Header

```
// A.hpp
class A {
  public:
     void foo1();      <- Member funct
  private:
     int data;
};
```

Source

```
// A.cpp
#include "A.hpp"

void A::foo1()
{
   ...
}
```

Static

Usage

```
main()
{
   A a;   // Instantiation
   a.foo1();
}  <- Destructor
```

(class → A, object → a)

A a; → Constructor of A

(Static) → allocated in the stack

C++ Vanilla functions:

1. Default Constructor → Initialization
2. Destructor → Cleanup.
3. Copy constructor
4. Assignment operator.

Dynamic Allocation

```
{
   A *a = new A();
   a->foo1();

   delete a;
}
```

```
A *a = new A[n];
      // Calls default constr. n times
delete [] a;
```

```
class B;
class A {
  public:
     // Nothing
  private:
     int mInt;          <- Member var.
     int * mPtrToInt;
     B mB;
};
```

```
int a;
int *p=&a;
```

[a] <- p

```
// Def. Constructor
A::A() : mB() {
   // mB -> init. thru' its constr.
   // No init for mInt
   // No init for mPtrToInt;
}
```

```
// Def. Destructor
A::~A() {
   mB.~B();
}
```

Reference = Alias

C++

```
int b = 6;
int & a = b;
int *p = &a;
```

[a, b]

a, b: [6] <- p

// Default copy constr.

Const

```
A a;
A a2;
```

A :: A (const A& other)

const → property of the path, not of the var. data type.

```
{
    A a;
    a = ...;    // Fine
    foo(a);     // const A& b = a;
    a = ...;
}
```

```
void foo(const A& b)
{
    ...
}
```

// Read-only

```
const float g = 9.8;
g = ...  //
```

a, b, **

a = a2; // Assignment

```
A a3 = a;
A a3(a);
```

formal

```
void foo(A aArg)
{
    ...
}
main() {
    A b;
    foo(b);  // A aArg = b
}
```

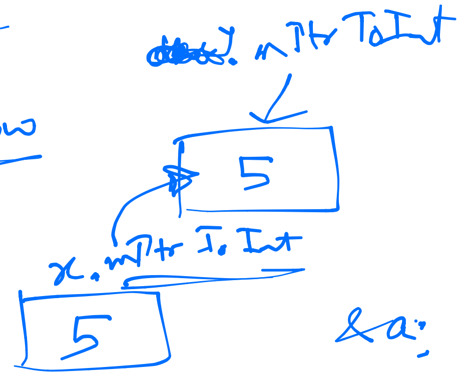
actual

```
foo(const A *_a);
foo(A const * a);
foo(A const * const a);
foo(A* const a);
```

Default Copy constructor

```
A::A(const A& other)
{
    mInt = other.mInt;
    mPtrToInt = other.mPtrToInt;
}
```

→ shallow



A x = y;

Default Assign operator

```
A& A::operator=(const A& other)
{
    mInt = other.mInt;
    mPtrToInt = other.mPtrToInt;
    mB(other.mB);  // mB = other.mB.
    return *this;
}.
```

```
class A {
private: int a;
public:
```

A* const this

a = b = c

A a(6);
int y;
a.foo(y);

void foo(int y)
{
this->a = 6;
}

this = 0x50200;

A const * const this

A a; // constr.
A b; // constr.
a = b; // a.operator=(b);
a(b); // a.operator=(b);

A a; // Def.
A b = a; // copy constr.
A b(a);

Operator overloading

A a, b;
a + b ⟹

a.operator+(b); // Member fun.

OR

operator+(a, b); // Non-member fun.

Chaining

x + y + z

⟹ ((x+y) + z)

⟹ (x.operator+(y)).operator+(z);

Signature

A operator+(const A& other);

A

A& operator = (const A& other);

A a;
cin >> a;

int x;
cin >> x;

Standard input

std::cin >> x >> y ;

⇒

operator >> (operator >> (std::cin, x), y)

std::cin

Signature : std::istream& operator >> (std::istream& A& a);

Standard output : std::cout

cout << x << y

std::ostream& operator << (std::ostream& s, const A& a);

Prefix / Postfix operators

++x ⇒

x.operator++();

x++ ⇒

x.operator++(0);

Signatures

A& operator++()
{
//Increment logic
return *this;
}

A operator++(int)
{
A result(*this);

```
++(*this);
return result;
}
```

A a;

MemberType m;

a[i] = m; // a.operator[](i) = m

const A a;

a[i] = m; // Illegal.

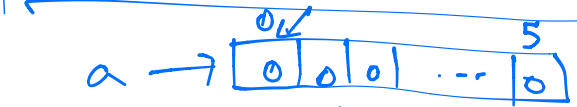
receiver is a const

Signature: const MemberType & operator[](int index) const;

MemberType & operator[](int index);

Receiver is not const.

```
int * a = new int[6];
```



&a[0]

```
for(int i=0; i<6; ++i){
    a[i] = 0;
}
foo(a, 6); // int *b = a;
```

```
void foo(int *b, int n)
{
    for(int i=0; i<n; ++i)
    {
    }
}
```

class Array {

private:

```
int *mA;
int mSize;
```

};

Array a;

foo(a);