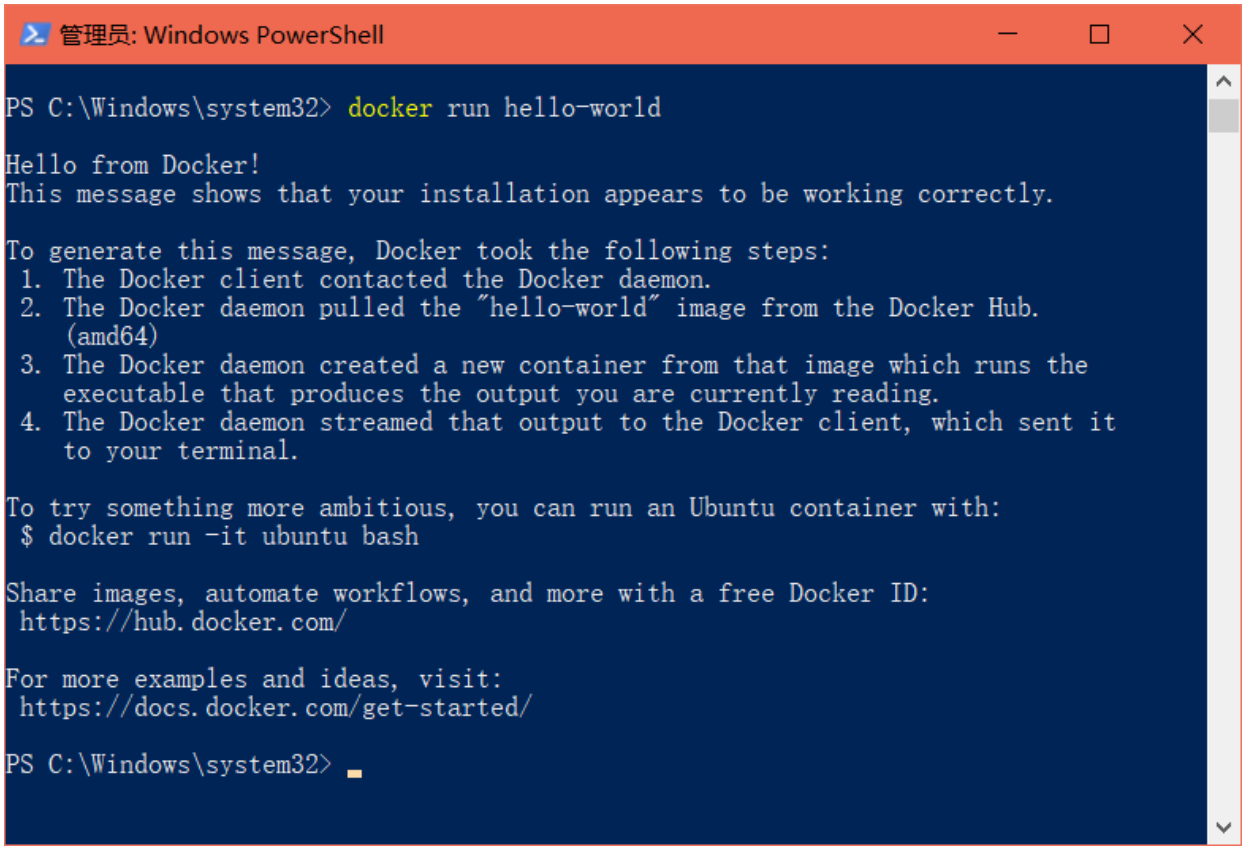
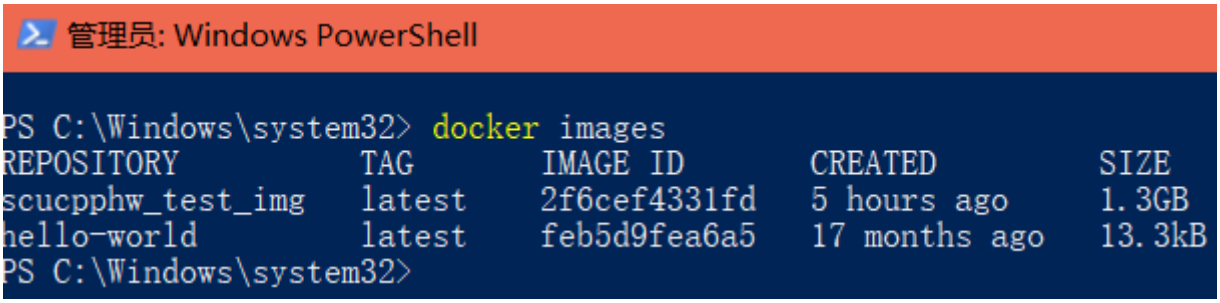
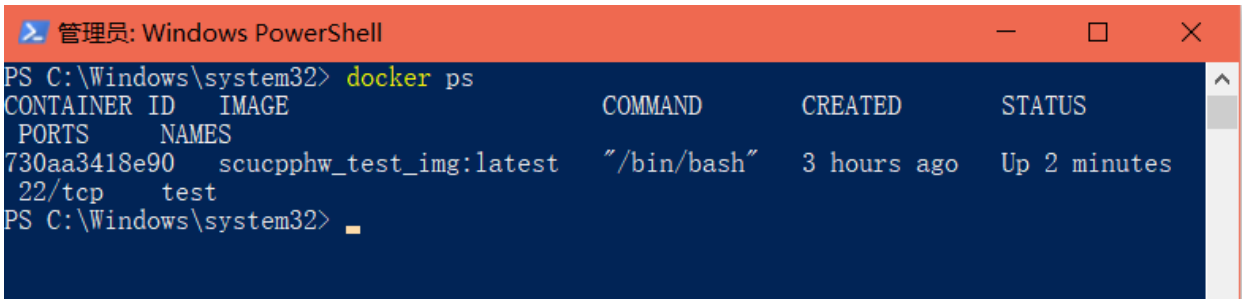
2022141650160 杨欣渝 homework1

2 配置对应的 docker 环境

1 终端执行 docker run hello-world,docker images,docker ps 的运行截图







---

2 使用 docker exec -it /bin/bash 进入 docker 后 cd /ws/code/的截图

```
管理员: Windows PowerShell

PS C:\Windows\system32> docker exec -it test /bin/bash
root@730aa3418e90:/# cd /ws/code
root@730aa3418e90:/ws/code#
```

---

3 利用 g++ 编译单文件

1 单步与分步编译 test.cpp 文件，并利用 ls 指令打印出相应的过程文件并截图，执行单步编译得到的 a.out 可执行文件, 与分步编译得到的 test 可执行文件，给出相应的运行结果

```
问题    输出    调试控制台    终端    端口

root@730aa3418e90:/ws# g++ test.cpp
root@730aa3418e90:/ws# ls
a.out  code  test.cpp
root@730aa3418e90:/ws# g++ -E test.cpp -o test.i
root@730aa3418e90:/ws# ls
a.out  code  test.cpp  test.i
root@730aa3418e90:/ws# g++ -S test.i -o test.s
root@730aa3418e90:/ws# ls
a.out  code  test.cpp  test.i  test.s
root@730aa3418e90:/ws# g++ -c test.s -o test.o
root@730aa3418e90:/ws# ls
a.out  code  test.cpp  test.i  test.o  test.s
root@730aa3418e90:/ws# g++ test.o -o test
root@730aa3418e90:/ws# ls
a.out  code  test  test.cpp  test.i  test.o  test.s
root@730aa3418e90:/ws# .\a.out
bash: .a.out: command not found
root@730aa3418e90:/ws# ./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@730aa3418e90:/ws# ./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@730aa3418e90:/ws#
```

---

2 给出利用 time 命令打印 inefficency.cpp 优化编译与非优化编译的执行信息

```
问题    输出    调试控制台    终端    端口

root@730aa3418e90:/ws# g++ inefficency.cpp -o without_o.out
root@730aa3418e90:/ws# g++ inefficency.cpp -o2 with_o.out
/usr/bin/ld: cannot find with_o.out: No such file or directory
collect2: error: ld returned 1 exit status
root@730aa3418e90:/ws# g++ inefficency.cpp -O2 with_o.out
/usr/bin/ld: cannot find with_o.out: No such file or directory
collect2: error: ld returned 1 exit status
root@730aa3418e90:/ws# g++ inefficency.cpp -O2 -o with_o.out
root@730aa3418e90:/ws# time ./with_o.out
result = 100904034

real    0m0.023s
user    0m0.017s
sys     0m0.000s
root@730aa3418e90:/ws# time ./without_o.out
result = 100904034

real    0m4.092s
user    0m4.074s
sys     0m0.012s
root@730aa3418e90:/ws#
```

---

3 选择 3 到 4 个其他的 test 文件，尝试利用不同的 g++ 参数进行编译，给出对应生成文件与最终的执行截图


```
问题    输出    调试控制台    终端    端口

root@730aa3418e90:/ws# g++ inefficency.cpp -o without_o.out
root@730aa3418e90:/ws# g++ inefficency.cpp -o2 with_o.out
/usr/bin/ld: cannot find with_o.out: No such file or directory
collect2: error: ld returned 1 exit status
root@730aa3418e90:/ws# g++ inefficency.cpp -O2 with_o.out
/usr/bin/ld: cannot find with_o.out: No such file or directory
collect2: error: ld returned 1 exit status
root@730aa3418e90:/ws# g++ inefficency.cpp -O2 -o with_o.out
root@730aa3418e90:/ws# time ./with_o.out
result = 100904034

real    0m0.023s
user    0m0.017s
sys     0m0.000s
root@730aa3418e90:/ws# time ./without_o.out
result = 100904034

real    0m4.092s
user    0m4.074s
sys     0m0.012s
root@730aa3418e90:/ws#
```

```
root@730aa3418e90:/ws# g++ -Wall test_class.cpp -o test
root@730aa3418e90:/ws# ./test
1       #21325302 is created
1       #58320212 is created
5       #21325302       5000    5000
25      #58320212       10000   10000
45      #21325302       5500    10500
60      #58320212       -4000   6000
90      #21325302       27.64   10527.6
90      #58320212       21.78   6021.78
#21325302       Balance: 10527.6
#58320212       Balance: 6021.78
```

```
root@730aa3418e90:/ws# g++ -g test_default_parameter.cpp -o test
root@730aa3418e90:/ws# readelf -S test | grep -i debug
  [26] .debug_aranges    PROGBITS         0000000000000000  00003082
  [27] .debug_info       PROGBITS         0000000000000000  000030c2
  [28] .debug_abbrev     PROGBITS         0000000000000000  000057f0
  [29] .debug_line       PROGBITS         0000000000000000  00005de9
  [30] .debug_str        PROGBITS         0000000000000000  00005faf
  [31] .debug_rnglists   PROGBITS         0000000000000000  000072bb
  [32] .debug_line_str   PROGBITS         0000000000000000  000072dd
root@730aa3418e90:/ws# ./test
Some box data is    10   12   15        1800
Some box data is    10   12    3        360
Some box data is    10    2    3        60
```

---

4 利用 GDB 调试文件

1 对于代码片段一，分别追踪前后两次调用函数 sumOfSquare() 时函数调用的栈帧与层级关系，并给出过程的相关截图

```
(gdb) b sumOfSquare(double,double)
Breakpoint 1 at 0x4011e0: file test_function_overload.cpp, line 9.
(gdb) b sumOfSquare(int,int)
Breakpoint 2 at 0x4011c0: file test_function_overload.cpp, line 5.
(gdb) r
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' s
et to "$debugdir:$datadir/auto-load".
To enable execution of this file add
        add-auto-load-safe-path /usr/local/lib64/libstdc++.so.6.0.29-gdb.py
line to your configuration file "/root/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/root/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
        info "(gdb)Auto-loading safe path"
Enter two integer: 1 2

Breakpoint 2, sumOfSquare (a=1, b=2) at test_function_overload.cpp:5
5               return a * a + b * b;
(gdb) bt
#0  sumOfSquare (a=1, b=2) at test_function_overload.cpp:5
#1  0x0000000000401262 in main () at test_function_overload.cpp:16
(gdb) c
Continuing.
Their sum of square: 5
Enter two real number: 2.1 2.3

Breakpoint 1, sumOfSquare (a=2.1000000000000001, b=2.2999999999999998) at test_function_overload.cpp:9
9               return a * a + b * b;
(gdb) bt
#0  sumOfSquare (a=2.1000000000000001, b=2.2999999999999998) at test_function_overload.cpp:9
#1  0x00000000004012d4 in main () at test_function_overload.cpp:21
(gdb) c
Continuing.
Their sum of square: 9.7
[Inferior 1 (process 19983) exited normally]
(gdb)
```

---

2 对于代码片段二，我们希望您能够追踪每次循环的变量 y 的具体变量值：

```
Temporary 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) display y
1: y = 4214944
(gdb) n
13              cout << 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) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 4
(gdb) n
13              cout << y << " ";
1: y = 5
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 5
(gdb) n
13              cout << y << " ";
1: y = 6
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 6
(gdb) n
13              cout << y << " ";
1: y = 7
```

```
13              cout << y << " ";
1: y = 7
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 7
(gdb) n
13              cout << y << " ";
1: y = 8
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 8
(gdb) n
13              cout << y << " ";
1: y = 9
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 9
(gdb) n
13              cout << y << " ";
1: y = 10
(gdb) n
11          for( int y : x ) { // Access by value using a copy declared as a specific type.
1: y = 10
(gdb) n
15          cout << endl;
(gdb) n
1 2 3 4 5 6 7 8 9 10
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
(gdb) n
18              cout << y << " ";
(gdb) display y
2: y = 1
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 1
(gdb) n
18              cout << y << " ";
2: y = 2
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
```

```
2: y = 2
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 2
(gdb) n
18              cout << y << " ";
2: y = 3
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 3
(gdb) n
18              cout << y << " ";
2: y = 4
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 4
(gdb) n
18              cout << y << " ";
2: y = 5
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 5
(gdb) n
18              cout << y << " ";
2: y = 6
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 6
(gdb) n
18              cout << y << " ";
2: y = 7
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 7
(gdb) n
18              cout << y << " ";
2: y = 8
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 8
(gdb) n
18              cout << y << " ";
```

```
2: y = 9
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 9
(gdb) n
18              cout << y << " ";
2: y = 10
(gdb) n
17          for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 10
(gdb) n
20          cout << endl;
(gdb) n
1 2 3 4 5 6 7 8 9 10
22          for( auto &y : x ) { // Type inference by reference.
(gdb) n
23              cout << y << " ";
(gdb) display y
3: y = (int &) @0x7fffe924cff0: 1
(gdb) n
22          for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7fffe924cff0: 1
(gdb) n
23              cout << y << " ";
3: y = (int &) @0x7fffe924cff4: 2
(gdb) n
22          for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7fffe924cff4: 2
(gdb) n
23              cout << y << " ";
3: y = (int &) @0x7fffe924cff8: 3
(gdb) n
22          for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7fffe924cff8: 3
(gdb) n
23              cout << y << " ";
3: y = (int &) @0x7fffe924cffc: 4
(gdb) n
22          for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7fffe924cffc: 4
(gdb) n
```

```
(gdb) n
27          for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffe924cffc: 4
(gdb) n
28              cout << y << " ";
4: y = (const int &) @0x7fffe924d000: 5
(gdb) n
27          for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffe924d000: 5
(gdb) n
28              cout << y << " ";
4: y = (const int &) @0x7fffe924d004: 6
(gdb) n
27          for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffe924d004: 6
(gdb) n
28              cout << y << " ";
4: y = (const int &) @0x7fffe924d008: 7
(gdb) n
27          for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffe924d008: 7
(gdb) n
28              cout << y << " ";
4: y = (const int &) @0x7fffe924d00c: 8
(gdb) n
27          for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffe924d00c: 8
(gdb) n
28              cout << y << " ";
4: y = (const int &) @0x7fffe924d010: 9
(gdb) n
27          for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffe924d010: 9
(gdb) n
28              cout << y << " ";
4: y = (const int &) @0x7fffe924d014: 10
(gdb) n
27          for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffe924d014: 10
(gdb) n
30          cout << endl;
```

并思考最后打印的 j 数值的意义

```
0.14159 1.14159 2.14159 3.14159 4.14159 5.14159 6.14159 7.14159 8.14159 9.14159
end of vector test
```

j数值的意义：把vector v中存了的数挨个赋值到 j 上，从v[0]到v[9]分别是 “0.14159，1.14159，…… 9.14159”

---

3 对于代码片段三，我们希望您能根据调试与代码中的提示修改代码让其正常运行，并告诉我们 const,enum，define 三者中哪一个有地址，并将其地址打印出来。

答：

- `const` 有地址，我的理解是 `const` 的名字虽然叫常量，但是本质上还是个变量，只是不能修改，比如 `const int` 本质上还是一个不能改变的 `int` 型变量，需要分配空间，所以必然有地址。
- `define` 没有地址，它的话比较像是预处理，做的是字符串替代，我感觉编译过后就不存在了，所以没地址。

- `enum` 没有地址，`enum` 只声明不定义，`enum` 和 `define` 就像一个真正的常量一样，比如一个数，`printf("%d",1)` ，这个 1 当然是个常量，但它没有地址。简单来说就是，`enum` 只声明不定义，也只是一个像数一样的常值，编译过后感觉就没有了。就是把这个数开个名字记录，所以 `enum YXY{a=3}` 和 `#define a 3` 差不多，然后 `YXY b=a` 和 `#define b 3` 差不多，都不需要分配空间。

代码与运行结果如图

test_const.cpp > Max<T>(const T &, const T &)

```cpp
#include <iostream>
using namespace std;
struct A {
#define p "hello"
};
class C {
public:
    static const int NUM = 3;
    enum con {
        NUM1 = 3
    };
};
#define MAX(a,b) ((a) > (b) ? (a) : (b))
template<typename T>
inline int Max(const T& a, const T& b){
    return (a>b ? a:b);
}
const int C::NUM;
int main() {
    cout << p << endl;
    cout << &C::NUM << endl;
    cout << C::NUM1 << endl;
}
```

问题　输出　调试控制台　终端　端口

```
root@730aa3418e90:/ws# ./test
hello
0x402004
3
root@730aa3418e90:/ws#
```

关于不能用 `&c.NUM` 来取地址的原因：

因为C::NUM不是 c独有的，访问这个静态变量才是说明将他作为静态属性访问，这才能代表这个静态变量是这个类的，所有对象共享的，而不是单个对象特有的，就是用c.NUM是没问题的，但不能给他取地址，因为这么访问时并不是因为在这个对象里给他分配了地址，所以不行。