



Section 1: Overview of the assignment content

因 docker 故障采用远程 ssh 连接 Ubuntu 虚拟机的方案代替

Section 2: Configure the appropriate docker environment

因 docker 故障采用远程 ssh 连接 Ubuntu 虚拟机的方案代替

Section 3: Compiling single files with g++

3.1: 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

answer:

```
root@wyb01: /ws/code2
root@wyb01:/ws/code2# ls
test.cpp
root@wyb01:/ws/code2# g++ -E test.cpp -o test.i
root@wyb01:/ws/code2# g++ -S test.i -o test.s
root@wyb01:/ws/code2# g++ -c test.s -o test.o
root@wyb01:/ws/code2# g++ test.o -o test
root@wyb01:/ws/code2# ls
test  test.cpp  test.i  test.o  test.s
root@wyb01:/ws/code2# ./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@wyb01:/ws/code2# g++ test.cpp
root@wyb01:/ws/code2# ls
a.out  test  test.cpp  test.i  test.o  test.s
root@wyb01:/ws/code2# ./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@wyb01:/ws/code2#
```

图 1: Single-step and step-by-step compilation

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

answer:

```
root@wyb01: /ws/code2
rm: missing operand
Try 'rm --help' for more information.
root@wyb01: /ws/code2# rm -r code2
rm: cannot remove 'code2': No such file or directory
root@wyb01: /ws/code2# cd
root@wyb01: ~# cd /ws
root@wyb01: /ws# rm -r code2
root@wyb01: /ws# mkdir code2
root@wyb01: /ws# cd code2
root@wyb01: /ws/code2# touch inefficiency.cpp
root@wyb01: /ws/code2# vi inefficiency.cpp
root@wyb01: /ws/code2# g++ inefficiency.cpp -O2 -o with_o.out
root@wyb01: /ws/code2# ls
inefficiency.cpp  with_o.out
root@wyb01: /ws/code2# g++ inefficiency.cpp -o without_o.out
root@wyb01: /ws/code2# ls
inefficiency.cpp  with_o.out  without_o.out
root@wyb01: /ws/code2# time ./without_o.out
result = 100904034

real    0m2.261s
user    0m2.260s
sys     0m0.001s
root@wyb01: /ws/code2# time ./with_o.out
result = 100904034

real    0m0.002s
user    0m0.001s
sys     0m0.001s
root@wyb01: /ws/code2#
```

图 2: inefficiency.cpp execution information for optimized and non-optimized compilation

3.3: 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

answer:

3.3.1:

```
root@wyb01: /ws/code2# g++ -Wall test1.cpp
root@wyb01: /ws/code2# g++ -g test1.cpp -o test1
root@wyb01: /ws/code2# g++ -std=c++17 test1.cpp
root@wyb01: /ws/code2# ls
a.out  test1  test1.cpp
root@wyb01: /ws/code2# ./test1
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@wyb01: /ws/code2# ./a.out
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@wyb01: /ws/code2#
```

图 3: First:Compile with different g++ parameters and print the final result

3.3.2:

```
root@wyb01:/ws/code# touch main.cpp
root@wyb01:/ws/code# vi main.cpp
root@wyb01:/ws/code# g++ -std=c++17 main.cpp
root@wyb01:/ws/code# g++ -Wall main.cpp
root@wyb01:/ws/code# g++ -g main.cpp -o main
root@wyb01:/ws/code# readelf -S main | grep -i debug

Command 'readlef' not found, did you mean:

  command 'readelf' from deb binutils (2.38-4ubuntu2.1)

Try: apt install <deb name>

root@wyb01:/ws/code# readelf -S main | grep -i debug
[28] .debug_aranges    PROGBITS     0000000000000000 0000303b
[29] .debug_info        PROGBITS     0000000000000000 000030db
[30] .debug_abbrev       PROGBITS     0000000000000000 00003df6
[31] .debug_line         PROGBITS     0000000000000000 000096a6
[32] .debug_str          PROGBITS     0000000000000000 000099cc
[33] .debug_line_str     PROGBITS     0000000000000000 000188ba
[34] .debug_rnglists    PROGBITS     0000000000000000 00010bee
root@wyb01:/ws/code# |
```

图 4: Second:Compile with different g++ parameters and print the final result

3.3.3:

```
root@wyb01:/ws/code# touch main2.cpp
root@wyb01:/ws/code# vi main2.cpp
root@wyb01:/ws/code# g++ -std=c++17 main2.cpp
root@wyb01:/ws/code# g++ -Wall main2.cpp
root@wyb01:/ws/code# g++ -g main2.cpp -o main
root@wyb01:/ws/code# readelf -S main2 | grep -i debug
readelf: Error: 'main2': No such file
root@wyb01:/ws/code# readelf -S main | grep -i debug
[29] .debug_aranges    PROGBITS     0000000000000000 0000d043
[30] .debug_info        PROGBITS     0000000000000000 0000df33
[31] .debug_abbrev       PROGBITS     0000000000000000 0001f905
[32] .debug_line         PROGBITS     0000000000000000 00020660
[33] .debug_str          PROGBITS     0000000000000000 00023526
[34] .debug_line_str     PROGBITS     0000000000000000 0003f3cb
[35] .debug_rnglists    PROGBITS     0000000000000000 0003f7fd
root@wyb01:/ws/code# |
```

图 5: Third:Compile with different g++ parameters and print the final result

Section 4: Debugging files with GDB

4.1: 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

answer:

4.1.1:

```
root@wyb01:/ws/code# g++ -g main.cpp -o main
root@wyb01:/ws/code# ls
main  main.cpp
root@wyb01:/ws/code# gdb main
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...
```

图 6: Compilation and gdb debugging

4.1.2:

```
(gdb) b sumOfSquare
Breakpoint 1 at 0x1257: sumOfSquare. (2 locations)
(gdb) info breakpoints
Num      Type             Disp Enb Address              What
1        breakpoint      keep y   <MULTIPLE>
1.1      y                   0x0000000000001257 in sumOfSquare(int, int) at main.cpp:5
1.2      y                   0x000000000000127b in sumOfSquare(double, double) at main.cpp:9
(gdb) run
```

图 7: Set the breakpoints and run the code

4.1.3:

```
Starting program: /ws/code/main
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Enter two integer: 2 3

Breakpoint 1, sumOfSquare (a=2, b=3) at main.cpp:5
5      return a * a + b * b;
(gdb) bt
#0  sumOfSquare (a=2, b=3) at main.cpp:5
#1  0x0000555555555329 in main () at main.cpp:16
(gdb) next
6
(gdb)
Their sum of square: 13
main () at main.cpp:19
19      cout << "Enter two real number: ";
(gdb)
20      cin >> x >> y;
(gdb)
Enter two real number: 2.2 1.1
21      cout << "Their sum of square: " << sumOfSquare(x, y) << endl;
(gdb) step

Breakpoint 1, sumOfSquare (a=2.2000000000000002, b=1.1000000000000001) at main.cpp:9
9      return a * a + b * b;
(gdb) bt
#0  sumOfSquare (a=2.2000000000000002, b=1.1000000000000001) at main.cpp:9
#1  0x00005555555553b9 in main () at main.cpp:21
```

图 8: View the stack frame and hierarchy relationship on two function calls separately (backtrace command)

4.2: 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，并思考最后打印的 j 数值的意义

answer:

4.2.1:

```
(gdb) b 9
Note: breakpoint 1 also set at pc 0x5555555533a4.
Breakpoint 2 at 0x5555555533a4: file range_based_for.cpp, line 9.
(gdb) run
Starting program: /ws/code2/test
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 2, main () at range_based_for.cpp:9
9      for (int y : x) { // Access by value using a copy declared as a specific type.
(gdb) display y
1: y = 0
(gdb) next
11      cout << y << " ";
(gdb)
12      y = 1
(gdb)
9      for (int y : x) { // Access by value using a copy declared as a specific type.
(gdb)
11      cout << y << " ";
(gdb)
12      y = 2
(gdb)
9      for (int y : x) { // Access by value using a copy declared as a specific type.
(gdb)
11      cout << y << " ";
(gdb)
12      y = 3
(gdb)
```

图 9: Set a breakpoint to monitor the change in y : the first loop

4.2.2:

```
root@wyb0t:~/src/code2
(gdb)
15      for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 6
(gdb)
16      cout << y << " ";
2: y = 7
(gdb)
15      for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 7
(gdb)
16      cout << y << " ";
2: y = 8
(gdb)
15      for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 8
(gdb)
16      cout << y << " ";
2: y = 9
(gdb)
15      for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 9
(gdb)
16      cout << y << " ";
2: y = 10
(gdb)
15      for( auto y : x ) { // Copy of 'x', almost always undesirable
2: y = 10
(gdb)
```

图 10: Set a breakpoint to monitor the change in y: the second loop

4.2.3:

```
root@wyb0t:~/src/code2
21      cout << y << " ";
3: y = (int &) @0x7fffffffe494: 6
(gdb)
20      for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7fffffffe494: 6
(gdb)
21      cout << y << " ";
3: y = (int &) @0x7fffffffe498: 7
(gdb)
20      for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7fffffffe498: 7
(gdb)
21      cout << y << " ";
3: y = (int &) @0x7fffffffe49c: 8
(gdb)
20      for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7fffffffe49c: 8
(gdb)
21      cout << y << " ";
3: y = (int &) @0x7fffffffe4a0: 9
(gdb)
20      for( auto &y : x ) { // Type inference by reference.
3: y = (int &) @0x7fffffffe4a0: 9
(gdb)
21      cout << y << " ";
3: y = (int &) @0x7fffffffe4a4: 10
(gdb)
```

图 11: Set a breakpoint to monitor the change in y: the third loop(Passing by reference)

4.2.4:

```
root@wyb0t:~/src/code2
26      cout << y << " ";
4: y = (const int &) @0x7fffffffe494: 6
(gdb)
25      for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffffffe494: 6
(gdb)
26      cout << y << " ";
4: y = (const int &) @0x7fffffffe498: 7
(gdb)
25      for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffffffe498: 7
(gdb)
26      cout << y << " ";
4: y = (const int &) @0x7fffffffe49c: 8
(gdb)
25      for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffffffe49c: 8
(gdb)
26      cout << y << " ";
4: y = (const int &) @0x7fffffffe4a0: 9
(gdb)
25      for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffffffe4a0: 9
(gdb)
26      cout << y << " ";
4: y = (const int &) @0x7fffffffe4a4: 10
(gdb)
25      for( const auto &y : x ) { // Type inference by const reference.
4: y = (const int &) @0x7fffffffe4a4: 10
(gdb)
```

图 12: Set a breakpoint to monitor the change in y: the forth loop(Passing by reference)

4.2.5:

```
(gdb)
0.14159 1.14159 2.14159 3.14159 4.14159 5.14159 6.14159 7.14159 8.14159 9.14159
41      cout << "end of vector test" << endl;
(gdb)
end of vector test
```

图 13: Print the value of each j

SUM: The meaning of j is to print out the data stored in the vector container

4.3: 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。

answer:

```
(gdb) b 21
Breakpoint 1 at 0x1220: file main.cpp, line 21.
(gdb) run
Starting program: /ws/code2/main
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at main.cpp:21
21      cout << p << endl;
(gdb) print &"hello"
$1 = (char (*)[6]) 0x55555556aeb0
(gdb) next
hello
      cout << &c.NUM << endl;
23      (gdb) print &c.NUM
Attempt to take address of value not located in memory.
(gdb) next
0x55555556004
25      cout << C::NUM1 << endl;
(gdb) print &C::NUM1
Can't take address of "C::NUM1" which isn't an lvalue.
(gdb) |
```

图 14: Print the address of the three

Conclusion:

in this problem enum does not have an address, const and define do

My interpretation:

After setting the breakpoint, you can see that p is replaced with "hello" in Pre-processing, so the value printed by define p has the address

c.NUM is a static const int type, and you can find the address printed out.

But C::NUM1 is an enum type, it doesn't print out the address and output: Can't take address of "C::NUM1" which isn't an lvalue.