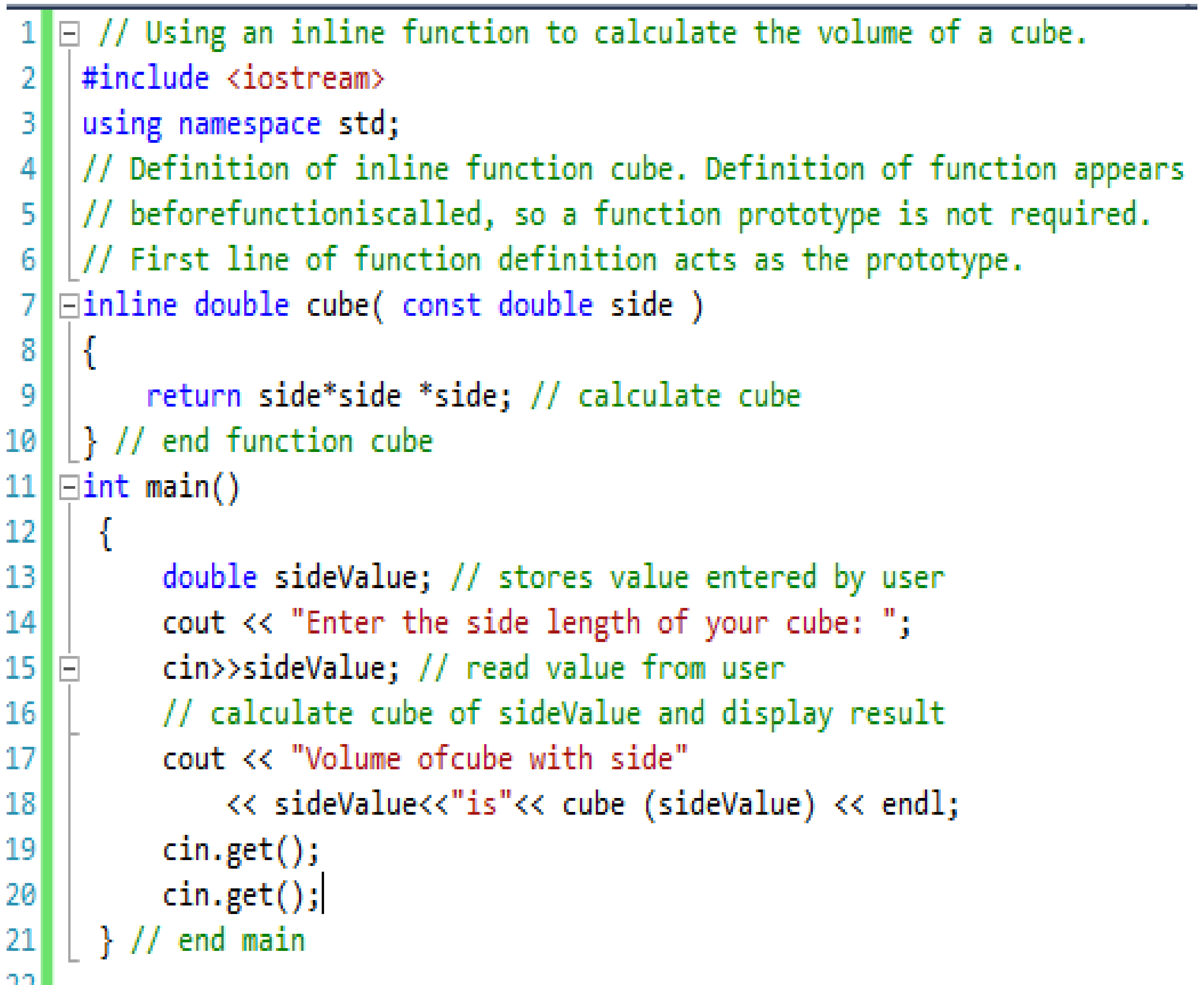
Functions and Introduction to Recursion

# FUNKSIONET PA PARAMETËR



```cpp
 // Functions that take no arguments.
#include <iostream>
using namespace std;

void function1(); // function that takes no arguments
void function2( void ); // function that takes no arguments

int main()
{
    function1(); // call function1 with no arguments
    function2(); // call function2 with no arguments

} // end main

 // function1 uses an empty parameter list to specify that
 // the function receives no arguments
void function1()
{

    cout << "function1 takes no arguments" << endl;
} // end function1
// function2 uses avoid parameter list to specify that
// the function receives no arguments
void function2()
{
    cout << "function2 also takes no arguments" << endl;

} // end function2
```

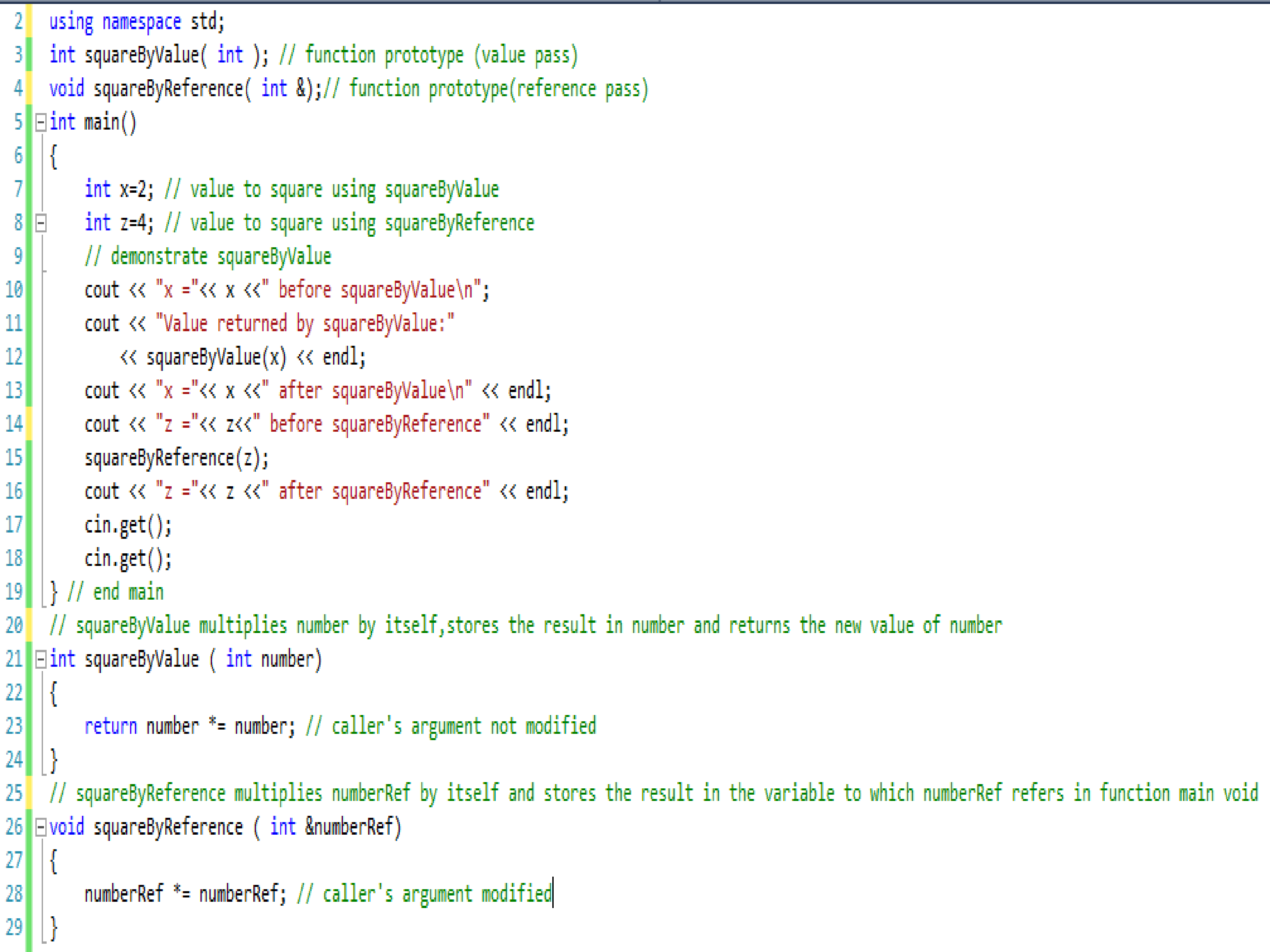
# FUNKSIONI INLINE



```cpp
// Using an inline function to calculate the volume of a cube.
#include <iostream>
using namespace std;
// Definition of inline function cube. Definition of function appears
// beforefunctioniscalled, so a function prototype is not required.
// First line of function definition acts as the prototype.
inline double cube( const double side )
{
    return side*side *side; // calculate cube
} // end function cube
int main()
 {
     double sideValue; // stores value entered by user
     cout << "Enter the side length of your cube: ";
     cin>>sideValue; // read value from user
     // calculate cube of sideValue and display result
     cout << "Volume ofcube with side"
         << sideValue<<"is"<< cube (sideValue) << endl;
     cin.get();
     cin.get();
 } // end main
```

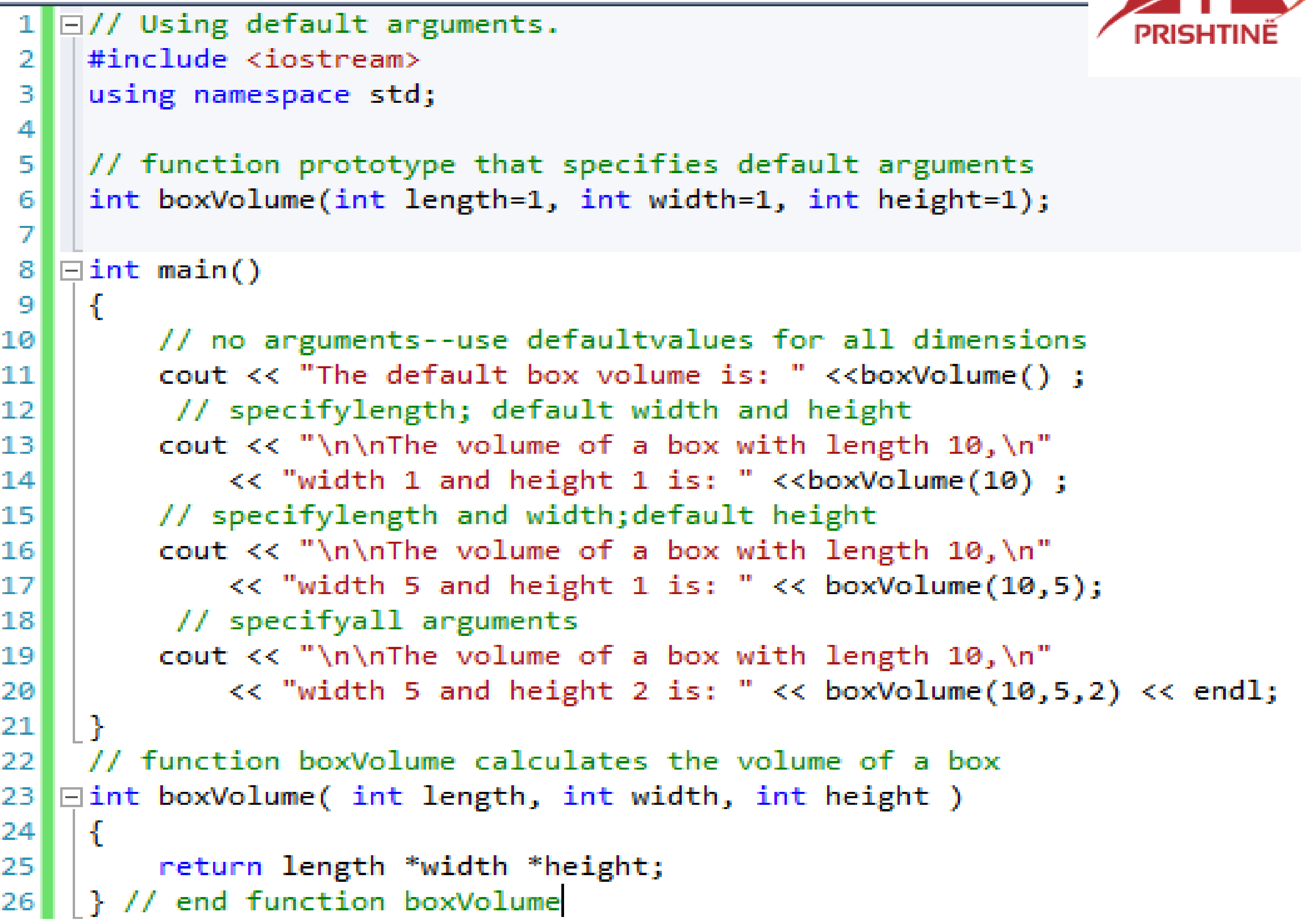
## REFERENCES AND REFERENCE PARAMETERS

- Comparing pass-by-value and pass-by-reference with references



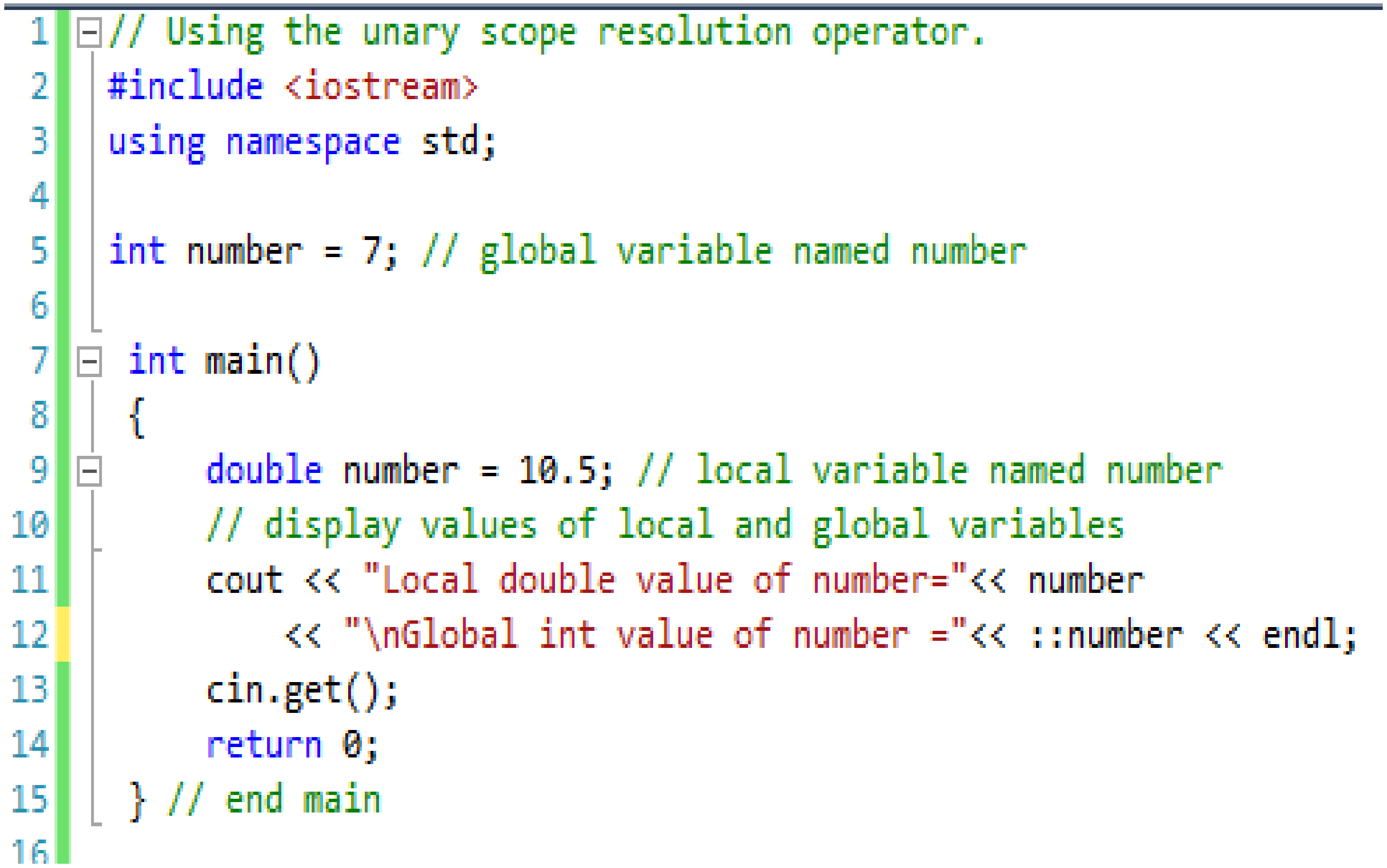
```cpp
using namespace std;
int squareByValue( int ); // function prototype (value pass)
void squareByReference( int &);// function prototype(reference pass)
int main()
{
    int x=2; // value to square using squareByValue
    int z=4; // value to square using squareByReference
    // demonstrate squareByValue
    cout << "x ="<< x <<" before squareByValue\n";
    cout << "Value returned by squareByValue:"
        << squareByValue(x) << endl;
    cout << "x ="<< x <<" after squareByValue\n" << endl;
    cout << "z ="<< z<<" before squareByReference" << endl;
    squareByReference(z);
    cout << "z ="<< z <<" after squareByReference" << endl;
    cin.get();
    cin.get();
} // end main
// squareByValue multiplies number by itself,stores the result in number and returns the new value of number
int squareByValue ( int number)
{
    return number *= number; // caller's argument not modified
}
// squareByReference multiplies numberRef by itself and stores the result in the variable to which numberRef refers in function main void
void squareByReference ( int &numberRef)
{
    numberRef *= numberRef; // caller's argument modified
}
```

# DEFAULT ARGUMENTS



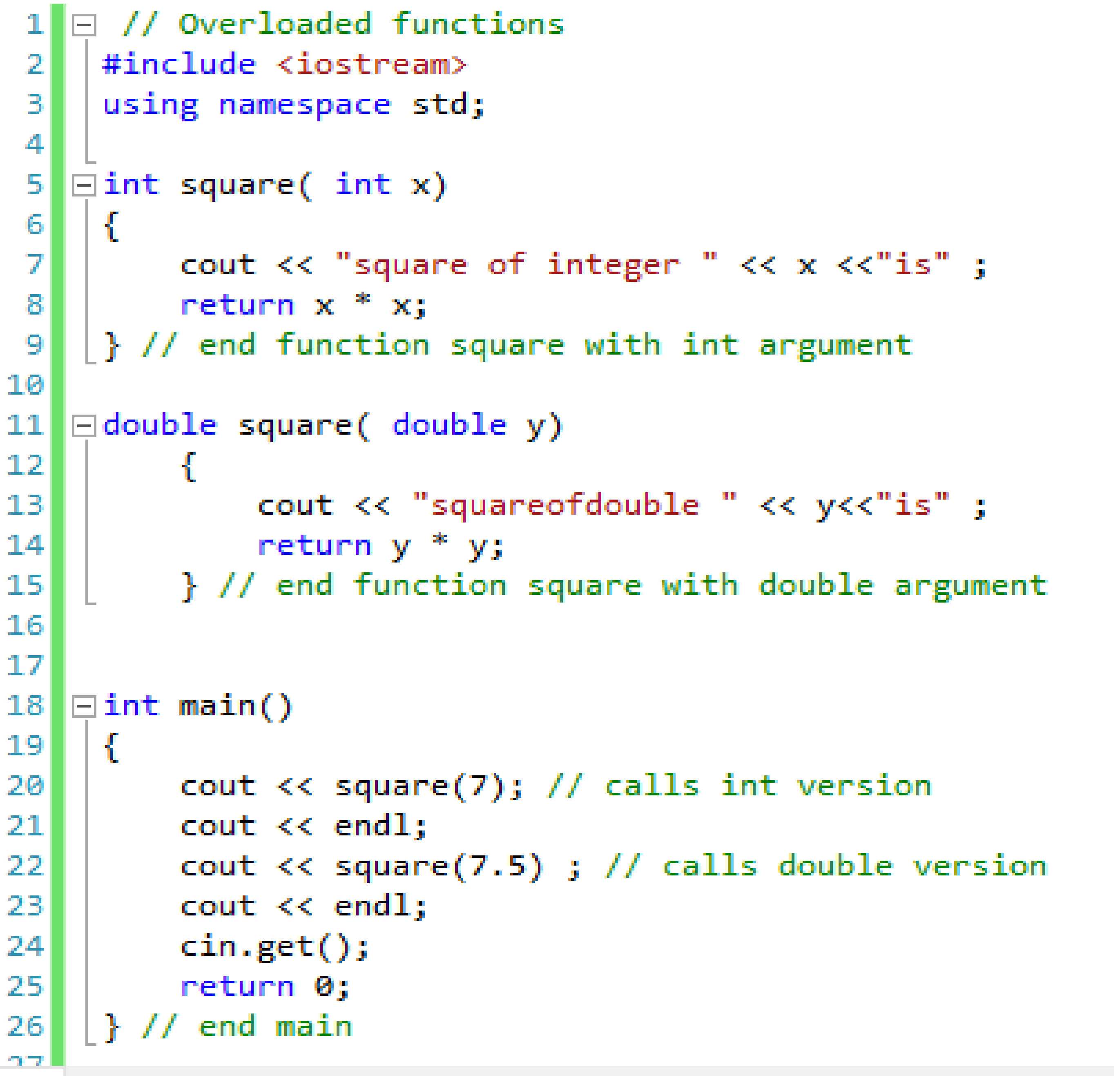
```cpp
// Using default arguments.
#include <iostream>
using namespace std;

// function prototype that specifies default arguments
int boxVolume(int length=1, int width=1, int height=1);

int main()
{
    // no arguments--use defaultvalues for all dimensions
    cout << "The default box volume is: " <<boxVolume() ;
     // specifylength; default width and height
    cout << "\n\nThe volume of a box with length 10,\n"
        << "width 1 and height 1 is: " <<boxVolume(10) ;
    // specifylength and width;default height
    cout << "\n\nThe volume of a box with length 10,\n"
        << "width 5 and height 1 is: " << boxVolume(10,5);
     // specifyall arguments
    cout << "\n\nThe volume of a box with length 10,\n"
        << "width 5 and height 2 is: " << boxVolume(10,5,2) << endl;
}
// function boxVolume calculates the volume of a box
int boxVolume( int length, int width, int height )
{
    return length *width *height;
} // end function boxVolume
```

## UNARY SCOPE RESOLUTION OPERATOR



```cpp
// Using the unary scope resolution operator.
#include <iostream>
using namespace std;

int number = 7; // global variable named number

 int main()
 {
     double number = 10.5; // local variable named number
     // display values of local and global variables
     cout << "Local double value of number="<< number
         << "\nGlobal int value of number ="<< ::number << endl;
     cin.get();
     return 0;
 } // end main
```

# FUNCTION OVERLOADING



```cpp
 // Overloaded functions
#include <iostream>
using namespace std;

int square( int x)
{
    cout << "square of integer " << x <<"is" ;
    return x * x;
} // end function square with int argument

double square( double y)
    {
        cout << "squareofdouble " << y<<"is" ;
        return y * y;
    } // end function square with double argument


int main()
{
    cout << square(7); // calls int version
    cout << endl;
    cout << square(7.5) ; // calls double version
    cout << endl;
    cin.get();
    return 0;
} // end main
```

```
char <= short <= int <= long <= long long
```

where:

```
char      >= 8 bits
short     >= 16 bits
int       >= 16 bits
long      >= 32 bits
long long >= 64 bits
```

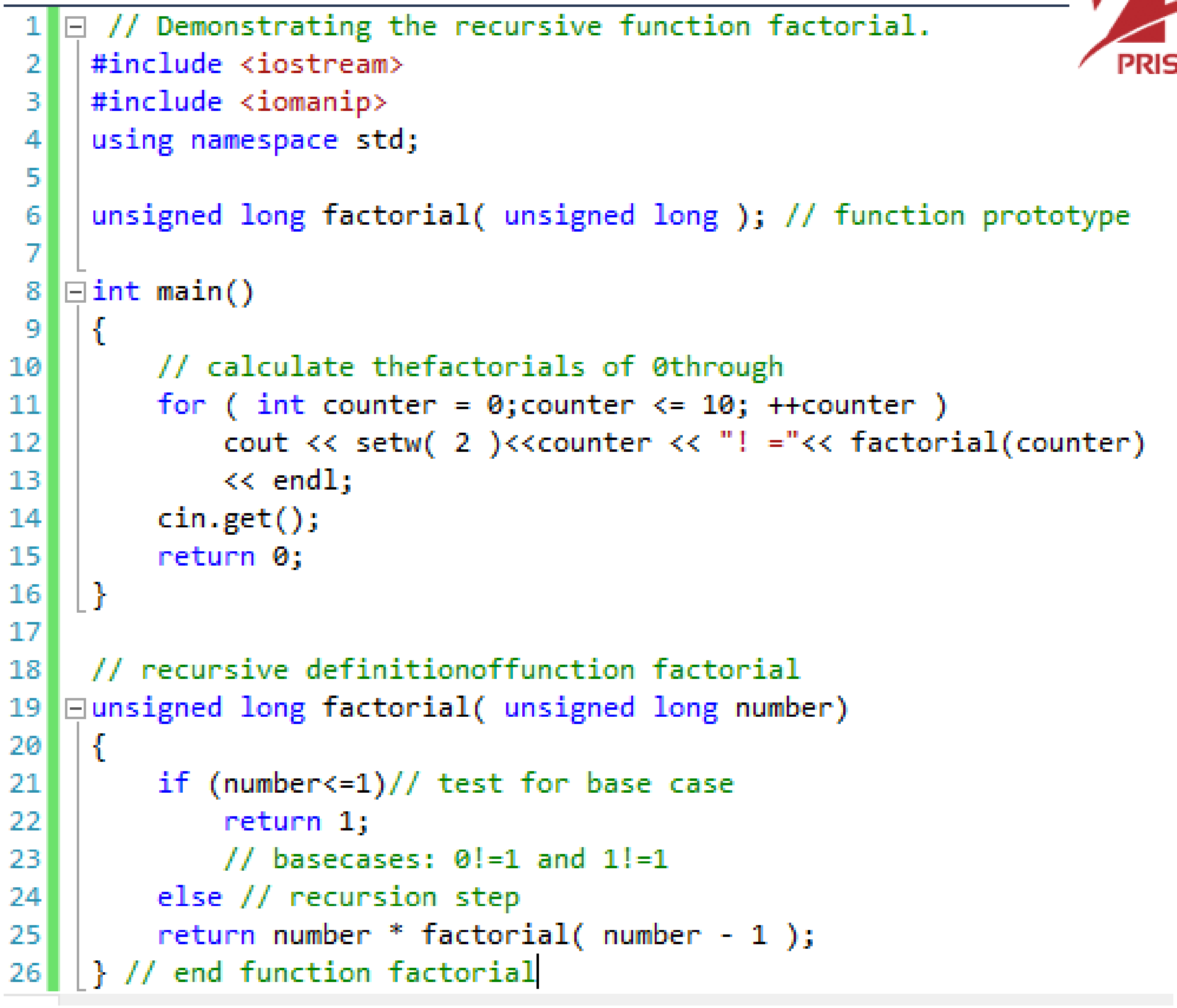
An Unsigned Variable Type of int can hold zero and positive numbers but a signed int holds negative, zero or positive numbers.

In 32 bits integers, an unsigned int has a range of 0 to $2^{32}$-1 = 0 to 4,294,967,295. While the signed version goes from -$2^{31}$-1 to $2^{31}$, i.e. –2,147,483,648 to 2,147,483,647. An int type in C, C++ and C# is signed by default.

| Operatori | Operacioni | Shembull | Rezultati |
|---|---|---|---|
| && | Konjuksioni, AND | `(x < 7) && (y ==5)` | true |
| \|\| | Disjunksioni, OR | `(x != 2) \|\| (x > 3)` | false |
| ! | Negacioni, NOT | `!(y > 4)` | false |

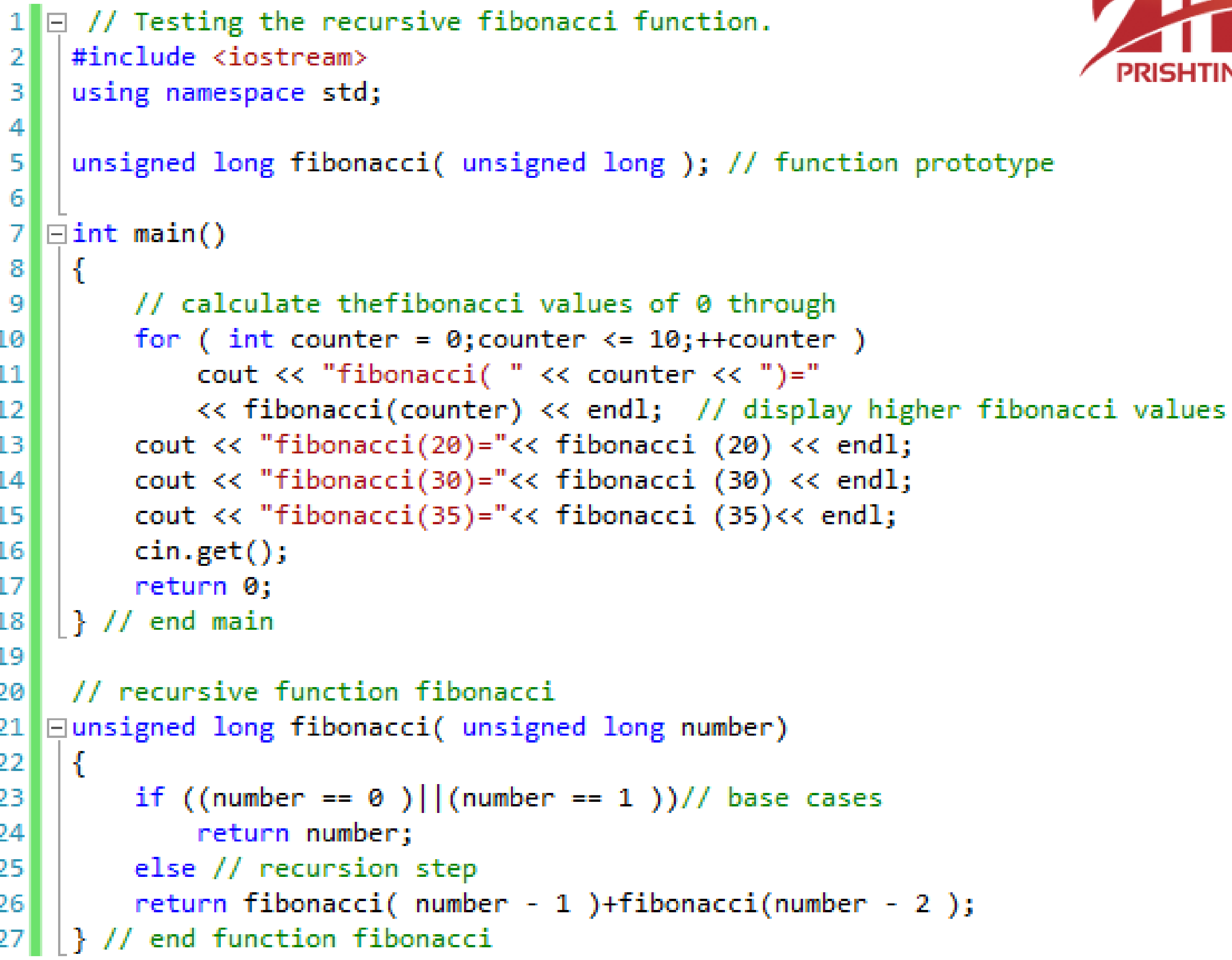
*Fig.2.21 Operatorët logjikë*

# RECURSIVE FAKTORIAL



```cpp
 // Demonstrating the recursive function factorial.
#include <iostream>
#include <iomanip>
using namespace std;

unsigned long factorial( unsigned long ); // function prototype

int main()
{
    // calculate thefactorials of 0through
    for ( int counter = 0;counter <= 10; ++counter )
        cout << setw( 2 )<<counter << "! ="<< factorial(counter)
        << endl;
    cin.get();
    return 0;
}

// recursive definitionoffunction factorial
unsigned long factorial( unsigned long number)
{
    if (number<=1)// test for base case
        return 1;
        // basecases: 0!=1 and 1!=1
    else // recursion step
    return number * factorial( number - 1 );
} // end function factorial
```

# RECURSIVE FIBONACCI



```cpp
 // Testing the recursive fibonacci function.
#include <iostream>
using namespace std;

unsigned long fibonacci( unsigned long ); // function prototype

int main()
{
    // calculate thefibonacci values of 0 through
    for ( int counter = 0;counter <= 10;++counter )
        cout << "fibonacci( " << counter << ")="
        << fibonacci(counter) << endl;  // display higher fibonacci values
    cout << "fibonacci(20)="<< fibonacci (20) << endl;
    cout << "fibonacci(30)="<< fibonacci (30) << endl;
    cout << "fibonacci(35)="<< fibonacci (35)<< endl;
    cin.get();
    return 0;
} // end main

// recursive function fibonacci
unsigned long fibonacci( unsigned long number)
{
    if ((number == 0 )||(number == 1 ))// base cases
        return number;
    else // recursion step
    return fibonacci( number - 1 )+fibonacci(number - 2 );
} // end function fibonacci
```

PRISHTINË