

# OOP C++




Moatasem Elsayed

# Content( long session 😁 )

- Thinking with c++*
- Enum*
- Initialization*
- struct/class*
- OOP*
- constructor*
- Destructor*
- delegation*


# Thinking with c++

```
// C
char name[]="Moatasem";
char lname[]="Elsayed";
strcat(name,lname);
```



```
//C++
std::string Name="Moatasem";
std::string lname="Elsayed";
Name+=lname;
```

```
// C
int arr[]={10,12,310,100,200};
int max=arr[0];
for(int i=1;i<sizeof(arr)/sizeof(arr[0]);i++)
{
    if(arr[i]>max)
    {
        max=arr[i];
    }
}
std::cout <<max<<std::endl;
```



```
//C++
std::cout << *std::max_element(std::begin(arr),std::end(arr));
```

# Write a program to sort array ?

```
int i, j, a, n, number[30];
printf("Enter the value of N \n");
scanf("%d", &n);

printf("Enter the numbers \n");
for (i = 0; i < n; ++i)
    scanf("%d", &number[i]);

for (i = 0; i < n; ++i)
{
    for (j = i + 1; j < n; ++j)
    {
        if (number[i] > number[j])
        {
            a = number[i];
            number[i] = number[j];
            number[j] = a;
        }
    }
}
```

```
int s[] = {5, 7, 4, 2, 8, 6, 1, 9, 0, 3};

std::sort(std::begin(s), std::end(s));
for(auto& i :s)
{
    std::cout <<i<<" "; //0 1 2 3 4 5 6 7 8 9
}
```

# C++ is OOP language

```
//C
void fun(){
    std::cout <<"Hello World"<<std::endl;
}
```



```
//C++
class Greeting{
public:
    void fun(){
        std::cout <<"Hello World"<<std::endl;
    }
};
```

## 4- Initialization

*Initialization* of a variable provides its initial value at the time of construction.

The initial value may be provided in the initializer section of a [declarator](#) or a [new expression](#). It also takes place during function calls: function parameters and the function return values are also initialized.

For each declarator, the initializer may be one of the following:

---

( *expression-list* )      (1)

---

= *expression*      (2)

---

{ *initializer-list* }      (3)

---

1) comma-separated list of arbitrary expressions and braced-init-lists in parentheses

2) the equals sign followed by an expression

3) braced-init-list: possibly empty, comma-separated list of expressions and other braced-init-lists

Depending on context, the initializer may invoke:

- Value initialization, e.g. `std::string s{};`
- Direct initialization, e.g. `std::string s("hello");`
- Copy initialization, e.g. `std::string s = "hello";`
- List initialization, e.g. `std::string s{'a', 'b', 'c'};`
- Aggregate initialization, e.g. `char a[3] = {'a', 'b'};`
- Reference initialization, e.g. `char& c = a[0];`

If no initializer is provided, the rules of [default initialization](#) apply.

# Example on Initialization

```
//construcotr(int number, int value)
std::vector<int>v(2,3); // 3 3
//initliaizer_list
std::vector<int>v2{2,3}; // 2 3
```

```
5
6 struct Data{
7     int temp;
8 };
9 int main()
10 {
11     // 1- all of them can assign value
12     int n; //default
13     int x=10; // copy
14     int y(10); // direct
15     int z{10}; // value
16
17     // 2- {} narrow conversion
18     float dec=3.5;
19     int value{dec};
20
21     // 3- vexing parse
22     int v{};
23     int v2(); //prototype
24     int v3=0;
25
26     // 4- synthesize constructor
27     Data d; //garbage
28     Data d2{}; //zeros
29     std::cout <<d.temp<<std::endl;
30     std::cout <<d2.temp<<std::endl;
31     return 0;
```

# Enum ( C Style)

```
4  enum Traffic
5  {
6      RED,
7      YELLOW,
8      GREEN
9  };
10
11 int main()
12 {
13     Traffic obj;
14     std::cout << obj << std::endl; // 1- default =0
15     std::cout << RED << std::endl; // 2- classic enum ->access direct
16     int x = RED;                    // 3- class enum -> convert enum to int
17     std::cout << x << std::endl;
18     // Traffic obj2=1;              // 4- #ERROR cannot convert int to classic enum
19     return 0;
20 }
```



# Enum Class

```
enum class Traffic : unsigned char           // 1- specify size optional
{
    RED,
    YELLOW,
    GREEN
};

int main()
{
    Traffic obj;
    // std::cout << obj << std::endl;           // 2- cannot print obj till operator overloading exist
    std::cout << (int)obj << std::endl;         // 3- to print use casting --- till we explain static_cast<int>(obj)

    // std::cout << RED << std::endl;           // 4- #ERROR you cannot access literals without class name
    std::cout << (int)Traffic::RED << std::endl; // 5- access enum class

    // int x = Traffic::RED;                     // 6- cannot convert from enum to int
    // Traffic obj2=1;                           // 7- #ERROR cannot convert int to classic enum

    int y = static_cast<int>(Traffic::RED);      // 8- conversion happen through casting
    Traffic obj2 = static_cast<Traffic>(1);
    std::cout << (int)obj2 << " "<< y << std::endl;
    //auto compelet

    Traffic::
        return 0;
    GREEN
    RED
    YELLOW
}
```

enum class Traffic::GREEN = 2U

File: test.cpp

# When we use you classic enum over enum class ?

```
//-----  
//! \namespace Software Update  
//!  
//! \brief Software Update namespace  
//!  
enum SwUpdateParam { DigBridge, OtaBridge, FlashingApp };  
//
```

If no need for that then  
use enum class whenever  
you need to use enum



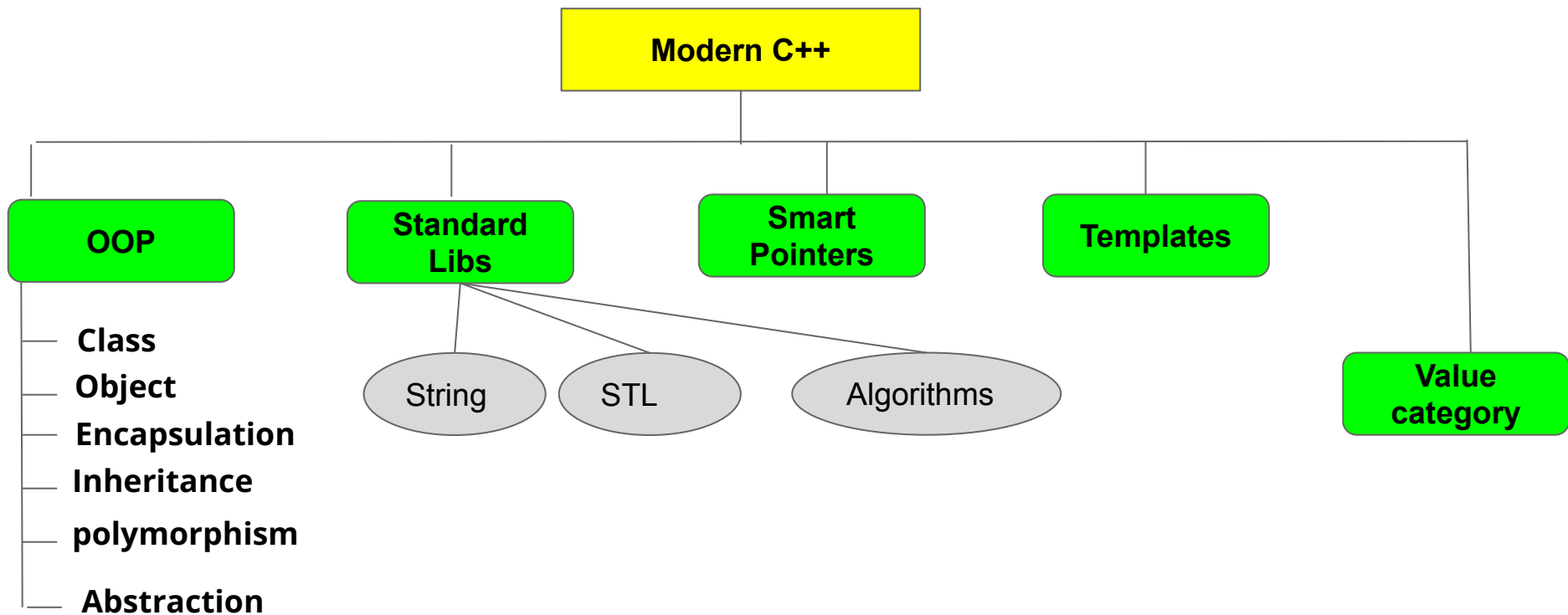
```
std::string SwUpdateJsonParse::converKeyToString(SwUpdateParam key) {  
    #define MYCase(X) \  
        case X: \  
            return std::string(#X)  
    switch (key) {  
        MYCase(DigBridge);  
        MYCase(OtaBridge);  
        MYCase(FlashingApp);  
    }  
    #undef MYCase  
    return "";  
}
```

# Struct

Function

Identifier is the type

```
2  using namespace std;
3
4  struct Data{
5      int x;
6      int y;
7      void fun(){
8          std::cout <<"HEllo"<<std::endl;
9      }
10 };
11 int main()
12 {
13     Data d;
14     d.fun();
15
16     return 0;
17 }
```

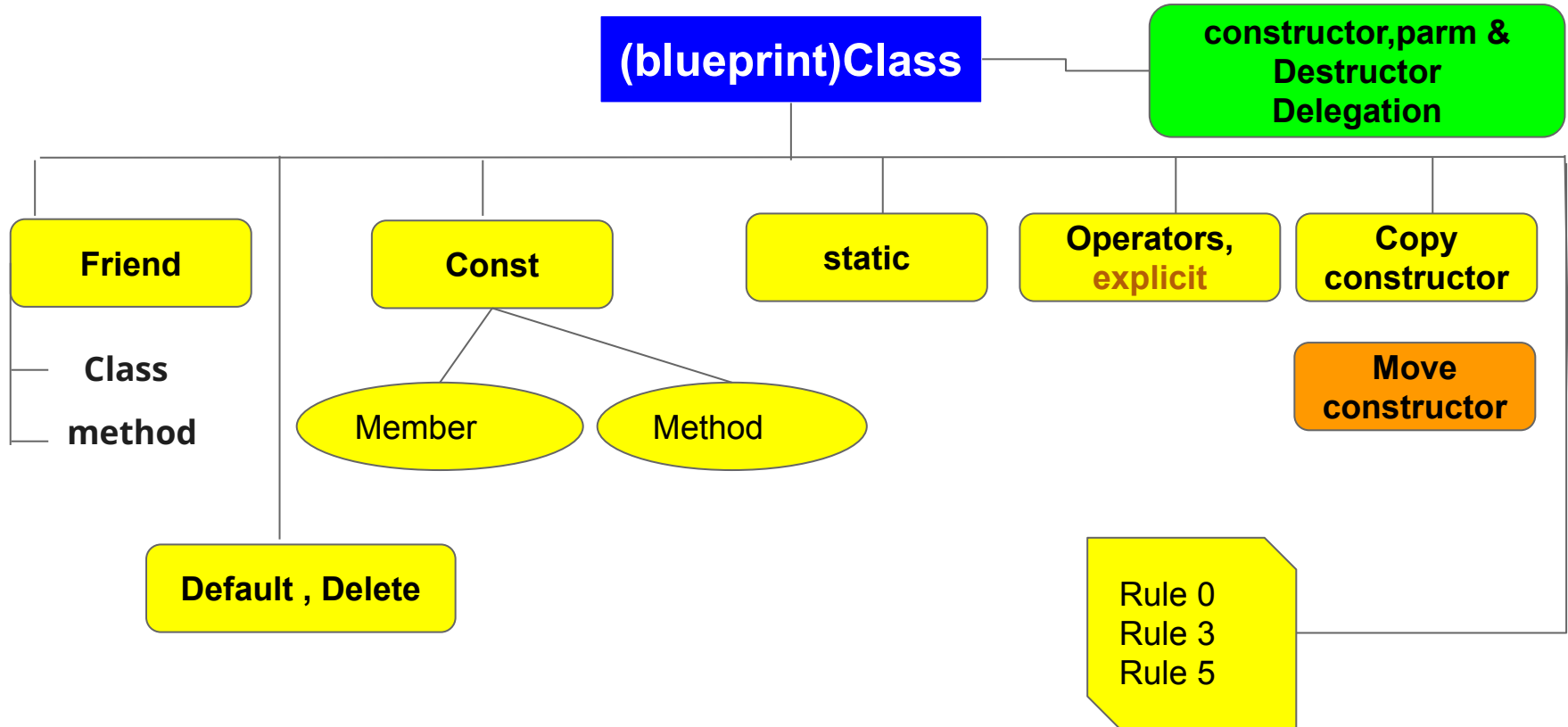


# OOP

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic.

An object can be defined as a data field that has unique attributes and behavior.

# Features of class



# Class blueprint

## Encapsulation

Methods

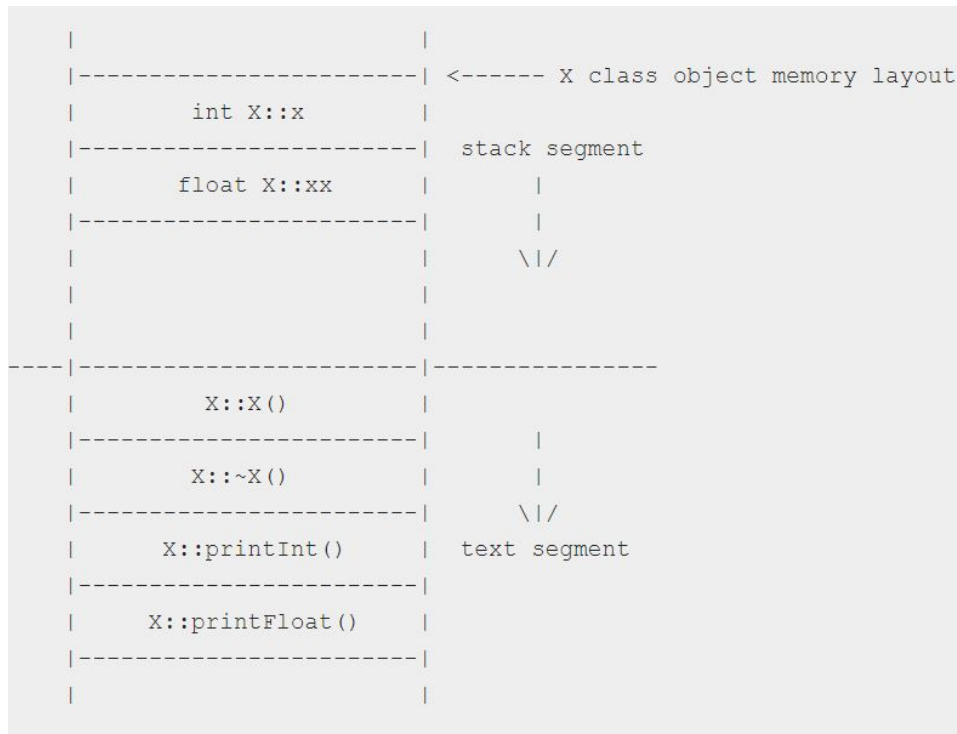
Variables

```
9  class LED{
10
11     public:
12         void TurnLedOn();
13         void TurnLedOff();
14     private:
15         uint8_t ledPin=10;
16 };
17 void LED::TurnLedOn()
18 {
19     std::cout <<"LED is HIGH "<<ledPin<<std::endl;
20 }
21
22 void LED::TurnLedOff()
23 {
24     std::cout <<"LED is LOW "<<ledPin<<std::endl;
25 }
26 }
27
28 int main() {
29     LED d;           //1- create object in stack
30     d.TurnLedOn();    //2- access only public section
31     d.TurnLedOff();
32     // d.LedPin=12;    //3- ERROR LED::LedPin' is private within this co
33     LED().TurnLedOff(); //4- temp object
34
35     //5- sizeof datatype + vp ptr if exist + padding
36     std::cout<<sizeof(d)<<std::endl;
37     LED *ptr=&d;
38     ptr->TurnLedOff(); //6- access through pointer
39 }
```

# Memory layout

Taken from : <http://www.vishalchovatiya.com/memory-layout-of-cpp-object/>

```
class X {  
    int    x;  
    float  xx;  
  
public:  
    X() {}  
    ~X() {}  
  
    void printInt() {}  
    void printFloat() {}  
};
```





# External vs inline function

```
4
5  class X {
6  public:
7      int x=10;
8      void PrintInt();
9  };
10 void X::PrintInt(){
11
12 }
13 int main()
14 {
15
16     return 0;
17 }
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   GITLENS

```
moatasem@CAI1-L14000:~/vsomeIp$ objdump -t --demangle a.out | grep -i "/*PrintInt*"
000000000000011d5 l      F .text 00000000000000019          _GLOBAL__sub_I__ZN1X8PrintIntEv
0000000000000116a g      F .text 0000000000000000f          X::PrintInt()
moatasem@CAI1-L14000:~/vsomeIp$
```

# Inline function

```
4
5 class X {
6 public:
7     int x=10;
8     void PrintInt()
9     {
10         cout << " In class A\n";
11     }
12 };
13
14 int main()
15 {
16     X obj;
17     obj.PrintInt();
18     return 0;
19 }
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

```
moatasem@CAI11-L14000:~/vsomeIp$ objdump -t --demangle a.out | grep -i "/*PrintInt*"
000000000001258 w F .text 0000000000000026 X::PrintInt()
moatasem@CAI11-L14000:~/vsomeIp$
```

```
4
5 class X {
6 public:
7     int x=10;
8     void PrintInt()
9     {
10         cout << " In class A\n";
11     }
12 };
13
14 int main()
15 {
16
17     return 0;
18 }
19
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

```
moatasem@CAI11-L14000:~/vsomeIp$ g++ -g test.cpp -std=c++14 -O0 && ./a.out
moatasem@CAI11-L14000:~/vsomeIp$ objdump -t --demangle | grep -i PrintInt
moatasem@CAI11-L14000:~/vsomeIp$
```

# Constructor(default - param)

What is “this “ ?

```
21      Data d;
    0x00000000080011be <+27>: lea    -0xc(%rbp),%rax
    0x00000000080011c2 <+31>: mov    %rax,%rdi
    0x00000000080011c5 <+34>: callq 0x800118a <Data::Data()>

22      return 0;
=> 0x00000000080011ca <+39>: mov    $0x0,%eax

23  }
    0x00000000080011cf <+44>: mov    -0x8(%rbp),%rdx
    0x00000000080011d3 <+48>:

--Type <RET> for more, q to quit
Quit
(gdb) p &d
$1 = (Data *) 0x7fffffffed9b4
(gdb) p %rax
A syntax error in expression, n
(gdb) p $rax
$2 = 140737488279988
(gdb) 
```

Calculator

Programmer

7FFF FFFE D9B4

HEX	7FFF FFFE D9B4
DEC	140,737,488,279,988

```
class LED{

public:
    LED()=default; // LED(){} //1- Default
    LED(int pin); //2-paramter
    void TurnLedOn();
    void TurnLedOff();
private:
    uint8_t ledPin=10;
};

LED::LED(int pin) :ledPin(pin) // 3- initializer list
{
    std::cout <<"Constructor " <<ledPin<<std::endl;
    // ledPin=pin; //4- instead of initializer list
}
```

## Once you create parameterized constructor , you need to create default

```
4
5  class Data {
6  public:
7      bool flag=1;
8      int value=0;
9      Data(bool flag);
10
11     void PrintInt()
12     {
13         cout << " In class A\n";
14     }
15 };
16 Data::Data(bool flag)
17 {
18     this->flag=flag;
19 }
20
21 int main()
22 {
23     Data d;
24     return 0;
25 }
26
```

test.cpp: In function 'int main()':

test.cpp:23:10: error: no matching function for call to 'Data::Data()'

```
23 |     Data d;
    |         ^
```

test.cpp:16:1: note: candidate: Data::Data(bool)

# Initializer list of constructor

```
4
5 class Data {
6 public:
7     bool flag=1;
8     int value=0;
9     Data(bool flag);
10
11 void PrintInt()
12 {
13     cout << " In class A\n";
14 }
15 };
16 Data::Data(bool flag)
17 {
18     this->flag=flag;
19 }
```

```
(gdb) disassemble /s 0x800118a
Dump of assembler code for function Data::Data(bool):
test.cpp:
16     Data::Data(bool flag)
    0x000000000800118a <+0>:    endbr64
    0x000000000800118e <+4>:    push    %rbp
    0x000000000800118f <+5>:    mov     %rsp,%rbp
    0x0000000008001192 <+8>:    mov     %rdi,-0x8(%rbp)
    0x0000000008001196 <+12>:   mov     %esi,%eax
    0x0000000008001198 <+14>:   mov     %al,-0xc(%rbp)
    0x000000000800119b <+17>:   mov     -0x8(%rbp),%rax
    0x000000000800119f <+21>:   movb    $0x1,(%rax)
    0x00000000080011a2 <+24>:   mov     -0x8(%rbp),%rax
    0x00000000080011a6 <+28>:   movl    $0x0,0x4(%rax)

17     {
18         this->flag=flag;
    0x00000000080011ad <+35>:   mov     -0x8(%rbp),%rax
    0x00000000080011b1 <+39>:   movzbl  -0xc(%rbp),%edx
    0x00000000080011b5 <+43>:   mov     %dl,(%rax)

19     }
    0x00000000080011b7 <+45>:   nop
```

# Initializer list of constructor

## Mandatory Usage (ref, const, base)

```
5  class Data {
6  public:
7      bool flag=1;
8      int value=0;
9      Data(bool flag);
10
11     void PrintInt()
12     {
13         cout << " In class A\n";
14     }
15 };
16 Data::Data(bool flag):flag(flag)
17 {
18 }
```

```
22     Data a(flag);
(gdb) disassemble /s Data::Data(bool)
Dump of assembler code for function Data::Data(bool):
test.cpp:
16     Data::Data(bool flag):flag(flag)
    0x0000000000800118a <+0>:      endbr64
    0x0000000000800118e <+4>:      push    %rbp
    0x0000000000800118f <+5>:      mov     %rsp,%rbp
    0x00000000008001192 <+8>:      mov     %rdi,-0x8(%rbp)
    0x00000000008001196 <+12>:     mov     %esi,%eax
    0x00000000008001198 <+14>:     mov     %al,-0xc(%rbp)
    0x0000000000800119b <+17>:     mov     -0x8(%rbp),%rax
    0x0000000000800119f <+21>:     movzbl -0xc(%rbp),%edx
    0x000000000080011a3 <+25>:     mov     %dl,(%rax)
    0x000000000080011a5 <+27>:     mov     -0x8(%rbp),%rax
    0x000000000080011a9 <+31>:     movl    $0x0,0x4(%rax)

17     {
18     }
    0x000000000080011b0 <+38>:     nop
    0x000000000080011b1 <+39>:     pop     %rbp
    0x000000000080011b2 <+40>:     retq

End of assembler dump.
```



# What is happening here ?

```
public : Thing(int _foo, int _bar){  
    member1 = _foo;  
}
```

```
class Data {  
public:  
    bool flag=1;  
    int value=0;  
    Data(bool flag);  
  
    void PrintInt()
```

```
(gdb) disassemble /s 0x800118a  
Dump of assembler code for function Data::Data(bool):  
test.cpp:
```

```
16      Data::Data(bool flag)  
    0x000000000800118a <+0>:      endbr64  
    0x000000000800118e <+4>:      push   %rbp  
    0x000000000800118f <+5>:      mov    %rsp,%rbp  
    0x0000000008001192 <+8>:      mov    %rdi,-0x8(%rbp)  
    0x0000000008001196 <+12>:     mov    %esi,%eax  
    0x0000000008001198 <+14>:     mov    %al,-0xc(%rbp)  
    0x000000000800119b <+17>:     mov    -0x8(%rbp),%rax  
    0x000000000800119f <+21>:     push   %rax
```

```
rc/service/receiver/DiagReceiver.cpp:57:5: performance: Variable 'm_Command1_CreateUploadLog' is assigned in constructor body. Consider performing initialization in initialization list. [useInitializationList]  
    m_Command1_CreateUploadLog = std::make_shared<Command1_CreateUploadLog>();  
    ^  
rc/service/receiver/DiagReceiver.cpp:58:5: performance: Variable 'm_command2_checkNetwork' is assigned in constructor body. Consider performing initialization in initialization list. [useInitializationList]  
    m_command2_checkNetwork = std::make_shared<Command2_NetworkConnection>();  
    ^  
rc/service/receiver/DiagReceiver.cpp:59:5: performance: Variable 'm_Command3_handleLogUpload' is assigned in constructor body. Consider performing initialization in initialization list. [useInitializationList]  
    m_Command3_handleLogUpload = std::make_shared<Command3_handleLogUpload>();  
    ^  
rc/service/receiver/DiagReceiver.cpp:60:5: performance: Variable 'm_Command4_diagAction' is assigned in constructor body. Consider performing initialization in initialization list. [useInitializationList]  
    m_Command4_diagAction = std::make_shared<Command4_DiagAction>();  
    ^  
rc/service/receiver/DiagReceiver.cpp:61:5: performance: Variable 'm_Command6_playVoice' is assigned in constructor body. Consider performing initialization in initialization list. [useInitializationList]  
    m_Command6_playVoice = std::make_shared<Command6_playVoice>();  
    ^
```

cppcheck

# Delegation

```
moatasem@CAI1-L14000:~/vsomeIp$ g++ -g test.cpp
moatasem@CAI1-L14000:~/vsomeIp$ ./a.out
Default
1 Param
2 Param
moatasem@CAI1-L14000:~/vsomeIp$
```

```
};
Data::Data()
{
    std::cout << "Default" << std::endl;
}
Data::Data(bool flag):flag(flag),Data()
{
}
```

class Data  
a delegating constructor cannot have other mem-initializers C/C++(2447)  
[View Problem \(Alt+F8\)](#) [Quick Fix... \(Ctrl+.\)](#)

```
4 class Data {
5 public:
6     bool flag=1;
7     int value=0;
8     Data();
9     Data(bool flag);
10    Data(bool flag,int value);
11
12    void PrintInt()
13    {
14        cout << " In class A\n";
15    }
16 };
17
18 Data::Data()
19 {
20     std::cout << "Default" << std::endl;
21 }
22 Data::Data(bool flag):Data()
23 {
24     this->flag=flag;
25     std::cout << "1 Param" << std::endl;
26 }
27
28 Data::Data(bool flag,int value):Data(flag)
29 {
30     this->value=value;
31     std::cout << "2 Param" << std::endl;
32 }
33
```



# Delete

```
class Data {  
public:  
    bool flag=1;  
    int value=0;  
    Data()=delete;  
    Data(int flag);  
    void PrintInt()  
    {  
        cout << " In class A\n";  
    }  
};  
Data::Data()  
{  
    std::cout <<"Default"<<std::endl;  
}  
  
Data::Data(int flag)  
{  
  
}
```

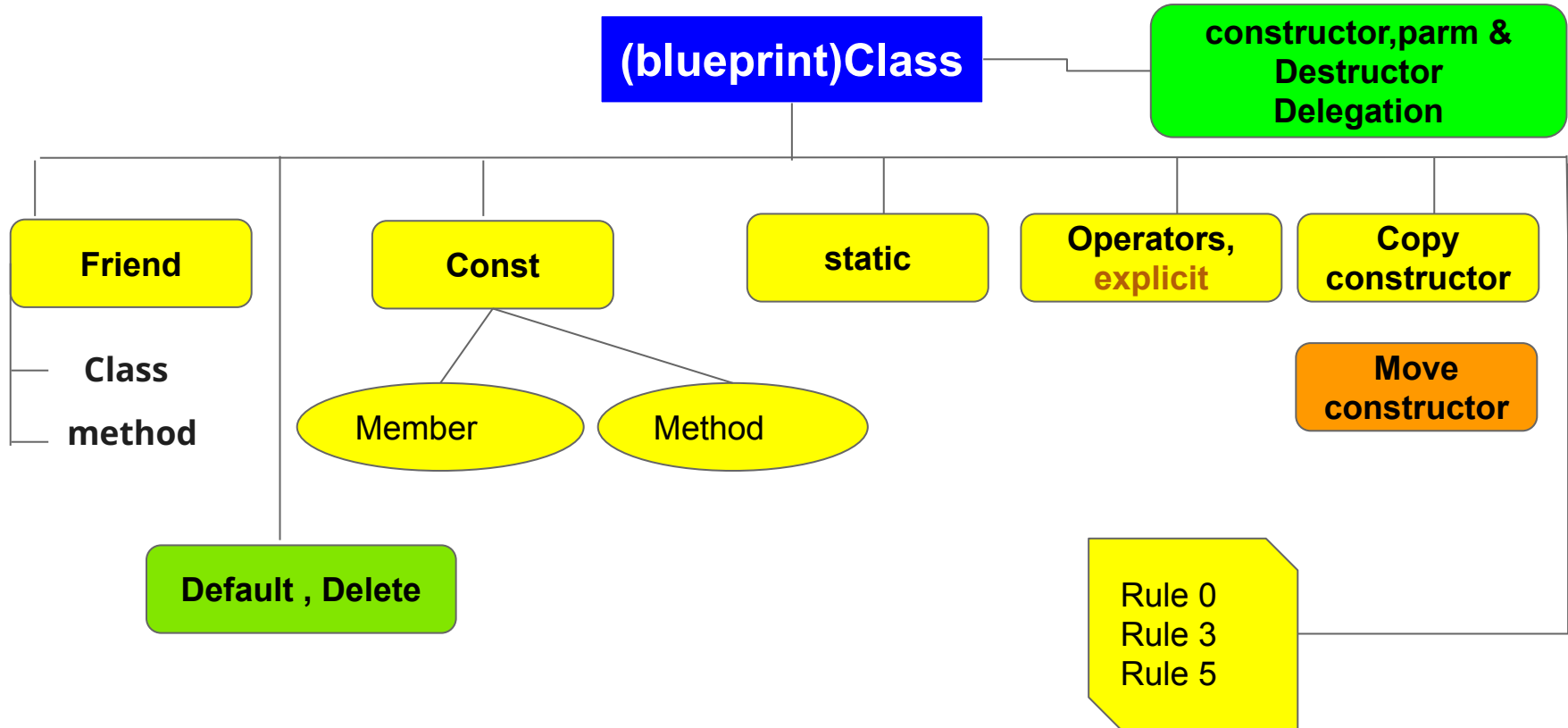
```
int main(  
{  
    Data d;  
    return 0;  
}
```

Data d

the default constructor of "Data" cannot be referenced -- it is a deleted function (

[View Problem \(Alt+F8\)](#) [Quick Fix... \(Ctrl+.\)](#)

# Features of class



# Const member

1- initialize in class itself

2- initialized in initializer list of constructor

```
class Data
{
    const int value1 ;           // 1-assign in constructor
    const int value2 = 10;       // 2-assign here it will replace with constructor

public:
    Data(int value1) : value1(value1)
    {
    }
    // Data(){}                  //3- Error should initliaize const member value1
};
```

```
test.cpp:
14      Data(int value1) : value1(value1)
    0x00000000000001236 <+0>:    endbr64
    0x0000000000000123a <+4>:    push    %rbp
    0x0000000000000123b <+5>:    mov     %rsp,%rbp
    0x0000000000000123e <+8>:    mov     %rdi,-0x8(%rbp)
    0x00000000000001242 <+12>:   mov     %esi,-0xc(%rbp)
    0x00000000000001245 <+15>:   mov     -0x8(%rbp),%rax
    0x00000000000001249 <+19>:   mov     -0xc(%rbp),%edx
    0x0000000000000124c <+22>:   mov     %edx,(%rax)
    0x0000000000000124e <+24>:   mov     -0x8(%rbp),%rax
    0x00000000000001252 <+28>:   movl    $0xa,0x4(%rax)

15      {
16      }
    0x00000000000001259 <+35>:   nop
```

```
(gdb) p &d.value1
$1 = (const int *) 0x7fffffffed9a0
(gdb) p &d.value2
$2 = (const int *) 0x7fffffffed9a4
(gdb) x/32w &d
0x7fffffffed9a0: 2      10      -1138036736
0x7fffffffed9b0: 0      0      -12763005
0x7fffffffed9c0: -10892416 32767   -75
0x7fffffffed9d0: 72704   1      134222217
```

# Const method

- 1- normal instance can call everything
- 2- const instance see only const methods
- 3- const method cannot write on attributes

## Mutable

```
private:
    mutable int x;
public:
    Data()
    {
    }
    void fun(int temp) const{
        x=10;
```

```
8  class Data
9  {
10 private:
11     int x;
12 public:
13     Data()
14     {
15     }
16     void fun(int temp) const{
17         // x=10;    //1- cant change member method
18         int value; //2- create local variable
19         value=temp; // 3- access local
20         temp=12;    // 4- access param
21         std::cout <<"Hello from const method"<<std::endl;
22     }
23     void method1(){}
24 };
25 int main()
26 {
27
28     Data d;
29     d.fun(2);    // 5- normal instance can access const
30     d.method1(); // 6- normal instance cann access non-const method
31
32     const Data k;
33     k.fun(2);    //7- const instance can access only const method
34     return 0;
35 }
```

# Overload const method

```
5 }  
6 void fun(int temp) const{  
7     std::cout <<"Hello from const method"<<std::endl;  
8 }  
9 void fun(int temp) {  
0     std::cout <<"Hello from method"<<std::endl;  
1 }  
2 void method1(){}  
3 };  
4 int main()  
5 {  
6  
7     Data d;  
8     d.fun(1); //Hello from method  
9  
0     const Data k;  
1     k.fun(1); //Hello from const method  
2  
3     return 0;  
4 }
```

```
moatasem@CAI1-L14000:~/vsomeIp$ g++ -g test.cpp -fverbose-asm -std=c++14  
moatasem@CAI1-L14000:~/vsomeIp$ objdump -t --demangle | grep fun  
00000000000012b0 w F .text 000000000000003e Data::fun(int) const  
00000000000012ee w F .text 000000000000003e Data::fun(int)  
moatasem@CAI1-L14000:~/vsomeIp$ objdump -t | grep fun  
00000000000012b0 w F .text 000000000000003e _ZNK4Data3funEi  
00000000000012ee w F .text 000000000000003e _ZN4Data3funEi  
moatasem@CAI1-L14000:~/vsomeIp$ objdump -d --demangle | grep fun  
11fc: e8 ed 00 00 00 callq 12ee <Data::fun(int)>  
1219: e8 92 00 00 00 callq 12b0 <Data::fun(int) const>  
00000000000012b0 <Data::fun(int) const>:  
00000000000012ee <Data::fun(int)>:  
moatasem@CAI1-L14000:~/vsomeIp$
```

**Rule 9-3-1 (Required) const member functions shall not return non-const pointers or references to *class-data*.**

```
class Data
{
public:
    Data(int32_t& b_) : a(new int32_t[10]), b(b_)
    {
    }
    int32_t* getA() const // Non-compliant
    // Returns non const pointer to data
    {
        return a;
    }
    int32_t* getB() const // Non-compliant
    // Returns non const pointer to data
    {
        return &b;
    }
    const int32_t* getC() const // Compliant
    // Returns const pointer to data
    {
        return &b;
    }
private:
    int32_t* a;
    int32_t& b;
};
```

**Rule 9-3-2 (Required) Member functions shall not return non-const handles to *class-data*.**

### Rationale

By implementing class interfaces with member functions the implementation retains more control over how the object state can be modified and helps to allow a class to be maintained without affecting clients. Returning a *handle* to *class-data* allows for clients to modify the state of the object without using any interfaces.

### Example

```
class C
{
public:
    int32_t & getA () // Non-compliant
    {
        return a;
    }
private:
    int32_t a;
};

void b ( C & c )
{
    int32_t & a_ref = c.getA ();
    a_ref = 10; // External modification of private C::a
}
```

# Good Practis

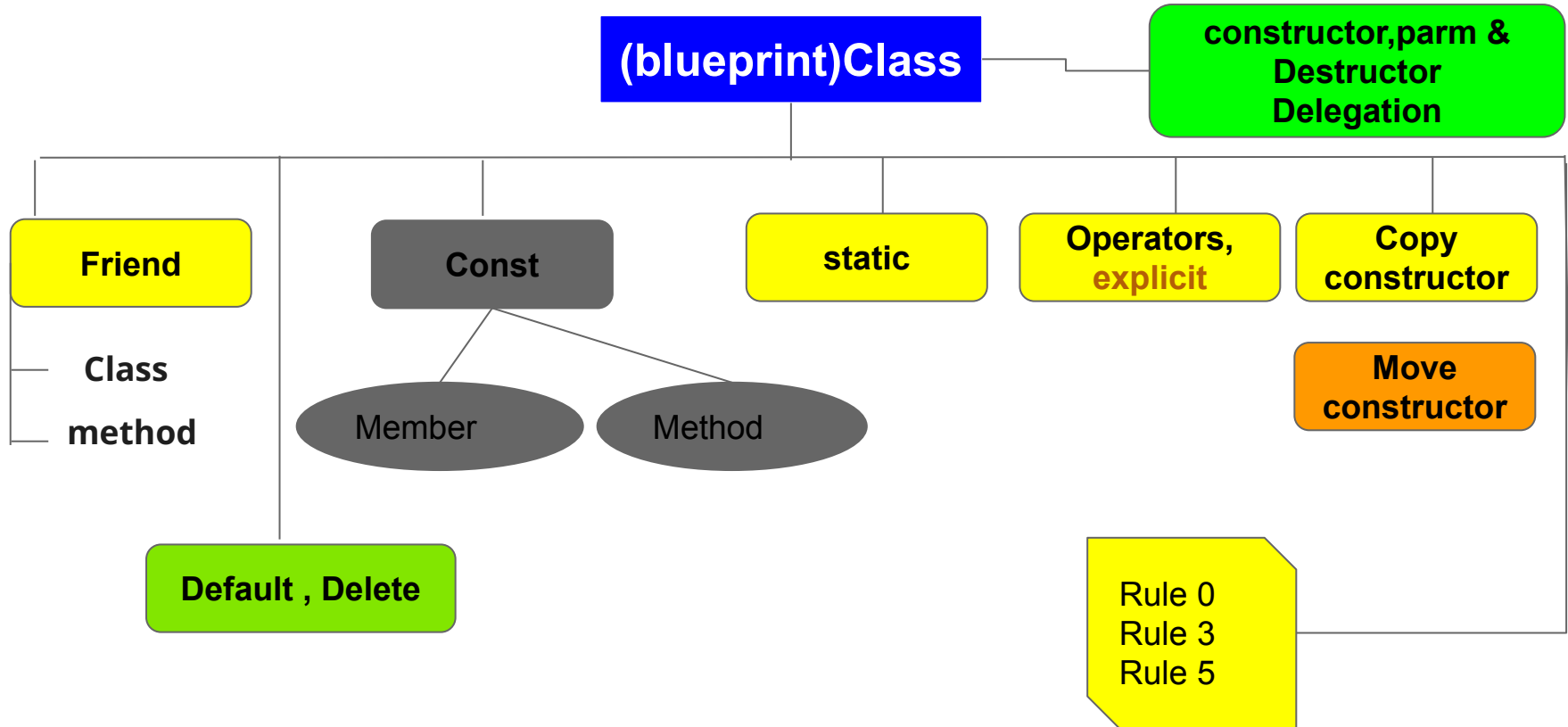
## Const as Much as Possible

`const` tells the compiler that a variable or method is immutable. This helps the compiler optimize the code and helps the developer know if a function has a side effect. Also, using `const &` prevents the compiler from copying data unnecessarily. The [comments on `const` from John Carmack](#) are also a good read.

```
// Bad Idea
class MyClass
{
public:
    void do_something(int i);
    void do_something(std::string str);
};

// Good Idea
class MyClass
{
public:
    void do_something(const int i);
    void do_something(const std::string &str);
};
```

# Features of class

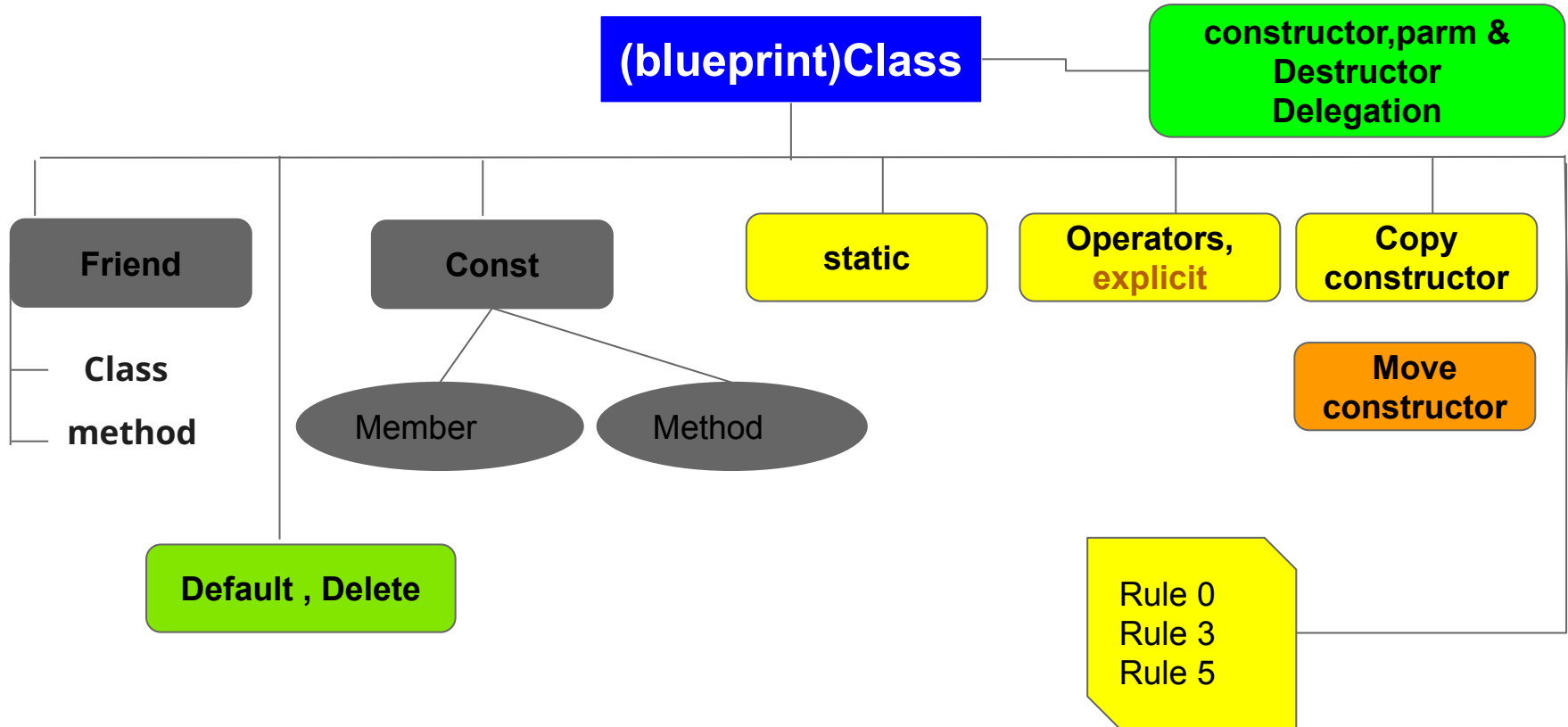




# Friend class

```
7
8  class Test; //forward decleration
9  class Data
10 {
11 public:
12     Data()
13     {
14     }
15 private:
16     friend class Test; //1- giving permission to access the private member
17     int value=0;
18 };
19 class Test{
20 public:
21     void fun(){
22         d.value=12; //2- works
23     }
24 private:
25     Data d;
26 };
```

# Features of class

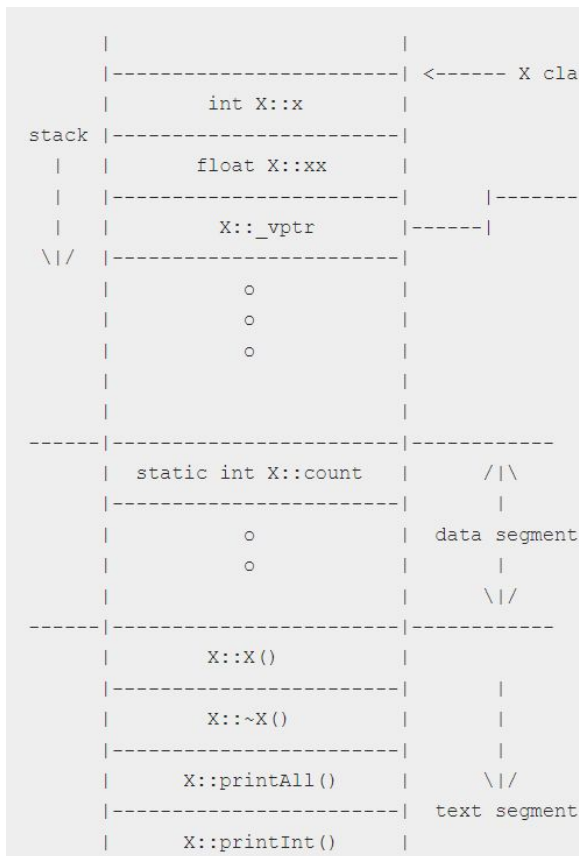


# Static member

```
9  class Data
10 {
11 public:
12     Data()
13     {
14         value++;
15     }
16     void print(){
17         std::cout <<value2<<std::endl;
18         std::cout <<value<<std::endl;
19     }
20 static int valuep;
21 private:
22     // static member not related to instance related to class itself
23     static int value;           // 1- must be defined outside
24     static const int value2=5;  // 2- can be defined here if const
25     // static const int value3; //3- ERROR Data(int m):value3(m)
26
27 };
28 int Data::value=0;
29 int Data::valuep=1000;
30
31 int main()
32 {
33     Data d;
34     Data d2;
35     d.print();
36     std::cout << Data::valuep; // 4-public can access with Data::valuep
37     std::cout << d.valuep;     // 5- public can access with object
38 }
```

# Layout memory

```
class X {  
    int      x;  
    float    xx;  
    static int count;  
  
public:  
    X() {}  
    virtual ~X() {}  
  
    virtual void printAll() {}  
    void printInt() {}  
    void printFloat() {}  
    static void printCount() {}  
};
```



# BSS / Data Segment

```
int Data::value=10; |
int Data::valuep=1000;

00000000000001238 1      r .text 000000000000000019      _GLOBAL__sub_I_
00000000000004014 g      0 .data 000000000000000004      Data::valuep
00000000000004010 g      0 .data 000000000000000004      Data::value
moatasem@CAI1-L14000:~/vsomeIp$
```

```
int Data::value=0;
int Data::valuep=0;

00000000000001298 1      r .text 000000000000000019      _GLOBAL__sub_I_
00000000000004158 g      0 .bss 000000000000000004      Data::valuep
00000000000004154 g      0 .bss 000000000000000004      Data::value
moatasem@CAI1-L14000:~/vsomeIp$
```

# Static const

## Constant static members

If a static data member of integral or enumeration type is declared `const` (and not `volatile`), it can be initialized with an initializer in which every expression is a constant expression, right inside the class definition:

```
struct X
{
    const static int n = 1;
    const static int m{2}; // since C++11
    const static int k;
};
const int X::k = 3;
```

```
4 static const int value2; // 2- can be defined here if const
5 // static const int value3; //3- ERROR Data(int m):value3(m)
6
7 };
8 int Data::value=0;
9 int Data::valuep=0;
10 const int Data::value2=0;
```

000000000000020b0	g	0 .rodata	0000000000000004	Data::value2
00000000000004154	g	0 .bss	0000000000000004	Data::value

```
10 ~ class foo
11 {
12     public :
13         static const int value=1 ;
14 };
15
16 ~ int main()
17 {
18     foo d;
```

```
1 moatasem@CAI1-L14000:~/vsomeIp$ objdump -t --demangle | grep value
2 moatasem@CAI1-L14000:~/vsomeIp$
```

# Tricky

C++

```
#include <iostream>

class foo
{
public :
    static const int f = 5;
};

void bar(const int& b)
{

}

int main()
{
    bar(foo::f); //undefined reference to foo::f
}
```

```
class foo
{
public :
    static const int f ;
};
const int foo::f = 5;
void bar(const int& b)
{

}
```

.rodata foo::f

```
int main()
{
    bar(foo::f);
}
```

```
class Data
{
public:
    static const int vtest = 10;
};
int main()
{
    Data d;
    std::cout << Data::vtest << std::endl; // 10
}
```

It will be const expr in assembly , not represent as Lvalue

# Static method

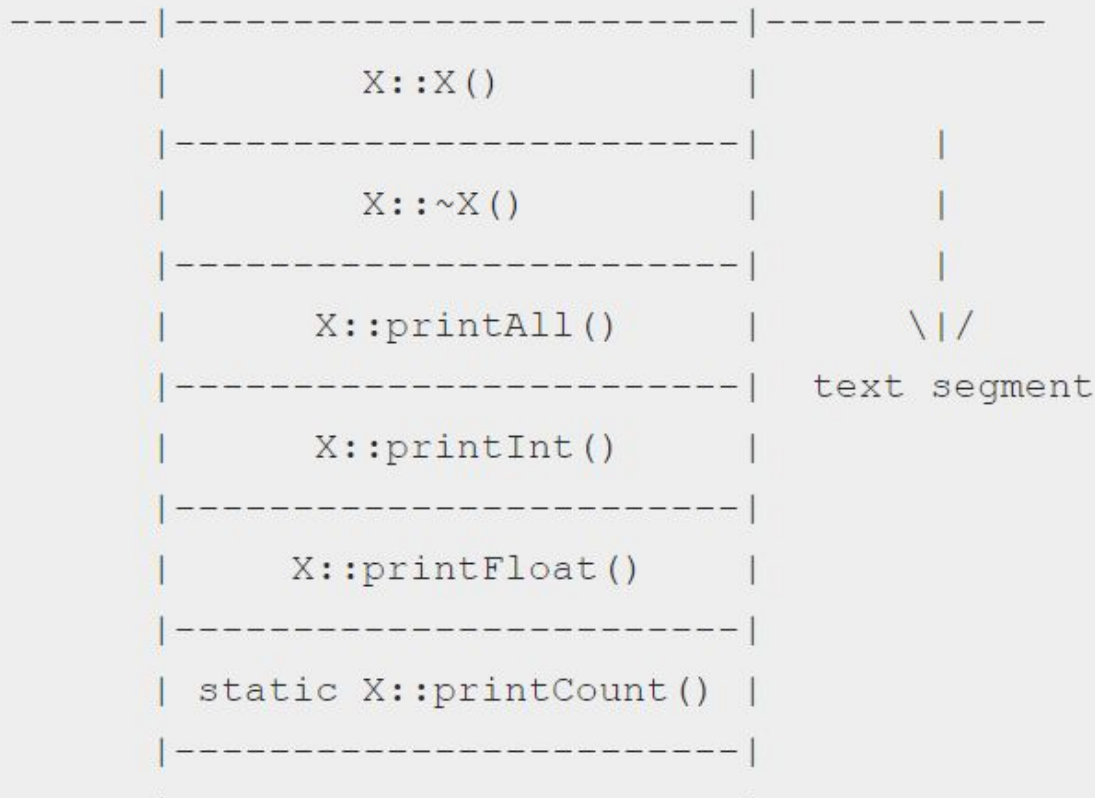
```
0  class foo
1  {
2  private:
3      int m_num=12;
4      static int temp;
5  public :
6      static void fun(){
7          int value=12;    //1- create local
8          // m_num=123;    //2-Error cannot access this
9          temp++;          //3- access static members
10     }
11 };
12 int foo::temp=1;
13 int main()
14 {
15     foo d;
16     foo::fun();           //4- calling from class
17     d.fun();              //5- calling from instance
18 }
```

```
20     }
21     static void test();
22 };
23 int foo::temp=1; // no static word
24 void foo::test() // no static word
25 {
26
27 }
28
```

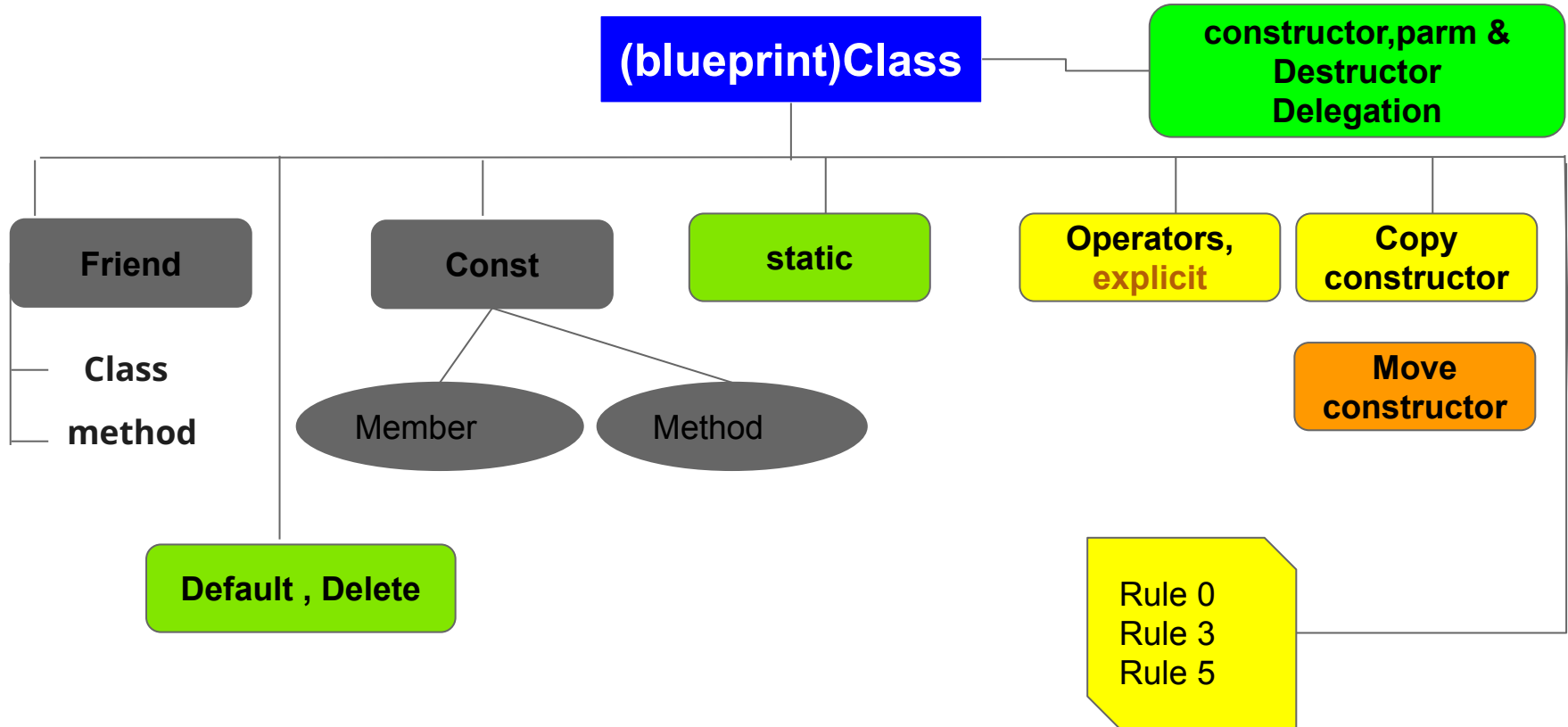


# Memory layout

```
class X {  
    int      x;  
    float    xx;  
    static int count;  
  
public:  
    X() {}  
    virtual ~X() {}  
  
    virtual void printAll() {}  
    void printInt() {}  
    void printFloat() {}  
    static void printCount() {}  
};
```



# Features of class



# Q&A

## Parsing

```
std::string frame="Moatasem 1234 99.9";
```

```
std::string name;  
int ID ;  
float score ;  
std::string frame="Moatasem 1234 99.9";  
std::stringstream st(frame);
```

```
st>>name>>ID>>score;
```

```
std::cout <<name<<std::endl; //Moatasem  
std::cout <<ID<<std::endl; //1234  
std::cout <<score<<std::endl; //99.9
```

## regex

```
//string to be searched
```

```
std::string mystr = "Email: moatasem.elsayed@valeo.com ,Mobile:01112932885";
```

```
moatasem@CAI11-L14000:~/vsomeIp$  
Email: moatasem.elsayed@valeo.com  
Mobile:01112932885
```

# Legacy code

```
#include <iostream>
#include <algorithm>
#include <string.h>
#include <vector>
extern "C"
{
    #include "cfile.h"
}
using namespace std;

void fun(int x)
{
    std::cout <<"hello world"<<std::endl;
}
void fun(int x , int y)
{
    std::cout <<"hello world"<<std::endl;
}

int main()
{
    fun(2);
    fun(2,3);
    display();

    return 0;
}
```

```
moatasem@CAI1-L14000:~/vsomeIp$ objdump -t --demangle | grep 'fun\|display'
00000000000012d1 l      F .text 0000000000000019      _GLOBAL__sub_I__Z3funi
00000000000011c9 g      F .text 0000000000000017      display
000000000000121a g      F .text 000000000000003d      fun(int, int)
00000000000011e0 g      F .text 000000000000003a      fun(int)
moatasem@CAI1-L14000:~/vsomeIp$ objdump -t | grep 'fun\|display'
00000000000012d1 l      F .text 0000000000000019      _GLOBAL__sub_I__Z3funi
00000000000011c9 g      F .text 0000000000000017      display
000000000000121a g      F .text 000000000000003d      _Z3funii
00000000000011e0 g      F .text 000000000000003a      _Z3funi
moatasem@CAI1-L14000:~/vsomeIp$
```

## Q&A

### Convert from decimal to binary?

```
12
13 int main()
14 {
15     int x=10;
16     std::cout <<std::bitset<8>(x)<<std::endl;
17     return 0;
18 }
```

### Find Number in array ?

```
int s[] = {5, 7, 4, 2, 8, 6, 1, 9, 0, 3};

auto v=std::find(std::begin(s), std::end(s),5);
std::cout <<*v<<std::endl;//5
```

### Convert from string to data type ?

```
auto i=std::stol("123");
std::cout << typeid(i).name()<<std::endl;//Long
```

```
std::stoi
return 0
std
stof
stoi
stol
stold
stoll
stoul
stoull
```

### Compare Struct

```
struct Data{
    int x;
    int y;
    bool operator==(const Data& temp)
    {
        return( (temp.x== x) && (temp.y == y));
    }
}d1,d2;
int main()
{
    if(d1 == d2)//Equal
    {
        std::cout <<"Equal"<<std::endl;
    }
}
```

# Method looks like more abstract

```
bool equal_strings(const string& lhs, const string& rhs) {  
    // Copy the arguments  
    string lhs_copy{lhs};  
    string rhs_copy{rhs};  
  
    // Convert to upper case  
    transform(begin(lhs_copy), end(lhs_copy), begin(lhs_copy), ::toupper);  
    transform(begin(rhs_copy), end(rhs_copy), begin(rhs_copy), ::toupper);  
  
    // Compare the results  
    return lhs_copy == rhs_copy;  
}
```

# Tasks

- check if the character is digit ?
- check if all the array is even ?
- check if there is any value of array is even ?
- write string class which has**  
**Members { length - string}**

# Tasks

1-handle interrupt signal like (ctrl+c)

2-

```
1 #include <iostream>
2 int &f() {
3     static int x=0;
4     std::cout <<x<<std::endl;
5     return x;
6 }
7 main() {
8     f() = 10;
9     f()=0;
10 }
```

3- fill array from 10 to 10000 sequentially

```
{10, 11, 12, 13, 14, ...}
```

4-try and / or

```
int main()
{
    int a = 10;
    if (a < 20 and a > 5)
        cout << "Yes";
    return 0;
}
```

5- calculate accumulate of array

```
>>> 1+2+3+4+5+6+7+8+9
45
```



# Tasks

Create A class that can be use to make backtrace  
for calling functions

```
moatasem@CA11-L14000:~/sta
Enter to [main]
Enter to [fun1]
Enter to [fun2]
Enter to [fun3]
Backtrace as follows :
0- main
1- fun1
2- fun2
3- fun3
Back Trace is Finished
Exit From [fun3]
Exit From [fun2]
Exit From [fun1]
Exit From [main]
```

```
2 #include "backtrace.hpp"
3 void fun2(int x);
4 void fun3(int x);
5 void fun1(int x)
6 {
7     EnterFn;
8     fun2(2);
9     ExitFn;
10 }
11 void fun2(int x)
12 {
13     EnterFn;
14     fun3(3);
15
16     ExitFn;
17 }
18 void fun3(int x)
19 {
20     EnterFn;
21     PRINT_BT;
22
23     ExitFn;
24 }
25
26 int main()
27 {
28     EnterFn;
29     fun1(3);
30
31     ExitFn;
32 }
```

# references

1-<http://www.vishalchovatiya.com/>

2-MISRA-CPP-2008-STANDARD.pdf

3-[https://lefticus.gitbooks.io/cpp-best-practices/content/05-Considering\\_Maintainability.html](https://lefticus.gitbooks.io/cpp-best-practices/content/05-Considering_Maintainability.html)

4-[www.fluentcpp.com](http://www.fluentcpp.com)