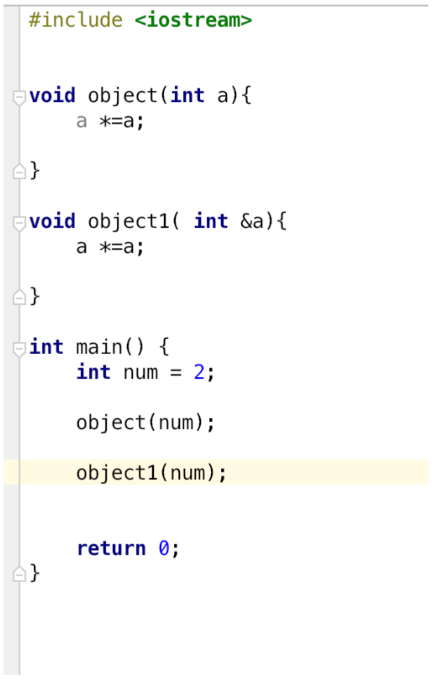
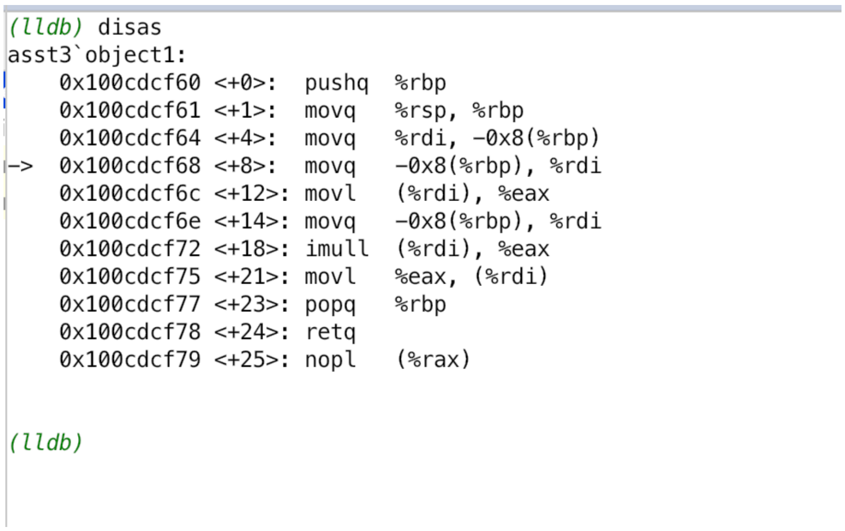
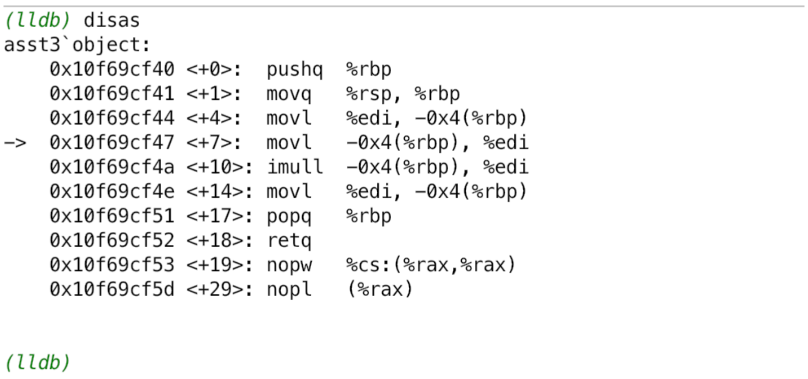
3.



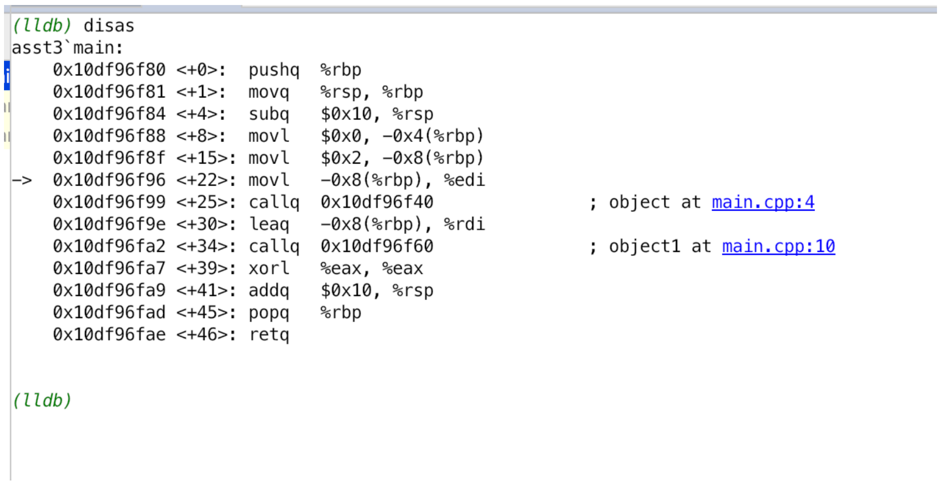
```cpp
#include <iostream>


void object(int a){
    a *=a;

}

void object1( int &a){
    a *=a;

}

int main() {
    int num = 2;

    object(num);

    object1(num);


    return 0;
}
```



```
(lldb) disas
asst3`object1:
    0x100cdcf60 <+0>:  pushq  %rbp
    0x100cdcf61 <+1>:  movq   %rsp, %rbp
    0x100cdcf64 <+4>:  movq   %rdi, -0x8(%rbp)
->  0x100cdcf68 <+8>:  movq   -0x8(%rbp), %rdi
    0x100cdcf6c <+12>: movl   (%rdi), %eax
    0x100cdcf6e <+14>: movq   -0x8(%rbp), %rdi
    0x100cdcf72 <+18>: imull  (%rdi), %eax
    0x100cdcf75 <+21>: movl   %eax, (%rdi)
    0x100cdcf77 <+23>: popq   %rbp
    0x100cdcf78 <+24>: retq
    0x100cdcf79 <+25>: nopl   (%rax)


(lldb)
```



```
(lldb) disas
asst3`object:
    0x10f69cf40 <+0>:  pushq  %rbp
    0x10f69cf41 <+1>:  movq   %rsp, %rbp
    0x10f69cf44 <+4>:  movl   %edi, -0x4(%rbp)
->  0x10f69cf47 <+7>:  movl   -0x4(%rbp), %edi
    0x10f69cf4a <+10>: imull  -0x4(%rbp), %edi
    0x10f69cf4e <+14>: movl   %edi, -0x4(%rbp)
    0x10f69cf51 <+17>: popq   %rbp
    0x10f69cf52 <+18>: retq
    0x10f69cf53 <+19>: nopw   %cs:(%rax,%rax)
    0x10f69cf5d <+29>: nopl   (%rax)


(lldb)
```



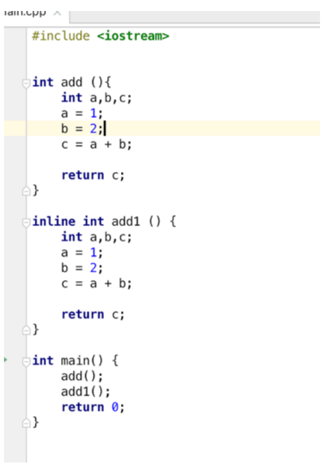
```
(lldb) disas
asst3`main:
    0x10df96f80 <+0>:  pushq  %rbp
    0x10df96f81 <+1>:  movq   %rsp, %rbp
    0x10df96f84 <+4>:  subq   $0x10, %rsp
    0x10df96f88 <+8>:  movl   $0x0, -0x4(%rbp)
    0x10df96f8f <+15>: movl   $0x2, -0x8(%rbp)
->  0x10df96f96 <+22>: movl   -0x8(%rbp), %edi
    0x10df96f99 <+25>: callq  0x10df96f40               ; object at main.cpp:4
    0x10df96f9e <+30>: leaq   -0x8(%rbp), %rdi
    0x10df96fa2 <+34>: callq  0x10df96f60               ; object1 at main.cpp:10
    0x10df96fa7 <+39>: xorl   %eax, %eax
    0x10df96fa9 <+41>: addq   $0x10, %rsp
    0x10df96fad <+45>: popq   %rbp
    0x10df96fae <+46>: retq


(lldb)
```

The referenced parameter passing is the passed address, which belongs to the address delivery.

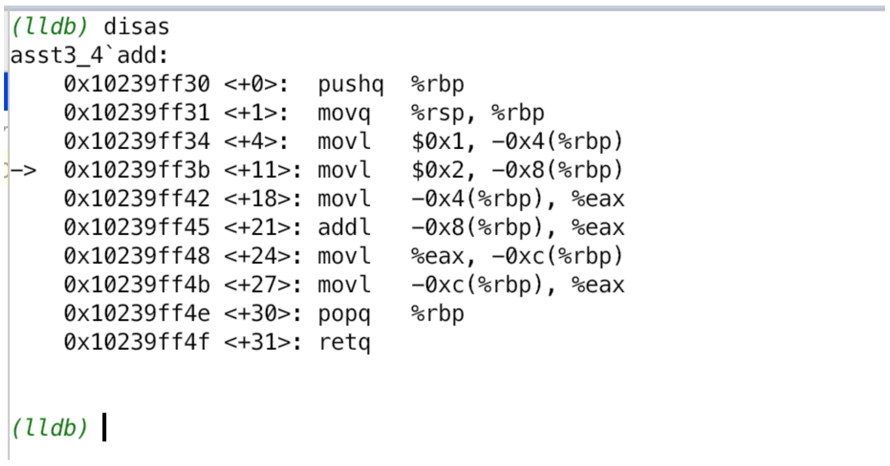
The value passed is a value. In the case of value passing, the address of the stack of the main function and the sub function is not the same.

Referenced parameter passing use Lea, and Lea will pass the address like&. In the second picture shows register %rdi keep the address of –(0x8) %rbp, and(%rdi) will show the value of the address. Value parameter passing only use Mov to pass the value. Since the address of the stack of the main function and the sub function is not the same. The value “a” only do “a*=a in sub” function, and when back to main function the address change, so a does not change in main function. However, if we pass the address the address will keep same, and value “a” can change in both functions.

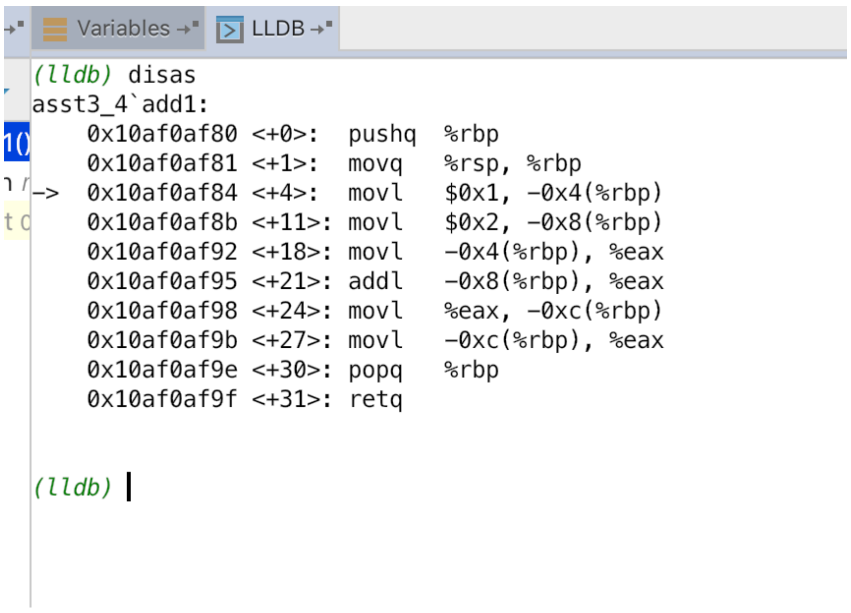
4.



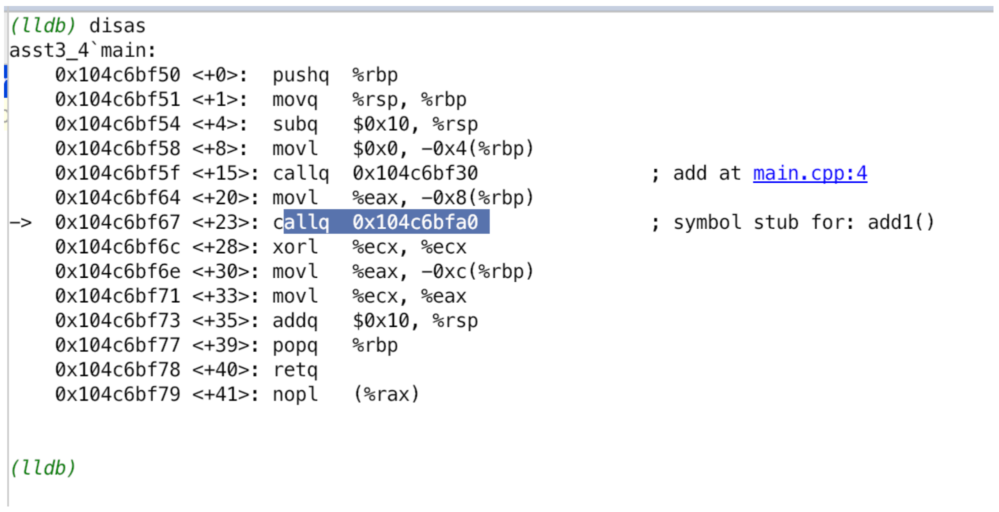
```
#include <iostream>


int add (){
    int a,b,c;
    a = 1;
    b = 2;
    c = a + b;

    return c;
}

inline int add1 () {
    int a,b,c;
    a = 1;
    b = 2;
    c = a + b;

    return c;
}

int main() {
    add();
    add1();
    return 0;
}
```



```
(lldb) disas
asst3_4`add:
    0x10239ff30 <+0>:  pushq  %rbp
    0x10239ff31 <+1>:  movq   %rsp, %rbp
    0x10239ff34 <+4>:  movl   $0x1, -0x4(%rbp)
->  0x10239ff3b <+11>: movl   $0x2, -0x8(%rbp)
    0x10239ff42 <+18>: movl   -0x4(%rbp), %eax
    0x10239ff45 <+21>: addl   -0x8(%rbp), %eax
    0x10239ff48 <+24>: movl   %eax, -0xc(%rbp)
    0x10239ff4b <+27>: movl   -0xc(%rbp), %eax
    0x10239ff4e <+30>: popq   %rbp
    0x10239ff4f <+31>: retq


(lldb)
```



```
(lldb) disas
asst3_4`add1:
    0x10af0af80 <+0>:  pushq  %rbp
    0x10af0af81 <+1>:  movq   %rsp, %rbp
->  0x10af0af84 <+4>:  movl   $0x1, -0x4(%rbp)
    0x10af0af8b <+11>: movl   $0x2, -0x8(%rbp)
    0x10af0af92 <+18>: movl   -0x4(%rbp), %eax
    0x10af0af95 <+21>: addl   -0x8(%rbp), %eax
    0x10af0af98 <+24>: movl   %eax, -0xc(%rbp)
    0x10af0af9b <+27>: movl   -0xc(%rbp), %eax
    0x10af0af9e <+30>: popq   %rbp
    0x10af0af9f <+31>: retq


(lldb)
```



```
(lldb) disas
asst3_4`main:
    0x104c6bf50 <+0>:  pushq  %rbp
    0x104c6bf51 <+1>:  movq   %rsp, %rbp
    0x104c6bf54 <+4>:  subq   $0x10, %rsp
    0x104c6bf58 <+8>:  movl   $0x0, -0x4(%rbp)
    0x104c6bf5f <+15>: callq  0x104c6bf30               ; add at main.cpp:4
    0x104c6bf64 <+20>: movl   %eax, -0x8(%rbp)
->  0x104c6bf67 <+23>: callq  0x104c6bfa0               ; symbol stub for: add1()
    0x104c6bf6c <+28>: xorl   %ecx, %ecx
    0x104c6bf6e <+30>: movl   %eax, -0xc(%rbp)
    0x104c6bf71 <+33>: movl   %ecx, %eax
    0x104c6bf73 <+35>: addq   $0x10, %rsp
    0x104c6bf77 <+39>: popq   %rbp
    0x104c6bf78 <+40>: retq
    0x104c6bf79 <+41>: nopl   (%rax)


(lldb)
```

The inline function reduces the parameter passing process. It's faster than normal function. The inline function uses different call, and it likes code the function in the internal of main function not external. It uses a symbol to replace a label.