Student Name: 杨轶涵

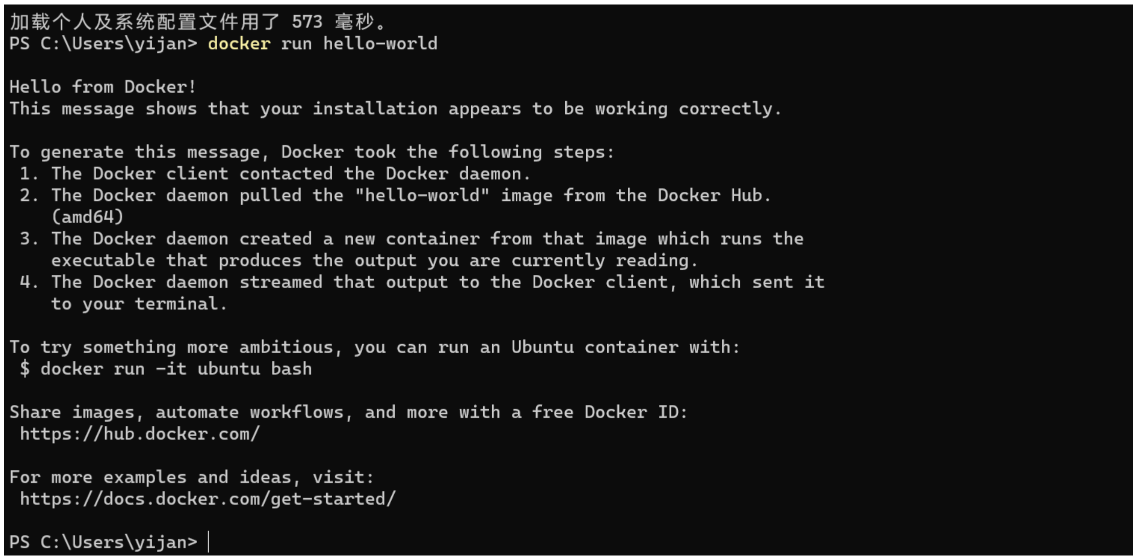
Student ID: 2022141410379

面向对象程序设计

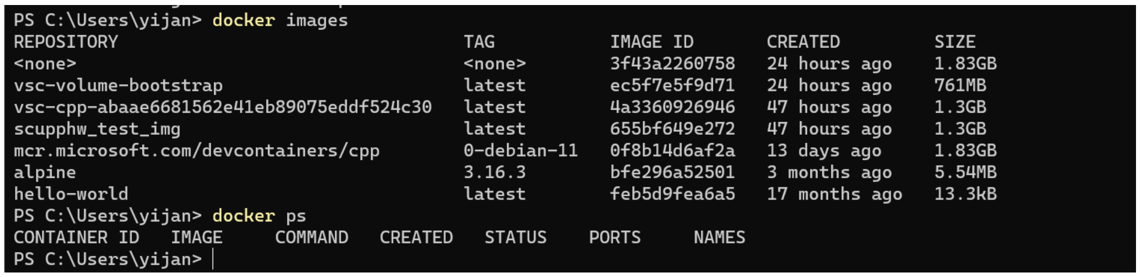
Assignment 1

1 配置对应的 docker 环境

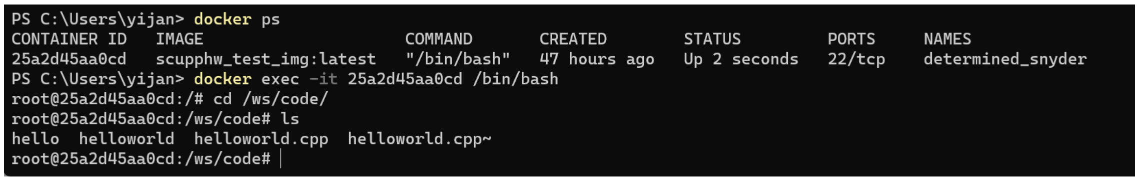
命令运行结果：



```
加载个人及系统配置文件用了 573 毫秒。
PS C:\Users\yijan> docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

PS C:\Users\yijan>
```



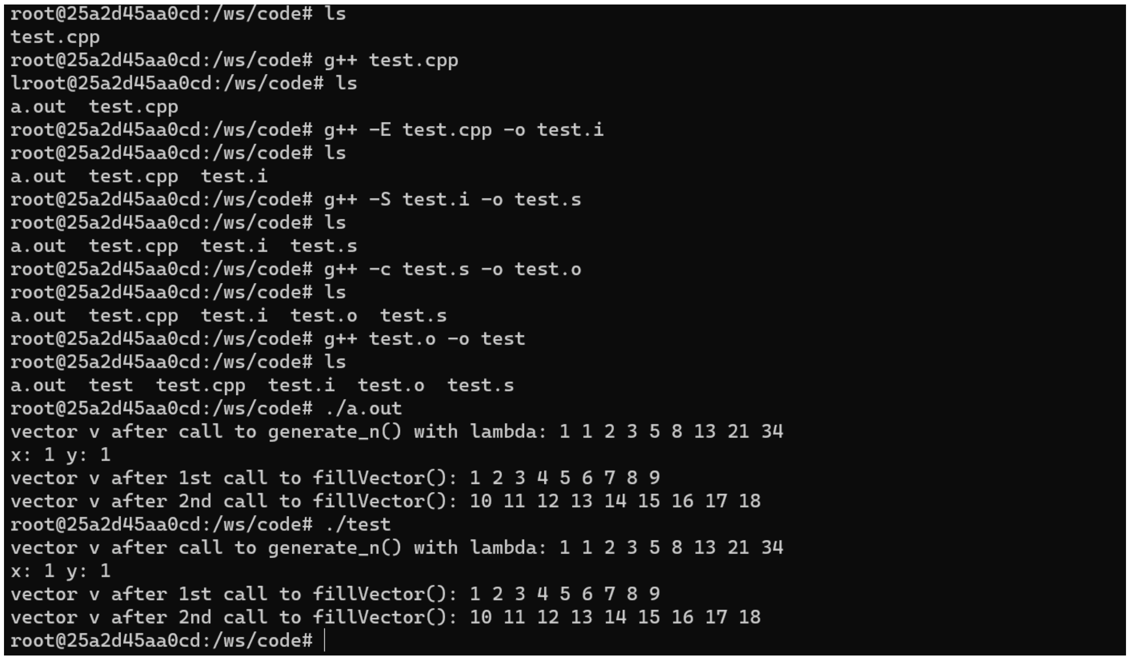
```
PS C:\Users\yijan> docker images
REPOSITORY                                   TAG          IMAGE ID       CREATED          SIZE
<none>                                       <none>       3f43a2260758   24 hours ago     1.83GB
vsc-volume-bootstrap                         latest       ec5f7e5f9d71   24 hours ago     761MB
vsc-cpp-abaae6681562e41eb89075eddf524c30     latest       4a3360926946   47 hours ago     1.3GB
scupphw_test_img                             latest       655bf649e272   47 hours ago     1.3GB
mcr.microsoft.com/devcontainers/cpp          0-debian-11  0f8b14d6af2a   13 days ago      1.83GB
alpine                                       3.16.3       bfe296a52501   3 months ago     5.54MB
hello-world                                  latest       feb5d9fea6a5   17 months ago    13.3kB
PS C:\Users\yijan> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\yijan>
```



```
PS C:\Users\yijan> docker ps
CONTAINER ID   IMAGE                     COMMAND       CREATED        STATUS         PORTS     NAMES
25a2d45aa0cd   scupphw_test_img:latest   "/bin/bash"   47 hours ago   Up 2 seconds   22/tcp    determined_snyder
PS C:\Users\yijan> docker exec -it 25a2d45aa0cd /bin/bash
root@25a2d45aa0cd:/# cd /ws/code/
root@25a2d45aa0cd:/ws/code# ls
hello  helloworld  helloworld.cpp  helloworld.cpp~
root@25a2d45aa0cd:/ws/code#
```

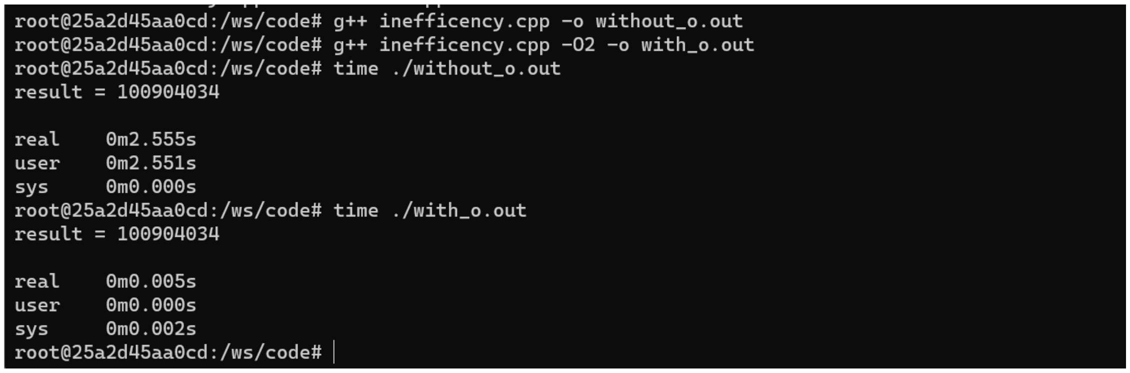
2 利用 g++ 编译单文件

过程如图:



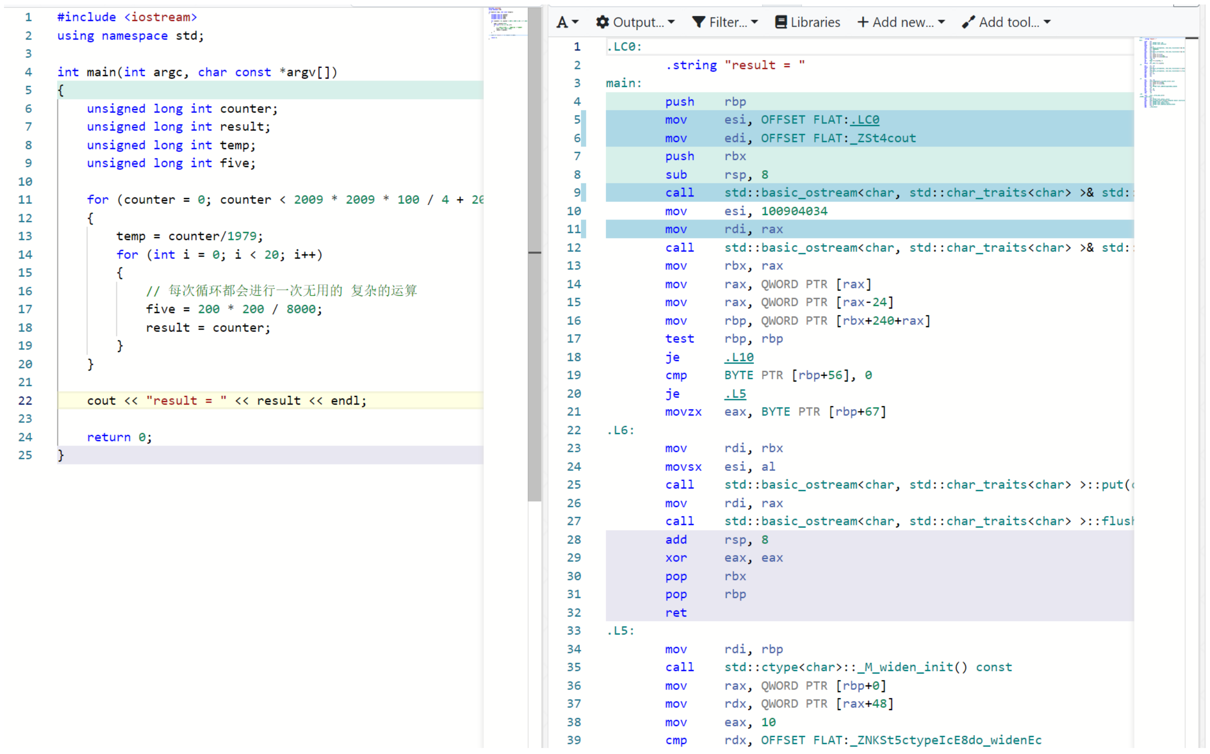
```
root@25a2d45aa0cd:/ws/code# ls
test.cpp
root@25a2d45aa0cd:/ws/code# g++ test.cpp
lroot@25a2d45aa0cd:/ws/code# ls
a.out  test.cpp
root@25a2d45aa0cd:/ws/code# g++ -E test.cpp -o test.i
root@25a2d45aa0cd:/ws/code# ls
a.out  test.cpp  test.i
root@25a2d45aa0cd:/ws/code# g++ -S test.i -o test.s
root@25a2d45aa0cd:/ws/code# ls
a.out  test.cpp  test.i  test.s
root@25a2d45aa0cd:/ws/code# g++ -c test.s -o test.o
root@25a2d45aa0cd:/ws/code# ls
a.out  test.cpp  test.i  test.o  test.s
root@25a2d45aa0cd:/ws/code# g++ test.o -o test
root@25a2d45aa0cd:/ws/code# ls
a.out  test  test.cpp  test.i  test.o  test.s
root@25a2d45aa0cd:/ws/code# ./a.out
vector v after call to generate_n() with lambda: 1 1 2 3 5 8 13 21 34
x: 1 y: 1
vector v after 1st call to fillVector(): 1 2 3 4 5 6 7 8 9
vector v after 2nd call to fillVector(): 10 11 12 13 14 15 16 17 18
root@25a2d45aa0cd:/ws/code# ./test
vector v after call to generate_n() with lambda: 1 1 2 3 5 8 13 21 34
x: 1 y: 1
vector v after 1st call to fillVector(): 1 2 3 4 5 6 7 8 9
vector v after 2nd call to fillVector(): 10 11 12 13 14 15 16 17 18
root@25a2d45aa0cd:/ws/code# 
```

打开与不打开 O2 优化时间对比:



```
root@25a2d45aa0cd:/ws/code# g++ inefficency.cpp -o without_o.out
root@25a2d45aa0cd:/ws/code# g++ inefficency.cpp -O2 -o with_o.out
root@25a2d45aa0cd:/ws/code# time ./without_o.out
result = 100904034

real    0m2.555s
user    0m2.551s
sys     0m0.000s
root@25a2d45aa0cd:/ws/code# time ./with_o.out
result = 100904034

real    0m0.005s
user    0m0.000s
sys     0m0.002s
root@25a2d45aa0cd:/ws/code# 
```

可以从汇编发现，在打开 O2 优化时无用的计算部分直接被跳过。



```cpp
#include <iostream>
using namespace std;

int main(int argc, char const *argv[])
{
    unsigned long int counter;
    unsigned long int result;
    unsigned long int temp;
    unsigned long int five;

    for (counter = 0; counter < 2009 * 2009 * 100 / 4 + 20
    {
        temp = counter/1979;
        for (int i = 0; i < 20; i++)
        {
            // 每次循环都会进行一次无用的 复杂的运算
            five = 200 * 200 / 8000;
            result = counter;
        }
    }

    cout << "result = " << result << endl;

    return 0;
}
```

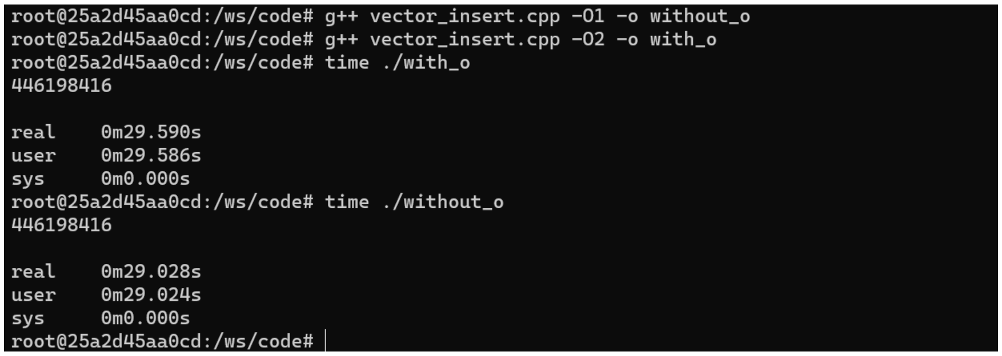
```asm
.LC0:
        .string "result = "
main:
        push    rbp
        mov     esi, OFFSET FLAT:.LC0
        mov     edi, OFFSET FLAT:_ZSt4cout
        push    rbx
        sub     rsp, 8
        call    std::basic_ostream<char, std::char_traits<char> >& std::
        mov     esi, 100904034
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char> >& std::
        mov     rbx, rax
        mov     rax, QWORD PTR [rax]
        mov     rax, QWORD PTR [rax-24]
        mov     rbp, QWORD PTR [rbx+240+rax]
        test    rbp, rbp
        je      .L10
        cmp     BYTE PTR [rbp+56], 0
        je      .L5
        movzx   eax, BYTE PTR [rbp+67]
.L6:
        mov     rdi, rbx
        movsx   esi, al
        call    std::basic_ostream<char, std::char_traits<char> >::put(
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char> >::flus
        add     rsp, 8
        xor     eax, eax
        pop     rbx
        pop     rbp
        ret
.L5:
        mov     rdi, rbp
        call    std::ctype<char>::_M_widen_init() const
        mov     rax, QWORD PTR [rbp+0]
        mov     rdx, QWORD PTR [rax+48]
        mov     eax, 10
        cmp     rdx, OFFSET FLAT:_ZNKSt5ctypeIcE8do_widenEc
```

其他实验

- 尝试对大小为 5×10^5 的 vector 进行 5×10^5 次向前插入数：

```cpp
#include <vector>
#include <iostream>
#include <numeric>

int main() {
    std::vector<int> vec( 500000 , 0 );
    for( int i = 1 ; i <= 500000 ; ++ i )
        vec.insert( vec.begin() , i );
    std::cout << std::accumulate( vec.begin() , vec.end() , 0 ) << std::endl;
}
```

运行结果：



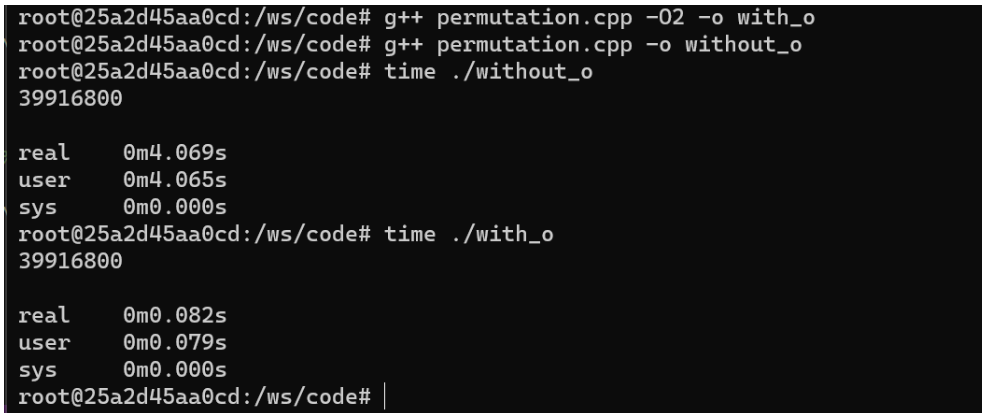
```
root@25a2d45aa0cd:/ws/code# g++ vector_insert.cpp -O1 -o without_o
root@25a2d45aa0cd:/ws/code# g++ vector_insert.cpp -O2 -o with_o
root@25a2d45aa0cd:/ws/code# time ./with_o
446198416

real    0m29.590s
user    0m29.586s
sys     0m0.000s
root@25a2d45aa0cd:/ws/code# time ./without_o
446198416

real    0m29.028s
user    0m29.024s
sys     0m0.000s
root@25a2d45aa0cd:/ws/code#
```

可以看出对于 `vector<T,Allocator>::insert` 优化几乎没有。

- 尝试通过 `next_permutation` 计数排列

```cpp
#include <iostream>
#include <algorithm>
#include <numeric>
#include <vector>

int main() {
    std::vector<int> vec( 11 , 0 );
    std::iota( vec.begin() , vec.end() , 1 );
    int cnt = 0;
    do {
        ++ cnt;
    } while( std::next_permutation( vec.begin() , vec.end() ) );
    std::cout << cnt << std::endl;
}
```

运行结果:



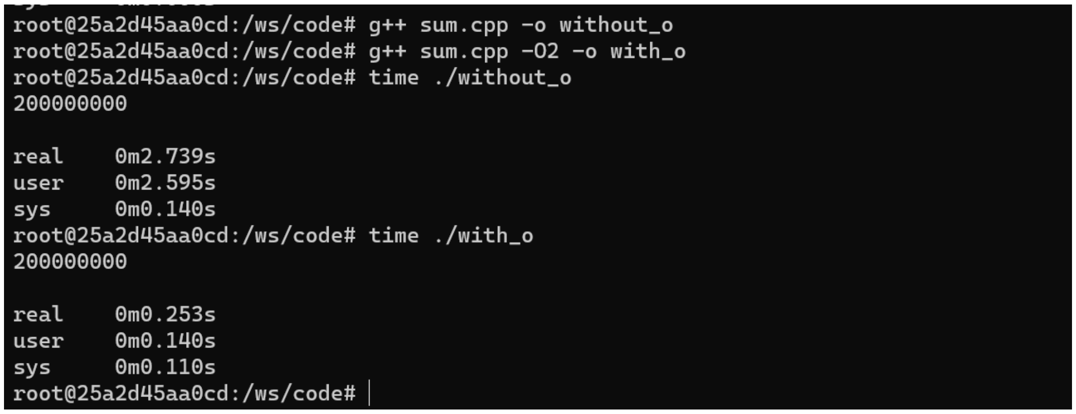
```
root@25a2d45aa0cd:/ws/code# g++ permutation.cpp -O2 -o with_o
root@25a2d45aa0cd:/ws/code# g++ permutation.cpp -o without_o
root@25a2d45aa0cd:/ws/code# time ./without_o
39916800

real    0m4.069s
user    0m4.065s
sys     0m0.000s
root@25a2d45aa0cd:/ws/code# time ./with_o
39916800

real    0m0.082s
user    0m0.079s
sys     0m0.000s
root@25a2d45aa0cd:/ws/code#
```

可以看出编译器对 `next_permutation` 有较优的优化。

- 对 2×10^8 个 1 求和

```cpp
#include <iostream>
#include <algorithm>
#include <numeric>
#include <vector>

int main() {
    std::vector<int> vec( 200000000 , 1 );
    std::cout << std::accumulate( vec.begin() , vec.end() , 0 ) << std::endl;
}
```

运行结果：



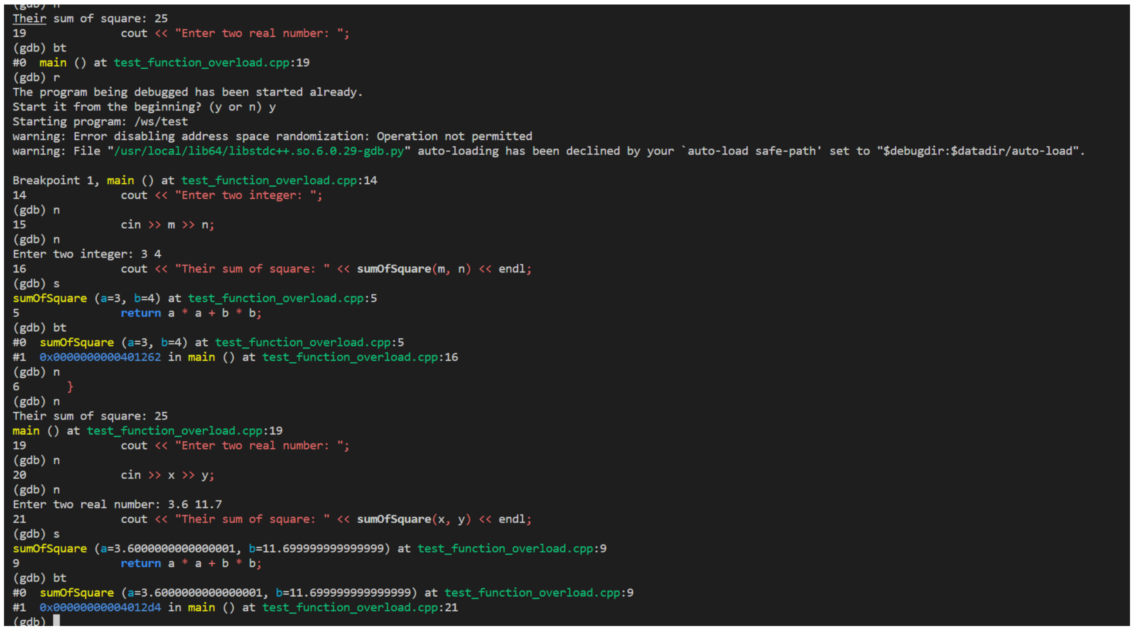
```
root@25a2d45aa0cd:/ws/code# g++ sum.cpp -o without_o
root@25a2d45aa0cd:/ws/code# g++ sum.cpp -O2 -o with_o
root@25a2d45aa0cd:/ws/code# time ./without_o
200000000

real    0m2.739s
user    0m2.595s
sys     0m0.140s
root@25a2d45aa0cd:/ws/code# time ./with_o
200000000

real    0m0.253s
user    0m0.140s
sys     0m0.110s
root@25a2d45aa0cd:/ws/code#
```

3 利用 gdb 调试文件

3.1 test function overload

利用 `n,s` 命令来单步调试/进入函数。

利用 verb|bt| 来查看调用的栈帧和层级关系。



```
(gdb) n
Their sum of square: 25
19              cout << "Enter two real number: ";
(gdb) bt
#0  main () at test_function_overload.cpp:19
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /ws/test
warning: Error disabling address space randomization: Operation not permitted
warning: File "/usr/local/lib64/libstdc++.so.6.0.29-gdb.py" auto-loading has been declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-load".

Breakpoint 1, main () at test_function_overload.cpp:14
14              cout << "Enter two integer: ";
(gdb) n
15              cin >> m >> n;
(gdb) n
Enter two integer: 3 4
16              cout << "Their sum of square: " << sumOfSquare(m, n) << endl;
(gdb) s
sumOfSquare (a=3, b=4) at test_function_overload.cpp:5
5               return a * a + b * b;
(gdb) bt
#0  sumOfSquare (a=3, b=4) at test_function_overload.cpp:5
#1  0x0000000000401262 in main () at test_function_overload.cpp:16
(gdb) n
6       }
(gdb) n
Their sum of square: 25
main () at test_function_overload.cpp:19
19              cout << "Enter two real number: ";
(gdb) n
20              cin >> x >> y;
(gdb) n
Enter two real number: 3.6 11.7
21              cout << "Their sum of square: " << sumOfSquare(x, y) << endl;
(gdb) s
sumOfSquare (a=3.6000000000000001, b=11.699999999999999) at test_function_overload.cpp:9
9               return a * a + b * b;
(gdb) bt
#0  sumOfSquare (a=3.6000000000000001, b=11.699999999999999) at test_function_overload.cpp:9
#1  0x00000000004012d4 in main () at test_function_overload.cpp:21
(gdb) 
```

3.2 test range base

`range based for` 是一种遍历一个 `range` 的方法，例如遍历一个容器。

`range based for` 等价于通过迭代器或指针对容器或 `range` 进行遍历，并将循环的指示变量设置为迭代器或指针指向的位置。

`range based for` 中，指示变量的类型可以是访问元素的类型，此时会调用 `copy constructor` 对容器或 `range` 内的值进行复制并访问。因此在循环内部对该 declaration 的修改不会对容器本身造成影响。

指示的类型可以是访问元素的类型对应的左值引用，此时 declaration 会是一个引用，对其修改可直接修改容器内的值。同时，因为没有复制过程，可以得到更快的速度。

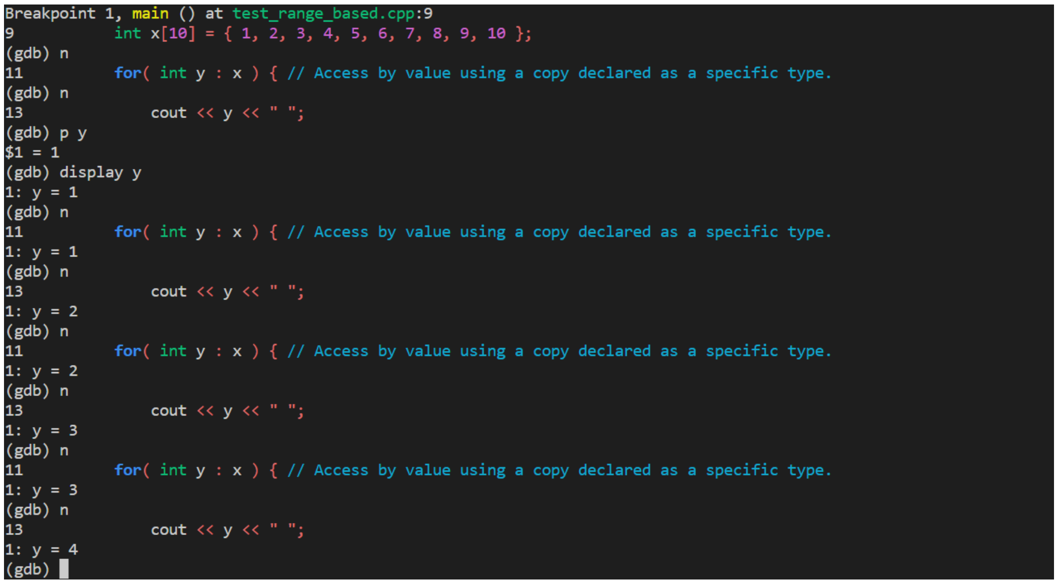
指示类型可以是 `const-qualified` 的，此时无论该类型是否是引用，都无法对原容器造成修改。

经过实验，当容器是 `const` 的，如果以引用形式访问，必须具有 `const` 标识。

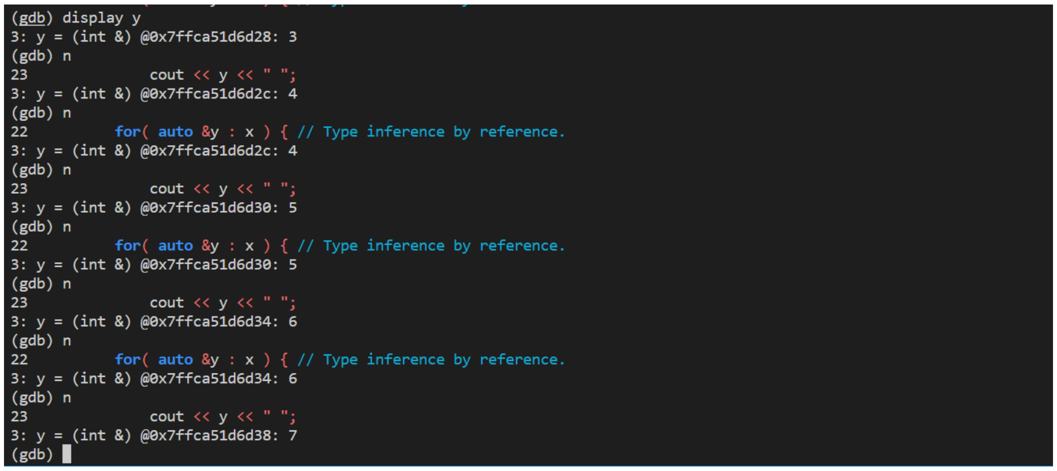
可以利用 `auto` 动态绑定类型，`auto&` 绑定类型并得到引用。也可以自动推断容器的 `cv-qualification` 属性。

在通过 gdb 调试的时候，可以发现引用的访问可以得到地址，而数值没有。

可通过 `display` 跟踪某个变量的值，在离开其所属的 `scope` 后会取消跟踪。



```
Breakpoint 1, main () at test_range_based.cpp:9
9           int x[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
(gdb) n
13              cout << y << " ";
(gdb) p y
$1 = 1
(gdb) display y
1: y = 1
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 1
(gdb) n
13              cout << y << " ";
1: y = 2
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 2
(gdb) n
13              cout << y << " ";
1: y = 3
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 3
(gdb) n
13              cout << y << " ";
1: y = 4
(gdb) 
```



```
(gdb) display y
3: y = (int &) @0x7ffca51d6d28: 3
(gdb) n
23              cout << y << " ";
3: y = (int &) @0x7ffca51d6d2c: 4
(gdb) n
22          for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7ffca51d6d2c: 4
(gdb) n
23              cout << y << " ";
3: y = (int &) @0x7ffca51d6d30: 5
(gdb) n
22          for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7ffca51d6d30: 5
(gdb) n
23              cout << y << " ";
3: y = (int &) @0x7ffca51d6d34: 6
(gdb) n
22          for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7ffca51d6d34: 6
(gdb) n
23              cout << y << " ";
3: y = (int &) @0x7ffca51d6d38: 7
(gdb) 
```

3.3 const test

编译错误：函数名大小写敏感、无法对 enum 内的东西取地址。

对于第三份代码，p 是一个宏定义，无论出现在哪里都是全局的。

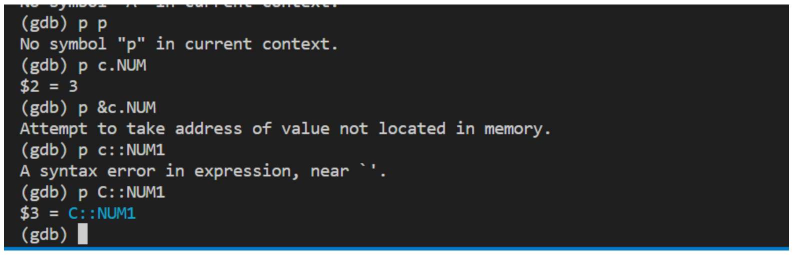
c.NUM 是一个左值，可以直接访问地址并输出。

C::NUM1 是 class 中的一个 enumorator ，可以直接得到值。

但是尝试对 enum 取地址会得到报错。可以参考[dcl.enum]，得知 enumerator 会被认为是一个常量，因此是一个右值，不能被取地址。

模板函数在进行类型推断的时候不会计算参数的值，因此第一次输出时只被增加了一次，即在调用函数时进行一次增加。第二次调用函数值会再被增加一次。

通过 gdb 输出地址：



```
No symbol "A" in current context.
(gdb) p p
No symbol "p" in current context.
(gdb) p c.NUM
$2 = 3
(gdb) p &c.NUM
Attempt to take address of value not located in memory.
(gdb) p c::NUM1
A syntax error in expression, near `'.
(gdb) p C::NUM1
$3 = C::NUM1
(gdb)
```