

Operator overloading

Bachelor of Science - École polytechnique

gael.thomas@inria.fr

Key concepts

■ You can overload the C++ operators

- Return a new object by copy

```
c_t operator + (const c_t& a, const c_t& b)
```

- Return one of the parameter by reference

```
c_t& operator += (c_t& a, const c_t& b)
```

■ A cast operator is used to convert an object to another type

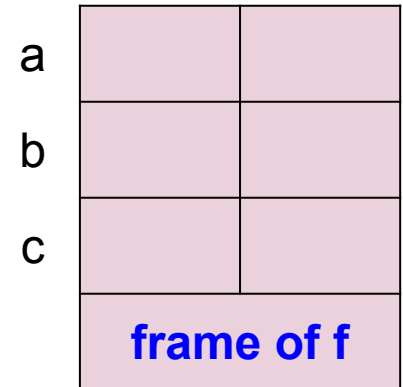
■ You can print your own objects by overloading <<

```
std::ostream& operator << (std::ostream& os, const c_t& c)
```

Operator overloading

- You can overload any operator in C++

```
struct c_t {  
    double r;  
    double i;  
};  
  
c_t operator + (const c_t& a, const c_t& b) {  
    c_t r { a.r + b.r, a.i + b.i };  
    return r;  
}
```



Operator overloading

- An overloaded operator can be used with the new types

```
struct c_t {  
    double r;  
    double i;  
};  
  
c_t operator + (const c_t& a, const c_t& b) {  
    c_t r { a.r + b.r, a.i + b.i };  
    return r;  
}  
  
void f() {  
    c_t a { 1, 2 }, b { 10, 12 };  
    c_t c = a + b;  
    ...  
}
```

Operator overloading

- a and b are local variables => allocated in the frame

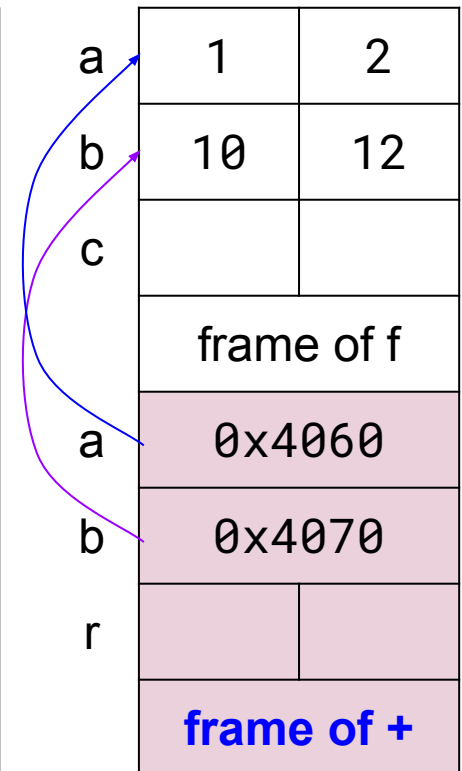
```
struct c_t {  
    double r;  
    double i;  
};  
  
c_t operator + (const c_t& a, const c_t& b) {  
    c_t r { a.r + b.r, a.i + b.i };  
    return r;  
}  
  
void f() {  
    c_t a { 1, 2 }, b { 10, 12 };  
    c_t c = a + b;  
    ...  
}
```

a	1	2
b	10	12
c		
frame of f		

Operator overloading

- a and b are passed by reference => + receives pointers

```
struct c_t {  
    double r;  
    double i;  
};  
  
c_t operator + (const c_t& a, const c_t& b) {  
    c_t r { a.r + b.r, a.i + b.i };  
    return r;  
}  
  
void f() {  
    c_t a { 1, 2 }, b { 10, 12 };  
    c_t c = a + b;  
    ...  
}
```



Operator overloading

- + uses the a and b of f to compute r

```
struct c_t {  
    double r;  
    double i;  
};  
  
c_t operator + (const c_t& a, const c_t& b) {  
    c_t r { a.r + b.r, a.i + b.i };  
    return r;  
}  
  
void f() {  
    c_t a { 1, 2 }, b { 10, 12 };  
    c_t c = a + b;  
    ...  
}
```

a	1	2
b	10	12
c		
	frame of f	
a	0x4060	
b	0x4070	
r	11	14
	frame of +	

Operator overloading

- The result of + is an object (not a reference, not a pointer)
=> the fields of r are copied into the fields of c

```
struct c_t {  
    double r;  
    double i;  
};  
  
c_t operator + (const c_t& a, const c_t& b) {  
    c_t r { a.r + b.r, a.i + b.i };  
    return r;  
}  
  
void f() {  
    c_t a { 1, 2 }, b { 10, 12 };  
    c_t c = a + b;  
    ...  
}
```

a	1	2
b	10	12
c	11	14
frame of f		

Operator overloading and return type

■ Return by copy versus return by reference

```
// returns a copy when the arguments are not modified
c_t operator + (const c_t& a, const c_t& b) {
    c_t r { a.r + b.r, a.i + b.i };
    return r; // returns a copy of r
}
```

```
// returns a reference when one of the argument is modified
c_t& operator += (c_t& a, const c_t& b) {
    a.r += b.r;
    a.i += b.i;
    return a; // returns a reference to a, not a copy of a
}
```

Cast operator

- A cast operator is used to convert a type to another

```
struct c_t {  
    double r;  
    double i;  
  
    operator double() const { return r; }  
};  
  
int main(int argc, char** argv) {  
    c_t c { 1, 2 };  
    double r = c; // => 1  
    return 0;  
}
```

Printing an object

- You can print your own objects thanks to operator overloading

```
struct c_t {  
    double r;  
    double i;  
};  
  
std::ostream& operator << (std::ostream& os, const c_t& c) {  
    os << "(" << c.r << ", " << c.i << ")";  
    return os;  
}  
  
int main(int argc, char** argv) {  
    c_t a { 3, 4 };  
    std::cout << a << std::endl; // => (3, 4)  
    return 0;  
}
```

Key concepts

■ You can overload the C++ operators

- Return a new object by copy

`c_t operator + (const c_t& a, const c_t& b)`

- Return one of the parameter by reference

`c_t& operator += (c_t& a, const c_t& b)`

■ A cast operator is used to convert an object to another type

■ You can print your own objects by overloading <<

`std::ostream& operator << (std::ostream& os, const c_t& c)`