Data Structures and Object Oriented Programming

Lecture 23

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Template (Multiple Arguments)

Template (Multiple Arguments)

- We can use more than one generic data type in a class template
- Declared as a comma-separated list
- **Example:**

```
template< typename T, typename U >
void add( T t, U u )
{
    cout<< t+u;
}

int main()
{
    double d = 10.5674;
    int i = 10;
    add(d, i);
    return 0;
}
```

Template (Multiple Arguments)

- We can use more than one generic data type in a class template
- Declared as a comma-separated list
- **Example:**

```
template< typename T, typename U >

T my_cast( U u )
{
    return (T)u;
}

int main()
{
    double d = 10.5674;

    int j = my_cast( d ); //Error
    int i = my_cast<int>( d ); //OK
    int k = my_cast<int, double>( d ); //OK
    return 0;
}
```

Reason: **my_cast** function needs two types as an argument

Template on User-defined types

User-Defined Types

- Besides primitive types, user-defined types can also be passed as type arguments to templates

- <u>Example</u>

```
class myclass {
    int x, y;
public:
    myclass(int a=0, int b=0);
    friend myclass operator+ (myclass, myclass);
};

myclass::myclass(int a=0, int b=0){
    x = a;
    y = b;
}
myclass operator+ (myclass p1, myclass p2) {
    myclass temp;
    temp.x = p1.x + p2.x;
    temp.y = p1.y + p2.y;
    return temp;
}
```

```
template< typename T>

T add( T a, T b )
{
    return (a + b);
}


int main() {
    int d = add(10,20)

    myclass x(1, 2);
    myclass y(3, 4);
    myclass z = add(x, y);

    return 0;
}
```

Executing on built-in data types

Executing on user-defined data types

Template vs. Overloading

Overloading vs Templates

- Different data types, **similar** operation

➢ **Needs function overloading**

- Different data types, **identical** operation

➢ **Needs function templates**

Overloading vs Templates

- Templates provide an advantage when you want to perform the same action on types that can be different.

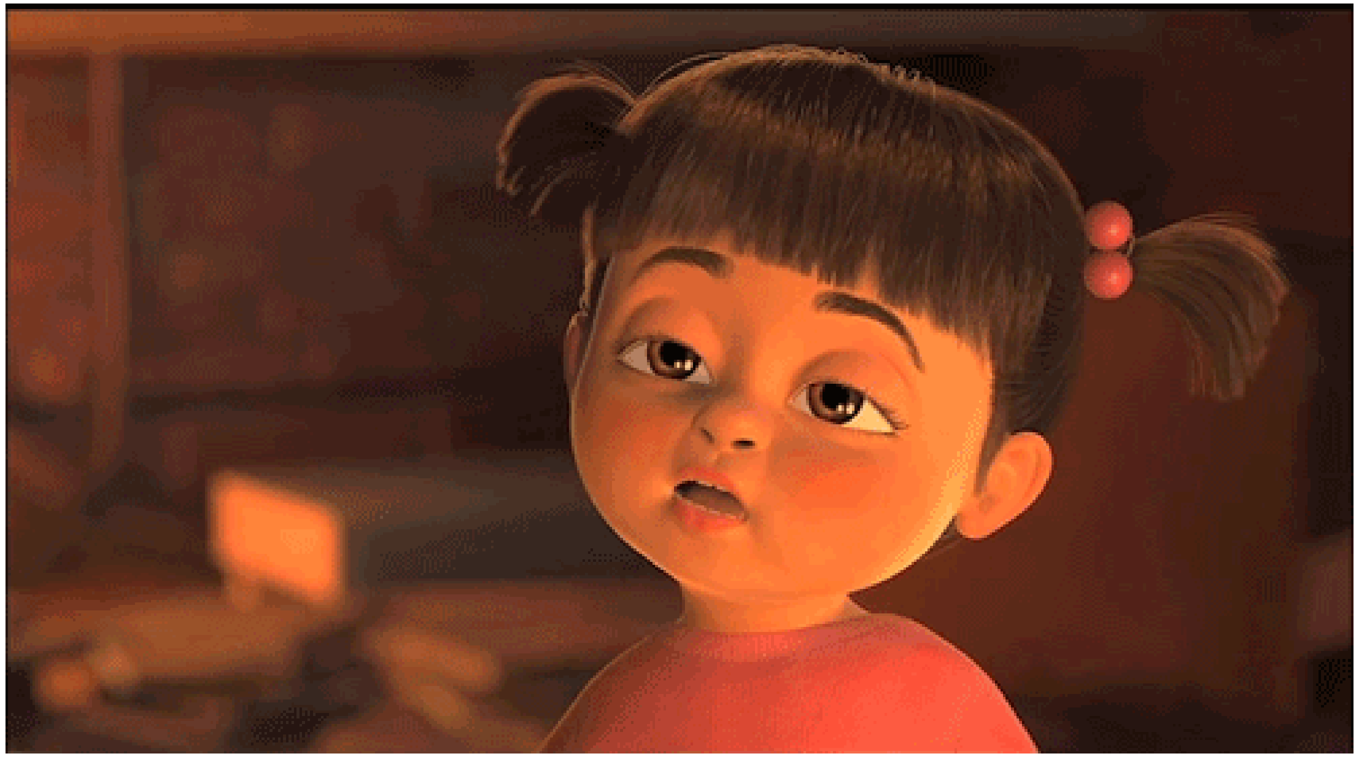
- You can use overloading when you want to apply different operations depending on the type

```
void print(int i)
{ cout << "i = " << i << "\n"; }

void print(myclass m)
{ m.print(); }
```

- Templates cannot take varying numbers of arguments. Overloads can

Thanks a lot



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