



Java™ Education & Technology Services

Object Oriented programming Using C++

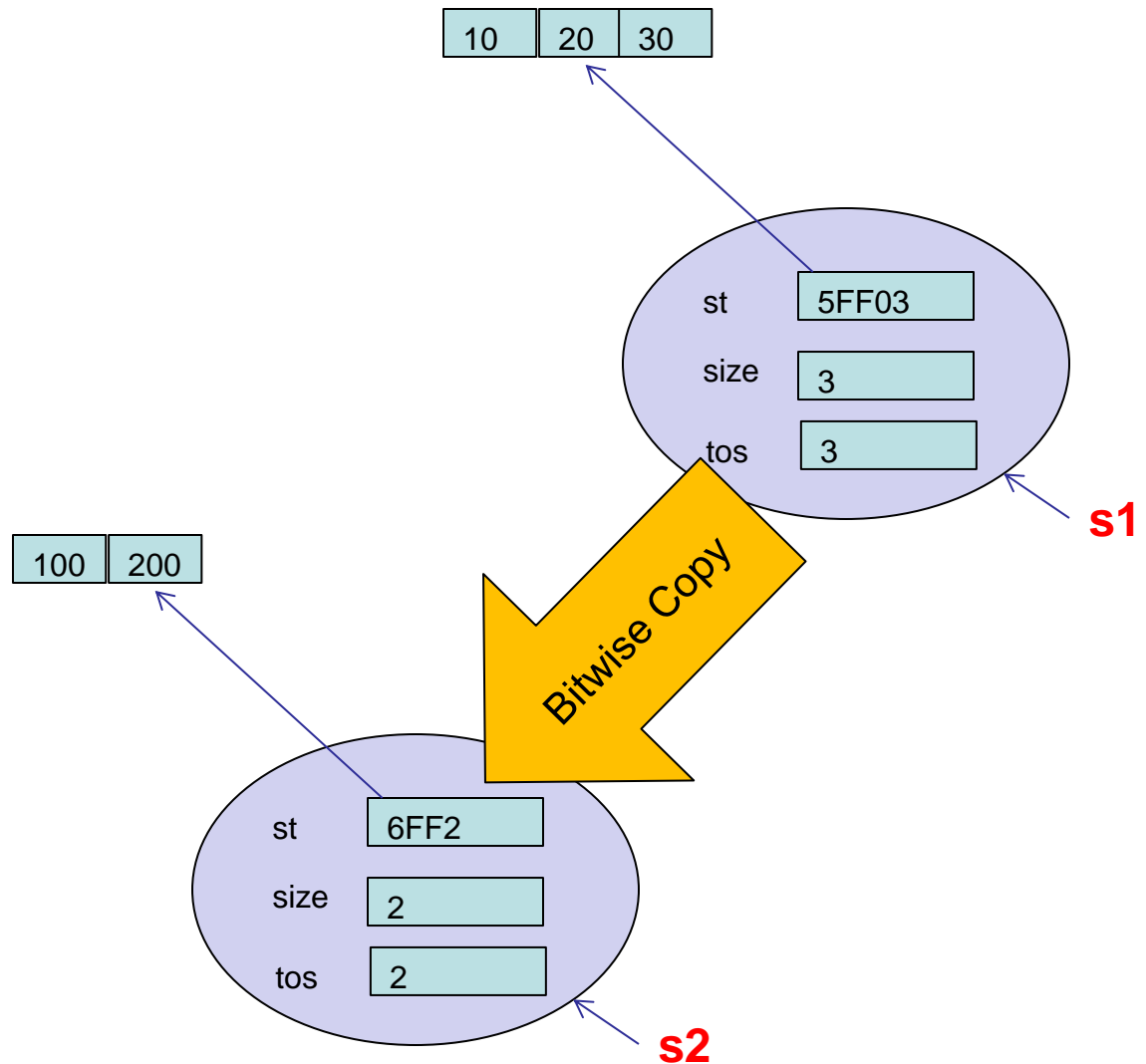


Stack Example

```
int main() {  
    int x = 5;  
    :  
    int y = 7;  
    :  
    y = x;  
    cout << x ;    5  
    cout << y;    5  
    x= 3;  
    cout << x;    3  
    cout << y;    5  
}
```

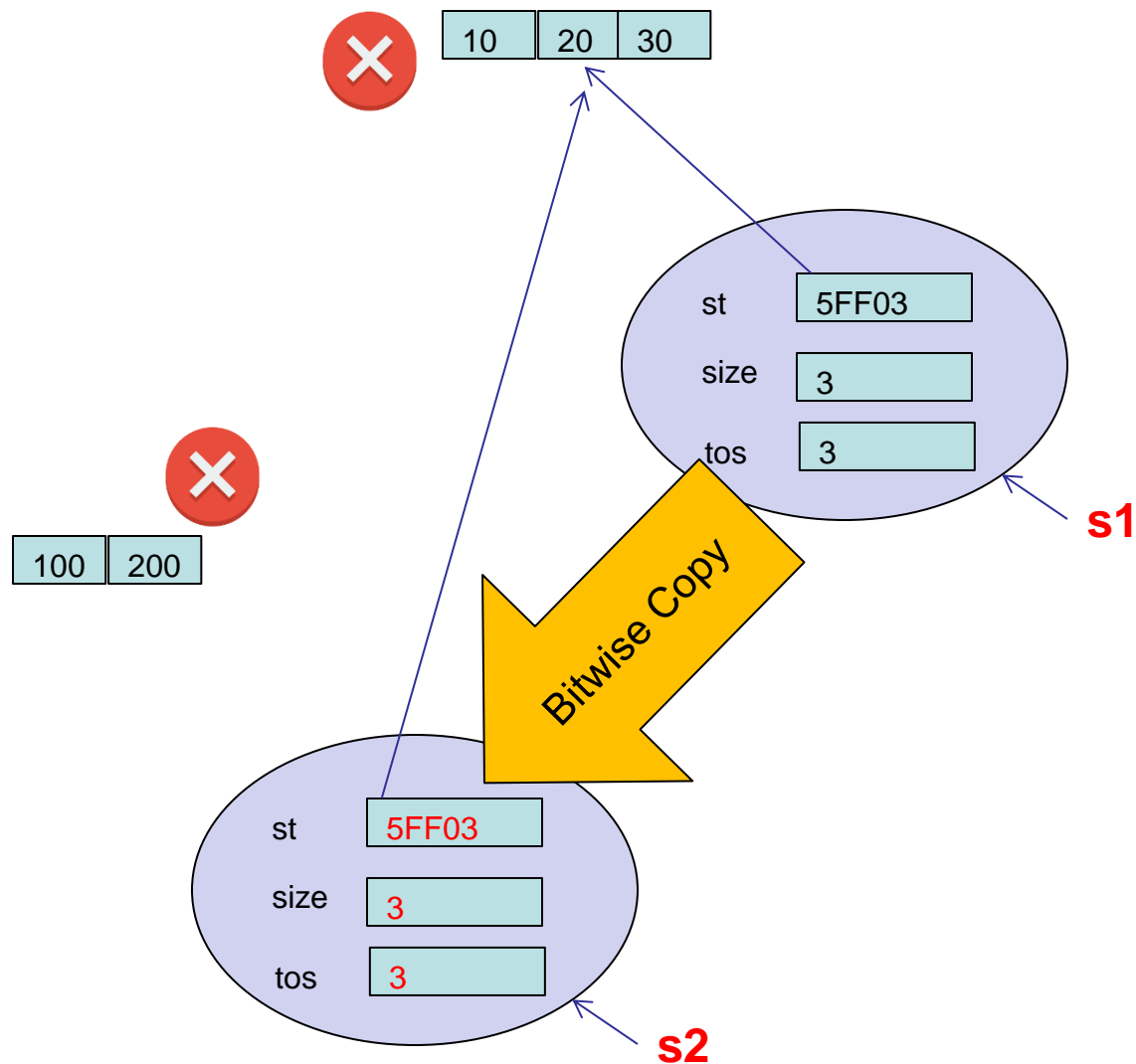
Stack Example

```
int main() {
    Stack s1(3);
    :
    Stack s2 (2);
    :
    s2=s1;
```



Stack Example

```
int main() {
    Stack s1(3);
    :
    Stack s2 (2);
    :
    s2=s1;
    s2.pop();
    s1.push(1);
}
```



Using the normal = operator

1. Bitwise copy between `s1` and `s2` (dynamic area problem)
2. `s1` and `s2` have one stack area.
3. this stack area will be deleted twice when `s1` and `s2` are destroyed.



Try to Overload the = operator for class stack

To extend the functionality of the operator.

Most of operators can be overloaded except `[. ? : :: *]`

= for Stack

```
class Stack{
```

```
:
```

```
void operator = ( Stack s ) ;
```

```
};
```

= overload function

```
void Stack :: operator = ( Stack s ) {
```

```
delete[] this->st; // for caller this
```

```
this-> tos = s.tos;
```

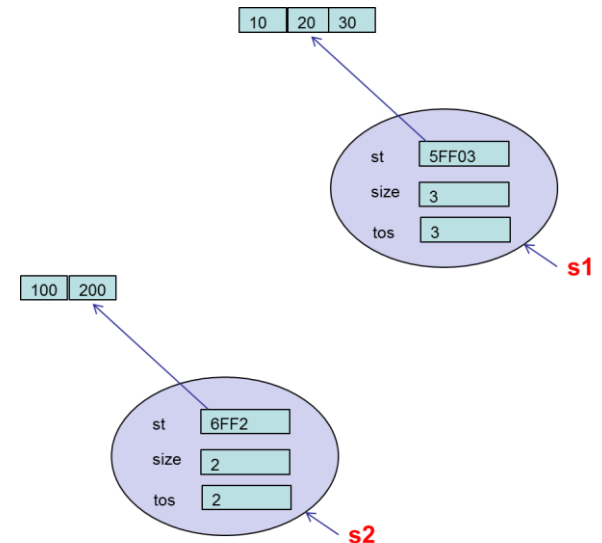
```
this-> size = s.size;
```

```
this-> st = new int [size];
```

```
for(int i=0; i< tos;i++)
```

```
    this-> st[i] = s.st[i];
```

```
}
```



```
int main() {
```

```
    Stack s1(3);
```

```
    Stack s2 (2);
```

```
    s2=s1;
```

```
    // this : s2
```

```
    // s: s1
```

= for Stack

```
class Stack{
    :
    void operator = ( Stack s ) ;
};
```

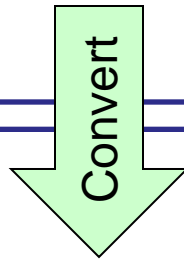
```
int main() {
    Stack s1(3);
    Stack s2 (2);
    s2=s1;
```

1. Object s1 send by **value** :
 - so the **copy constructor** will run to copy from s1 to s
2. we can call it by **const reference** to protect it from changing inside the function body.

```
class Stack{
    :
    void operator = (const Stack &s ) ;
};
```

= for Stack

```
class Stack{
    :
    void operator = (const Stack &s) ;
};
```



```
class Stack{
    :
    stack& operator = (const Stack &s) ;
};
```

```
int main() {
    Stack s1(3);
    Stack s2 (2);
    s2=s1;
    s3=s2=s1;
```




Operators overload for Complex

```
class Complex{
    :
    Complex operator + (Complex c ) ;
};
```

```
Complex Complex :: operator + (Complex c )
{
    Complex b;
    b.real = real + c.real;
    b.img = img + c.img;
    return b;
}
```

```
int main() {
    Complex c1(10,3);
    Complex c2(5,2);
    Complex c3;
    c3 = c1 + c2;
    c3. print () ;
}
```

Test : c4 = c1 + c2 + c3

1. need to overload **+ operator** for Class Complex
-  2. **c1** is the caller of **+ operator** and **c2** is the parameter
3. No need to overload **= operator** for Class Complex

Operators overload for Complex

```
class Complex{
    Complex operator + (Complex c ) ;
    Complex operator + (float x ) ;
};
```

```
Complex Complex :: operator + (float x)
{
    Complex b;
    b.real = real + x;
    b.img = img;
    return b;
}
```

```
int main() {
    Complex c1(10,3);
    Complex c2(5,2);
    Complex c3;
    c3 = c1 + 5;
    c3. print () ;
}
```

Test : c2 = 5 + c1 ;

1. need to overload extra **+ operator** for Class Complex
2. **c1** is the caller of **+ operator** and **5** is the parameter

Operators overload for Complex

```
class Complex{
    Complex operator + (Complex c ) ;
    Complex operator + (float x ) ;
    friend Complex operator + (float x ,Complex c ) ;
};
```

```
int main() {
    Complex c1(10,3);
    Complex c2(5,2);
    Complex c3;
    c3 = c1 + 5;
    c3 = 5+c1;
    c3. print () ;
}
```

```
Complex operator + (float x , Complex c )
{
    Complex b;
    b.real = c.real + x;
    b.img = c.img;
    return b;
}
```

1. need to overload extra **+ operator** as a **friend** function to Class Complex.
2. **+** operator parameters are **5** and **c1**

Operators overload for Complex

```
int main() {
    Complex c1(10,3);
    Complex c2(5,2);
    Complex c3;
    c3 = c1 + c2;
    c3 = c1 - c2;
    c3 = c1 + 5;
    c3 = c1 - 5;
    c3 = 5 + c1;
    c3 = 5 - c1;
    c1 == c2;
    c1 += c2;
```

```
class Complex{
    Complex operator+(Complex);
    Complex operator-(Complex);
    Complex operator+(float);
    friend Complex operator+(float, Complex); // friend function
    friend Complex operator-(float, Complex); // friend function
    Complex operator+=(Complex);
    int operator==(Complex);
};
```



Operators overload for Complex

```
int main() {  
    :  
    c3 = c1 + c2;  
    c3 = c1 - c2;  
    c3 = c1 + 5;  
    c3 = c1 - 5;  
    c3 = 5 + c1;  
    c3 = 5 - c1;  
    c1 == c2;  
    c1 += c2;  
  
    ++c1;  
    c1 ++ ;  
    (float) c1;
```

```
class Complex{  
    Complex operator+(Complex);  
    Complex operator-(Complex);  
    Complex operator+(float);  
    friend Complex operator+(float, Complex); // friend function  
    friend Complex operator-(float, Complex); // friend function  
    Complex operator+=(Complex);  
    int operator==(Complex);  
  
    Complex operator++(); //Prefix  
    Complex operator++(int); //Postfix  
    operator float(); //casting operator  
};
```

Operators overload for Complex

```
int main() {
    :
    ++c1; //prefix
    // increment the real part
    c2 = ++c1;
```

```
int main() {
    :
    c1++; //postfix
    // increment the real part
    c2 = c1 ++;
```

```
Complex Complex::operator++()
{
    real++;
    return *this;
}
```

```
Complex Complex::operator++(int)
{
    Complex temp = *this;
    real++;
    return temp; // return old value
}
```

the compiler uses this **int** parameter to distinguish between the prefix and post fix

Operators overload for Complex

```
int main() {  
    :  
    cout<< (float) c1;  
  
    float x = (float) c1;
```

```
class Complex{  
    :  
    operator float(); //casting operator  
};  
  
Complex::operator float()  
{  
    return real;  
}
```

No return type for casting operator as it automatically returns the type casting to.



Lab Exercise

- **1st Assignment :**

1. **Complete Stack Class:**

1. = operator overload.

2. **Complete Complex Class for:**

```
c3 = c1 + c2;
```

```
c3 = c1 - c2;
```

```
c3 = c1 + 5;
```

```
c3 = c1 - 5;
```

```
c3 = 5 + c1;
```

```
c3 = 5 - c1;
```

```
c1 == c2;
```

```
c1 += c2;
```

```
++c1; //--c1;
```

```
c1 ++ ; // c1--;
```

```
(float) c1;
```