COURSE CODE: CSE 205
COURSE TITLE: OBJECT ORIENTED PROGRAMMING

Prepared by- ***Sumaiya Afroz Mila***

Inline Function

How function call works

- Argument of the function is stored in stack
- Control transferred to the func being called.
- CPU then executes the functions code
- Stores the return value in a predefined memory register
- Control is returned to the calling function
- overhead of the function call increases if the function is called a lot of time

```
int odd(int x)
{
        return x%2;
}
int main()
{       int result;
        for(int  i=0; i<=10000; i++){
                result = odd(i) ;
                if(result==1)
                cout <<i<< "  is odd";
        }
}
```

Solution?

Inline Function - Inline function is a function that is expanded in line when it is called.

- When the inline function is called, whole code of the inline function gets inserted or substituted at the point of inline function call.
- This substitution is performed by the C++ compiler at compile time.

Inline function syntax

```
inline return_type function_name(parameters)
{
            // function code
}
```

Inline Function

```
inline int odd(int x)
{
        return x%2;
}
int main()
{       int result;
        for(int  i=0; i<=10000; i++){
                result = odd(i) ;
                if(result==1)
                cout <<i<< “  is odd”;
        }
}
```

Function statement is copied by the compiler

Advantages

- Function call overhead doesn’t occur. Much faster than normal functions when it is small
- It also saves the overhead of push/pop variables on the stack when function is called.
- It also saves overhead of a return call from a function.
- Increases locality of reference by utilizing instruction cache
- Once inlining is done, compiler can perform intra-procedural optimization if specified.
- Inline function may be useful (if it is small) for embedded systems because inline can yield less code than the function call preamble and return.

Disadvantages

- If Inline functions are too large and called too often, program grows larger and Overhead increases.
- If too many inline functions are used, codes become larger.

Therefore only short functions are declared as inline functions.

Restrictions

- An inline function must be defined before it is first called.
- Inline specifier is a request, not a command.
- If, for various reasons, the compiler is unable to fulfill the request, the function is compiled as a normal function and the inline request is ignored.
- Some more restrictions: functions should not consists of
  - Static variable
  - A loop statement
  - A switch or a goto statement
  - Recursive function

Example 1-Inlining Member Function

```
class samp
{
        int i, j;
public:
        initialValue (int a, int b);
        int divisible();
};
samp:: initialValue (int a, int b)
{
        i = a;
        j = b;

}
```

```
inline int samp::divisible(){
        return !(i%j);
}
int main()
{
        samp ob1(10, 2), ob2(10, 3);
        ob1. initialValue (10, 2);
        ob2. initialValue (10, 3);
        if(ob1.divisible())
                cout<<“10 is divisible by 2”;
}
```

Example 2

```
inline int min (int a, int b)
{
        return a<b ? a : b;
}
inline long min (long a, long b)
{
        return a<b ? a : b;
}
 inline double min (double a, double b)
{
        return a<b ? a : b;
}
```

```
int main()
{
    cout << min (-10, 10);
    cout << min (-10.01, 100.002);
    cout << min (-10L, 12L);

    return 0;
}
```

Automatic Inline Function

Automatic In-Lining

- If a member function’s definition is short enough, the definition can be included inside the class declaration. It causes the function to automatically become an in-line function.

- When a function is defined within a class declaration, the inline keyword is no longer necessary. However, it is not an error to use it in this situation.

Example 1

```
class samp
{
        int i, j;
public:
         initialValue (int a, int b);
        int divisible()
                {
                        return !(i%j);
                }
};
```

```
samp:: initialValue (int a, int b)
{
        i = a;
        j = b;

}
int main(){
        samp ob1;
        ob1. initialValue (10, 2);
        if(ob1.divisible())
                cout<<“10 is divisible by 2”;
}
```

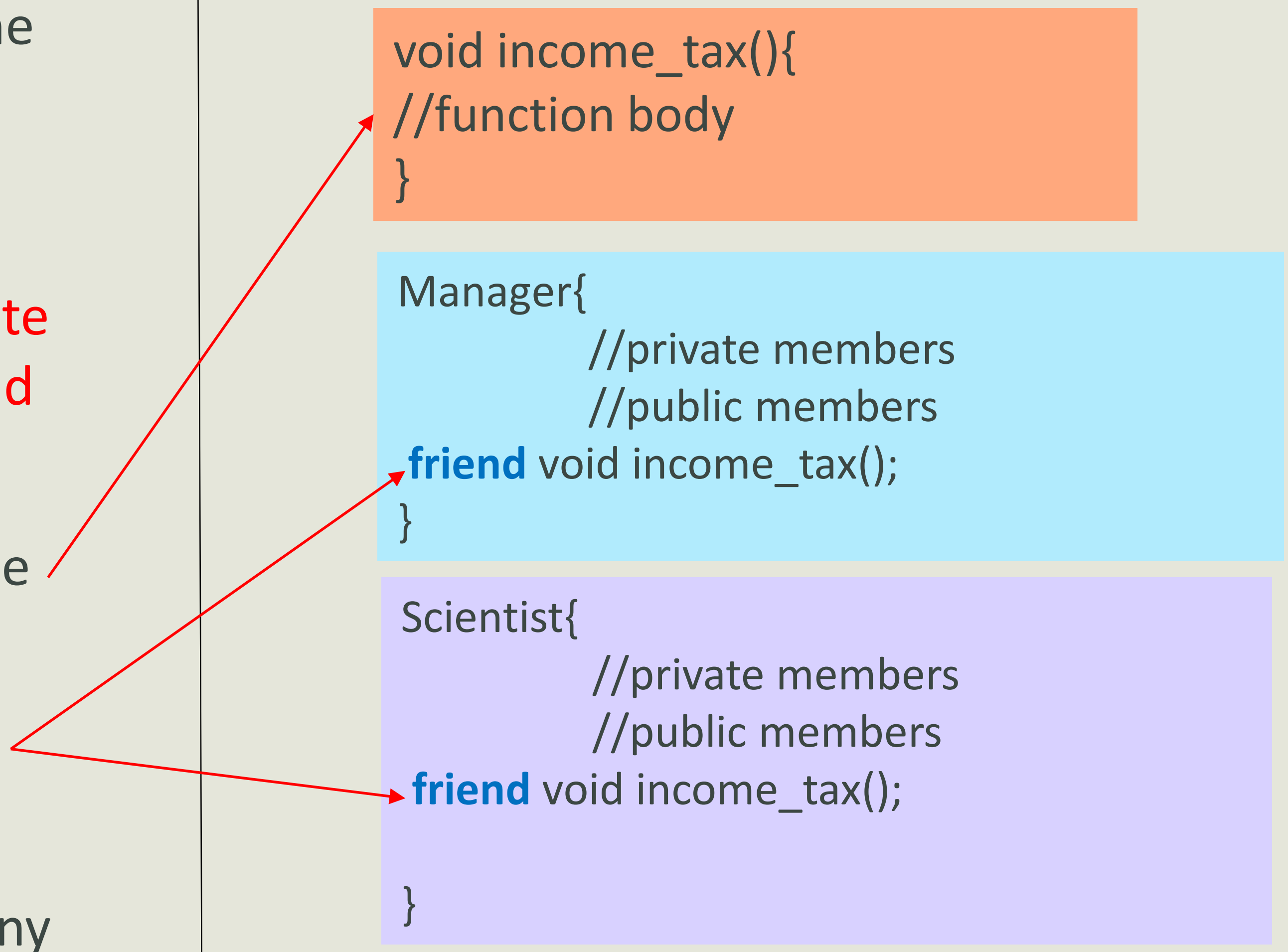
Friend Function

Introduction to Friend Function

- Can private members of a class be accessed from outside the class by a non member function of that class?
- Consider two classes, “**manager**” and “**scientists**”. You need a function **income_tax()** to perform some operations on objects of both the classes. What should you do?
  - Write **income_tax()** function in both the classes?
  - Is it redundant?
  - Is there any way to optimize the situation?
  - **“Friend Function”** is the solution

Friend Function

- Friend function is not a member of the class
- It is defined outside of the class
- A friend function can access the private members of the class, it is made friend to
- A friend function is defined exactly the same way as other functions
- Just the function is declared as friend inside the class
- A func can be declared as friends in any number of classes



```
void income_tax(){
//function body
}
```

```
Manager{
            //private members
            //public members
friend void income_tax();
}
```

```
Scientist{
            //private members
            //public members
friend void income_tax();


}
```

Special Characteristics of Friend Functions

- Is not a member of the class to which it is made friend with. Hence friend func is not in the scope of the class.
- As it is not a member, friend func can not be called using the object of that class.
- Unlike member functions, it can not access the members directly. Has to use an object name and the dot operator to access.

```
void income_tax(Manager ob){
            salary = 1000;
            ob.salary = 1000;
}
```

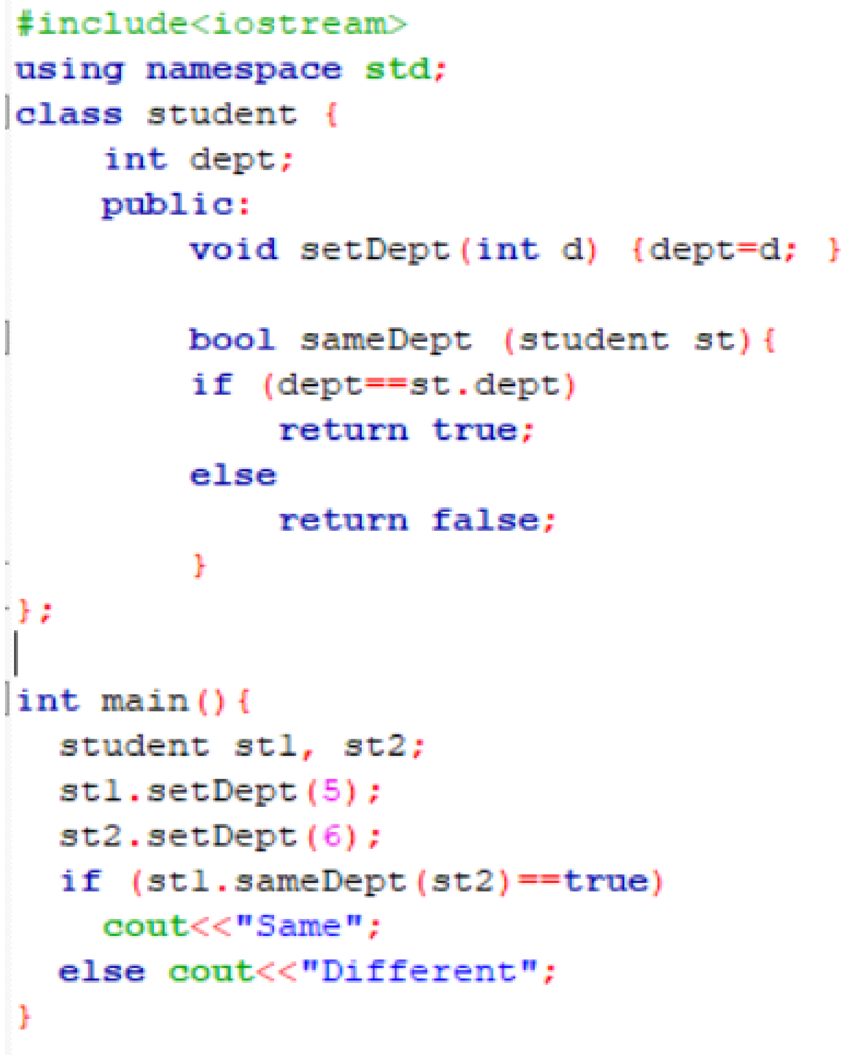
```
Manager{
private:
 double salary;
 friend void income_tax(); //Just declared as friend
                             // not member of the class
}
```

```
main{
            Manager ob;
            ob.income_tax();
            income_tax(ob);
}
```

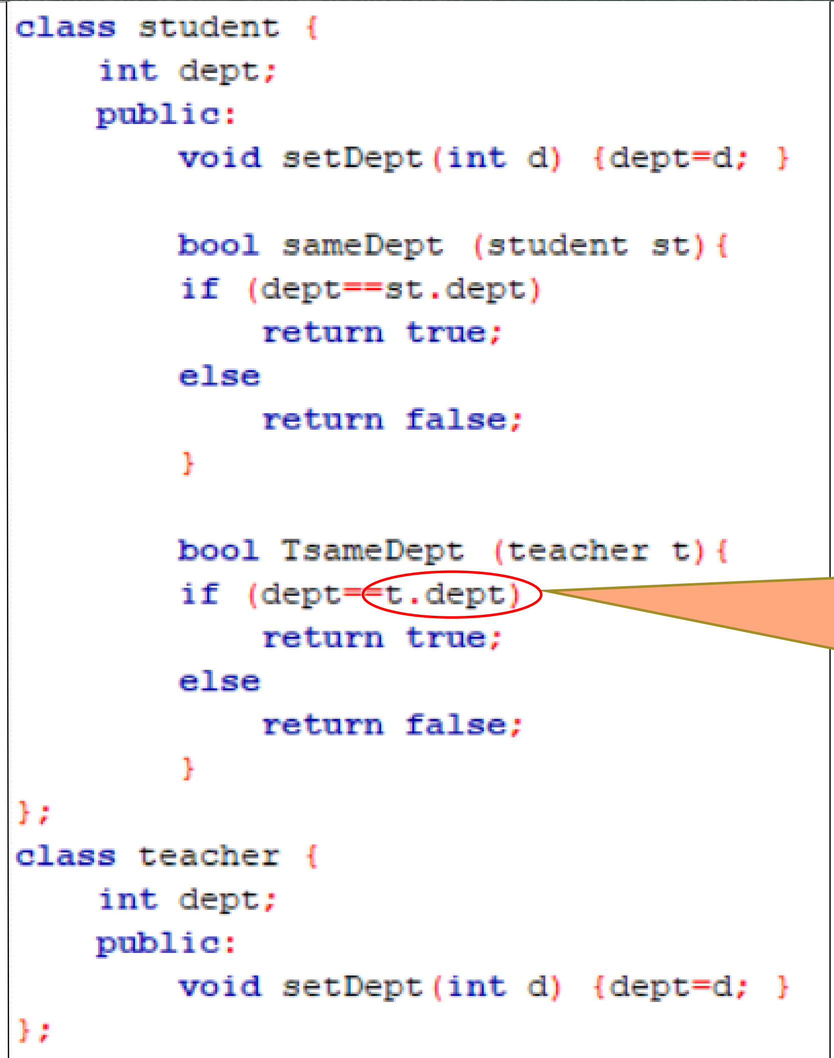
Special Characteristics of Friend Functions

- It can be declared either in the public or in the private part of the class without affecting its meaning
- Can be friend of more than one class
- Does not have “this” pointer
- Friend function is not inherited.

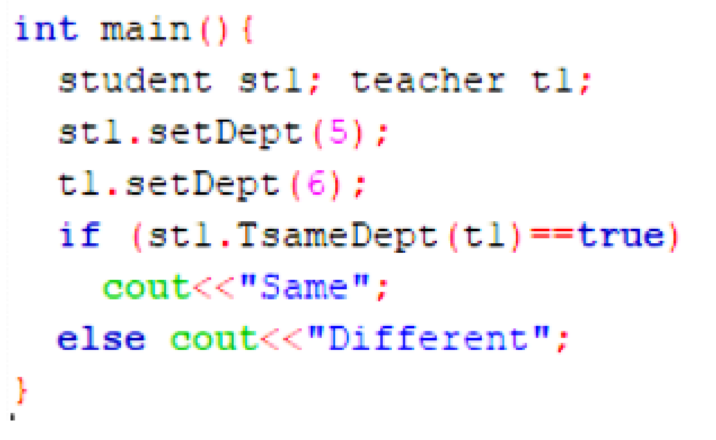
Friend Function Example



```
#include<iostream>
using namespace std;
class student {
     int dept;
     public:
          void setDept(int d) {dept=d; }

          bool sameDept (student st){
          if (dept==st.dept)
               return true;
          else
               return false;
          }
};

int main(){
  student st1, st2;
  st1.setDept(5);
  st2.setDept(6);
  if (st1.sameDept(st2)==true)
    cout<<"Same";
  else cout<<"Different";
}
```



```
class student {
     int dept;
     public:
          void setDept(int d) {dept=d; }

          bool sameDept (student st){
          if (dept==st.dept)
               return true;
          else
               return false;
          }

          bool TsameDept (teacher t){
          if (dept==t.dept)
               return true;
          else
               return false;
          }
};
class teacher {
     int dept;
     public:
          void setDept(int d) {dept=d; }
};
```



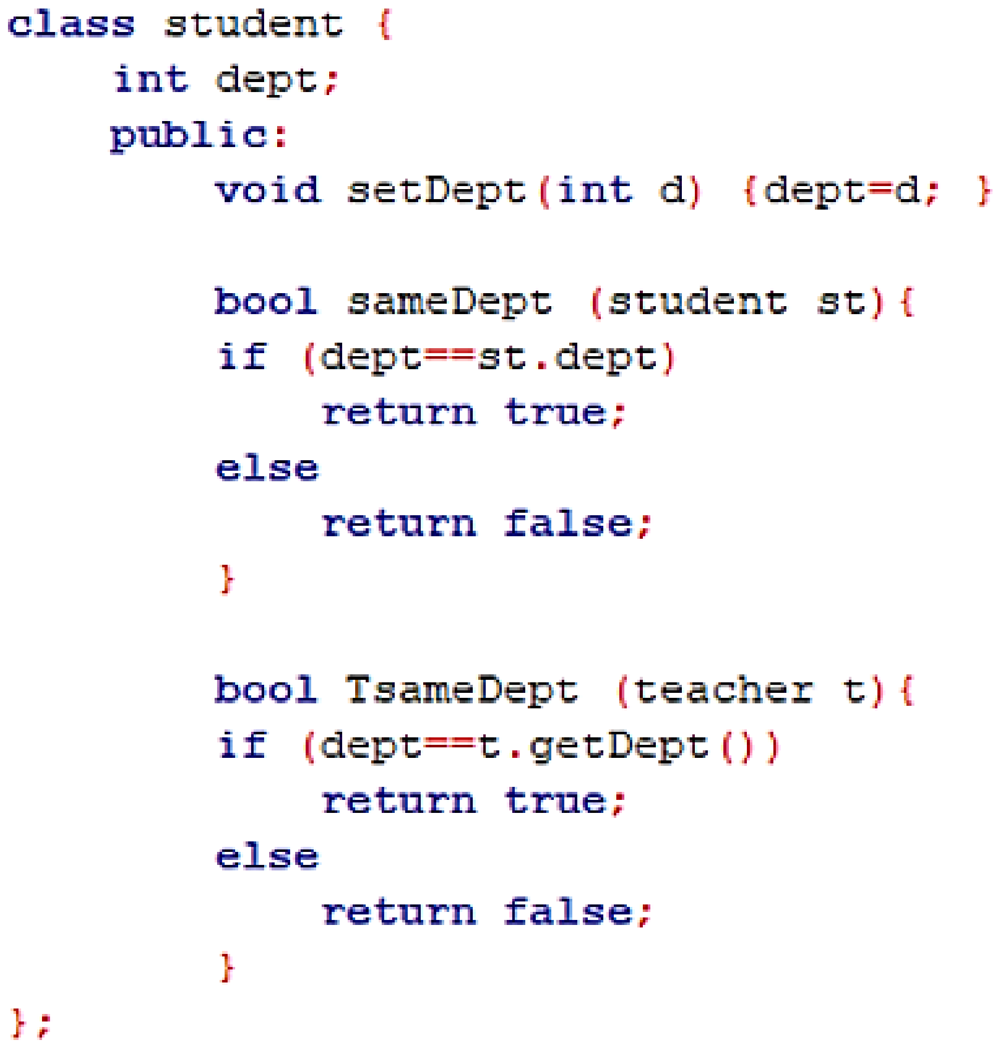
```
int main(){
  student st1; teacher t1;
  st1.setDept(5);
  t1.setDept(6);
  if (st1.TsameDept(t1)==true)
    cout<<"Same";
  else cout<<"Different";
}
```

ERROR

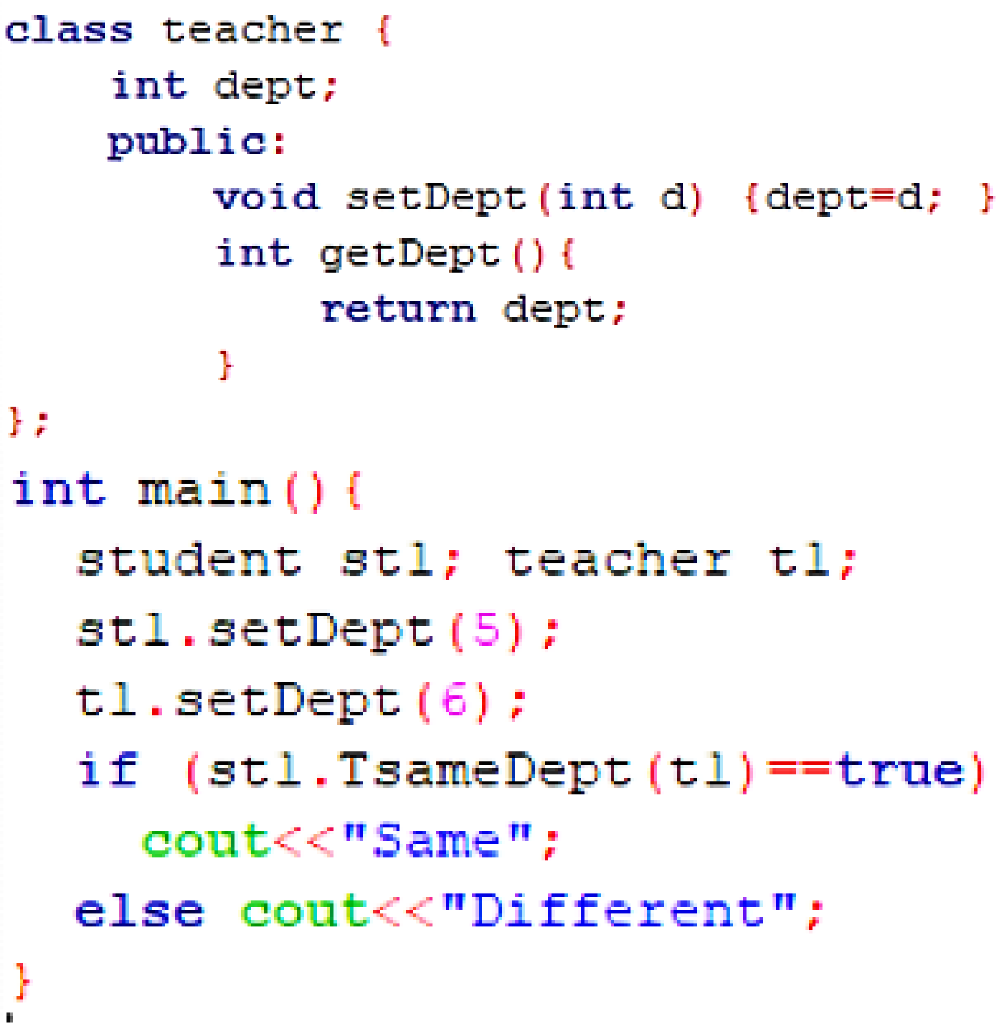
Dept is private in class teacher. Can not be accessed from nonmember of the class

Solution???

Solution 1

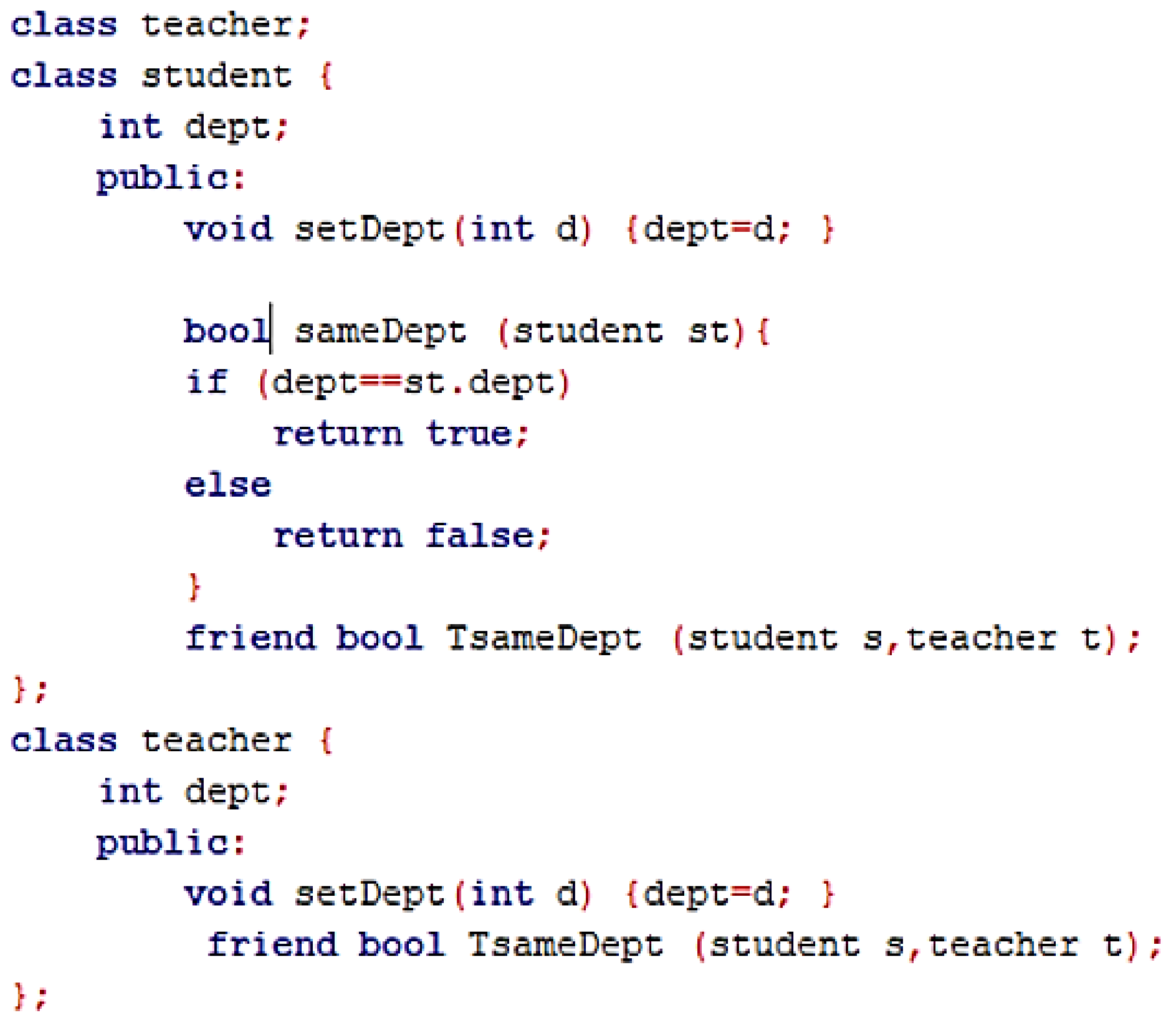


```
class student {
    int dept;
    public:
        void setDept(int d) {dept=d; }

        bool sameDept (student st){
        if (dept==st.dept)
            return true;
        else
            return false;
        }

        bool TsameDept (teacher t){
        if (dept==t.getDept())
            return true;
        else
            return false;
        }
};
```

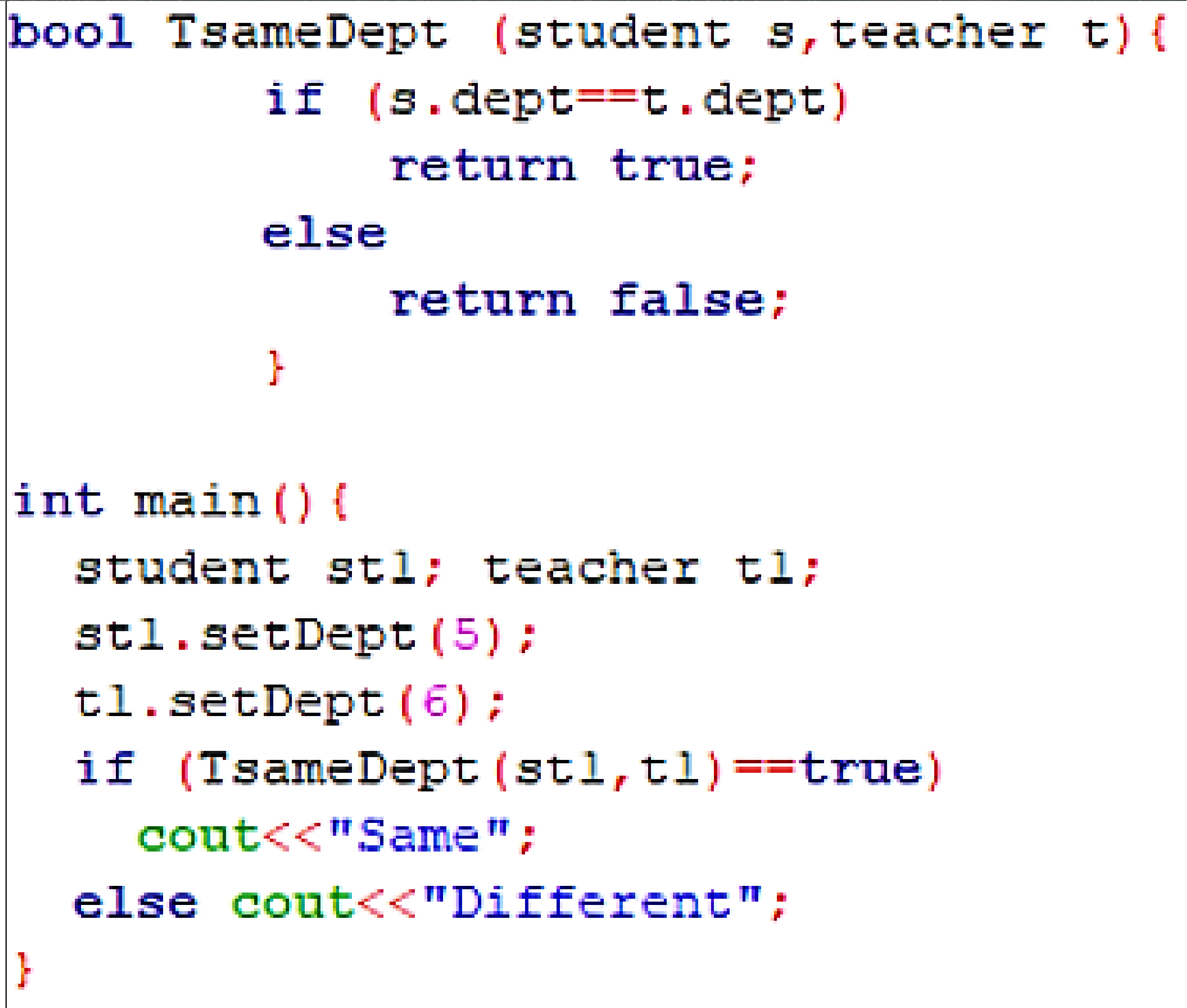


```
class teacher {
    int dept;
    public:
        void setDept(int d) {dept=d; }
        int getDept(){
            return dept;
        }
};
int main(){
  student st1; teacher t1;
  st1.setDept(5);
  t1.setDept(6);
  if (st1.TsameDept(t1)==true)
    cout<<"Same";
  else cout<<"Different";
}
```

Solution 2 using Friend Function

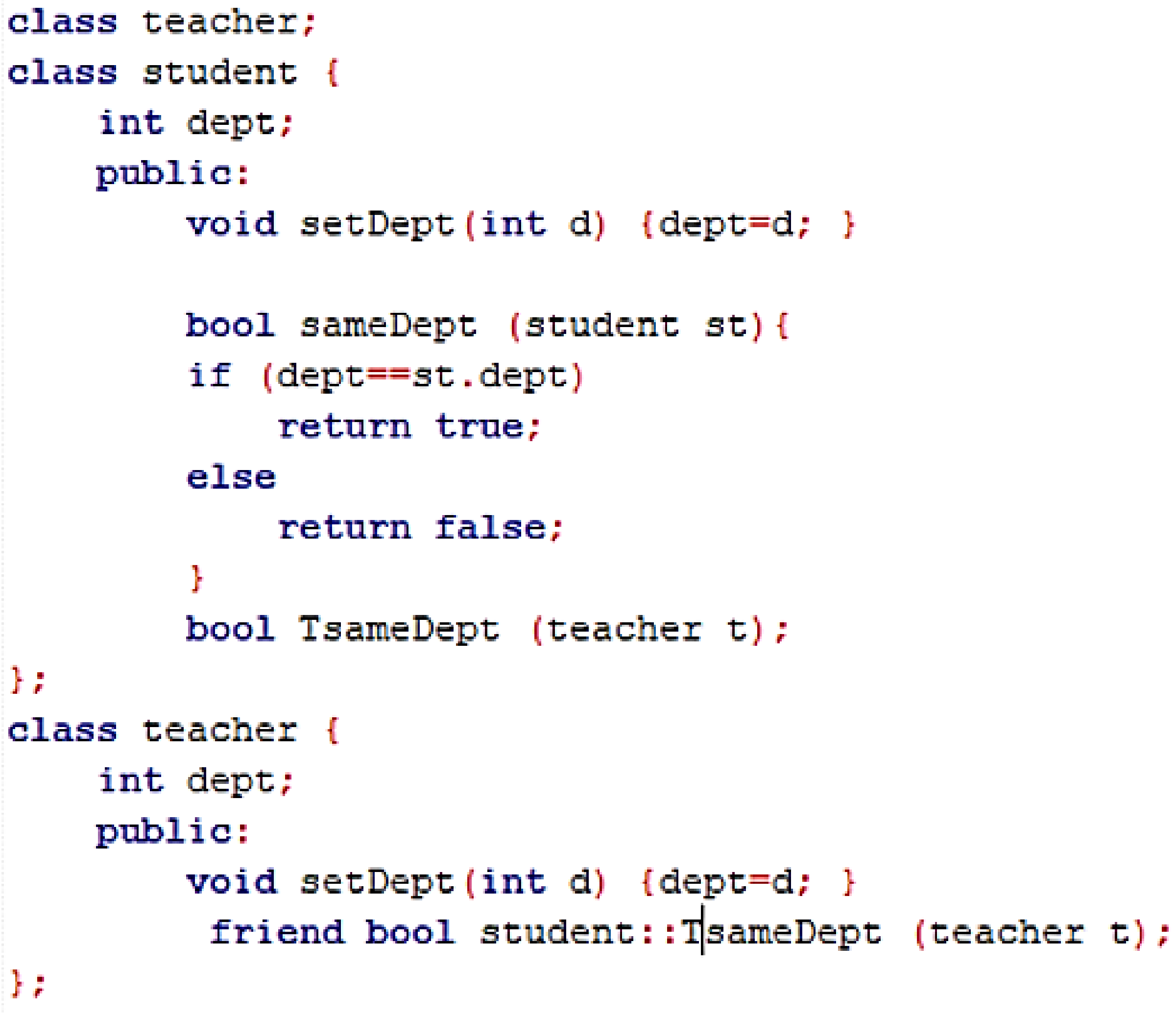


```
class teacher;
class student {
    int dept;
    public:
        void setDept(int d) {dept=d; }

        bool sameDept (student st){
        if (dept==st.dept)
            return true;
        else
            return false;
        }
        friend bool TsameDept (student s,teacher t);
};
class teacher {
    int dept;
    public:
        void setDept(int d) {dept=d; }
         friend bool TsameDept (student s,teacher t);
};
```

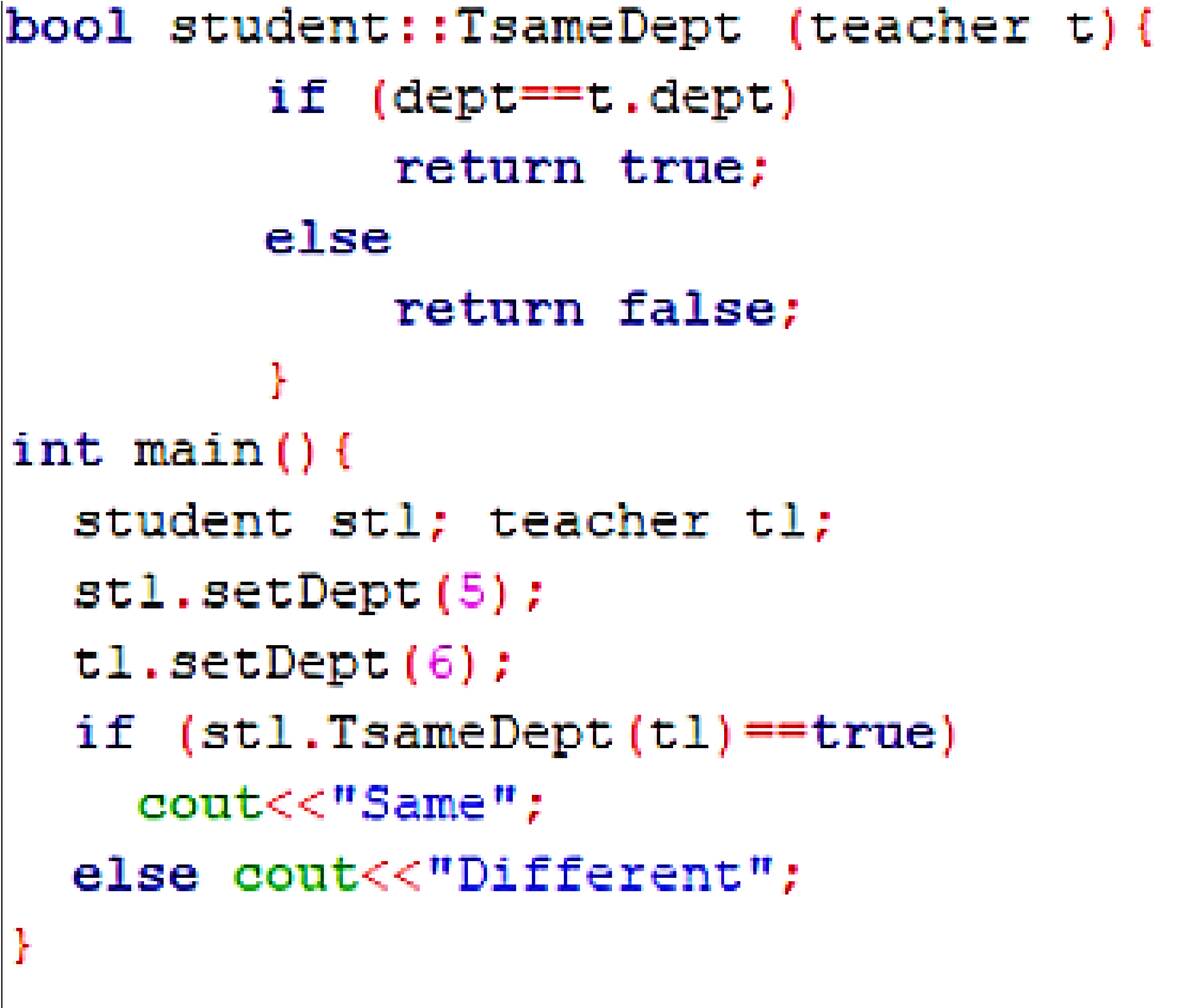


```
bool TsameDept (student s,teacher t){
        if (s.dept==t.dept)
            return true;
        else
            return false;
        }

int main(){
  student st1; teacher t1;
  st1.setDept(5);
  t1.setDept(6);
  if (TsameDept(st1,t1)==true)
    cout<<"Same";
  else cout<<"Different";
}
```

Solution 3 using Friend Function



```
class teacher;
class student {
    int dept;
    public:
        void setDept(int d) {dept=d; }

        bool sameDept (student st){
        if (dept==st.dept)
            return true;
        else
            return false;
        }
        bool TsameDept (teacher t);
};
class teacher {
    int dept;
    public:
        void setDept(int d) {dept=d; }
         friend bool student::TsameDept (teacher t);
};
```



```
bool student::TsameDept (teacher t){
        if (dept==t.dept)
            return true;
        else
            return false;
        }
int main(){
  student st1; teacher t1;
  st1.setDept(5);
  t1.setDept(6);
  if (st1.TsameDept(t1)==true)
    cout<<"Same";
  else cout<<"Different";
}
```

Thank you!