Student Name: 马越
Student ID: 2022141460097

C++ 面向对象程序设计
Assignment 1

1. Configure docker environment

1. Screenshots of executing `docker run hello-world`, `docker images`, and `docker ps` commands on the terminal.

```
PS C:\Users\myyquq> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:6e8b6f026e0b9c419ea0fd02d3905dd0952ad1feea67543f525c73a0a790fefb
Status: Downloaded newer image for hello-world:latest

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/
```

图 1: docker run hello-world

```
PS C:\Users\myyquq> docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
scucpp        latest    701599c859d5   3 hours ago      1.65GB
hello-world   latest    feb5d9fea6a5   17 months ago    13.3kB
```

图 2: docker images

```
PS C:\Users\myyquq> docker ps
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS       PORTS     NAMES
118e996adc3e   scucpp:latest  "/bin/bash"   3 hours ago   Up 3 hours   22/tcp    hopeful_hodgkin
```

图 3: docker ps

2. Screenshot of executing `docker exec -it <container id> /bin/bash` to enter docker container and executing `cd /ws/code/`.

```
PS C:\Users\myyquq> docker exec -it 118e996adc3e /bin/bash
root@118e996adc3e:/# cd /ws/code
root@118e996adc3e:/ws/code# cat hello_world.cpp
// ======================== C++ Way ==============================
#include <iostream>

void hello_cpp() {
}

// ======================== C Way ==============================

#include "stdio.h"
#include "stdlib.h"

void hello_c() {
 printf("%s", "Hello, world!\n");
}


// ======================== Assembly Way ==============================

#include "stdio.h"
#include "stdlib.h"
```

图 4: screenshot of the docker terminal

2. g++ compilation

Compile the `test.cpp` file with single-step and step-by-step compilation, use `ls` command to print out the corresponding process files and take a screenshot. Execute `a.out` from single-step compilation and `test` from step-by-step compilation and provide the corresponding execution results.

1.1 Single-step compilation and execution results:

```
root@118e996adc3e:/ws/code# ls
hello_world.cpp  inefficency.cpp  test.cpp
root@118e996adc3e:/ws/code# g++ test.cpp
root@118e996adc3e:/ws/code# ls
a.out  hello_world.cpp  inefficency.cpp  test.cpp
root@118e996adc3e:/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
```

图 5: g++ compilation in single-step mode

1.2 Step-by-step compilation and execution results:

```
root@118e996adc3e:/ws/code# ls
hello_world.cpp  inefficency.cpp  test.cpp
root@118e996adc3e:/ws/code# g++ -E test.cpp -o test.i
root@118e996adc3e:/ws/code# g++ -S test.i -o test.s
root@118e996adc3e:/ws/code# g++ -c test.s -o test.o
root@118e996adc3e:/ws/code# g++ test.o -o test
root@118e996adc3e:/ws/code# ls
hello_world.cpp  inefficency.cpp  test  test.cpp  test.i  test.o  test.s
root@118e996adc3e:/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
```

图 6: g++ compilation in step-by-step mode

2. Compile optimization:

```
root@118e996adc3e:/ws/code# g++ inefficency.cpp -o without_o.out
root@118e996adc3e:/ws/code# g++ inefficency.cpp -O2 -o with_o.out
root@118e996adc3e:/ws/code# time ./with_o.out
result = 100904034

real    0m0.011s
user    0m0.004s
sys     0m0.001s
root@118e996adc3e:/ws/code# time ./without_o.out
result = 100904034

real    0m1.865s
user    0m1.855s
sys     0m0.001s
```

图 7: Comparison between compile optimization levels

3. g++ compilation using different parameters:

The file structure of the working directory and the shell script written for testing run time of programs are shown as follows.

```
root@118e996adc3e:/ws/code# tree
.
|-- batch.sh
`-- testcpps
    |-- inefficency.cpp
    |-- test_class.cpp
    |-- test_class_size.cpp
    |-- test_default_parameter.cpp
    |-- test_move.cpp
    |-- test_noexcept.cpp
    |-- test_ptr.cpp
    `-- test_raii.cpp

1 directory, 9 files
```

图 8: file structure

Listing 1: batch.sh

```
OPT_LEVELS=(0 1 2 3)
INPUT_DIR="./testcpps"
OUTPUT_FILE="./result"

if [ -f "$OUTPUT_FILE" ]; then
    rm "$OUTPUT_FILE"
fi

for file in "$INPUT_DIR"/*.cpp; do
    filename=$(basename -- "$file")
    filename="${filename%.*}"
    printf "Testing %s...\n" "$filename"
    printf "%s:\n" "$filename" >> "$OUTPUT_FILE
    for opt_level in "${OPT_LEVELS[@]}"; do
        g++ -O$opt_level "$file" -o "$filename"_O$opt_level
        result=$({ time ./"$filename"_O$opt_level; } 2>&1 |grep '^real' |awk '{print $2}' |tr -d '\n')
        printf "\tO%s:\t%s\n" "$opt_level" "$result" >> "$OUTPUT_FILE"
        rm "$filename"_O$opt_level
    done
done
```

Make the following modifications to the test source code:

Add the string header file to `test_raii.cpp` by `#include<string>` . Modify the `printf_s` function in `test_noexcept.cpp` to `printf`, delete `__declspec(nothrow)` from the function signature of `f2`, and set the compiler language standard to `C++11`.

All code can be compiled and run without issue after modifications. The results of the time tests are as follows:

```
root@118e996adc3e:/ws/code# ./batch.sh
Testing inefficency...
Testing test_class...
Testing test_class_size...
Testing test_default_parameter...
Testing test_move...
Testing test_noexcept...
Testing test_ptr...
Testing test_raii...
root@118e996adc3e:/ws/code# cat result
inefficency:
        O0:     0m1.811s
        O1:     0m0.422s
        O2:     0m0.004s
        O3:     0m0.004s
test_class:
        O0:     0m0.004s
        O1:     0m0.004s
        O2:     0m0.006s
        O3:     0m0.004s
test_class_size:
        O0:     0m0.003s
        O1:     0m0.007s
        O2:     0m0.003s
        O3:     0m0.004s
test_default_parameter:
        O0:     0m0.004s
        O1:     0m0.004s
        O2:     0m0.005s
        O3:     0m0.004s
test_move:
        O0:     0m0.003s
        O1:     0m0.004s
        O2:     0m0.006s
        O3:     0m0.004s
```

图 9: screenshot of the terminal

```
inefficency:
      O0:   0m1.811s
      O1:   0m0.422s
      O2:   0m0.004s
      O3:   0m0.004s
test_class:
      O0:   0m0.004s
      O1:   0m0.004s
      O2:   0m0.006s
      O3:   0m0.004s
test_class_size:
      O0:   0m0.003s
      O1:   0m0.007s
      O2:   0m0.003s
      O3:   0m0.004s
test_default_parameter:
      O0:   0m0.004s
      O1:   0m0.004s
      O2:   0m0.005s
      O3:   0m0.004s
test_move:
      O0:   0m0.003s
      O1:   0m0.004s
      O2:   0m0.006s
      O3:   0m0.004s
test_noexcept:
      O0:   0m0.004s
      O1:   0m0.004s
      O2:   0m0.005s
      O3:   0m0.004s
test_ptr:
      O0:   0m0.005s
      O1:   0m0.004s
      O2:   0m0.006s
      O3:   0m0.004s
test_raii:
      O0:   0m0.004s
      O1:   0m0.004s
      O2:   0m0.007s
      O3:   0m0.004s
```

图 10: all test results

The results show that for simple code, different compiler optimization levels did not demonstrate significant differences in run time.

3. gdb debugging

1.

```
root@118e996adc3e:/ws/code# gdb test1
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
Copyright (C) 2021 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 test1...
(gdb) break sumOfSquare
Breakpoint 1 at 0x4011c0: sumOfSquare. (2 locations)
(gdb) run
Starting program: /ws/code/test1
warning: Error disabling address space randomization: Operation not permitted
Enter two integer: 1 2

Breakpoint 1, sumOfSquare (a=1, b=2) at test_function_overload.cpp:7
7           return a * a + b * b;
(gdb) backtrace
#0  sumOfSquare (a=1, b=2) at test_function_overload.cpp:7
#1  0x0000000000401262 in main () at test_function_overload.cpp:18
(gdb) continue
Continuing.
Their sum of square: 5
Enter two real number: 1.0 2.0

Breakpoint 1, sumOfSquare (a=1, b=2) at test_function_overload.cpp:11
11          return a * a + b * b;
(gdb) backtrace
#0  sumOfSquare (a=1, b=2) at test_function_overload.cpp:11
#1  0x00000000004012d4 in main () at test_function_overload.cpp:22
(gdb) continue
Continuing.
Their sum of square: 5
[Inferior 1 (process 11914) exited normally]
(gdb)
```

图 11: gdb debugging 1

2. Considering that the function `operator<<(ostream&, int)` is called every time the loop variable `y` is updated, a breakpoint is set here, and the command `display $rsi` is used to print the value of the second parameter, which is exactly the value of the variable `y`.

```
root@118e996adc3e:/ws/code# gdb test2
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
Copyright (C) 2021 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 test2...
(gdb) break operator<<(int)
Breakpoint 1 at 0x4010f0
(gdb) run
Starting program: /ws/code/test2
warning: Error disabling address space randomization: Operation not permitted

Breakpoint 1, 0x00007fa4c277f5a0 in std::ostream::operator<<(int) () from /usr/local/lib64/libstdc++.so.6
(gdb) display $rsi
1: $rsi = 1
(gdb) continue
Continuing.

Breakpoint 1, 0x00007fa4c277f5a0 in std::ostream::operator<<(int) () from /usr/local/lib64/libstdc++.so.6
1: $rsi = 2
(gdb)
Continuing.

Breakpoint 1, 0x00007fa4c277f5a0 in std::ostream::operator<<(int) () from /usr/local/lib64/libstdc++.so.6
1: $rsi = 3
(gdb)
Continuing.

Breakpoint 1, 0x00007fa4c277f5a0 in std::ostream::operator<<(int) () from /usr/local/lib64/libstdc++.so.6
1: $rsi = 4
(gdb)
Continuing.

Breakpoint 1, 0x00007fa4c277f5a0 in std::ostream::operator<<(int) () from /usr/local/lib64/libstdc++.so.6
1: $rsi = 5
(gdb)
```

图 12: gdb debugging 2.1

Similar intermediate steps are omitted...

```
Breakpoint 1, 0x00007fa4c277f5a0 in std::ostream::operator<<(int) () from /usr/local/lib64/libstdc++.so.6
1: $rsi = 8
(gdb)
Continuing.

Breakpoint 1, 0x00007fa4c277f5a0 in std::ostream::operator<<(int) () from /usr/local/lib64/libstdc++.so.6
1: $rsi = 9
(gdb)
Continuing.

Breakpoint 1, 0x00007fa4c277f5a0 in std::ostream::operator<<(int) () from /usr/local/lib64/libstdc++.so.6
1: $rsi = 10
(gdb)
Continuing.
1 2 3 4 5 6 7 8 9 10
end of integer array test

0.14159 1.14159 2.14159 3.14159 4.14159 5.14159 6.14159 7.14159 8.14159 9.14159
end of vector test
[Inferior 1 (process 328) exited normally]
(gdb)
```

图 13: gdb debugging 2.2

3. Using range-based for loop to obtain each element in the vector `v` through constant reference and printing out the result, where each `j` corresponds to the array index plus `0.14159`.

`#define` is a preprocessor directive, and all occurrences of `p` are replaced before runtime, so `p` does not have an address.

Determine whether `NUM1` is an lvalue or an rvalue using the following code.

Listing 2: testNUM1.cpp

```cpp
#include <iostream>

class A {
public:
    enum con : int {
        NUM1 = 3
    };
};

void foo(int &a) {
    std::cout << "NUM1 is an lvalue\n";
}

void foo(int &&a) {
    std::cout << "NUM1 is an rvalue\n";
}

int main() {
    foo(A::NUM1);  /// Output "NUM1 is an rvalue"
    return 0;
}
```

According to the result, `NUM1` is an rvalue, so it does not have an address and cannot be referenced.

Making the following changes to the code:
Change `cout << &c.NUM << endl;` to `cout << &C::NUM << endl;`.
Comment out `cout << &C:NUM1 << endl;`.

```
root@118e996adc3e:/ws/code# g++ -g test_const.cpp -o test3
root@118e996adc3e:/ws/code# ./test3
hello
0x402004
3
7
10
6
```

图 14: gdb debugging 3

The address of `NUM` is: `0x402004`.