# Data Structures and Object Oriented Programming

## Lecture 2

Dr. Naveed Anwar Bhatti

**Webpage:** naveedanwarbhatti.github.io

# User-defined data types

# User defined data types

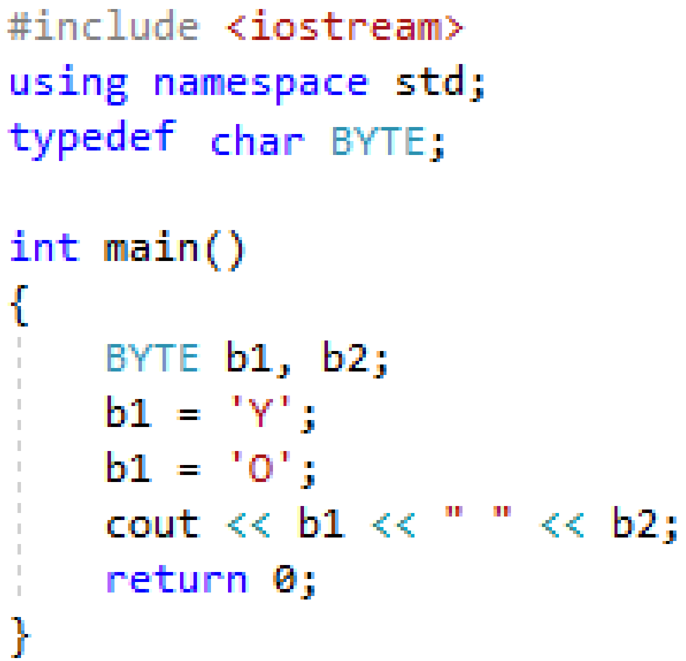
- The data types that are defined by the user are called the ***derived datatype (***or ***user-defined derived data type / user-defined data type)***

- These types include:

  ❑ Typedef
  ❑ Structure
  ❑ Union
  ❑ Class

# Typedef

- Allows you to define explicitly new data type names by using the keyword typedef
- Does not actually create a new data class, rather it defines a name for an existing type



```
#include <iostream>
using namespace std;
typedef  char BYTE;

int main()
{
    BYTE b1, b2;
    b1 = 'Y';
    b1 = 'O';
    cout << b1 << " " << b2;
    return 0;
}
```

**Output: Y O**

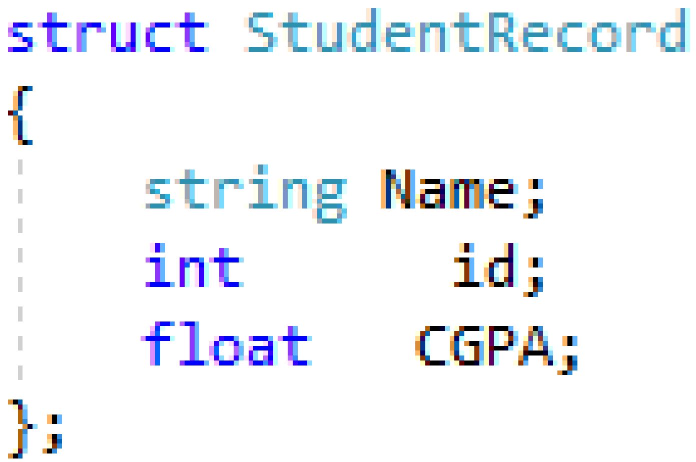
# Structures

- A ***struct*** (structure) is a collection of information of different data types (heterogeneous). The fields of a struct are referred to as ***members***.

- Defining a Structure:

```
struct StructName
{
    dataType memberName;
    …
    …
};
```

**Example:**

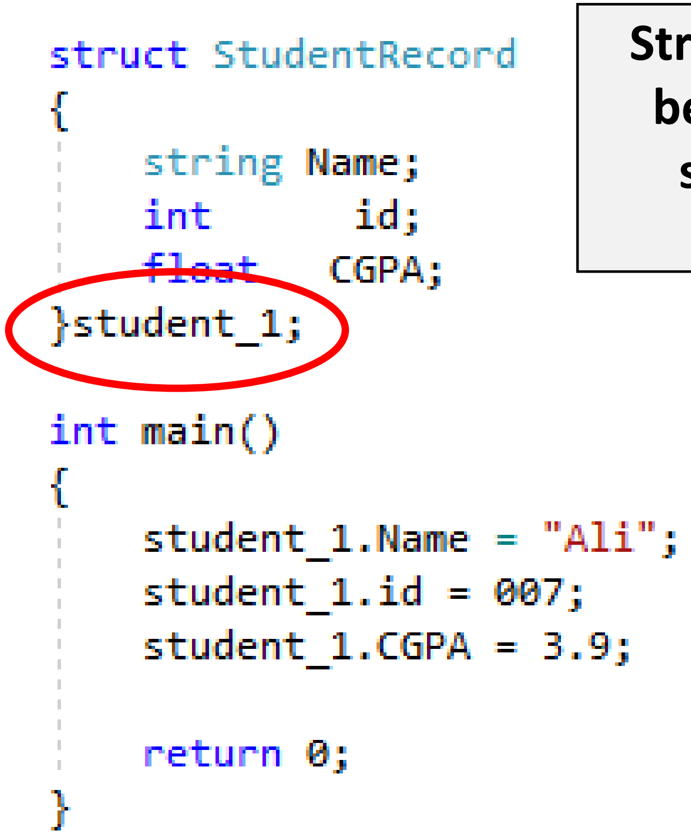


```
struct StudentRecord
{
    string Name;
    int        id;
    float    CGPA;
};
```

# Structures

- Two ways to create instance of Structure and accessing the Data Members

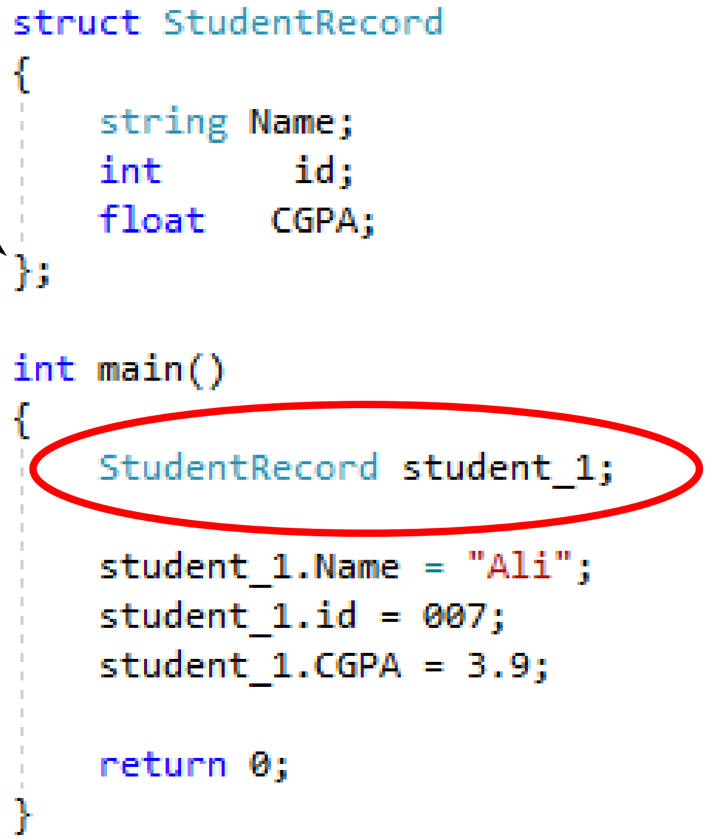
## Option 1



```
struct StudentRecord
{
    string Name;
    int        id;
    float    CGPA;
}student_1;

int main()
{
    student_1.Name = "Ali";
    student_1.id = 007;
    student_1.CGPA = 3.9;

    return 0;
}
```

**Structure definition must be followed either by a semicolon or a list of declarations**

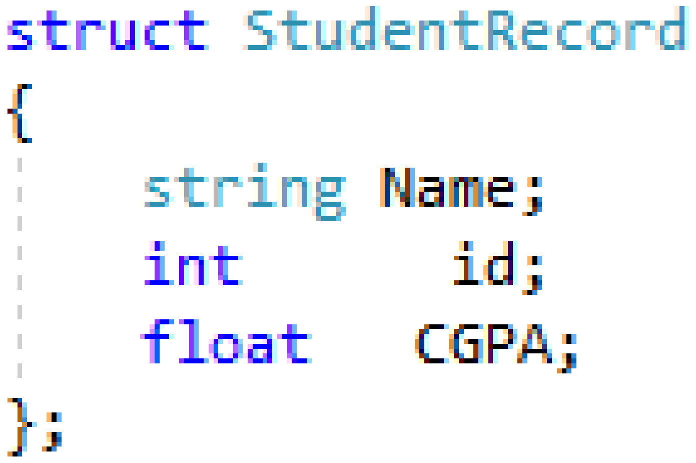
## Option 2



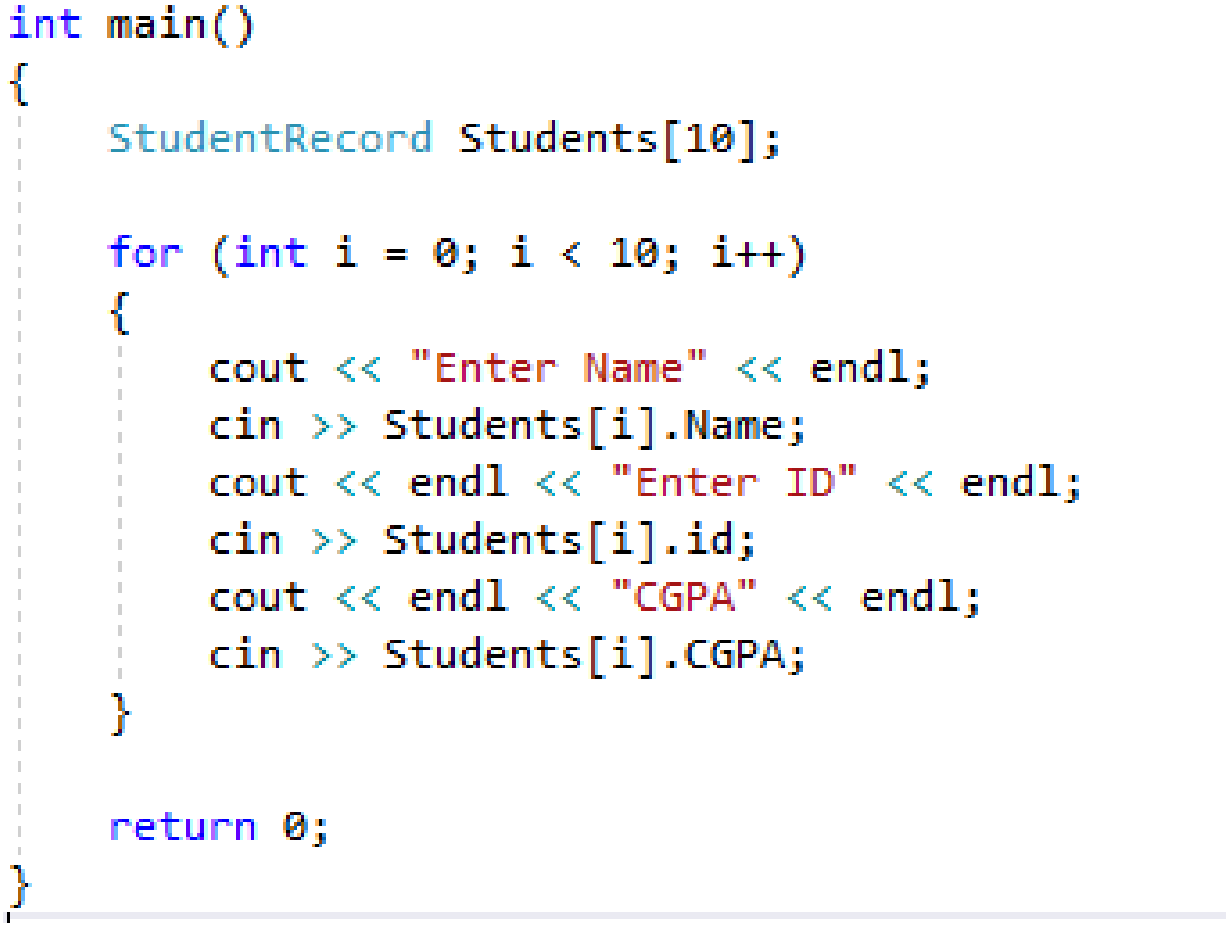
```
struct StudentRecord
{
    string Name;
    int        id;
    float    CGPA;
};

int main()
{
    StudentRecord student_1;

    student_1.Name = "Ali";
    student_1.id = 007;
    student_1.CGPA = 3.9;

    return 0;
}
```

# Structures (Recap)

- Exercise: Create an array of “*StudentRecord*” structure and insert data of 10 students in it.



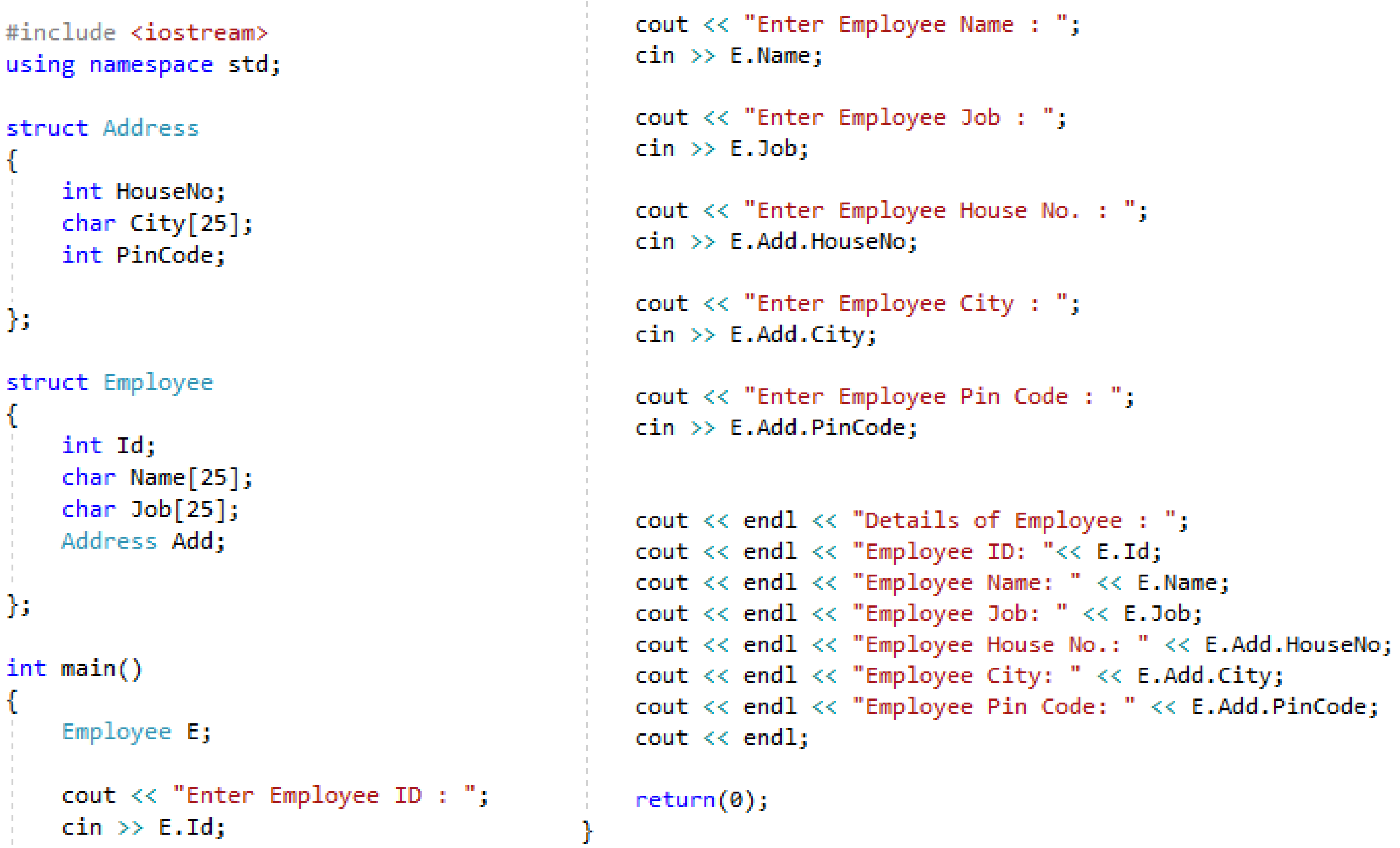
```cpp
struct StudentRecord
{
    string Name;
    int      id;
    float   CGPA;
};
```



```cpp
int main()
{
    StudentRecord Students[10];

    for (int i = 0; i < 10; i++)
    {
        cout << "Enter Name" << endl;
        cin >> Students[i].Name;
        cout << endl << "Enter ID" << endl;
        cin >> Students[i].id;
        cout << endl << "CGPA" << endl;
        cin >> Students[i].CGPA;
    }

    return 0;
}
```

# Nested Structures (Recap)

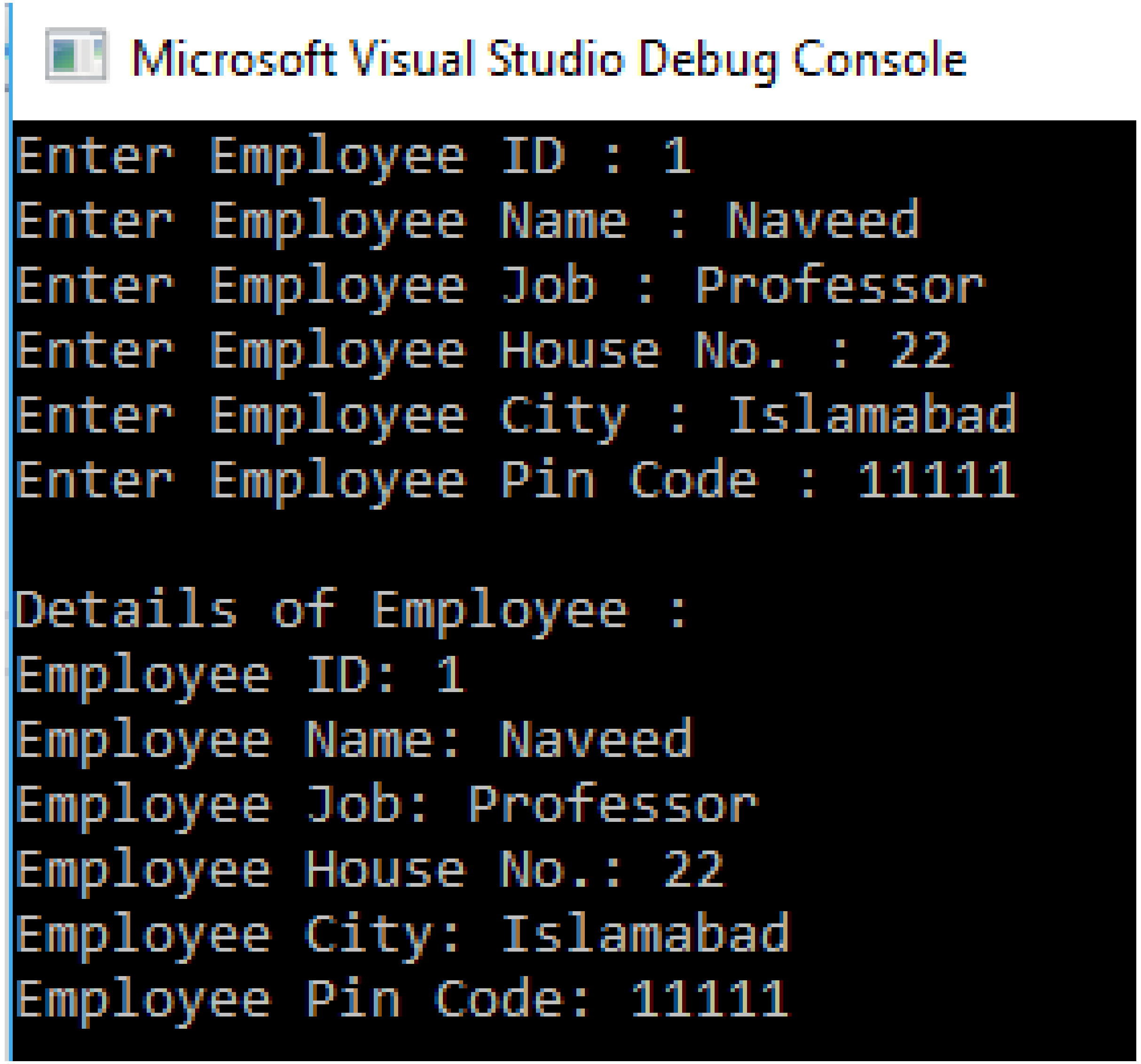
- Example



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

struct Address
{
    int HouseNo;
    char City[25];
    int PinCode;

};

struct Employee
{
    int Id;
    char Name[25];
    char Job[25];
    Address Add;

};

int main()
{
    Employee E;

    cout << "Enter Employee ID : ";
    cin >> E.Id;

    cout << "Enter Employee Name : ";
    cin >> E.Name;

    cout << "Enter Employee Job : ";
    cin >> E.Job;

    cout << "Enter Employee House No. : ";
    cin >> E.Add.HouseNo;

    cout << "Enter Employee City : ";
    cin >> E.Add.City;

    cout << "Enter Employee Pin Code : ";
    cin >> E.Add.PinCode;


    cout << endl << "Details of Employee : ";
    cout << endl << "Employee ID: "<< E.Id;
    cout << endl << "Employee Name: " << E.Name;
    cout << endl << "Employee Job: " << E.Job;
    cout << endl << "Employee House No.: " << E.Add.HouseNo;
    cout << endl << "Employee City: " << E.Add.City;
    cout << endl << "Employee Pin Code: " << E.Add.PinCode;
    cout << endl;

    return(0);
}
```

# Structures (Recap)

- Example (continued…)



Microsoft Visual Studio Debug Console

```
Enter Employee ID : 1
Enter Employee Name : Naveed
Enter Employee Job : Professor
Enter Employee House No. : 22
Enter Employee City : Islamabad
Enter Employee Pin Code : 11111

Details of Employee :
Employee ID: 1
Employee Name: Naveed
Employee Job: Professor
Employee House No.: 22
Employee City: Islamabad
Employee Pin Code: 11111
```

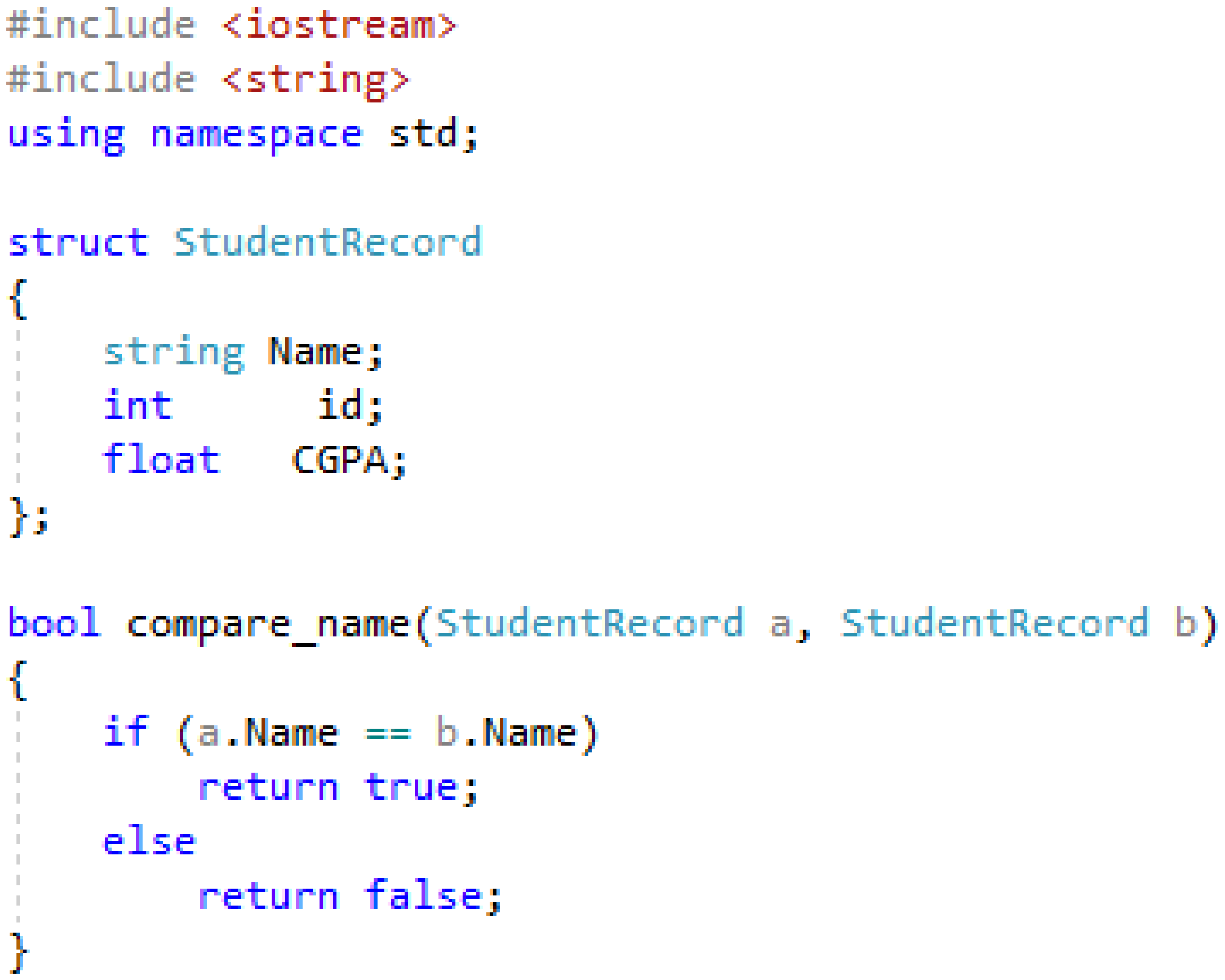
# Structures (Recap)

- Some important points to remember:

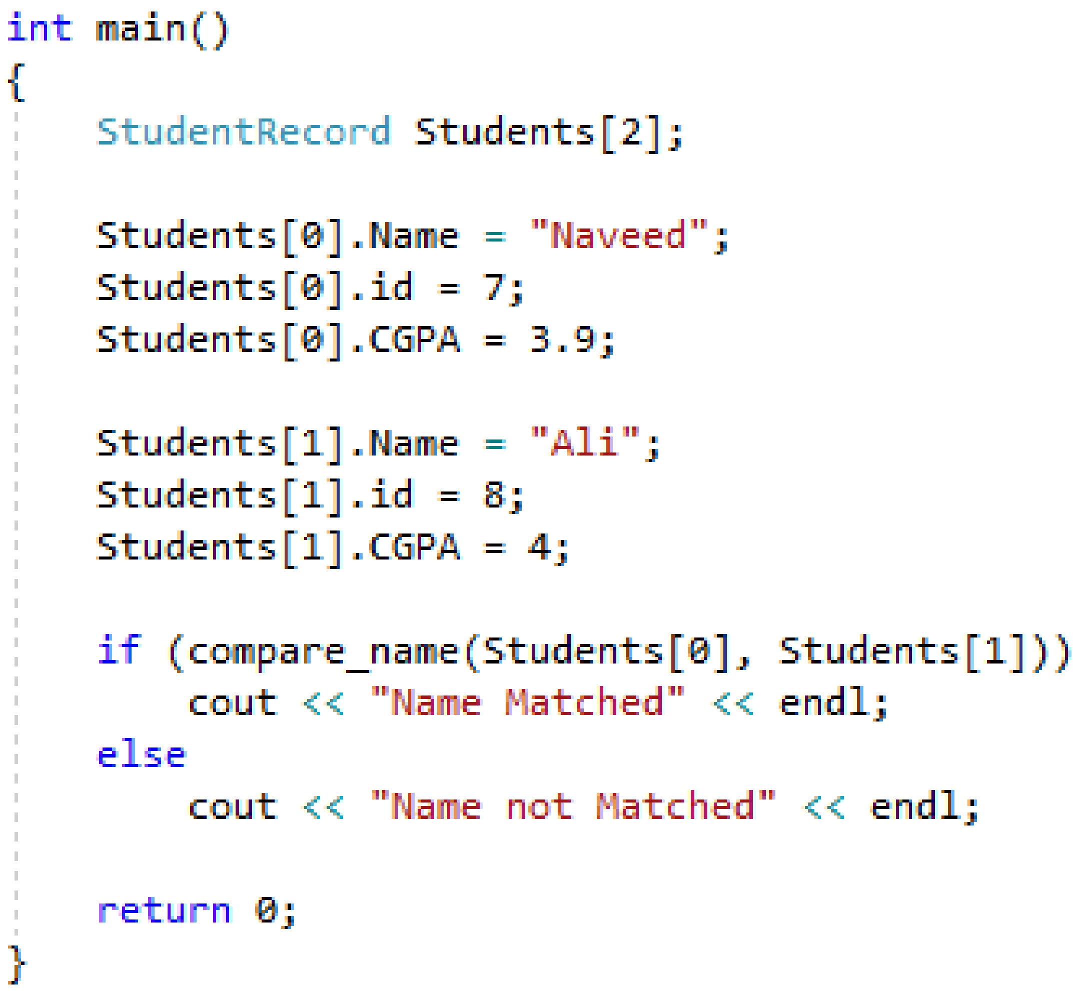
  - ❑ Aggregate I/O is **not allowed**. I/O must be performed on a member by member basis.
  - ❑ Aggregate assignment is allowed. All data members (fields) are copied (**if both structure variables are of same type**)
  - ❑ Aggregate arithmetic is **not allowed**.
  - ❑ Aggregate comparison is **not allowed**. Comparisons must be performed on a member by member basis.
  - ❑ A struct is a valid return type for a value returning function.

# Passing structure to function

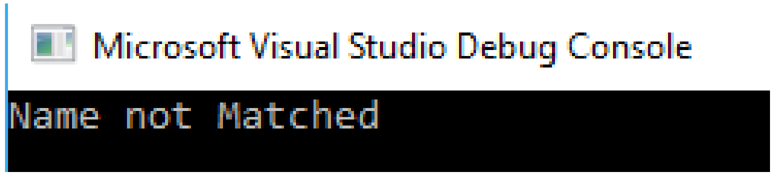
- Example of comparison:



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

struct StudentRecord
{
    string Name;
    int       id;
    float    CGPA;
};

bool compare_name(StudentRecord a, StudentRecord b)
{
    if (a.Name == b.Name)
        return true;
    else
        return false;
}
```



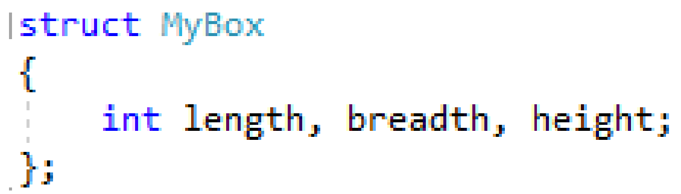
```cpp
int main()
{
    StudentRecord Students[2];

    Students[0].Name = "Naveed";
    Students[0].id = 7;
    Students[0].CGPA = 3.9;

    Students[1].Name = "Ali";
    Students[1].id = 8;
    Students[1].CGPA = 4;

    if (compare_name(Students[0], Students[1]))
        cout << "Name Matched" << endl;
    else
        cout << "Name not Matched" << endl;

    return 0;
}
```



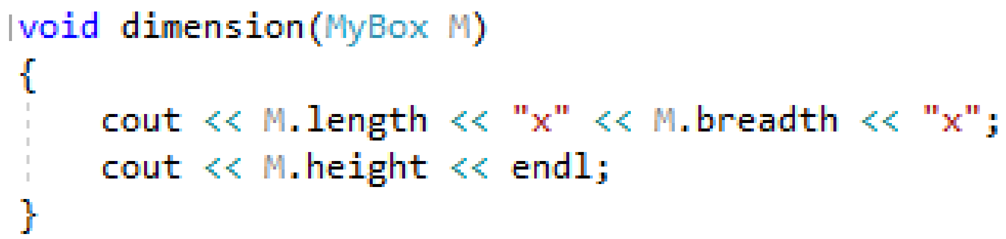
```
Name not Matched
```

# Passing structure to function

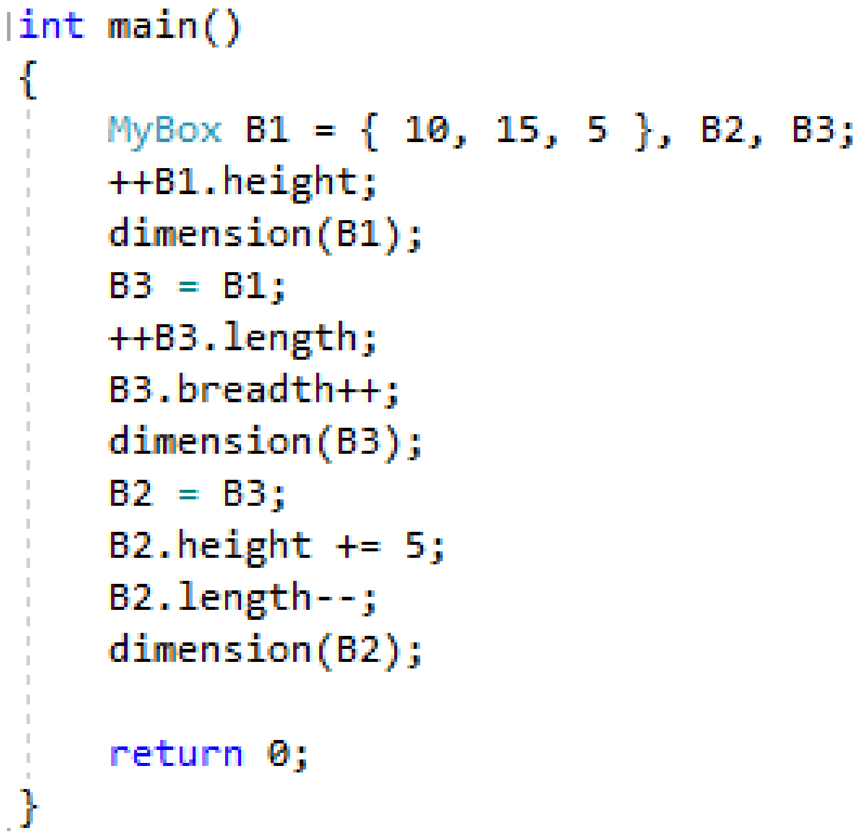
- Exercise: Find the output of the following program



```
struct MyBox
{
    int length, breadth, height;
};
```

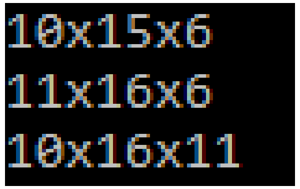


```
void dimension(MyBox M)
{
    cout << M.length << "x" << M.breadth << "x";
    cout << M.height << endl;
}
```



```
int main()
{
    MyBox B1 = { 10, 15, 5 }, B2, B3;
    ++B1.height;
    dimension(B1);
    B3 = B1;
    ++B3.length;
    B3.breadth++;
    dimension(B3);
    B2 = B3;
    B2.height += 5;
    B2.length--;
    dimension(B2);

    return 0;
}
```

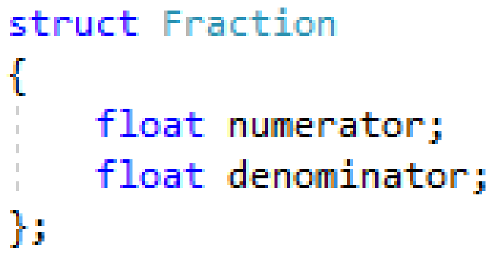
**Output:**



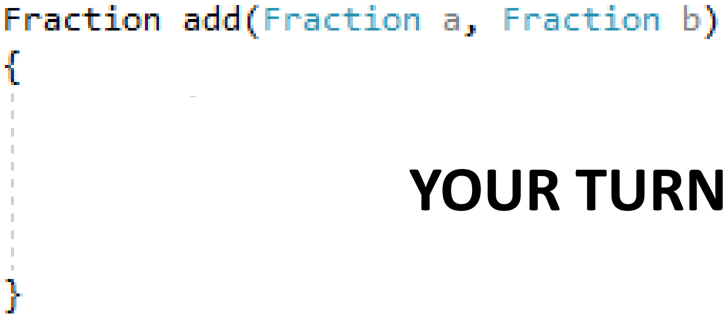
```
10x15x6
11x16x6
10x16x11
```

# Passing structure to function

- Example of addition:

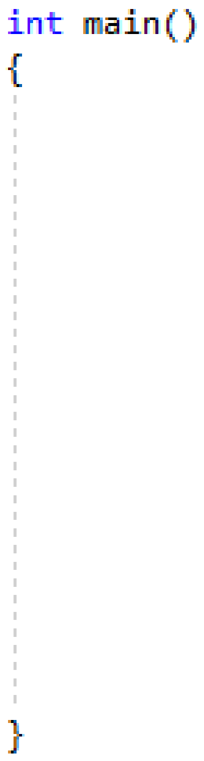


```
struct Fraction
{
    float numerator;
    float denominator;
};
```



```
Fraction add(Fraction a, Fraction b)
{

}
```

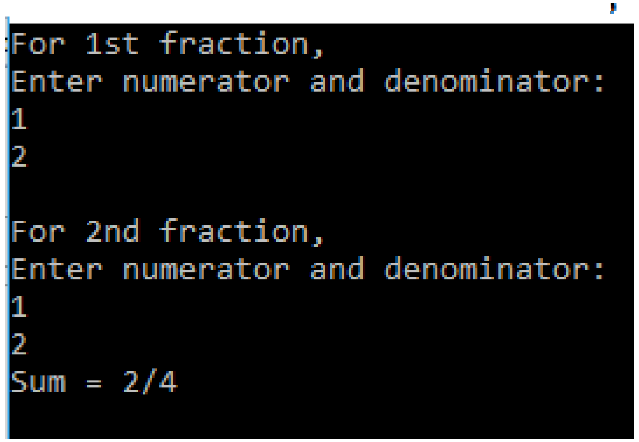
**YOUR TURN**



```
int main()
{

                                                              ;
}
```

**YOUR TURN**

**Output:**



```
For 1st fraction,
Enter numerator and denominator:
1
2

For 2nd fraction,
Enter numerator and denominator:
1
2
Sum = 2/4
```

# Passing Structure Array to Function

## Option 1

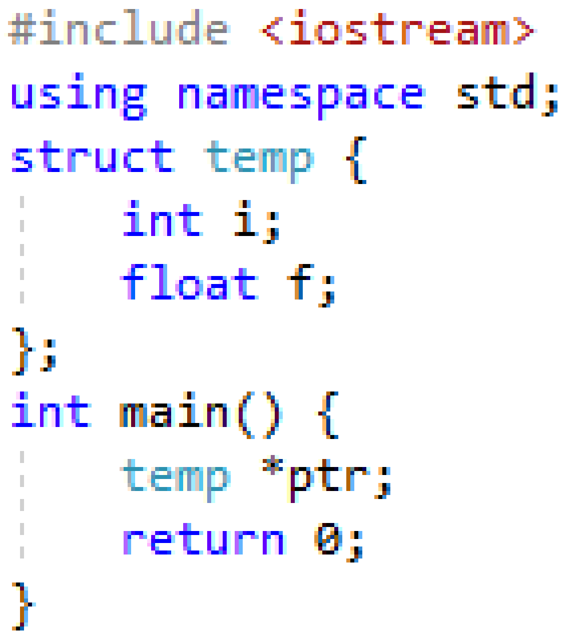
```
void myFunction(StudentRecord Student[10])
{
   .
   .
   .
}
```

## Option 1

```
void myFunction(StudentRecord Student[], int size)
{
   .
   .
   .
}
```

# Pointers to Structure

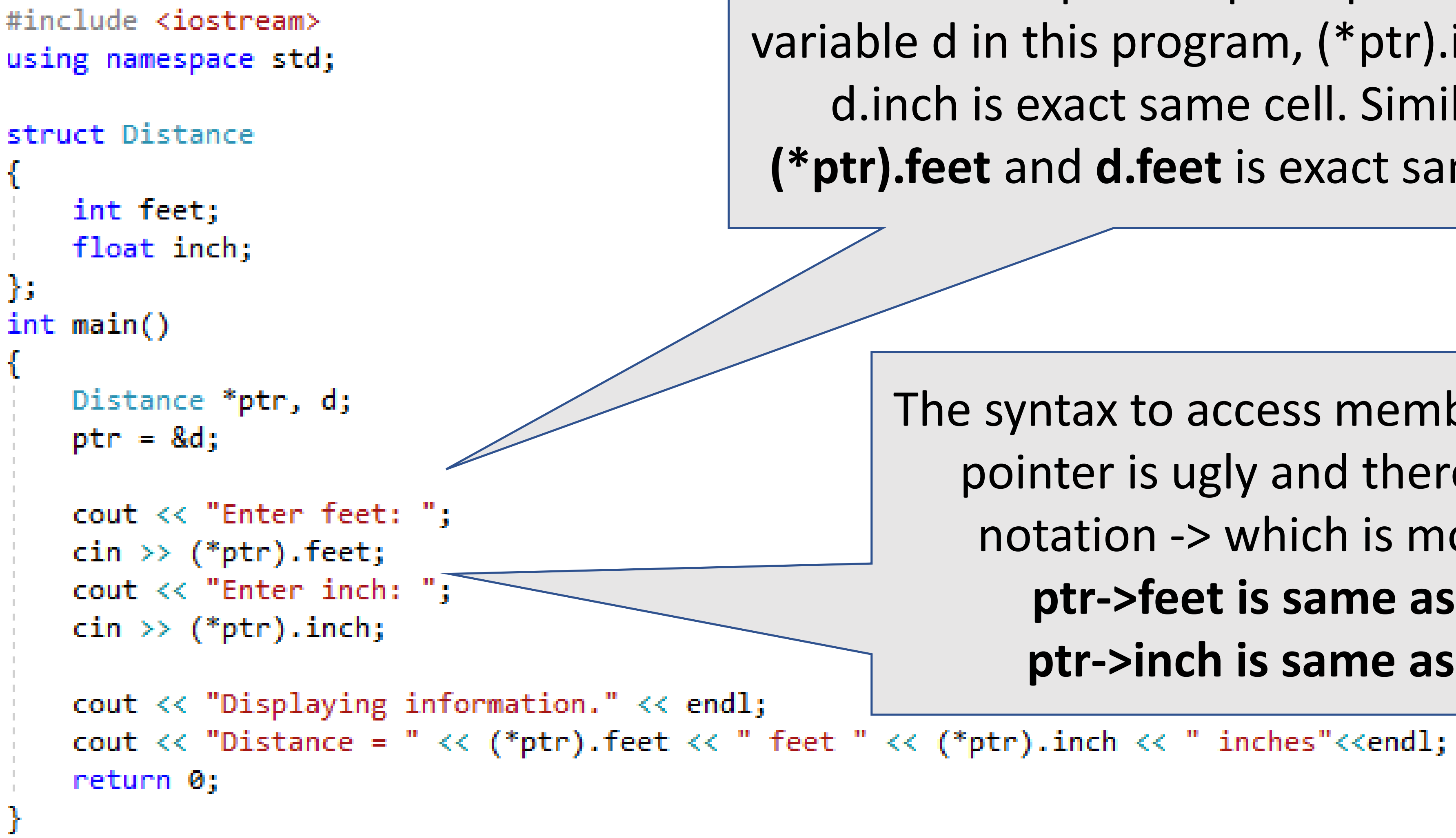
Here is how you can create pointer for structures:



```
#include <iostream>
using namespace std;
struct temp {
    int i;
    float f;
};
int main() {
    temp *ptr;
    return 0;
}
```

# Pointers to Structure

- Example



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

struct Distance
{
    int feet;
    float inch;
};
int main()
{
    Distance *ptr, d;
    ptr = &d;

    cout << "Enter feet: ";
    cin >> (*ptr).feet;
    cout << "Enter inch: ";
    cin >> (*ptr).inch;

    cout << "Displaying information." << endl;
    cout << "Distance = " << (*ptr).feet << " feet " << (*ptr).inch << " inches"<<endl;
    return 0;
}
```

**Note:** Since pointer ptr is pointing to variable d in this program, (*ptr).inch and d.inch is exact same cell. Similarly, **(*ptr).feet** and **d.feet** is exact same cell.

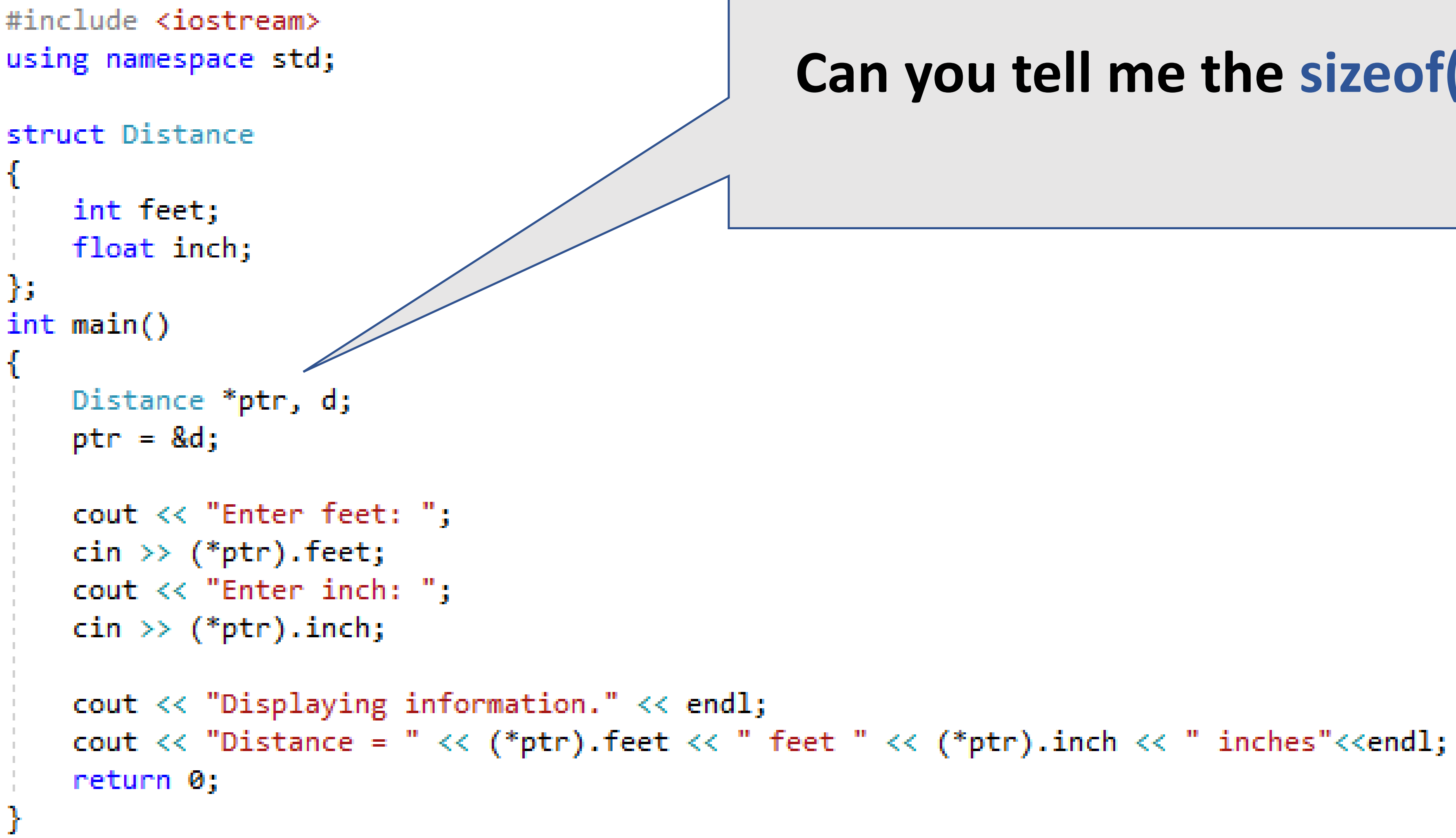
The syntax to access member function using pointer is ugly and there is alternative notation -> which is more common..

**ptr->feet is same as (*ptr).feet**

**ptr->inch is same as (*ptr).inch**

# Pointers to Structure

- Example



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

struct Distance
{
    int feet;
    float inch;
};
int main()
{
    Distance *ptr, d;
    ptr = &d;

    cout << "Enter feet: ";
    cin >> (*ptr).feet;
    cout << "Enter inch: ";
    cin >> (*ptr).inch;

    cout << "Displaying information." << endl;
    cout << "Distance = " << (*ptr).feet << " feet " << (*ptr).inch << " inches"<<endl;
    return 0;
}
```

**Can you tell me the sizeof(ptr)?**

# Unions

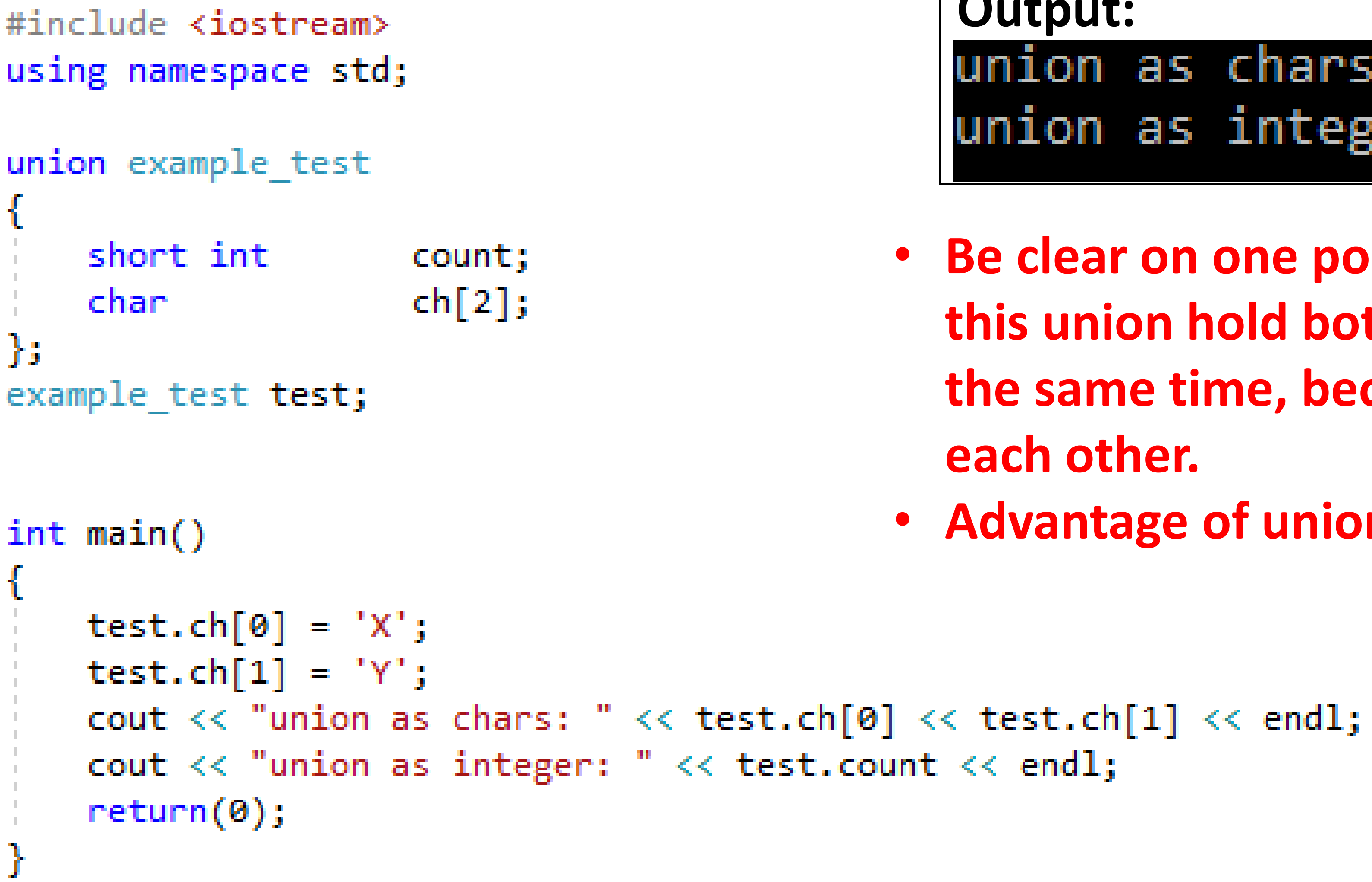
- A union is comprised of two or more variables that share the same memory location.

- A union declaration is similar to that of a structure, as shown below:

```
union example
{
    int      a;
    double b;
    char    c;
};
```

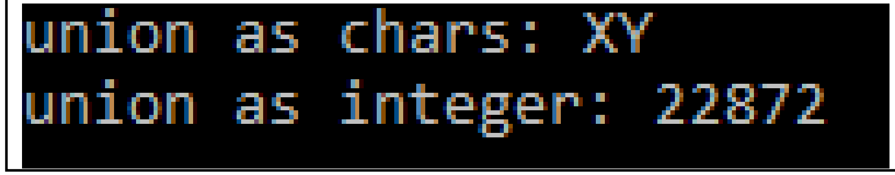
# Unions

- Example



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

union example_test
{
    short int       count;
    char            ch[2];
};
example_test test;


int main()
{
    test.ch[0] = 'X';
    test.ch[1] = 'Y';
    cout << "union as chars: " << test.ch[0] << test.ch[1] << endl;
    cout << "union as integer: " << test.count << endl;
    return(0);
}
```

**Output:**

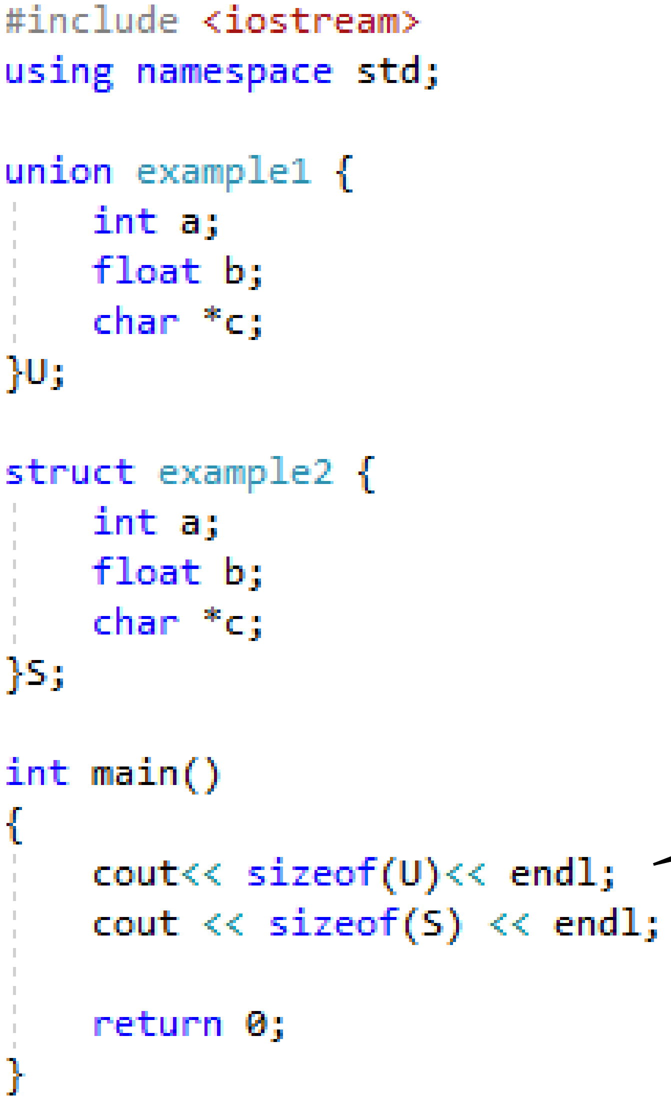


```
union as chars: XY
union as integer: 22872
```

- **Be clear on one point: It is not possible to have this union hold both an integer and a character at the same time, because *count* and *ch* overlay each other.**
- **Advantage of union?**

# Unions

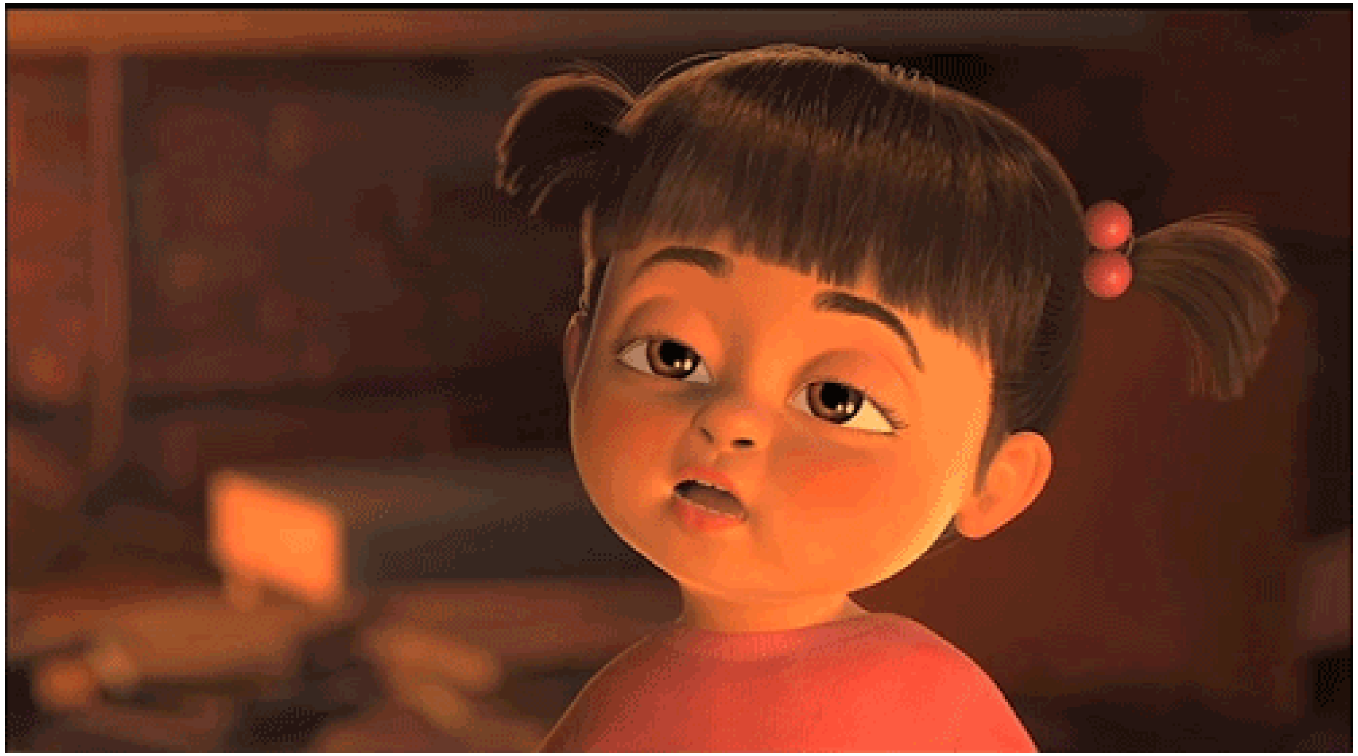
- Example



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

union example1 {
    int a;
    float b;
    char *c;
}U;

struct example2 {
    int a;
    float b;
    char *c;
}S;

int main()
{
    cout<< sizeof(U)<< endl;
    cout << sizeof(S) << endl;

    return 0;
}
```

**Can you tell me the output?**

# Thanks a lot



If you are taking a Nap, **wake up**.......Lecture Over