# CSE2001 OBJECT ORIENTED PROGRAMMING WITH C++

Shreyasi

# INTRODUCTION TO OBJECT ORIENTED APPROACH

## WHY OBJECT ORIENTED PROGRAMMING?

- Object oriented programming was used most frequently for two major types of applications:
  - Computer simulations
  - Graphical user interfaces

## CLASSES AND OBJECTS

- Class
  - Describes objects with common properties
  - User-defined data type with a set of data members and member functions.
- Attributes
  - Characteristics that define an object
  - Differentiate objects of same class
- Object
  - Specific instance of a class

# FEATURES OF OBJECT ORIENTED PROGRAMMING

- Encapsulation
  - Hides internal values and methods from outside sources
  - Provides security
  - Keeps data and methods safe from unintended changes
- Inheritance
  - Classes can inherit properties from existing classes.
  - Classes can share attributes and methods of existing classes and also get customized with more specific features
  - Helps you understand real-world objects
- Polymorphism
  - Means “Many forms”
  - Allows for different implementations of the same method name.

## DATA ABSTRACTION

- It essentially involves hiding the internal implementation details and revealing only the essential information.
- For example, if you take the example of a car, you know that to stop the car you need to apply brakes, but you are not fully familiar with the implementation of the braking mechanism. So the complexity of the process is not visible to you.

## MERITS OF OOP

- Improved productivity in software development – it provides improved productivity due to the modularity and extensibility of design.
- Due to reusability, it also enables faster development.
- The overall cost of development also reduces due to the reusability feature.

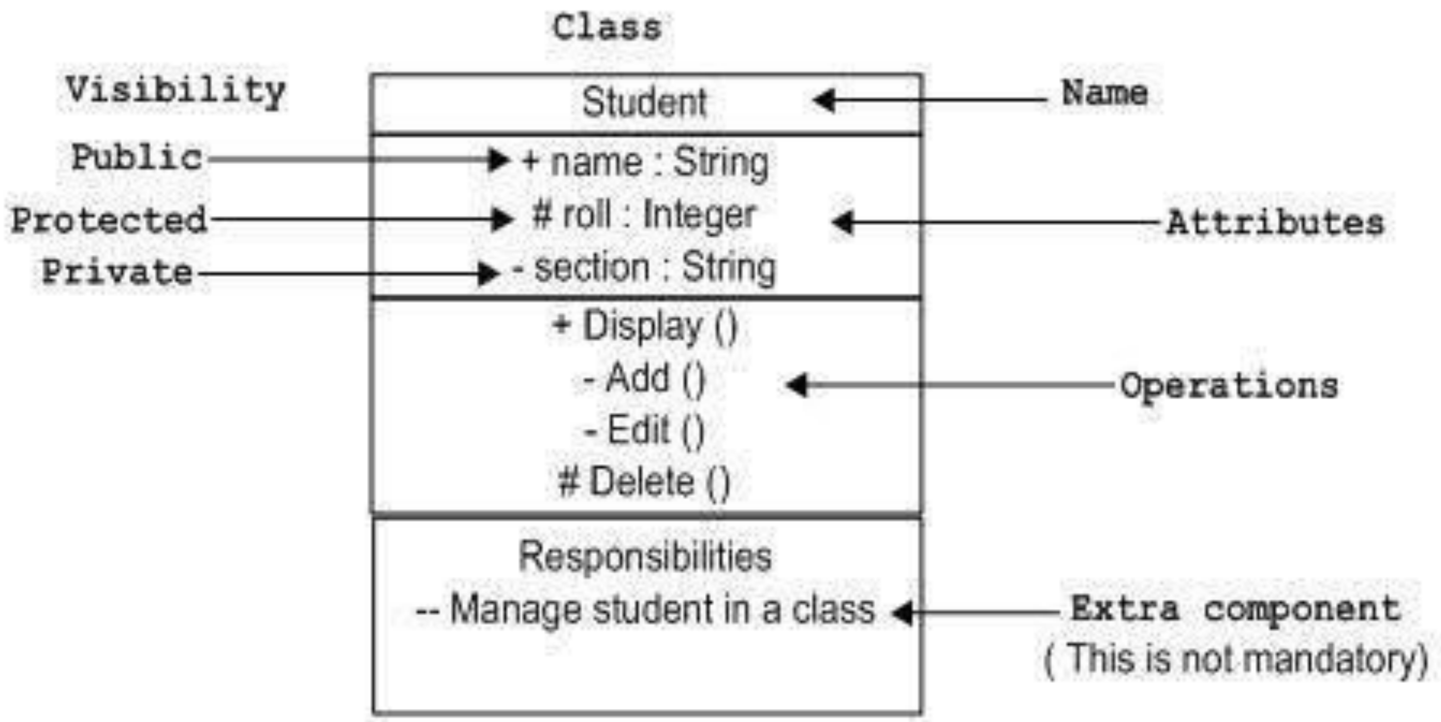
## DEMERITS OF OOP

- Object-oriented programs can be complex and thus may involve a steep learning curve.
- The program size may be larger.
- It may not be a suitable approach for certain problem statements.
- As it can involve more instructions as compared to a procedural program, the program may tend to be slower.

## UML

- It stands for Unified Modeling Language.
- It is the industry standard graphical language for modeling object-oriented systems.
- Each class is modeled as a rectangle with three components.
- The top compartment contains the class's name centered horizontally and in boldface type.
- The middle compartment contains the class's attributes, which correspond to data members in C++.
- The bottom compartment contains the class's operations, which correspond to member functions in C++.

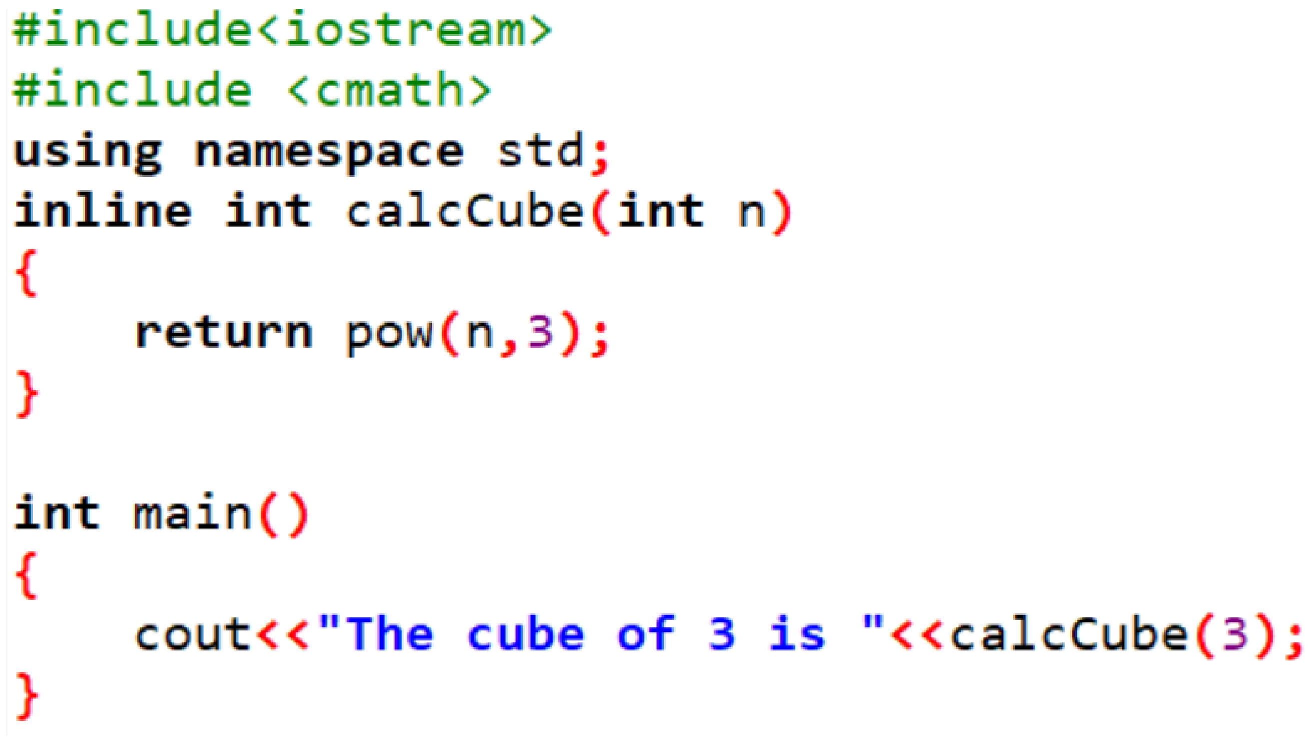
## CLASS DIAGRAM OF OOP



## INLINE FUNCTION

- Sometimes, the switching time from the calling function to the called function exceeds the time taken to execute the called function.
- Inline can be used to handle such function call overheads.
- It advises the compiler to generate a copy of the function's body code wherever the function is called to reduce the overhead.
- Providing the inline keyword is a request to the compiler and the compiler may choose to ignore it.

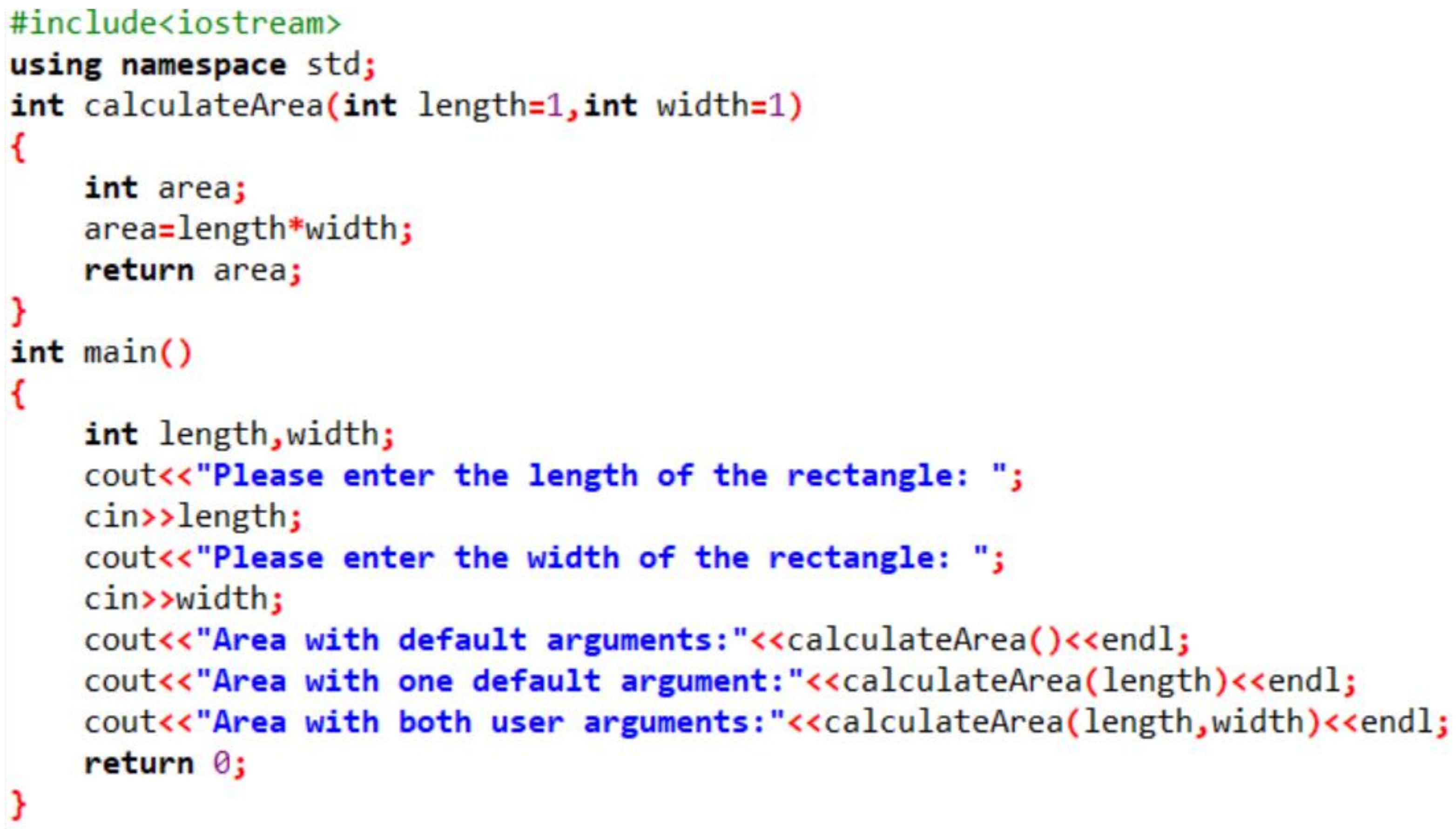
## INLINE FUNCTION(CONTD...)



```
#include<iostream>
#include <cmath>
using namespace std;
inline int calcCube(int n)
{
    return pow(n,3);
}

int main()
{
    cout<<"The cube of 3 is "<<calcCube(3);
}
```

## DEFAULT ARGUMENT FUNCTION

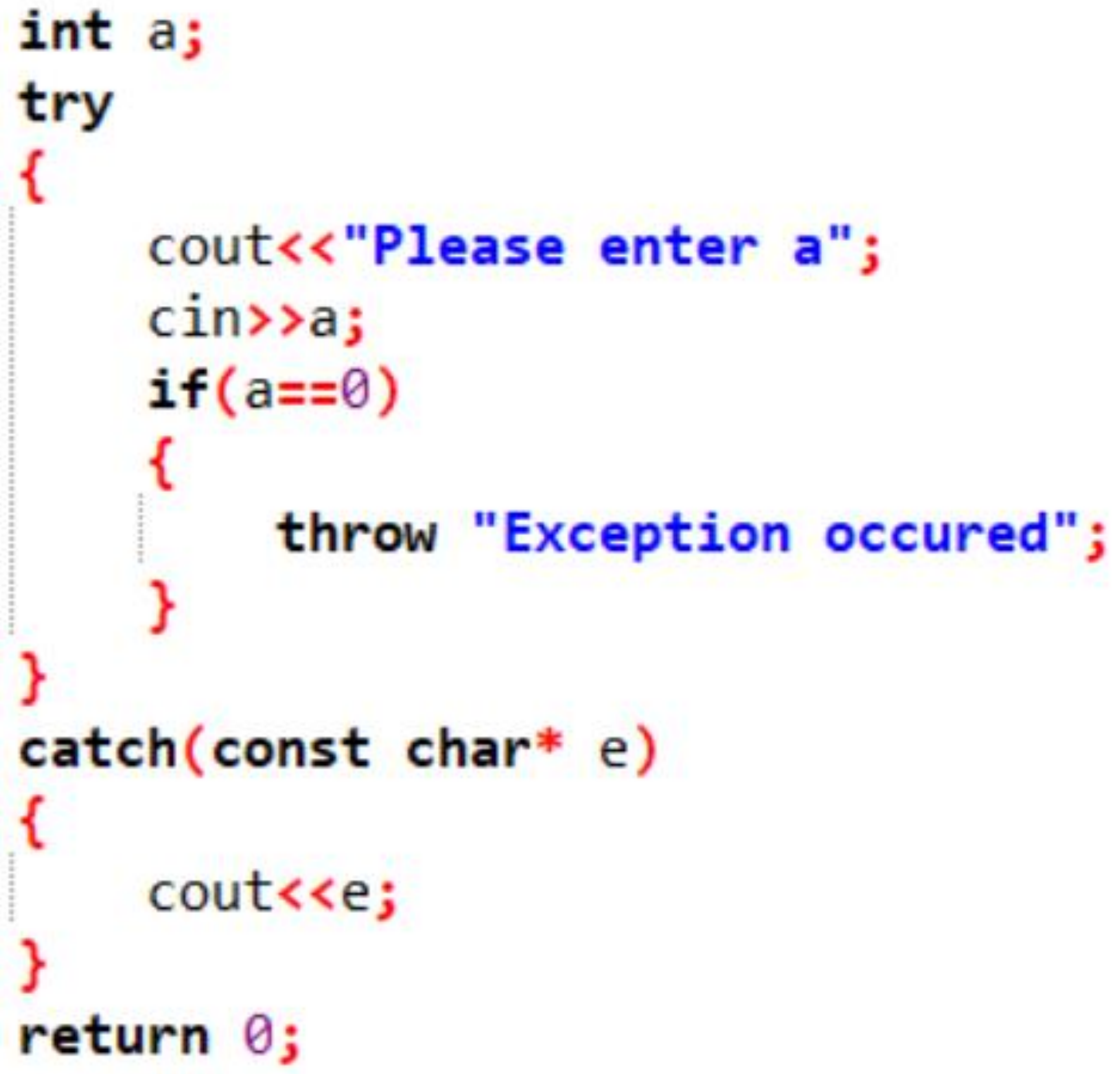
- If a program invokes a function repeatedly with the same argument value for a particular parameter, we can specify such parameters as default arguments.



```
#include<iostream>
using namespace std;
int calculateArea(int length=1,int width=1)
{
    int area;
    area=length*width;
    return area;
}
int main()
{
    int length,width;
    cout<<"Please enter the length of the rectangle: ";
    cin>>length;
    cout<<"Please enter the width of the rectangle: ";
    cin>>width;
    cout<<"Area with default arguments:"<<calculateArea()<<endl;
    cout<<"Area with one default argument:"<<calculateArea(length)<<endl;
    cout<<"Area with both user arguments:"<<calculateArea(length,width)<<endl;
    return 0;
}
```

## EXCEPTION HANDLING

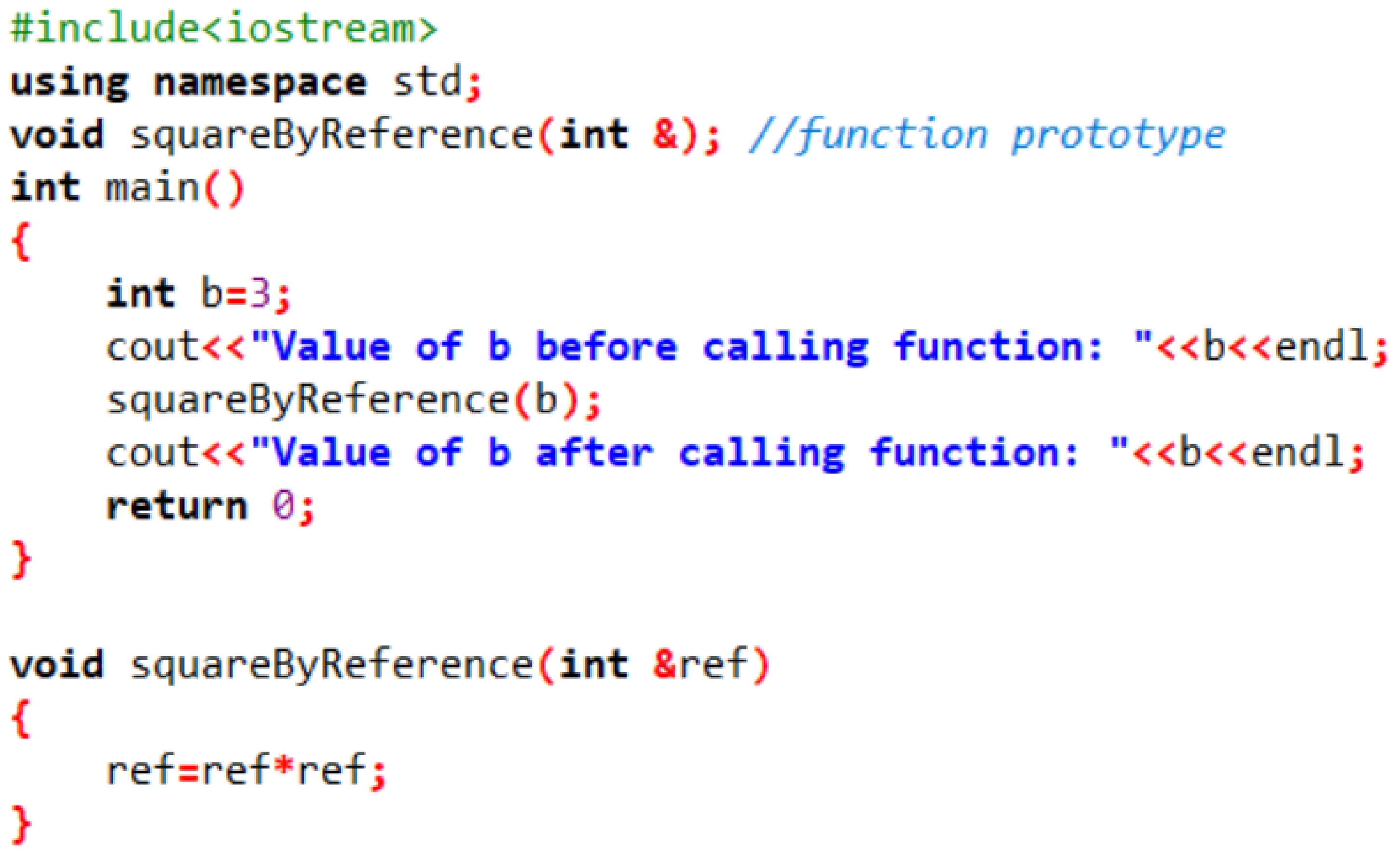
- It provides a mechanism for handling errors.



```
int a;
try
{
    cout<<"Please enter a";
    cin>>a;
    if(a==0)
    {
        throw "Exception occured";
    }
}
catch(const char* e)
{
    cout<<e;
}
return 0;
```

## PASS BY REFERENCE

- When an argument is passed by reference, any change made to it in the called function will get reflected in the calling function too.



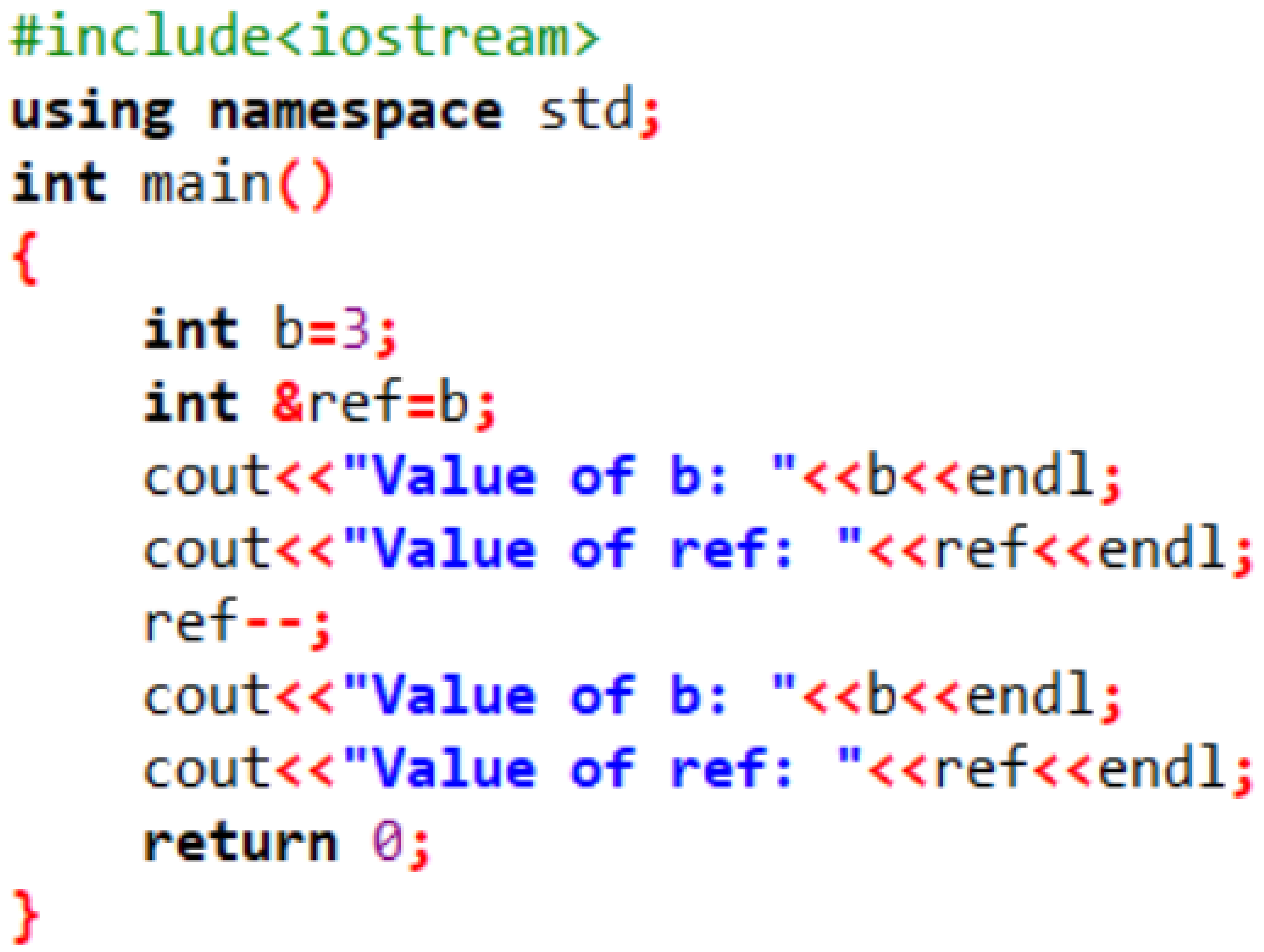
```cpp
#include<iostream>
using namespace std;
void squareByReference(int &); //function prototype
int main()
{
    int b=3;
    cout<<"Value of b before calling function: "<<b<<endl;
    squareByReference(b);
    cout<<"Value of b after calling function: "<<b<<endl;
    return 0;
}

void squareByReference(int &ref)
{
    ref=ref*ref;
}
```

## FUNCTION RETURNING REFERENCE

- Functions can also return references.

```
#include<iostream>
using namespace std;
/*This function returns a reference to the index
element of the array*/
int& getElement(int arr[10],int index)
{
    return arr[index];
}
int main()
{
    int arr[10];
    getElement(arr,3)=5;
    cout<<arr[3]<<endl;
    return 0;
}
```

## INDEPENDENT REFERENCE



```
#include<iostream>
using namespace std;
int main()
{
    int b=3;
    int &ref=b;
    cout<<"Value of b: "<<b<<endl;
    cout<<"Value of ref: "<<ref<<endl;
    ref--;
    cout<<"Value of b: "<<b<<endl;
    cout<<"Value of ref: "<<ref<<endl;
    return 0;
}
```