華中科技大學

操作系统原理实验报告

姓　　名：　　刘本嵩

学　　院：　　计算机科学与技术

专　　业：　　计算机科学与技术

班　　级：　　CS1601

学　　号：　　U201614531

指导教师：

分数	
教师签名	

年　月　日

目　录

1 实验一 进程控制

1.1 实验目的

1、加深对进程的理解,进一步认识并发执行的实质；

2、分析进程争用资源现象,学习解决进程互斥的方法；

3、掌握 Linux 进程基本控制；

4、掌握 Linux 系统中软中断和管道通信。

1.2 实验内容

编写程序，演示多进程并发执行和进程软中断、管道通信。

父进程使用系统调用 pipe( )建立一个管道,然后使用系统调用 fork()创建两个子进程：子进程 1 和子进程 2；

子进程 1 每隔 1 秒通过管道向子进程 2 发送数据:

I send you x times. (x 初值为 1，每次发送后做加一操作)

子进程 2 从管道读出信息，并显示在屏幕上。

父进程用系统调用 signal()捕捉来自键盘的中断信号（即按 CTRL+C 键）；当捕捉到中断信号后，父进程用系统调用 Kill()向两个子进程发出信号，子进程捕捉到信号后分别输出下列信息后终止：

Child Process l is Killed by Parent!
Child Process 2 is Killed by Parent!

父进程等待两个子进程终止后，释放管道并输出如下的信息后终止

Parent Process is Killed!

额外要求：15 秒后自动退出所有进程。

1.3 实验设计

1.3.1 开发环境

```
recolic@RECOLICPC
OS: Arch Linux
Kernel: x86_64 Linux 5.0-rc1
Uptime: 27d 2h 32m
Packages: 1795
Shell: fish 2.7.1
Resolution: 4480x1440
DE: GNOME
WM: GNOME Shell
WM Theme:
GTK Theme: Gnome-OSX-II-2-6 [GTK2/3]
Icon Theme: hicolor
Font: Cantarell 11
CPU: Intel Core i5-4200H @ 4x 3.4GHz [59.0°C]
GPU: GeForce GTX 950M
RAM: 2734MiB / 15489MiB
Disk: 1TiB ext4/btrfs/ntfs + 2TiB nfs
```

```
recolic@RECOLICMPC
OS: Arch Linux
Kernel: x86_64 Linux 4.20.0-arch1-1-ARCH
Uptime: 4d 16h 45m
Packages: 1201
Shell: fish 2.7.1
Resolution: 1920x1080
DE: GNOME
WM: GNOME Shell
WM Theme: Human
GTK Theme: Gn-OSX-HSierra-1.2.1-S [GTK2/3]
Icon Theme: la-capitaine-icon-theme
Font: Cantarell 11
CPU: Intel Core m3-7Y30 @ 4x 2.6GHz [26.0°C]
GPU: intel
RAM: 2002MiB / 3828MiB
Disk: 128GiB btrfs + 2TiB nfs
```

1.3.2 实验设计

显然，直接创建 3 个进程，两个子进程按要求进行工作即可。自动超时使用一个 timed_callback 来自动 SIGINT 父亲进程，handler 函数是为了作为 wrapper，使 linux sig handler 支持 std::function 和 lambda。两个进程的消息传递使用 rlib::fdIO 的库功能，这是为了简化不关键部分的逻辑。rlib 完全由我自己完成，在报告最后我将使用小字体粘贴 rlib 当前的全部代码。

1.4 实验调试

1.4.1 实验步骤

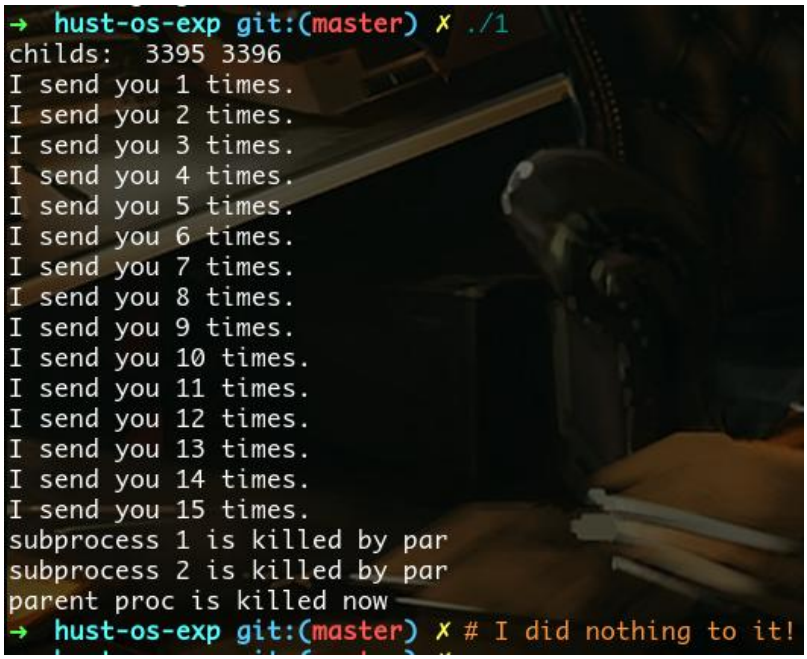
查阅`man 2 fork`，`man 2 wait`，`man 2 getpid`，`man 2 pipe`和`man 2 signal`，按照文档的接口写程序即可。详细内容见附录代码。

1.4.2 实验调试及心得

逻辑较简单，编译时错误修复完成后一次通过。注意，要求 linux 环境，要求支持 c++14 标准的编译器。

代码中忽略了很多不常见的错误检查，用 c assert 代替 exception 进行了一些错误检查，这都是不优雅的，但对于一个简单的实验似乎可以接受。

如测试图所示，用超时和 SIGINT 都能正常退出 1,实验要求已经满足。



附录 实验代码

```
#include <cassert>
#include <chrono>
#include <rlib/stdio.hpp>
#include <thread>

#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
```

```
#include <rlib/sys/sio.hpp>
using rlib::println;
using namespace rlib::literals;
using namespace std::chrono_literals;
```

```
template <typename FuncType, typename TimeType>
void timed_callback(FuncType func, TimeType time) {
```

```
    std::thread([&] {
        std::this_thread::sleep_for(time);
        func();
    })
     .detach();
}
```

```
std::function<void()> *real_handler = nullptr;
void handler(int) {
    (*real_handler)();
    exit(0);
}
```

```
int main() {
    fd_t fPipe[2];
    assert(0 == pipe(fPipe));
    assert(SIG_ERR != signal(SIGINT, &handler));
```

```
    auto pids = std::make_pair(fork(), fork());
    auto times = 0;
    if (pids.first + pids.second == 0)
        exit(0);
    if (pids.first == 0) {
        // child 1
        auto h = [] { println("subprocess 1 is killed by par"); };
        real_handler = new std::function<void()>(h);
        close(fPipe[0]);
        while (true) {
            const auto str = "I send you {} times"_format(++times);
            rlib::fdIO::quick_write(fPipe[1], str);
            std::this_thread::sleep_for(1s);
        }
    } else if (pids.second == 0) {
        // child 2
        auto h = [] { println("subprocess 2 is killed by par"); };
        real_handler = new std::function<void()>(h);
        close(fPipe[1]);
        while (true) {
            try {
                println(rlib::fdIO::quick_readall(fPipe[0]));
            } catch (...) {
            }
        }
    } else {
```

```
        // parent
        // no error check
        println("childs: ", pids.first, pids.second);
        auto h = [=] {
            kill(pids.first, SIGINT);
            waitpid(pids.first, NULL, NULL);
            kill(pids.second, SIGINT);
            waitpid(pids.second, NULL, NULL);
            println("parent proc is killed now");
        };
        real_handler = new std::function<void()>(h);
        timed_callback([] { kill(getpid(), SIGINT); }, 15s);
        while (true)
            ;
    }
}
```

2 实验二 线程同步与通信

2.1 实验目的

1、掌握 Linux 下线程的概念；

2、了解 Linux 线程同步与通信的主要机制；

3、通过信号灯操作实现线程间的同步与互斥。

2.2 实验内容

通过 Linux 多线程与信号灯机制，设计并实现计算机线程与 I/O 线程共享缓冲区的同步与通信。

程序要求:两个线程,共享公共变量 a

线程 1 负责计算(1 到 100 的累加，每次加一个数)

线程 2 负责打印（输出累加的中间结果)

额外要求：用两个进程实现此功能。

2.3 实验设计

2.3.1 开发环境

```
recolic@RECOLICPC
OS: Arch Linux
Kernel: x86_64 Linux 5.0-rc1
Uptime: 27d 2h 32m
Packages: 1795
Shell: fish 2.7.1
Resolution: 4480x1440
DE: GNOME
```

```
WM: GNOME Shell
WM Theme:
GTK Theme: Gnome-OSX-II-2-6 [GTK2/3]
Icon Theme: hicolor
Font: Cantarell 11
CPU: Intel Core i5-4200H @ 4x 3.4GHz [59.0°C]
GPU: GeForce GTX 950M
RAM: 2734MiB / 15489MiB
Disk: 1TiB ext4/btrfs/ntfs + 2TiB nfs


recolic@RECOLICMPC
OS: Arch Linux
Kernel: x86_64 Linux 4.20.0-arch1-1-ARCH
Uptime: 4d 16h 45m
Packages: 1201
Shell: fish 2.7.1
Resolution: 1920x1080
DE: GNOME
WM: GNOME Shell
WM Theme: Human
GTK Theme: Gn-OSX-HSierra-1.2.1-S [GTK2/3]
Icon Theme: la-capitaine-icon-theme
Font: Cantarell 11
CPU: Intel Core m3-7Y30 @ 4x 2.6GHz [26.0°C]
GPU: intel
RAM: 2002MiB / 3828MiB
Disk: 128GiB btrfs + 2TiB nfs
```

2.3.2 实验设计

对于多线程版本，这是一个标准的 provider-consumer model。使用一个 condition variable 即是此类问题的标准解决方案。对于此同步原语的详细解释，请参阅 wikipedia: https://en.wikipedia.org/wiki/Monitor_(synchronization)

对于多进程版本，可选的做法有 fifo、socket、pipe、消息队列。尽管 rlib 能够在 fifo 等流式传输工具上轻松实现信息包的传输，但我还是决定尝试古老的 sysetmV 消息队列。

2.4 实验调试

2.4.1 实验步骤

对于多线程版本，查阅 c++ 标准库对 condition variable 的使用说明文档 (https://en.cppreference.com/w/cpp/thread/condition_variable)，写好生产者和消费者的函数，开两个线程执行即可。

对于多进程版本，查阅`man 2 msgget`和`man 2 msgrcv`，按说明分别写好生产者消费者的函数，开两个进程执行即可。这两个进程使用不同的命令行参数来决定谁是生产者，同时设定相同的`--key`参数使得操作系统能够将其配对。这里依赖于 rlib 库的部分都将在实验报告的最后一部分找到源码。

2.4.2 实验调试及心得

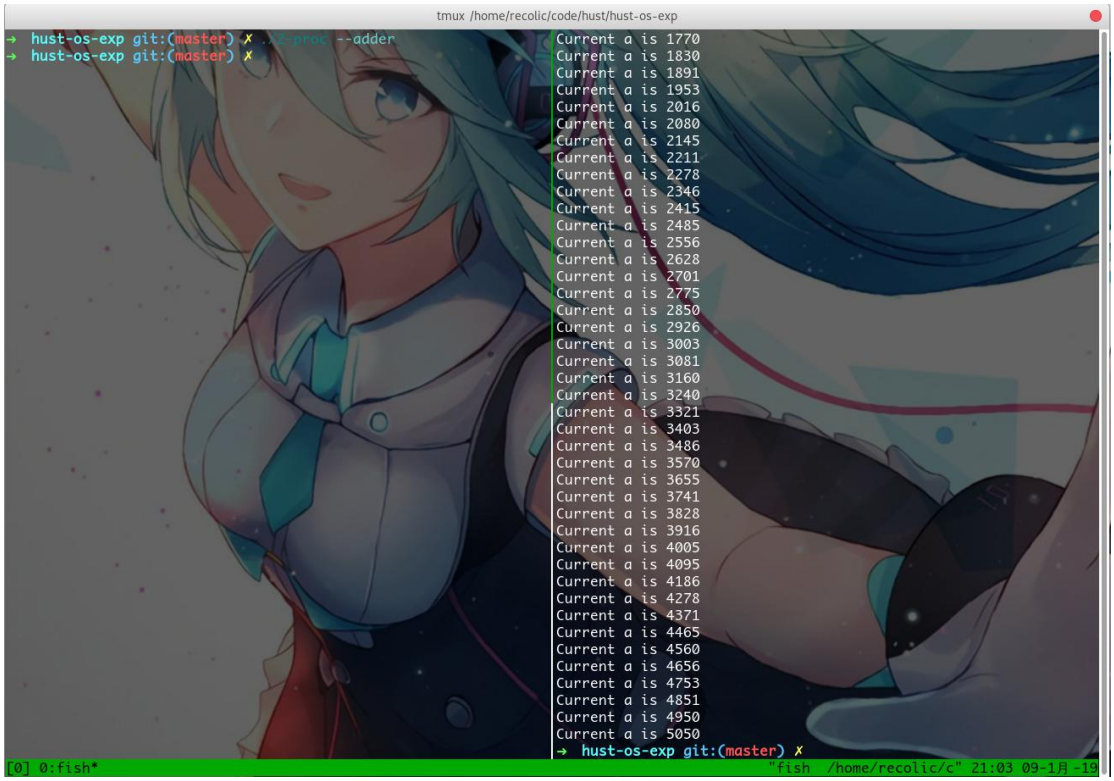
在修复所有编译错误后，一次通过。

在本次实验中感觉，c++ STL 的 condition variable 接口很不易于使用。使用者需要自己维护一个 bool 变量来防止 fake wake，并且使用者需要自己维护锁。如果能够与 std::lock_guard 等可以保护 critical section 的接口设计相融合就好了。但仔细思考似乎也并不容易作出很好的改进。

多进程版本的 system v api 是一如既往的 c 风格，虽然设计很标准但用起来很难受。。。

注意，要求 linux 环境，要求支持 c++14，c++17 或 c++20 标准的编译器。

如测试图所示，线程和进程版本的实验结果都正确。



附录 实验代码

多线程版本：

```
#include <condition_variable>
#include <mutex>
#include <thread>

#include <rlib/stdio.hpp>
```

```
int a = 0;
std::mutex a_m;
std::condition_variable a_cv;
```

```
bool processed = false;
```

```
void adder_thread() {
    for (auto cter = 1; cter < 101; ++cter) {
        std::unique_lock<std::mutex> lk(a_m);
        a_cv.wait(lk, [] { return processed; });
        a += cter;
        processed = false;
        lk.unlock();
        a_cv.notify_one();
    }
}
```

```
void printer_thread() {
    while (true) {
        std::unique_lock<std::mutex> lk(a_m);
        a_cv.wait(lk, [] { return !processed; });
        rlib::println("Current a is", a);
        processed = true;
        lk.unlock();
        a_cv.notify_one();
    }
}
```

```
int main() {
    std::thread(&printer_thread).detach();
    adder_thread();
}
```

多进程版本:

```
#include <cassert>
#include <rlib/opt.hpp>
#include <rlib/stdio.hpp>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <sys/types.h>

constexpr int msgsize = sizeof(int) / sizeof(char);
```

```
struct r_msgbuf {
    long mtype = 1; /* message type, must be > 0 */
    char mtext[msgsize]; /* message data */
};
```

```
void adder(int qid) {
    int a = 0;
    for (auto cter = 1; cter < 101; ++cter) {
        a += cter;
```

```
        r_msgbuf buf;
        std::memcpy(buf.mtext, &a, sizeof(a));
        assert(msgsnd(qid, (void *)&buf, msgsize, IPC_NOWAIT) !=
-1);
    }
    r_msgbuf buf{2}; // EOF message
    assert(msgsnd(qid, (void *)&buf, msgsize, IPC_NOWAIT) != -1);
}
```

```
void printer(int qid) {
    while (true) {
        r_msgbuf buf;
        msgrcv(qid, (void *)&buf, msgsize, 0, NULL);
```

```
        if (buf.mtype == 2)
            break; // EOF message type is 2.
        auto a = *(int *)(buf.mtext);
        rlib::println("Current a is", a);
    }
}
```

```
int main(int argc, char **argv) {
    rlib::opt_parser args(argc, argv);
    auto key = args.getValueArg("--key", false,
"25501").as<key_t>();
    auto qid = msgget(key, IPC_CREAT | 0666);
    assert(qid != -1);
```

```
    if (args.getBoolArg("--adder"))
        adder(qid);
    else if (args.getBoolArg("--printer"))
        printer(qid);
    else
        rlib::println("Usage: this --adder/--printer");
}
```

3 实验三 共享内存与进程同步

3.1 实验目的

1、掌握 Linux 下共享内存的概念与使用方法；

2、掌握环形缓冲的结构与使用方法；

3、掌握 Linux 下进程同步与通信的主要机制。

3.2 实验内容

利用多个共享内存（有限空间）构成的环形缓冲，将源文件复制到目标文件，实现两个进程的誊抄。

3.3 实验设计

3.3.1 开发环境

```
recolic@RECOLICPC
OS: Arch Linux
Kernel: x86_64 Linux 5.0-rc1
Uptime: 27d 2h 32m
Packages: 1795
Shell: fish 2.7.1
Resolution: 4480x1440
DE: GNOME
WM: GNOME Shell
WM Theme:
GTK Theme: Gnome-OSX-II-2-6 [GTK2/3]
Icon Theme: hicolor
Font: Cantarell 11
CPU: Intel Core i5-4200H @ 4x 3.4GHz [59.0°C]
GPU: GeForce GTX 950M
RAM: 2734MiB / 15489MiB
```

```
Disk: 1TiB ext4/btrfs/ntfs + 2TiB nfs


recolic@RECOLICMPC
OS: Arch Linux
Kernel: x86_64 Linux 4.20.0-arch1-1-ARCH
Uptime: 4d 16h 45m
Packages: 1201
Shell: fish 2.7.1
Resolution: 1920x1080
DE: GNOME
WM: GNOME Shell
WM Theme: Human
GTK Theme: Gn-OSX-HSierra-1.2.1-S [GTK2/3]
Icon Theme: la-capitaine-icon-theme
Font: Cantarell 11
CPU: Intel Core m3-7Y30 @ 4x 2.6GHz [26.0°C]
GPU: intel
RAM: 2002MiB / 3828MiB
Disk: 128GiB btrfs + 2TiB nfs
```

3.3.2 实验设计

得益于现代 C++标准库设计者的远见卓识，此次实验的代码实现**极其令人满意**。下面我将依次介绍。

首先，C++支持重载 operator new 和 operator delete。这使得共享内存对上层是透明的。我只需要求一个类的 operator new 使用我自己定义的 create_shared_memory 函数而不是 malloc/calloc 分配空间，使用我自己定义的函数而不是 free 来释放空间即可。

具体的操作是，create_shared_memory 使用 mmap 分配能够在进程之间共享的匿名内存，这段内存加一个偏移量后返回给 operator new，偏移的空间存储了分配空间的大小。这正如 malloc 在堆上所做的操作相似。同时，operator delete 根据预先存储的 meta 信息，调用 munmap 释放空间。这样之后，任何一个通过 c++ new 创建的对象都会自动在进程间共享。

为了在 stl 库中使用我想要的空间分配方法，我写了一个功能相同的 std::allocator，它被传给 stl 容器，以便告诉容器应当如何分配我想要的元素。他的名字是::mmap_allocator。

随后，由于我不了解 c++ std::mutex 的具体实现，由于这个奇怪的类甚至是 unmovable 的，我自己写了一个满足 std layout 的 rlib::stdlayout_mutex，放进我的 cache 类中。由于 stl lib 的良好实现，c++线程同步标准库的所有需要 mutex 的组件都可以使用我的 mutex 代替 std::mutex。这使得我不必自己实现一个 std::lock/std::lock_guard/std::unique_lock 等等。

然后，我创建了一个 std::list，这是一个连接着所有缓冲区的双向链表。我随意指定了一个长度 16。但是，我想要一个圆环状的链表。于是我写了一个 rlib::circularIterator 用来代替 std::iterator 参与迭代，这样这个链表“看起来”就是圆环状的了。父进程创建了 cache list 之后，两个子进程访问它们，别忘了 cache 这个 standard layout 的类的内存是在所有进程之间共享的(包括里面的 mutex)，因此我可以像对待两个线程一样对待它们。

利用如此丰富的工具，在两个线程之间同步一个环形缓冲区简直易如反掌。他们就像两只在圆环中游动追逐的贪吃蛇，只需要简单处理一下这它们咬到对方或自己的尾巴的情形就可以了。当所有数据发送完毕之后，发送者将在这个环内放置一个豆豆(EOF)然后逃走，接收者的那条蛇吃到这个豆豆便知道后面不会再有有用的数据了，于是它也离开现场，结束。

3.4 实验调试

3.4.1 实验步骤

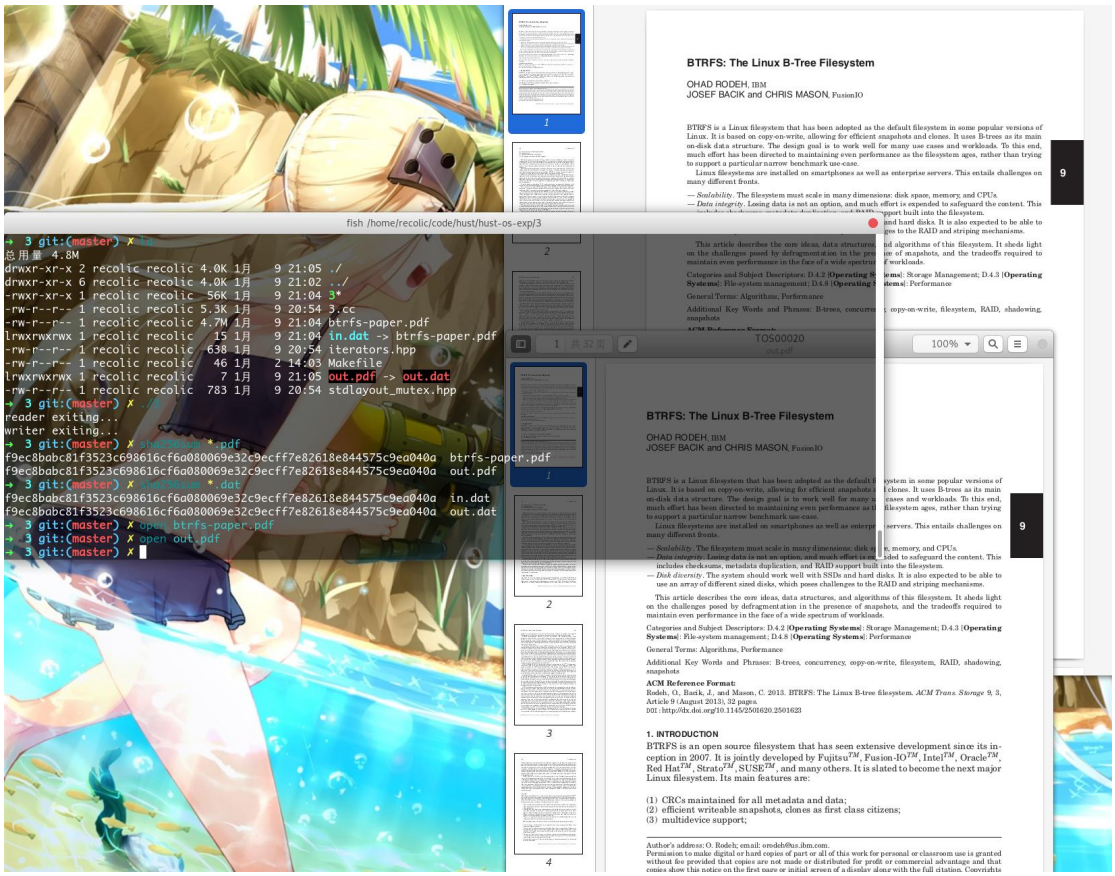
完成两个头文件，然后完成 3.cc 的所有逻辑。详细步骤在上一部分已经说明。

3.4.2 实验调试及心得

本次实验过程中遇到了一个错误（stl 的坑）。在重载了 operator new 之后，std::list<cache>没有使用重载之后的内存分配函数。我在调试时只能看到那一块内存始终为空，却不会有 segmentation fault 出现。花了几十分钟增加了很多 log 才发现，operator new 只对手写的 new 好用，stl 强制使用 allocator 而不是直接调用 operator new。

注意，要求 linux 环境，要求支持 c++14，c++17 或 c++20 标准的编译器。

如测试图所示，输入输出的两个 pdf 文件内容相同，sha256 校验和相等。



附录 实验代码

iterators.hpp 实现了环状迭代器

```
#ifndef RLIB_ITERATOR_HPP_
#define RLIB_ITERATOR_HPP_ 1

namespace rlib {
template <class baseIter> class circularIterator {
private:
  baseIter cur;
  baseIter begin;
  baseIter end;
```

```
public:
  circularIterator(baseIter b, baseIter e) : cur(b), begin(b), end(e)
{}
  circularIterator(baseIter b, baseIter e, baseIter c)
   : cur(c), begin(b), end(e) {}
```

```
  baseIter &operator++() {
    ++cur;
    if (cur == end) {
      cur = begin;
    }
    return cur;
  }
  typename baseIter::pointer operator->() { return &*cur; }
  typename baseIter::reference operator*() { return *cur; }
};
} // namespace rlib
```

```
#endif
```

stdlayout_mutex.hpp，实现了 standard layout 的 mutex.

```
#ifndef RLIB_NAIVE_MUTEX_HPP_
#define RLIB_NAIVE_MUTEX_HPP_ 1

#include <atomic>
#include <stdexcept>
```

```
namespace rlib {
```

```
//! Mutex with standard layout.
class stdlayout_mutex {
public:
  stdlayout_mutex() : locked(false) {}
  void lock() {
    while (true) {
      if (try_lock())
        return;
    }
  }
  bool try_lock() {
    bool expected = false;
    return locked.compare_exchange_strong(expected, true);
  }
  bool is_locked() { return locked; }
  void unlock() {
    if (not locked)
      throw std::logic_error("unlock a unlocked mutex.");
    bool expected = true;
```

```
    while (not locked.compare_exchange_strong(expected, false))
      ; // Try again
  }
```

```
private:
  std::atomic<bool> locked;
};
```

```
} // namespace rlib
```

```
#endif
```

3.cc，实现了其他逻辑

```
#include "iterators.hpp"
#include "stdlayout_mutex.hpp"
```

```
#include <chrono>
#include <fstream>
#include <list>
#include <memory>
#include <mutex>
#include <thread>
```

```
#include <sys/mman.h>
#include <sys/types.h>
#include <unistd.h>
```

```
#include <rlib/impl/traceable_list.hpp>
#include <rlib/stdio.hpp>
#include <rlib/string.hpp>
```

```
using namespace rlib::literals;
using namespace std::chrono_literals;
```

```
void *create_shared_memory(size_t size) {
    // Our memory buffer will be readable and writable:
    int protection = PROT_READ | PROT_WRITE;
```

```
    // The buffer will be shared (meaning other processes can access
it), but
    // anonymous (meaning third-party processes cannot obtain an
address for
```

```
    // it), so only this process and its children will be able to
use it:
    int visibility = MAP_ANONYMOUS | MAP_SHARED;

    // The remaining parameters to `mmap()` are not important for
this use case,
    // but the manpage for `mmap` explains their purpose.
    auto ptr = mmap(NULL, size, protection, visibility, 0, 0);

    if (ptr == MAP_FAILED)
        throw std::runtime_error(
         "mmap failed. system error: {}"_format(strerror(errno)));
    return ptr;
}

// See StackOverflow replies to this answer for important commentary
about
// inheriting from std::allocator before replicating this code.
template <typename T> class mmap_allocator : public
std::allocator<T> {
public:
    typedef size_t size_type;
    typedef T *pointer;
    typedef const T *const_pointer;

    template <typename _Tp1> struct rebind {
        typedef mmap_allocator<_Tp1> other;
    };

    pointer allocate(size_type size, const void *hint = 0) {
        void *ptr = create_shared_memory(size + sizeof(size));
        size_t *size_ptr = reinterpret_cast<size_t *>(ptr);
        *size_ptr = size;
        return (pointer)++size_ptr;
    }

    void deallocate(pointer ptr, size_type n) {
        size_t *size_ptr = reinterpret_cast<size_t *>(ptr);
        --size_ptr;
        auto res = munmap(size_ptr, *size_ptr);
        if (res == -1)
            throw std::runtime_error(
             "munmap failed. system error:
{}"_format(strerror(errno)));
```

```
    }
```

```
    mmap_allocator() throw() : std::allocator<T>() {}
    mmap_allocator(const mmap_allocator &a) throw() :
std::allocator<T>(a) {}
    template <class U>
    mmap_allocator(const mmap_allocator<U> &a) throw() :
std::allocator<T>(a) {}
    ~mmap_allocator() throw() {}
};
```

```
class cache {
public:
    static void *operator new(size_t size) {
        void *ptr = create_shared_memory(size + sizeof(size));
        size_t *size_ptr = reinterpret_cast<size_t *>(ptr);
        *size_ptr = size;
        return ++size_ptr;
    }
    static void *operator new[](size_t size) { return operator
new(size); }
    static void operator delete(void *ptr) {
        size_t *size_ptr = reinterpret_cast<size_t *>(ptr);
        --size_ptr;
        auto res = munmap(size_ptr, *size_ptr);
        if (res == -1)
            throw std::runtime_error(
             "munmap failed. system error:
{}"_format(strerror(errno)));
    }
    static void operator delete[](void *ptr) { return operator
delete(ptr); }
```

```
    bool dirty = false;
    rlib::stdlayout_mutex mut;
```

```
    std::array<char, 1024> data;
    size_t data_len;
```

```
    bool eof_flag = false;
```

```
private:
};
```

```
int main() {
    std::list<cache, mmap_allocator<cache>> caches(16);
```

```
    auto pid = fork();
    if (0 == pid) {
        // child 1
        // open file to write
        std::ofstream o("./out.dat",
         std::ios_base::binary | std::ios_base::trunc);
        rlib::circularIterator<decltype(caches.begin())> iter(
         caches.begin(), caches.end(), caches.begin());
        for (; true; ++iter) {
            std::lock_guard<rlib::stdlayout_mutex>
lock(iter->mut);
            if (iter->dirty) {
                // write into file
                o.write(iter->data.data(), iter->data_len);
                iter->dirty = false;
            }
            if (iter->eof_flag)
                break;
        }
        rlib::println("writer exiting...");
        return 0; // fileWriter done his job!
    } else {
        // parent
        // open file to read data into memory
        std::ifstream i("./in.dat", std::ios::binary |
std::ios::ate);
        if (not i)
            throw std::runtime_error("Failed to open
file ./in.dat");
        size_t fileSize = i.tellg();
        i.seekg(0);
```

```
        rlib::circularIterator<decltype(caches.begin())>
iter(caches.begin(),
         caches.end());
        for (; true; ++iter) {
            std::unique_lock<rlib::stdlayout_mutex>
lock(iter->mut);
            if (iter->dirty) {
                // Maybe: reader and writer hit their head!
                // wait for the writer plz...
```

```
                // Maybe: reader ate its own snake tail!
                lock.unlock();
                while (iter->dirty)
                    ;
                // writer finally catched...
                lock.lock();
            }
```

```
            // OK. do the file reading.
            auto blockSize = (fileSize > 1024) ? 1024 : fileSize;
            i.read(iter->data.data(), blockSize);
            iter->data_len = blockSize;
            iter->dirty = true;
            fileSize -= blockSize;
            lock.unlock();
            if (fileSize == 0) {
                iter->eof_flag = true;
                rlib::println("reader exiting...");
                return 0; // fileReader done his job!
            }
        }
    }
}
```

4 实验四 Linux 文件目录

4.1 实验目的

1、了解 Linux 文件系统与目录操作；

2、了解 Linux 文件系统目录结构；

3、掌握文件和目录的程序设计方法。

4.2 实验内容

编程实现目录查询功能：

功能类似 ls -lR；

查询指定目录下的文件及子目录信息；

显示文件的类型、大小、时间等信息；

递归显示子目录中的所有文件信息。

4.3 实验设计

4.3.1 开发环境

```
recolic@RECOLICPC
OS: Arch Linux
Kernel: x86_64 Linux 5.0-rc1
Uptime: 27d 2h 32m
Packages: 1795
Shell: fish 2.7.1
Resolution: 4480x1440
DE: GNOME
WM: GNOME Shell
WM Theme:
```

```
GTK Theme: Gnome-OSX-II-2-6 [GTK2/3]
Icon Theme: hicolor
Font: Cantarell 11
CPU: Intel Core i5-4200H @ 4x 3.4GHz [59.0°C]
GPU: GeForce GTX 950M
RAM: 2734MiB / 15489MiB
Disk: 1TiB ext4/btrfs/ntfs + 2TiB nfs


recolic@RECOLICMPC
OS: Arch Linux
Kernel: x86_64 Linux 4.20.0-arch1-1-ARCH
Uptime: 4d 16h 45m
Packages: 1201
Shell: fish 2.7.1
Resolution: 1920x1080
DE: GNOME
WM: GNOME Shell
WM Theme: Human
GTK Theme: Gn-OSX-HSierra-1.2.1-S [GTK2/3]
Icon Theme: la-capitaine-icon-theme
Font: Cantarell 11
CPU: Intel Core m3-7Y30 @ 4x 2.6GHz [26.0°C]
GPU: intel
RAM: 2002MiB / 3828MiB
Disk: 128GiB btrfs + 2TiB nfs
```

4.3.2 实验设计

此次实验使用了 C++17 新加入的 std::filesystem，其中仅有不到 10 行代码用于递归遍历目录，其余近 100 行代码用来使输出更美观。

那 10 行代码是极其简单的一个递归，列出并打印目录中的所有 entry，以及 main 函数的命令行解析。

其余的 100 行都是将各种数据类型转换成 std::string 或对 operator<<输出运算符的重载，逻辑非常简单，阅读源码即可。

4.4 实验调试

4.4.1 实验步骤

此次实验使用了 C++17 新加入的 std::filesystem，其中仅有不到 10 行代码用于递归遍历目录，其余近 100 行代码用来使输出更美观。

那 10 行代码是极其简单的一个递归，列出并打印目录中的所有 entry，以及 main 函数的命令行解析。

其余的 100 行都是将各种数据类型转换成 std::string 或对 operator<<输出运算符的重载，逻辑非常简单，阅读源码即可。

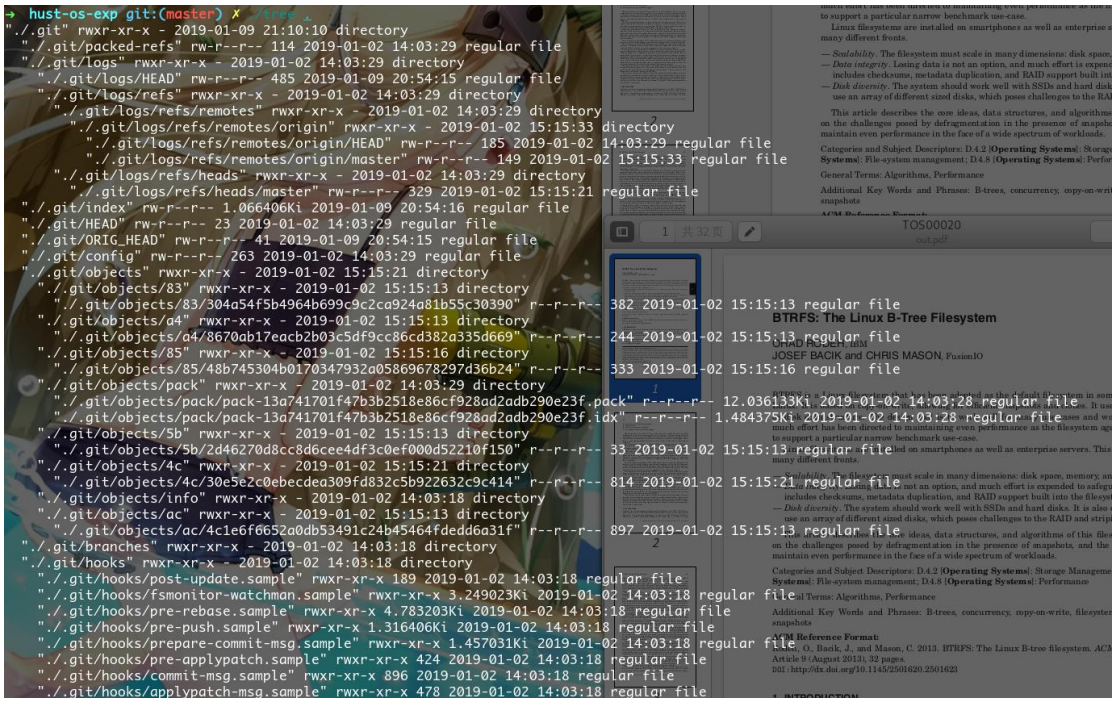
4.4.2 实验调试及心得

第一次编译完成后执行代码，发现其在/dev/fd/3 出发生死循环。由于这个目录对每个进程都不一样，其代表 3 号 file descriptor。没有进行详细的 debug，直接暴力 hardcode 跳过。

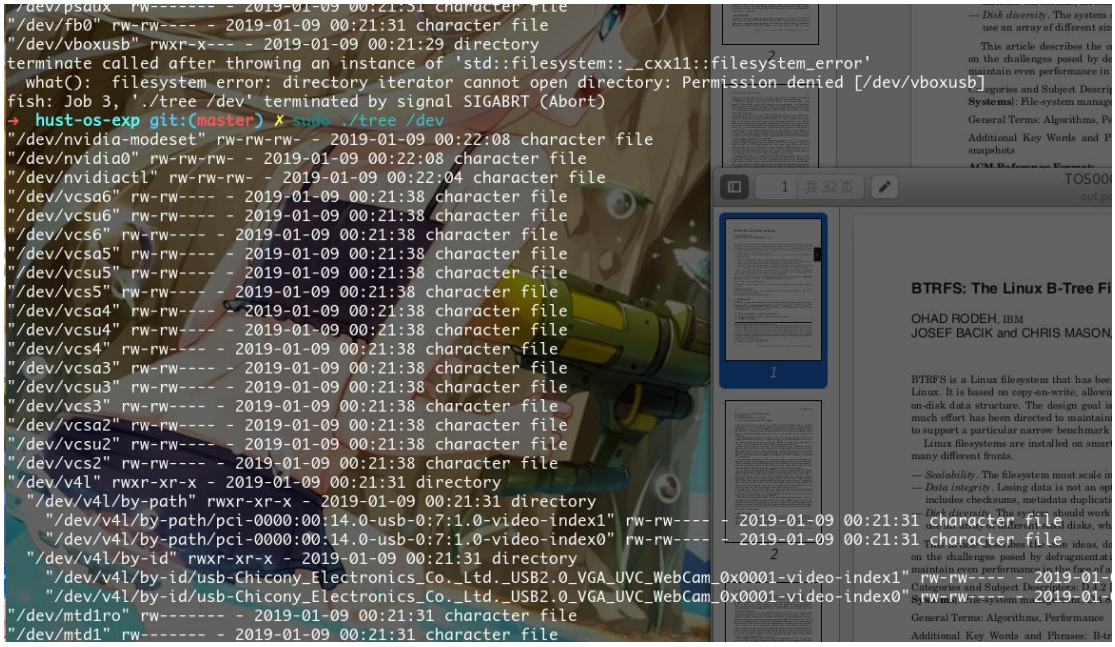
注意，要求 linux 环境，要求支持 **c++17 或 c++20** 标准的编译器。

下面是几张测试图，其结果都已经验证为正确。其能对我目前找到的所有 linux 目录正确操作，包括整个/dev 和/proc。

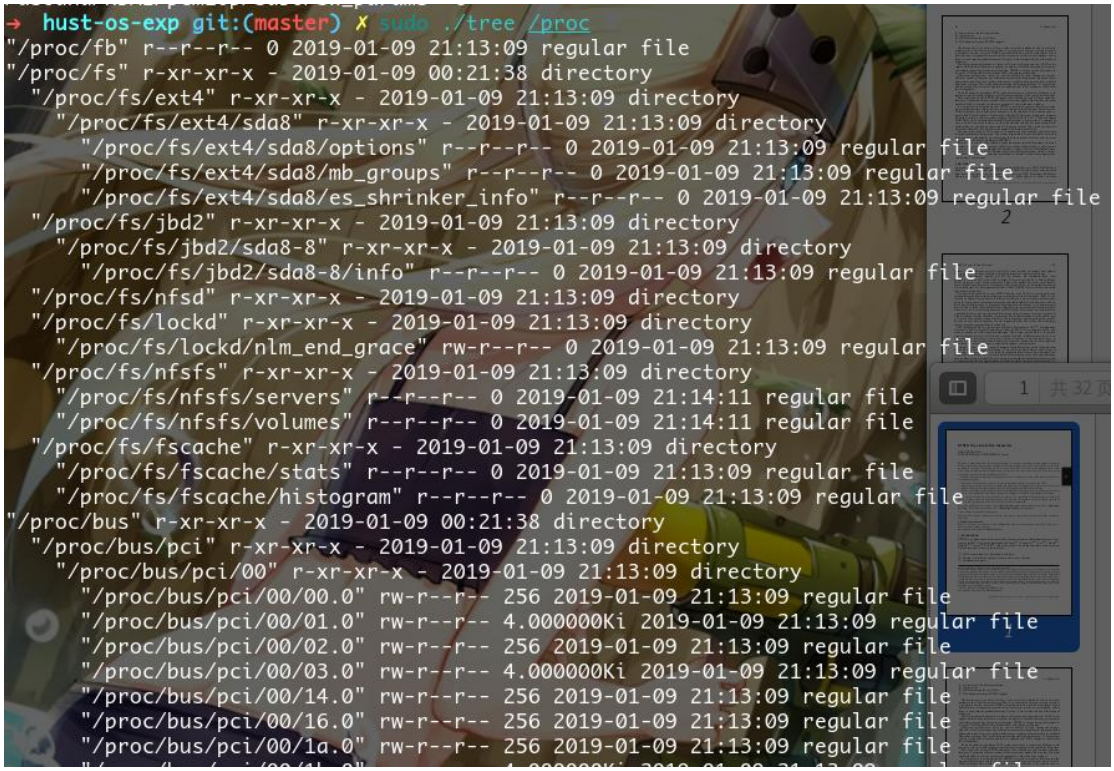
这是我的工程目录。



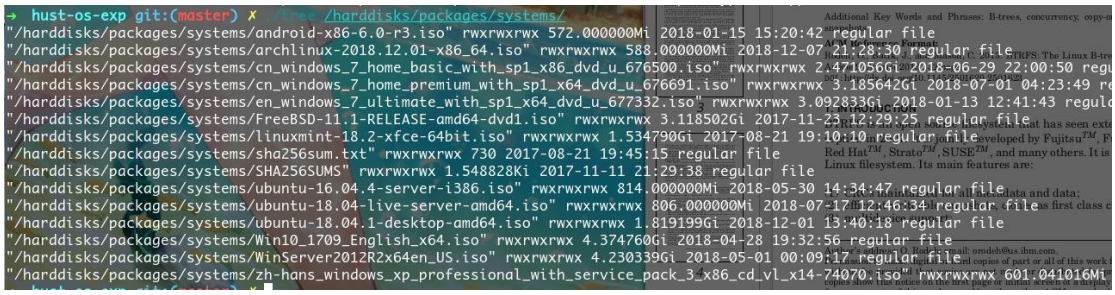
这是/dev。第一次运行没有提供 root 权限，因此程序抛出异常退出。



/proc/self/task/fd/3 也会出现类似/dev/fd/3 的无限循环问题，也是通过 hardcode 解决的。



对于大文件，其能够自动选择恰当的单位显示文件大小。



附录 实验代码

4.cc 主要逻辑在这里。

```
#include "fs_prettyprint.hpp"
#include <filesystem>
#include <rlib/opt.hpp>
#include <rlib/stdio.hpp>
```

```
using namespace rlib::literals;
using namespace rlib::prettyprint;
namespace fs = std::filesystem;
```

```
void list_dir(const fs::path &dir, const size_t depth) {
```

```
    if (std::string(dir).find("/dev/fd/") == 0)
        return;
    for (const auto &entry : fs::directory_iterator(dir)) {
        rlib::printfln("{}{} {}", std::string(depth * 2, ' '),
entry.path(),
         entry);
        if (entry.is_directory())
            list_dir(entry, depth + 1);
    }
}
```

```
int main(int argc, char **argv) {
    rlib::opt_parser args(argc, argv);
    std::string path = args.getSubCommand();
```

```
    list_dir(path, 0);
}
```

fs_prettyprint.hpp 用来为 stdc++fs 的对象提供更美观的输出。

```
#ifndef RLIB_FS_STATUS_PRETTYPRINT
#define RLIB_FS_STATUS_PRETTYPRINT

#include <filesystem>
#include <iomanip>
```

```
namespace rlib::prettyprint {
```

```
namespace impl {
inline std::string ftypeToString(const std::filesystem::file_type
&t) {
switch(t) {
case std::filesystem::file_type::regular: return "regular file";
case std::filesystem::file_type::directory: return "directory";
case std::filesystem::file_type::symlink: return "symlink";
case std::filesystem::file_type::block: return "block file";
case std::filesystem::file_type::character: return "character
file";
case std::filesystem::file_type::fifo: return "fifo";
case std::filesystem::file_type::socket: return "socket";
case std::filesystem::file_type::unknown: return "unknown";
case std::filesystem::file_type::none: return "none";
case std::filesystem::file_type::not_found: return "not_found";
}
```

```
return "";
}
```

```
inline std::string fsizeToString(const size_t fsize) {
if(fsize < 1024)
return std::to_string(fsize);
const auto KiB = (double)fsize / 1024.;
if(KiB < 1024) return std::to_string(KiB) + "Ki";
const auto MiB = KiB / 1024.;
if(MiB < 1024) return std::to_string(MiB) + "Mi";
const auto GiB = MiB / 1024.;
if(GiB < 1024) return std::to_string(GiB) + "Gi";
const auto TiB = GiB / 1024.;
if(TiB < 1024) return std::to_string(TiB) + "Ti";
const auto PiB = TiB / 1024.;
return std::to_string(PiB) + "Pi";
}
```

```
template <typename T>
std::string try_show_file_size(const T &arg) {
try {
return fsizeToString(std::filesystem::file_size(arg));
}
catch(...) {
return "-";
}
}
}
```

```
template<typename CharT, typename Traits>
std::basic_ostream<CharT, Traits>&
operator<< (std::basic_ostream<CharT, Traits> &os, const
std::filesystem::perms& p) {
return os
<< ((p & std::filesystem::perms::owner_read) !=
std::filesystem::perms::none ? "r" : "-")
<< ((p & std::filesystem::perms::owner_write) !=
std::filesystem::perms::none ? "w" : "-")
<< ((p & std::filesystem::perms::owner_exec) !=
std::filesystem::perms::none ? "x" : "-")
<< ((p & std::filesystem::perms::group_read) !=
std::filesystem::perms::none ? "r" : "-")
<< ((p & std::filesystem::perms::group_write) !=
std::filesystem::perms::none ? "w" : "-")
```

```
<< ((p & std::filesystem::perms::group_exec) !=
std::filesystem::perms::none ? "x" : "-")
<< ((p & std::filesystem::perms::others_read) !=
std::filesystem::perms::none ? "r" : "-")
<< ((p & std::filesystem::perms::others_write) !=
std::filesystem::perms::none ? "w" : "-")
<< ((p & std::filesystem::perms::others_exec) !=
std::filesystem::perms::none ? "x" : "-")
;
}
```

```
template<typename CharT, typename Traits>
std::basic_ostream<CharT, Traits>&
operator<< (std::basic_ostream<CharT, Traits> &os, const
std::filesystem::file_type& t) {
return os << impl::ftypeToString(t);
}
```

```
template<typename CharT, typename Traits>
std::basic_ostream<CharT, Traits>&
operator<< (std::basic_ostream<CharT, Traits> &os, const
std::filesystem::directory_entry& entry) {
auto lastWrite =
std::chrono::system_clock::to_time_t(entry.last_write_time());
return os << entry.status().permissions() << ' ' <<
impl::try_show_file_size(entry) << ' ' <<
std::put_time(std::localtime(&lastWrite), "%F %T") << ' ' <<
entry.status().type();
}
```

```
} // end namespace rlib::prettyprint
#endif
```

5 附录：编译使用的 Sconstruct

注：其中的 LIB`r`在附录 6 列出源码。

```
ccflags = '-std=c++17'

# No need to compile rlib from source
# rlibenv=Environment(CPPPATH='./lib/rlib', CPPFLAGS='-std=c++17')
# rlibenv.Library('r', ['lib/rlib/libr.cc'])

env=Environment(CPPDEFINES=[], LIBS=['r'], CPPFLAGS='-std=c++17')
env.Program('1', '1.cc', LIBS=['pthread','r'], CPPFLAGS='')
env.Program('2', '2.cc', LIBS=['pthread','r'])
env.Program('2-proc', '2-proc.cc', LIBS=['r'])
env.Program('3', '3.cc', LIBS=['r'])
env.Program('tree', '4.cc', LIBS=['stdc++fs','r'])
```

6 附录：rlib 快照 于 Jan 9 2019

```
commit 3a442c6dd8661d45cfe7528112b93c42ffa5d591 (HEAD -> master, origin/master, origin/HEAD)
Author: Recolic Keghart <root@recolic.net>
Date:   Tue Jan 8 19:31:15 2019 +0800

    fix a compilation time error: missing include. fix some warning
```

我使用如下命令生成快照：

```
for fl in (find . -type f  -not -path '*/\.*')
    echo "/******* $fl ****************/"
    cat $fl
end > /tmp/out.log
```

快照内容如下：

```
/******* ./traits.hpp ****************/
#ifndef RLIB_TRAITS_HPP
#define RLIB_TRAITS_HPP

#include <type_traits>

namespace rlib{
    namespace impl {
        template<typename T>
        struct is_callable_helper {
```

```
		private:
			typedef char(&yes)[1];
			typedef char(&no)[2];

			struct Fallback { void operator()(); };
			struct Derived : T, Fallback { };

			template<typename U, U> struct Check;

			template<typename>
			static yes test(...);

			template<typename C>
			static no test(Check<void (Fallback::*)(), &C::operator()>*);

		public:
			static constexpr bool value = sizeof(test<Derived>(0)) == sizeof(yes);
		};
	} //impl
} //rlib

namespace rlib {
	template<typename T>
	struct is_callable {
		using _impl = typename std::conditional<std::is_class<T>::value, impl::is_callable_helper<T>, std::is_function<T>>::type;
		static constexpr bool value() noexcept {
			return _impl::value;
		}
		constexpr operator bool() noexcept {
			return is_callable<T>::value();
		}
	};
}

#endif
/******* ./Makefile ****************/
CXX ?= g++
CC ?= gcc
AR ?= ar
CXXFLAGS = -O3 -std=c++1z -fPIC
CFLAGS =
ARFLAGS = rcs

PREFIX ?= /usr

def: compile_library

compile_library:
		$(CXX) $(CXXFLAGS) -c libr.cc -I . -o libr.o
		$(AR) $(ARFLAGS) libr.a libr.o

install_header:
		[ ! -d $(PREFIX)/include/rlib ] || rm -rf $(PREFIX)/include/rlib
		cp -r , $(PREFIX)/include/rlib
		rm -rf $(PREFIX)/include/rlib/test $(PREFIX)/include/rlib/.git

install_library: compile_library
		cp libr.a $(PREFIX)/lib/

install_cmake: install_library
		[ ! -d $(PREFIX)/lib/cmake/rlib ] || rm -rf $(PREFIX)/lib/cmake/rlib
		[ ! -d $(PREFIX)/lib/cmake ] || cp -r cmake $(PREFIX)/lib/cmake/rlib

install: install_header install_library install_cmake

uninstall:
		rm -rf $(PREFIX)/include/rlib $(PREFIX)/lib/cmake/rlib
		rm -f $(PREFIX)/lib/libr.a

clean:
		rm *.o *.a
/******* ./LICENSE ****************/
MIT License

Copyright (c) 2017-2018 Recolic Keghart <root@recolic.net>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
/******* ./string.hpp ****************/
/*
 *
 * string.hpp: string process utility.
 * Recolic Keghart <root@recolic.net>
 * MIT License
 *
 */

#ifndef R_STRING_HPP
#define R_STRING_HPP

#include <rlib/require/cxx14>
#include <rlib/class_decorator.hpp>
#include <rlib/sys/os.hpp>

#include <array>
#include <vector>
#include <string>
#include <cstdarg>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <stdexcept>
#include <sstream>
#include <type_traits>

namespace rlib {
	// literals::_format, format_string, string::format
	namespace impl {
#ifndef RLIB_MINGW_DISABLE_TLS
//#if RLIB_CXX_STD < 2017
// Intel C++ compiler has a pending bug for `thread_local inline` variable.
		thread_local extern std::stringstream to_string_by_sstream_ss;
		thread_local extern std::stringstream _format_string_helper_ss;
//#else
//		thread_local inline std::stringstream to_string_by_sstream_ss;
//		thread_local inline std::stringstream _format_string_helper_ss;
//#endif
#endif
		template <typename VarT>
		std::string to_string_by_sstream(VarT &thing) {
#ifdef RLIB_MINGW_DISABLE_TLS // Fix intel C++ bug https://software.intel.com/en-us/forums/intel-c-compiler/topic/784136
			// Also fix mingw bug. But much slower!
			std::stringstream ss;
#else
```

```
		auto &ss = to_string_by_sstream_ss;
		ss.str(std::string());
#endif
		ss << thing;
		return ss.str();
	}

	template<typename... Args>
	std::string _format_string_helper(const std::string &fmt, Args... args) {
#ifdef RLIB_MINGW_DISABLE_TLS // Fix intel C++ bug https://software.intel.com/en-us/forums/intel-c-compiler/topic/784136
		std::stringstream ss;
#else
		auto &ss = _format_string_helper_ss; // cached stringstream is much quicker.
		ss.str(std::string());
#endif
		size_t pos = 0, prev_pos = 0;
		std::array<std::string, sizeof...(args)> argsArr{to_string_by_sstream(args) ...};
		size_t current_used_arg = 0;
		bool discovered_escape_char = false;
		while((pos = fmt.find("{}", pos)) != std::string::npos) {
			if(pos != 0 && fmt[pos-1] == '\\') {
				// Invalid hit.
				discovered_escape_char = true;
				pos += 2;
			}
			else {
				std::string cutted_tmp_str = fmt.substr(prev_pos, pos - prev_pos);
				if(discovered_escape_char) {
					// hand-written string replace. Replace `\{}` to `{}`.
					size_t pos = 0;
					while((pos = cutted_tmp_str.find("\\{}", pos)) != std::string::npos) {
						cutted_tmp_str.erase(pos, 1);
						pos += 2;
					}
				}
				ss << cutted_tmp_str << argsArr[current_used_arg];
				pos += 2;
				prev_pos = pos;
				++current_used_arg;
			}
		}
		ss << fmt.substr(prev_pos);
		return ss.str();
	}
	template<typename... Args>
	inline std::string format_string(const std::string &fmt, Args... args) {
		return _format_string_helper(fmt, args...);
	}
	template<>
	inline std::string format_string<>(const std::string &fmt) {
		return fmt;
	}

	/*
	template<class MetaFmtArr, typename... Args>
	constexpr std::string format_string_meta(Args... args) {
		return (args + ...);
	}*/
}

// format_string_c, string::cformat
namespace impl {
	inline char *_format_string_c_helper(const char *fmt, ...)
	{
		int n;
		int size = std::strlen(fmt);
		char *p, *np;
		va_list ap;

		if ((p = (char *)malloc(size)) == NULL)
			throw std::runtime_error("malloc returns null.");

		while (1) {
			va_start(ap, fmt);
			n = vsnprintf(p, size, fmt, ap);
			va_end(ap);

			if (n < 0)
				throw std::runtime_error("vsnprintf returns " + std::to_string(n));
			if (n < size)
				return p;

			size = n + 1;

			if ((np = (char *)realloc (p, size)) == NULL) {
				free(p);
				throw std::runtime_error("make_message realloc failed.");
			} else {
				p = np;
			}
		}
	}
	template<typename... Args>
	std::string format_string_c(const std::string &fmt, Args... args)
	{
		char *res = _format_string_c_helper(fmt.c_str(), args ...);
		std::string s = res;
		free(res);
		return s;
	}
}

class string : public std::string {
public:
	using std::string::string;
	string() : std::string() {}
	string(const std::string &s) : std::string(s) {}
	string(std::string &&s) : std::string(std::forward<std::string>(s)) {}

private:
	template <typename T> struct as_helper {};
	template <typename T>
	T as(as_helper<T>) const {
		if(empty()) return T();
		return T(*this);
	}
	std::string as(as_helper<std::string>) const {
		return std::move(*this);
	}
	rlib::string as(as_helper<rlib::string>) const {
		return std::move(*this);
	}
	char as(as_helper<char>) const {
		if(size() > 1)
			throw std::invalid_argument("Can not convert rlib::string to char: size() > 1.");
		return size() == 0 ? '\0' : *cbegin();
	}
	unsigned char as(as_helper<unsigned char>) const {
		return static_cast<unsigned char>(as<char>());
	}
	bool as(as_helper<bool>) const {
		if(*this == "true") {
			return true;
		}
		else if(*this == "false") {
			return false;
		}
		// Nothing is slower than throw(); Just test more cases...
		else if(*this == "1" || *this == "True" || *this == "TRUE") {
```

```
                return true;
            }
            else if(*this == "0" || *this == "False" || *this == "FALSE") {
                return false;
            }
            throw std::invalid_argument("Can not convert rlib::string to bool. Not matching any template.");
        }

#define RLIB_IMPL_GEN_AS_NUMERIC(type, std_conv) \
        type as(as_helper<type>) const { \
            if(empty()) return 0; \
            return std::std_conv(*this); \
        }

        RLIB_IMPL_GEN_AS_NUMERIC(int, stoi)
        RLIB_IMPL_GEN_AS_NUMERIC(long, stol)
        RLIB_IMPL_GEN_AS_NUMERIC(unsigned long, stoul)
        RLIB_IMPL_GEN_AS_NUMERIC(unsigned long long, stoull)
        RLIB_IMPL_GEN_AS_NUMERIC(long long, stoll)
        RLIB_IMPL_GEN_AS_NUMERIC(float, stof)
        RLIB_IMPL_GEN_AS_NUMERIC(double, stod)
        RLIB_IMPL_GEN_AS_NUMERIC(long double, stold)

#define RLIB_IMPL_GEN_AS_ALIAS(new_type, old_type) \
        new_type as(as_helper<new_type>) const { \
            return static_cast<new_type>(as<old_type>()); \
        }

        RLIB_IMPL_GEN_AS_ALIAS(unsigned int, unsigned long)
        RLIB_IMPL_GEN_AS_ALIAS(unsigned short, unsigned long)
        //RLIB_IMPL_GEN_AS_ALIAS(uint8_t, unsigned long)

        RLIB_IMPL_GEN_AS_ALIAS(short, int)
        //RLIB_IMPL_GEN_AS_ALIAS(int8_t, int)

    public:
        template <typename T>
        T as() const {
            return std::forward<T>(as(as_helper<T>()));
        }

        std::vector<string> split(const char &divider = ' ') const {
            const string &toSplit = *this;
            std::vector<string> buf;
            size_t curr = 0, prev = 0;
            while((curr = toSplit.find(divider, curr)) != std::string::npos) {
                buf.push_back(toSplit.substr(prev, curr - prev));
                ++curr; // skip divider
                prev = curr;
            }
            buf.push_back(toSplit.substr(prev));
            return buf;
        }
        std::vector<string> split(const std::string &divider) const {
            const string &toSplit = *this;
            std::vector<string> buf;
            size_t curr = 0, prev = 0;
            while((curr = toSplit.find(divider, curr)) != std::string::npos) {
                buf.push_back(toSplit.substr(prev, curr - prev));
                curr += divider.size(); // skip divider
                prev = curr;
            }
            buf.push_back(toSplit.substr(prev));
            return buf;
        }
        template <typename T>
        std::vector<T> split_as(const char &divider = ' ') const {
            const string &toSplit = *this;
            std::vector<T> buf;
            size_t curr = 0, prev = 0;
            while((curr = toSplit.find(divider, curr)) != std::string::npos) {
                buf.push_back(string(toSplit.substr(prev, curr - prev)).as<T>());
                ++curr; // skip divider
                prev = curr;
            }
            buf.push_back(string(toSplit.substr(prev)).as<T>());
            return std::move(buf);
        }
        template <typename T>
        std::vector<T> split_as(const std::string &divider) const {
            const string &toSplit = *this;
            std::vector<T> buf;
            size_t curr = 0, prev = 0;
            while((curr = toSplit.find(divider, curr)) != std::string::npos) {
                buf.push_back(string(toSplit.substr(prev, curr - prev)).as<T>());
                curr += divider.size(); // skip divider
                prev = curr;
            }
            buf.push_back(string(toSplit.substr(prev)).as<T>());
            return std::move(buf);
        }

        template <class ForwardIterable>
        string &join(const ForwardIterable &buffer) {
            join(buffer.cbegin(), buffer.cend());
            return *this;
        }
        template <class ForwardIterator>
        string &join(ForwardIterator begin, ForwardIterator end) {
            const string &toJoin = *this;
            std::string result;
            for(ForwardIterator iter = begin; iter != end; ++iter) {
                if(iter != begin)
                    result += toJoin;
                result += *iter;
            }
            return operator=(std::move(result));
        }

        string &strip() {
            strip(" \t\r\n");
            return *this;
        }
        template <typename CharOrStringOrView>
        string &strip(const CharOrStringOrView &stripped) {
            size_t len = size();
            size_t begin = find_first_not_of(stripped);

            if(begin == std::string::npos) {
                clear();
                return *this;
            }
            size_t end = find_last_not_of(stripped);

            erase(end + 1, len - end - 1);
            erase(0, begin);
            return *this;
        }

        string &replace(const std::string &from, const std::string &to) {
            size_t _;
            replace(from, to, _);
            return *this;
        }
        string &replace(const std::string &from, const std::string &to, size_t &out_times) {
            if(from.empty())
                return *this;
            size_t start_pos = 0;
            size_t times = 0;
```

```
		while((start_pos = find(from, start_pos)) != std::string::npos)
		{
			++times;
			this->std::string::replace(start_pos, from.length(), to);
			start_pos += to.length(); // In case 'to' contains 'from', like replacing 'x' with 'yx'
		}
		out_times = times;
		return *this;
	}
	string &replace_once(const std::string &from, const std::string &to) {
		bool _;
		replace_once(from, to, _);
		return *this;
	}
	string &replace_once(const std::string &from, const std::string &to, bool &out_replaced) {
		size_t start_pos = find(from);
		if(start_pos == std::string::npos) {
			out_replaced = false;
		}
		else {
			this->std::string::replace(start_pos, from.length(), to);
			out_replaced = true;
		}
		return *this;
	}

	template <typename... Args>
	string &format(Args... args) {
		return operator=(std::move(impl::format_string(*this, args ...)));
	}
	template <typename... Args>
	string &cformat(Args... args) {
		return operator=(std::move(impl::format_string_c(*this, args ...)));
	}
};

namespace impl {
	struct formatter {
		formatter(const std::string &fmt) : fmt(fmt) {}
		formatter(std::string &&fmt) : fmt(fmt) {}
		template <typename... Args>
		std::string operator ()(Args... args) {
			return std::move(rlib::impl::format_string(fmt, args ...));
		}

		std::string fmt;
	};
}

namespace literals {
	inline impl::formatter operator "" _format (const char *str, size_t) {
		return impl::formatter(str);
	}
	inline rlib::string operator "" _rs (const char *str, size_t len) {
		return rlib::string(str, len);
	}
}
}

#endif
/******* ./sys/os.hpp ****************/
#ifndef R_OS_HPP
#define R_OS_HPP

#ifndef RLIB_OS_ID
#if defined(_Windows) || defined(__WIN32__) || defined(_WIN64) || defined(WIN32)
#	define RLIB_OS_ID OS_WINDOWS
#elif defined(__linux__) || defined(__linux)
#	define RLIB_OS_ID OS_LINUX
#elif defined(__APPLE__)
#	include "TargetConditionals.h"
#	if TARGET_IPHONE_SIMULATOR
#	define RLIB_OS_ID OS_IOS
#	elif TARGET_OS_IPHONE
#	define RLIB_OS_ID OS_IOS
#	elif TARGET_OS_MAC
#	define RLIB_OS_ID OS_MACOS
#	else
#	define RLIB_OS_ID OS_UNKNOWN
#	endif
#elif defined(__OS_ANDROID__)
#	define RLIB_OS_ID OS_ANDROID
#elif defined(__unix__) || defined(__unix)
#	define RLIB_OS_ID OS_UNIX
#else
#	define RLIB_OS_ID OS_UNKNOWN
#endif
#endif

#define RLIB_OS_ID_MAGIC 980427
#define OS_WINDOWS (RLIB_OS_ID_MAGIC + 1)
#define OS_LINUX     (RLIB_OS_ID_MAGIC + 2)
#define OS_MACOS     (RLIB_OS_ID_MAGIC + 3)
#define OS_BSD       (RLIB_OS_ID_MAGIC + 4)
#define OS_IOS       (RLIB_OS_ID_MAGIC + 5)
#define OS_ANDROID (RLIB_OS_ID_MAGIC + 6)
#define OS_UNIX      (RLIB_OS_ID_MAGIC + 7)
#define OS_UNKNOWN (RLIB_OS_ID_MAGIC + 8)

#include "compiler_detector"
// Define RLIB_COMPILER_ID and RLIB_COMPILER_VER

// shorthand for __cplusplus macro.
#ifndef RLIB_CXX_STD
#	if RLIB_COMPILER_ID == CC_MSVC
#		if defined(__cplusplus)
#		if __cplusplus == 199711L
#		pragma message (": warning MSVC_Wrong__cplusplus: Your MSVC is possibly set __cplusplus to wrong value. Please upgrade to VS2017 15.7 Preview 3 and recompile with /Zc:__cplusplus. Or I'll assume you support C++17 and set RLIB_CXX_STD to 2017. (refer to https://blogs.msdn.microsoft.com/vcblog/2018/04/09/msvc-now-correctly-reports-__cplusplus/)")
#		define RLIB_CXX_STD 2017
#		else
#		define RLIB_CXX_STD (__cplusplus / 100L)
#		endif
#		endif
#	else
#		if defined(__cplusplus)
#		define RLIB_CXX_STD (__cplusplus / 100L)
#		endif
#	endif
#endif

#if RLIB_CXX_STD >= 2011
namespace rlib {
	class os_info
	{
	public:
		enum class os_t {UNKNOWN = OS_UNKNOWN, WINDOWS = OS_WINDOWS, LINUX = OS_LINUX, MACOS = OS_MACOS, BSD = OS_BSD, IOS = OS_IOS, ANDROID = OS_ANDROID, UNIX = OS_UNIX};
		enum class compiler_t {UNKNOWN = CC_UNKNOWN, GCC = CC_GCC, CLANG = CC_CLANG, MSVC = CC_MSVC, ICC = CC_ICC, BORLAND = CC_BORLAND, IARC = CC_IARC, SOLARIS = CC_SOLARIS/*, ZAPCC = CC_ZAPCC*/};
		//C = CC_Compiler which not supports cxx1x yet is not listed here. 201708.

		static constexpr os_t os =
	#if defined(RLIB_OS_ID)
	(os_t)RLIB_OS_ID;
	#else
	os_t::UNKNOWN;
	#endif
```

```
        static constexpr compiler_t compiler =
    #if defined(RLIB_COMPILER_ID)
    (compiler_t)RLIB_COMPILER_ID;
    #else
    compiler_t::UNKNOWN;
    #endif
        static constexpr auto compiler_version =
    #if defined(RLIB_COMPILER_VER)
    RLIB_COMPILER_VER;
    #else
    0;
    #endif
    };
}

#endif

#endif
/******* ./sys/cc_codegen.py ****************/
#!/bin/env python3

rlib_cc_magic = 90713
def getSerialNumber():
    global rlib_cc_magic
    rlib_cc_magic += 1
    return rlib_cc_magic

def genDefList(idarr):
    s = '#if'
    cter = 1
    for i in idarr:
        s += ' defined(' + i + ')'
        if cter != len(idarr):
            s += ' ||'
        cter += 1
    return s

print('// Generated by cc_codegen.py. Do not edit it by hand.')

with open("cc_list") as fd:
    osarr=fd.read().split('\n')
    for i in osarr:
        if i == '':
            continue
        if i[0] == '#':
            continue
        iarr=i.split(' ')
        if len(iarr) < 2:
            continue
        print('#ifndef RLIB_COMPILER_ID')
        print(genDefList(iarr[:-1:]))
        print('#define RLIB_COMPILER_ID', 'CC_' + iarr[-1])
        print('#endif')
        print('#endif')
        print('#define CC_' + iarr[-1], getSerialNumber())
        print('')

print('#ifndef RLIB_COMPILER_ID')
print('#define RLIB_COMPILER_ID CC_UNKNOWN')
print('#endif')
print('#define CC_UNKNOWN', getSerialNumber())

/******* ./sys/fd.hpp ****************/
#ifndef RLIB_FD_HPP
#define RLIB_FD_HPP_

#include <rlib/sys/os.hpp>
#if RLIB_OS_ID == OS_WINDOWS
#include <Windows.h>
using fd_t = HANDLE;
using sockfd_t = SOCKET;
#else
using fd_t = int;
using sockfd_t = int;
#endif

#endif
/******* ./sys/rwlock.hpp ****************/
#ifndef R_SWLOCK_HPP
#define R_SWLOCK_HPP

#include <pthread.h>
namespace rlib {
    [[deprecated]] class RWLock
    {
    public:
        RWLock() : isFree(true) {pthread_rwlock_init(&m_lock, NULL);}
        ~RWLock() {pthread_rwlock_destroy(&m_lock);}
        void acquireShared() {pthread_rwlock_rdlock(&m_lock);isFree = false;}
        void acquireExclusive() {pthread_rwlock_wrlock(&m_lock);isFree = false;}
        void release() {pthread_rwlock_unlock(&m_lock);isFree = true;}
    //      bool tryAcquireShared() {return pthread_rwlock_tryrdlock(&m_lock) == 0;}
    //      bool tryAcquireExclusive() {return pthread_rwlock_trywrlock(&m_lock) == 0;}
    private:
        pthread_rwlock_t m_lock;
        bool isFree;
    };
}

#endif
/******* ./sys/compiler_detector ****************/
// Generated by cc_codegen.py. Do not edit it by hand.
#ifndef RLIB_COMPILER_ID
#if defined(__ACC__)
#define RLIB_COMPILER_ID CC_ACC
#endif
#endif
#define CC_ACC 90714

#ifndef RLIB_COMPILER_ID
#if defined(__CMB__)
#define RLIB_COMPILER_ID CC_ALTIUM_MICROBLAZE
#endif
#endif
#define CC_ALTIUM_MICROBLAZE 90715

#ifndef RLIB_COMPILER_ID
#if defined(__CHC__)
#define RLIB_COMPILER_ID CC_ALTIUM_HARDWARE
#endif
#endif
#define CC_ALTIUM_HARDWARE 90716

#ifndef RLIB_COMPILER_ID
#if defined(__ACK__)
#define RLIB_COMPILER_ID CC_AMSTERDAM
#endif
#endif
#define CC_AMSTERDAM 90717

#ifndef RLIB_COMPILER_ID
#if defined(__CC_ARM)
#define RLIB_COMPILER_ID CC_ARMCC
#endif
#endif
#define CC_ARMCC 90718

#ifndef RLIB_COMPILER_ID
#if defined(AZTEC_C) || defined(__AZTEC_C__)
```

```
#define RLIB_COMPILER_ID CC_AZTEC
#endif
#endif
#define CC_AZTEC 90719

#ifndef RLIB_COMPILER_ID
#if defined(__BORLANDC__) || defined(__CODEGEARC__)
#define RLIB_COMPILER_ID CC_BORLAND
#endif
#endif
#define CC_BORLAND 90720

#ifndef RLIB_COMPILER_ID
#if defined(__CC65__)
#define RLIB_COMPILER_ID CC_CC65
#endif
#endif
#define CC_CC65 90721

#ifndef RLIB_COMPILER_ID
#if defined(__clang__)
#define RLIB_COMPILER_ID CC_CLANG
#endif
#endif
#define CC_CLANG 90722

#ifndef RLIB_COMPILER_ID
#if defined(__COMO__)
#define RLIB_COMPILER_ID CC_COMEAU
#endif
#endif
#define CC_COMEAU 90723

#ifndef RLIB_COMPILER_ID
#if defined(__DECC) || defined(__DECCXX)
#define RLIB_COMPILER_ID CC_COMPAQ
#endif
#endif
#define CC_COMPAQ 90724

#ifndef RLIB_COMPILER_ID
#if defined(__convexc__)
#define RLIB_COMPILER_ID CC_CONVEX
#endif
#endif
#define CC_CONVEX 90725

#ifndef RLIB_COMPILER_ID
#if defined(__COMPCERT__)
#define RLIB_COMPILER_ID CC_COMPCERT
#endif
#endif
#define CC_COMPCERT 90726

#ifndef RLIB_COMPILER_ID
#if defined(__COVERITY__)
#define RLIB_COMPILER_ID CC_COVERITY
#endif
#endif
#define CC_COVERITY 90727

#ifndef RLIB_COMPILER_ID
#if defined(_CRAYC)
#define RLIB_COMPILER_ID CC_CRAY
#endif
#endif
#define CC_CRAY 90728

#ifndef RLIB_COMPILER_ID
#if defined(__DCC__)
#define RLIB_COMPILER_ID CC_DIAB
#endif
#endif
#define CC_DIAB 90729

#ifndef RLIB_COMPILER_ID
#if defined(_DICE)
#define RLIB_COMPILER_ID CC_DICE
#endif
#endif
#define CC_DICE 90730

#ifndef RLIB_COMPILER_ID
#if defined(__DMC__)
#define RLIB_COMPILER_ID CC_DIGITAL_MARS
#endif
#endif
#define CC_DIGITAL_MARS 90731

#ifndef RLIB_COMPILER_ID
#if defined(__SYSC__)
#define RLIB_COMPILER_ID CC_DIGNUS
#endif
#endif
#define CC_DIGNUS 90732

#ifndef RLIB_COMPILER_ID
#if defined(__DJGPP__)
#define RLIB_COMPILER_ID CC_DJGPP
#endif
#endif
#define CC_DJGPP 90733

#ifndef RLIB_COMPILER_ID
#if defined(__ICC) || defined(__INTEL_COMPILER)
#define RLIB_COMPILER_ID CC_ICC
#endif
#endif
#define CC_ICC 90734

#ifndef RLIB_COMPILER_ID
#if defined(__EDG__)
#define RLIB_COMPILER_ID CC_EDG
#endif
#endif
#define CC_EDG 90735

#ifndef RLIB_COMPILER_ID
#if defined(__PATHCC__)
#define RLIB_COMPILER_ID CC_EKOPATH
#endif
#endif
#define CC_EKOPATH 90736

#ifndef RLIB_COMPILER_ID
#if defined(__FCC_VERSION)
#define RLIB_COMPILER_ID CC_FUJITSU
#endif
#endif
#define CC_FUJITSU 90737

#ifndef RLIB_COMPILER_ID
#if defined(__GNUC__)
#define RLIB_COMPILER_ID CC_GCC
#endif
#endif
#define CC_GCC 90738

#ifndef RLIB_COMPILER_ID
```

```
#if defined(__ghs__)
#define RLIB_COMPILER_ID CC_GREENHILL
#endif
#endif
#define CC_GREENHILL 90739

#ifndef RLIB_COMPILER_ID
#if defined(__HP_cc)
#define RLIB_COMPILER_ID CC_HPC
#endif
#endif
#define CC_HPC 90740

#ifndef RLIB_COMPILER_ID
#if defined(__HP_aCC)
#define RLIB_COMPILER_ID CC_HPACXX
#endif
#endif
#define CC_HPACXX 90741

#ifndef RLIB_COMPILER_ID
#if defined(__IAR_SYSTEMS_ICC__)
#define RLIB_COMPILER_ID CC_IARC
#endif
#endif
#define CC_IARC 90742

#ifndef RLIB_COMPILER_ID
#if defined(__IBMCPP__) || defined(__IBMC__)
#define RLIB_COMPILER_ID CC_IBMC
#endif
#endif
#define CC_IBMC 90743

#ifndef RLIB_COMPILER_ID
#if defined(__IMAGECRAFT__)
#define RLIB_COMPILER_ID CC_IMAGECRAFT
#endif
#endif
#define CC_IMAGECRAFT 90744

#ifndef RLIB_COMPILER_ID
#if defined(__KCC)
#define RLIB_COMPILER_ID CC_KAICXX
#endif
#endif
#define CC_KAICXX 90745

#ifndef RLIB_COMPILER_ID
#if defined(__CA__) || defined(__KEIL__)
#define RLIB_COMPILER_ID CC_KEIL_CARM
#endif
#endif
#define CC_KEIL_CARM 90746

#ifndef RLIB_COMPILER_ID
#if defined(__C166__)
#define RLIB_COMPILER_ID CC_KEIL_C166
#endif
#endif
#define CC_KEIL_C166 90747

#ifndef RLIB_COMPILER_ID
#if defined(__C51__) || defined(__CX51__)
#define RLIB_COMPILER_ID CC_KEIL_C51
#endif
#endif
#define CC_KEIL_C51 90748

#ifndef RLIB_COMPILER_ID
#if defined(__LCC__)
#define RLIB_COMPILER_ID CC_LCC
#endif
#endif
#define CC_LCC 90749

#ifndef RLIB_COMPILER_ID
#if defined(__llvm__)
#define RLIB_COMPILER_ID CC_LLVM
#endif
#endif
#define CC_LLVM 90750

#ifndef RLIB_COMPILER_ID
#if defined(__MWERKS__) || defined(__CWCC__)
#define RLIB_COMPILER_ID CC_METROWERKS
#endif
#endif
#define CC_METROWERKS 90751

#ifndef RLIB_COMPILER_ID
#if defined(_MSC_VER)
#define RLIB_COMPILER_ID CC_MSVC
#endif
#endif
#define CC_MSVC 90752

#ifndef RLIB_COMPILER_ID
#if defined(_MRI)
#define RLIB_COMPILER_ID CC_MICROTEC
#endif
#endif
#define CC_MICROTEC 90753

#ifndef RLIB_COMPILER_ID
#if defined(__NDPC__) || defined(__NDPX__)
#define RLIB_COMPILER_ID CC_MICROWAY
#endif
#endif
#define CC_MICROWAY 90754

#ifndef RLIB_COMPILER_ID
#if defined(__sgi) || defined(sgi)
#define RLIB_COMPILER_ID CC_MIPSPRO
#endif
#endif
#define CC_MIPSPRO 90755

#ifndef RLIB_COMPILER_ID
#if defined(MIRACLE)
#define RLIB_COMPILER_ID CC_MIRACLE
#endif
#endif
#define CC_MIRACLE 90756

#ifndef RLIB_COMPILER_ID
#if defined(__MRC__) || defined(MPW_C) || defined(MPW_CPLUS)
#define RLIB_COMPILER_ID CC_MPW
#endif
#endif
#define CC_MPW 90757

#ifndef RLIB_COMPILER_ID
#if defined(__CC_NORCROFT)
#define RLIB_COMPILER_ID CC_NORCROFT
#endif
#endif
#define CC_NORCROFT 90758
```

```
#ifndef RLIB_COMPILER_ID
#if defined(__NWCC__)
#define RLIB_COMPILER_ID CC_NWCC
#endif
#endif
#define CC_NWCC 90759

#ifndef RLIB_COMPILER_ID
#if defined(__OPEN64__) || defined(__OPENCC__)
#define RLIB_COMPILER_ID CC_OPEN64
#endif
#endif
#define CC_OPEN64 90760

#ifndef RLIB_COMPILER_ID
#if defined(ORA_PROC)
#define RLIB_COMPILER_ID CC_ORACLE_PROC
#endif
#endif
#define CC_ORACLE_PROC 90761

#ifndef RLIB_COMPILER_ID
#if defined(__SUNPRO_C) || defined(__SUNPRO_CC)
#define RLIB_COMPILER_ID CC_SOLARIS
#endif
#endif
#define CC_SOLARIS 90762

#ifndef RLIB_COMPILER_ID
#if defined(__PACIFIC__)
#define RLIB_COMPILER_ID CC_PACIFIC
#endif
#endif
#define CC_PACIFIC 90763

#ifndef RLIB_COMPILER_ID
#if defined(PACC_VER)
#define RLIB_COMPILER_ID CC_PLAM
#endif
#endif
#define CC_PLAM 90764

#ifndef RLIB_COMPILER_ID
#if defined(__POCC__)
#define RLIB_COMPILER_ID CC_PELLES
#endif
#endif
#define CC_PELLES 90765

#ifndef RLIB_COMPILER_ID
#if defined(__PGI)
#define RLIB_COMPILER_ID CC_PORTLAND
#endif
#endif
#define CC_PORTLAND 90766

#ifndef RLIB_COMPILER_ID
#if defined(__RENESAS__) || defined(__HITACHI__)
#define RLIB_COMPILER_ID CC_RENESAS
#endif
#endif
#define CC_RENESAS 90767

#ifndef RLIB_COMPILER_ID
#if defined(SASC) || defined(__SASC) || defined(__SASC__)
#define RLIB_COMPILER_ID CC_SASC
#endif
#endif
#define CC_SASC 90768

#ifndef RLIB_COMPILER_ID
#if defined(_SCO_DS)
#define RLIB_COMPILER_ID CC_SCO_OPENSERVER
#endif
#endif
#define CC_SCO_OPENSERVER 90769

#ifndef RLIB_COMPILER_ID
#if defined(SDCC)
#define RLIB_COMPILER_ID CC_SDCC
#endif
#endif
#define CC_SDCC 90770

#ifndef RLIB_COMPILER_ID
#if defined(__SNC__)
#define RLIB_COMPILER_ID CC_SN
#endif
#endif
#define CC_SN 90771

#ifndef RLIB_COMPILER_ID
#if defined(__VOSC__)
#define RLIB_COMPILER_ID CC_STRATUS_VOS
#endif
#endif
#define CC_STRATUS_VOS 90772

#ifndef RLIB_COMPILER_ID
#if defined(__SC__)
#define RLIB_COMPILER_ID CC_SYMANTEC
#endif
#endif
#define CC_SYMANTEC 90773

#ifndef RLIB_COMPILER_ID
#if defined(__TenDRA__)
#define RLIB_COMPILER_ID CC_TENDRA
#endif
#endif
#define CC_TENDRA 90774

#ifndef RLIB_COMPILER_ID
#if defined(__TI_COMPILER_VERSION__) || defined(_TMS320C6X)
#define RLIB_COMPILER_ID CC_TEXAS
#endif
#endif
#define CC_TEXAS 90775

#ifndef RLIB_COMPILER_ID
#if defined(THINKC3) || defined(THINKC4)
#define RLIB_COMPILER_ID CC_THINK
#endif
#endif
#define CC_THINK 90776

#ifndef RLIB_COMPILER_ID
#if defined(__TINYC__)
#define RLIB_COMPILER_ID CC_TINYC
#endif
#endif
#define CC_TINYC 90777

#ifndef RLIB_COMPILER_ID
#if defined(__TURBOC__)
#define RLIB_COMPILER_ID CC_TURBOC
#endif
#endif
#define CC_TURBOC 90778
```

```
#ifndef RLIB_COMPILER_ID
#if defined(__UCC)
#define RLIB_COMPILER_ID CC_UCC
#endif
#endif
#define CC_UCC 90779

#ifndef RLIB_COMPILER_ID
#if defined(__USLC__)
#define RLIB_COMPILER_ID CC_USLC
#endif
#endif
#define CC_USLC 90780

#ifndef RLIB_COMPILER_ID
#if defined(__VBCC__)
#define RLIB_COMPILER_ID CC_VBCC
#endif
#endif
#define CC_VBCC 90781

#ifndef RLIB_COMPILER_ID
#if defined(__WATCOMC__)
#define RLIB_COMPILER_ID CC_WATCOM
#endif
#endif
#define CC_WATCOM 90782

#ifndef RLIB_COMPILER_ID
#if defined(__ZTC__)
#define RLIB_COMPILER_ID CC_ZORTECH
#endif
#endif
#define CC_ZORTECH 90783

#ifndef RLIB_COMPILER_ID
#define RLIB_COMPILER_ID CC_UNKNOWN
#endif
#define CC_UNKNOWN 90784
/******* ./sys/time.hpp ****************/
#ifndef RLIB_TIME_HPP
#define RLIB_TIME_HPP_

#include <chrono>
#include <ctime>
#include <iomanip>
namespace rlib {
	static inline std::string get_current_time_str() noexcept {
		std::chrono::system_clock::time_point now = std::chrono::system_clock::now();
		std::time_t now_c = std::chrono::system_clock::to_time_t(now - std::chrono::hours(24));
		static char mbstr[128];
		if (std::strftime(mbstr, sizeof(mbstr), "%c", std::localtime(&now_c))) {
			return mbstr;
		}
		throw std::overflow_error("on get_current_time: mbstr buffer is too small.");
	}
}
#endif
/******* ./sys/unix_handy.hpp ****************/
#ifndef RLIB_UNIX_HANDY_HPP
#define RLIB_UNIX_HANDY_HPP_

#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netdb.h>
#include <rlib/scope_guard.hpp>
#include <rlib/string.hpp>

#include <rlib/sys/os.hpp>
#if RLIB_OS_ID == OS_WINDOWS
#error rlib/sys/unix_handy.hpp is not for Windows.
#endif

namespace rlib {
	namespace impl {
		using rlib::literals::operator""_format;
		static inline fd unix_quick_listen(const std::string &addr, uint16_t port) {
			addrinfo *psaddr;
			addrinfo hints{0};
			fd listenfd;

			hints.ai_family = AF_UNSPEC;
			hints.ai_socktype = SOCK_STREAM;
			hints.ai_flags = AI_PASSIVE;	/* For wildcard IP address */
			auto _ = getaddrinfo(addr.c_str(), std::to_string(port).c_str(), &hints, &psaddr);
			if (_ != 0) throw std::runtime_error("Failed to getaddrinfo. returnval={}, check `man getaddrinfo`'s return value."_format(_));

			bool success = false;
			for (addrinfo *rp = psaddr; rp != nullptr; rp = rp->ai_next) {
				listenfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
				if (listenfd == -1)
					continue;
				int reuse = 1;
				if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, (const char *) &reuse, sizeof(int)) < 0)
					throw std::runtime_error("setsockopt(SO_REUSEADDR) failed");
				if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEPORT, (const char *) &reuse, sizeof(int)) < 0)
					throw std::runtime_error("setsockopt(SO_REUSEPORT) failed");
				if (bind(listenfd, rp->ai_addr, rp->ai_addrlen) == 0) {
					success = true;
					break;
				}
				close(listenfd);
			}
			if (!success) throw std::runtime_error("Failed to bind {}:{}."_format(addr, port));

			if (-1 == ::listen(listenfd, 16)) throw std::runtime_error("listen failed.");

			rlib_defer([psaddr] { freeaddrinfo(psaddr); });
			return listenfd;
		}

		static inline fd unix_quick_connect(const std::string &addr, uint16_t port) {
			addrinfo *paddr;
			addrinfo hints{0};
			fd sockfd;

			hints.ai_family = AF_UNSPEC;
			hints.ai_socktype = SOCK_STREAM;
			auto _ = getaddrinfo(addr.c_str(), std::to_string(port).c_str(), &hints, &paddr);
			if (_ != 0)
				throw std::runtime_error("getaddrinfo failed. Check network connection to {}:{}; returnval={}, check `man getaddrinfo`'s return
value."_format(
					addr.c_str(), port, _));
			rlib_defer([paddr] { freeaddrinfo(paddr); });

			bool success = false;
			for (addrinfo *rp = paddr; rp != NULL; rp = rp->ai_next) {
				sockfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
				if (sockfd == -1)
					continue;
				int reuse = 1;
				if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char *) &reuse, sizeof(int)) < 0)
					throw std::runtime_error("setsockopt(SO_REUSEADDR) failed");
				if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEPORT, (const char *) &reuse, sizeof(int)) < 0)
					throw std::runtime_error("setsockopt(SO_REUSEPORT) failed");
				if (connect(sockfd, rp->ai_addr, rp->ai_addrlen) == 0) {
					success = true;
```

```
                        break; /* Success */
                    }
                    close(sockfd);
                }
                if (!success) throw std::runtime_error("Failed to connect to any of these addr.");

                return sockfd;
            }
        }
        using impl::unix_quick_connect;
        using impl::unix_quick_listen;
}


// Unfinished. I'm not sure if I must implement it.
#if 1+1 == 3
#include <fstream>
#include <iostream>
#if RLIB_COMPILER_ID == CC_GCC
#include <ext/stdio_filebuf.h>
namespace rlib {
        inline std::istream & fd_to_istream(fd handle) {
                if(handle == STDIN_FILENO) return std::cin;
                __gnu_cxx::stdio_filebuf<char> filebuf(handle, std::ios::in);
                return std::istream(&filebuf);
        }
        inline std::ostream & fd_to_ostream(fd handle) {
                if(handle == STDOUT_FILENO) return std::cout;
                if(handle == STDERR_FILENO) return std::cerr;
                __gnu_cxx::stdio_filebuf<char> filebuf(handle, std::ios::out);
                return std::ostream(&filebuf);
        }
        inline std::iostream & fd_to_iostream(fd handle) {
                __gnu_cxx::stdio_filebuf<char> filebuf(handle, std::ios::in | std::ios::out);
                return std::iostream(&filebuf);
        }
} // rlib
#elif RLIB_COMPILER_ID == CC_MSVC
namespace rlib {
        inline std::istream & fd_to_istream(fd handle) {
                if(handle == STDIN_FILENO) return std::cin;
                ifstream fs(::_fdopen(handle, "r"));
                return fs;
        }
        inline std::ostream & fd_to_ostream(fd handle) {
                if(handle == STDOUT_FILENO) return std::cout;
                if(handle == STDERR_FILENO) return std::cerr;
                ofstream fs(::_fdopen(handle, "w"));
                return fs;
        }
        inline std::iostream & fd_to_iostream(fd handle) {
                fstream fs(::_fdopen(handle, "rw"));
                return fs;
        }
} // rlib
#else
namespace rlib {
        constexpr inline std::istream & fd_to_istream(fd handle) {
                if(handle == STDIN_FILENO) return std::cin;
                throw std::invalid_argument("fd != 0 to istream is not implemented except gcc/msvc.");
        }
        constexpr inline std::ostream & fd_to_ostream(fd handle) {
                if(handle == STDOUT_FILENO) return std::cout;
                if(handle == STDERR_FILENO) return std::cerr;
                throw std::invalid_argument("fd != 1/2 to ostream is not implemented except gcc/msvc.");
        }
        constexpr inline std::iostream & fd_to_iostream(fd handle) {
                throw std::invalid_argument("fd to iostream is not implemented except gcc/msvc.");
        }

} // rlib
#endif // if compiler
#endif // if 1+1==3

#endif // header guarder
/******* ./sys/cc_list ****************/
_ACC_ ACC
__CMB__ ALTIUM_MICROBLAZE
__CHC__ ALTIUM_HARDWARE
__ACK__ AMSTERDAM
__CC_ARM ARMCC
AZTEC_C __AZTEC_C__ AZTEC
__BORLANDC__ __CODEGEARC__ BORLAND
__CC65__ CC65
__clang__ CLANG
__COMO__ COMEAU
__DECC __DECCXX COMPAQ
__convexc__ CONVEX
__COMPCERT__ COMPCERT
__COVERITY__ COVERITY
_CRAYC CRAY
_ DCC__ DIAB
_DICE DICE
_ DMC__ DIGITAL_MARS
__SYSC__ DIGNUS
__DJGPP__ DJGPP
__ICC__ INTEL_COMPILER ICC
# ICC must be placed before: EDG, GCC
__EDG__ EDG
__PATHCC__ EKOPATH
__FCC_VERSION FUJITSU
__GNUC__ GCC
__ghs__ GREENHILL
__HP_cc HPC
__HP_aCC HPACXX
__IAR_SYSTEMS_ICC__ IARC
__IBMCPP__ IBMC__ IBMC
__IMAGECRAFT__ IMAGECRAFT
__KCC KAICXX
__CA__ KEIL__ KEIL_CARM
__C166__ KEIL_C166
__C51__ __CX51__ KEIL_C51
__LCC__ LCC
__llvm__ LLVM
__MWERKS__ __CWCC__ METROWERKS
_MSC_VER MSVC
_MRI MICROTEC
_ NDPC__ NDPX__ MICROWAY
__sgi sgi MIPSPRO
MIRACLE MIRACLE
__MRC__ MPW_C MPW_CPLUS MPW
__CC_NORCROFT NORCROFT
__NWCC__ NWCC
__OPEN64__ __OPENCC__ OPEN64
ORA_PROC ORACLE_PROC
__SUNPRO_C__ SUNPRO_CC SOLARIS
__PACIFIC__ PACIFIC
_PACC_VER PLAM
_ POCC__ PELLES
__PGI PORTLAND
__RENESAS__ __HITACHI__ RENESAS
SASC__ SASC __SASC__ SASC
_SCO_DS SCO_OPENSERVER
SDCC SDCC
__SNC__ SN
__VOSC__ STRATUS_VOS
__SC__ SYMANTEC
__tenDRA__ TENDRA
__TI_COMPILER_VERSION__ _TMS320C6X TEXAS
```

```
THINKC3 THINKC4 THINK
__TINYC__ __TINYC__
__TURBOC__ __TURBOC__
__UCC__ __UCC__
__USLC__ __USLC__
__VBCC__ __VBCC__
__WATCOMC__ __WATCOM__
__ZTC__ __ZORTECH__
/******* ./sys/sio.hpp ****************/
#ifndef R_SIO_HPP
#define R_SIO_HPP

#include <rlib/sys/os.hpp>

#if RLIB_OS_ID == OS_WINDOWS
#include <winsock2.h>
#include <windows.h>
#include <ws2tcpip.h>
#else
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <fcntl.h>
#include <arpa/inet.h>
#endif

// Include winsock2.h before windows.h
#include <cerrno>
#include <cstdlib>
#include <unistd.h>
#include <string>
#include <stdexcept>

#include <rlib/sys/fd.hpp>
#include <rlib/sys/os.hpp>
#include <rlib/string.hpp>
#include <rlib/scope_guard.hpp>


namespace rlib {
	// Both POSIX and Win32
	using rlib::literals::operator ""_format;
	static inline sockfd_t quick_accept(sockfd_t sock) {
		auto res = accept(sock, NULL, NULL);
		if(res == -1)
			throw std::runtime_error("accept failed. errno = {}"_format(strerror(errno)));
		return res;
	}
	static inline std::pair<std::string, uint16_t> get_peer_name(sockfd_t sock) {
		struct sockaddr_storage addr;
		socklen_t addrlen = sizeof(addr);
		auto res = getpeername(sock, (struct sockaddr *)&addr, &addrlen);
		if(res == -1)
			throw std::runtime_error("getpeername failed. errno = {}"_format(strerror(errno)));
		std::string ipstr;
		uint16_t port = 0;
		if(addr.ss_family == AF_INET) {
			struct sockaddr_in *s = (struct sockaddr_in *)&addr;
			char str[INET_ADDRSTRLEN];
			auto res = inet_ntop(AF_INET, &s->sin_addr, str, INET_ADDRSTRLEN);
			if(res == NULL)
				throw std::runtime_error("inet_ntop failed. errno = {}"_format(strerror(errno)));
			port = ntohs(s->sin_port);
			ipstr = str;
		}
		else {
			struct sockaddr_in6 *s = (struct sockaddr_in6 *)&addr;
			char str[INET6_ADDRSTRLEN];
			auto res = inet_ntop(AF_INET6, &s->sin6_addr, str, INET6_ADDRSTRLEN);
			if(res == NULL)
				throw std::runtime_error("inet_ntop failed. errno = {}"_format(strerror(errno)));
			port = ntohs(s->sin6_port);
			ipstr = std::string() + '[' + str + ']';
		}
		return {ipstr, port};


	}

#if RLIB_OS_ID != OS_WINDOWS
	namespace impl {
		inline void MakeNonBlocking(fd_t fd) {
			int flags, s;

			flags = fcntl (fd, F_GETFL, 0);
			if (flags == -1) {
				perror ("fcntl");
				exit(-1);
			}

			flags |= O_NONBLOCK;
			s = fcntl (fd, F_SETFL, flags);
			if (s == -1) {
				perror ("fcntl");
				exit(-1);
			}
		}
	}
#endif

#if RLIB_OS_ID == OS_WINDOWS
	template <bool doNotWSAStartup = false>
	static inline sockfd_t quick_listen(const std::string &addr, uint16_t port) {
		WSADATA wsaData;
		sockfd_t listenfd = INVALID_SOCKET;
		if(!doNotWSAStartup) {
			int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
			if (iResult != 0) throw std::runtime_error("WSAStartup failed with error: {}\n"_format(iResult));
		}

		addrinfo *psaddr;
		addrinfo hints { 0 };
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		hints.ai_flags = AI_PASSIVE;	/* For wildcard IP address */
		hints.ai_protocol = IPPROTO_TCP;
		auto _ = getaddrinfo(addr.c_str(), std::to_string(port).c_str(), &hints, &psaddr);
		if(_ != 0) {
			WSACleanup();
			throw std::runtime_error("Failed to getaddrinfo. returnval={}, check `man getaddrinfo`'s return value."_format(_));
		}

		bool success = false;
		for (addrinfo *rp = psaddr; rp != NULL; rp = rp->ai_next) {
			listenfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
			if (listenfd == INVALID_SOCKET)
				continue;
			int reuse = 1;
			if(setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&reuse, sizeof(int)) < 0) throw
std::runtime_error("setsockopt(SO_REUSEADDR) failed");
			if (bind(listenfd, rp->ai_addr, rp->ai_addrlen) != SOCKET_ERROR) {
				success = true;
				break; /* Success */
			}
			closesocket(listenfd);
		}
		if(!success) throw std::runtime_error("Failed to bind to any of these addr.");

		if(SOCKET_ERROR == ::listen(listenfd, 16)) throw std::runtime_error("listen failed. {}"_format(strerror(errno)));
```

```
        freeaddrinfo(psaddr);
        return listenfd;
    }

    template <bool doNotWSAStartup = false>
    static inline sockfd_t quick_connect(const std::string &addr, uint16_t port) {
        WSADATA wsaData;
        sockfd_t sockfd = INVALID_SOCKET;
        if(!doNotWSAStartup) {
            int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);
            if (iResult != 0) throw std::runtime_error("WSAStartup failed with error: {}\n"_format(iResult));
        }

        addrinfo *paddr;
        addrinfo hints { 0 };

        hints.ai_family = AF_UNSPEC;
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_protocol = IPPROTO_TCP;
        auto _ = getaddrinfo(addr.c_str(), std::to_string(port).c_str(), &hints, &paddr);
        if(_ != 0) {
            WSACleanup();
            throw std::runtime_error("getaddrinfo failed. Check network connection to {}:{}; returnval={}, check `man getaddrinfo`'s return
value."_format(addr.c_str(), port, _));
        }
        rlib_defer([p=paddr]{WSACleanup();freeaddrinfo(p);});

        bool success = false;
        for (addrinfo *rp = paddr; rp != NULL; rp = rp->ai_next) {
            sockfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
            if (sockfd == INVALID_SOCKET)
                continue;
            int reuse = 1;
            if(setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&reuse, sizeof(int)) < 0) throw
std::runtime_error("setsockopt(SO_REUSEADDR) failed");
            if (connect(sockfd, rp->ai_addr, rp->ai_addrlen) != SOCKET_ERROR) {
                success = true;
                break; /* Success */
            }
            closesocket(sockfd);
        }
        if(!success) throw std::runtime_error("Failed to connect to any of these addr.");

        freeaddrinfo(paddr);
        return sockfd;
    }

#else
    // POSIX version
    static inline fd_t quick_listen(const std::string &addr, uint16_t port) {
        addrinfo *psaddr;
        addrinfo hints{0};
        fd_t listenfd;

        hints.ai_family = AF_UNSPEC;
        hints.ai_socktype = SOCK_STREAM;
        hints.ai_flags = AI_PASSIVE;      /* For wildcard IP address */
        auto _ = getaddrinfo(addr.c_str(), std::to_string(port).c_str(), &hints, &psaddr);
        if (_ != 0) throw std::runtime_error("Failed to getaddrinfo. returnval={}, check `man getaddrinfo`'s return value."_format(_));

        bool success = false;
        for (addrinfo *rp = psaddr; rp != nullptr; rp = rp->ai_next) {
            listenfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
            if (listenfd == -1)
                continue;
            int reuse = 1;
            if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, (const char *) &reuse, sizeof(int)) < 0)
                throw std::runtime_error("setsockopt(SO_REUSEADDR) failed");
            if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEPORT, (const char *) &reuse, sizeof(int)) < 0)
                throw std::runtime_error("setsockopt(SO_REUSEPORT) failed");
            if (bind(listenfd, rp->ai_addr, rp->ai_addrlen) == 0) {
                success = true;
                break;
            }
            close(listenfd);
        }
        if (!success) throw std::runtime_error("Failed to bind {}:{}."_format(addr, port));

        if (-1 == ::listen(listenfd, 16)) throw std::runtime_error("listen failed. {}"_format(strerror(errno)));

        rlib_defer([psaddr] { freeaddrinfo(psaddr); });
        return listenfd;
    }

    static inline fd_t quick_connect(const std::string &addr, uint16_t port) {
        addrinfo *paddr;
        addrinfo hints{0};
        fd_t sockfd;

        hints.ai_family = AF_UNSPEC;
        hints.ai_socktype = SOCK_STREAM;
        auto _ = getaddrinfo(addr.c_str(), std::to_string(port).c_str(), &hints, &paddr);
        if (_ != 0)
            throw std::runtime_error("getaddrinfo failed. Check network connection to {}:{}; returnval={}, check `man getaddrinfo`'s return
value."_format(
                    addr.c_str(), port, _));
        rlib_defer([paddr] { freeaddrinfo(paddr); });

        bool success = false;
        for (addrinfo *rp = paddr; rp != NULL; rp = rp->ai_next) {
            sockfd = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);
            if (sockfd == -1)
                continue;
            int reuse = 1;
            if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char *) &reuse, sizeof(int)) < 0)
                throw std::runtime_error("setsockopt(SO_REUSEADDR) failed");
            if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEPORT, (const char *) &reuse, sizeof(int)) < 0)
                throw std::runtime_error("setsockopt(SO_REUSEPORT) failed");
            if (connect(sockfd, rp->ai_addr, rp->ai_addrlen) == 0) {
                success = true;
                break; /* Success */
            }
            close(sockfd);
        }
        if (!success) throw std::runtime_error("Failed to connect to any of these addr.");

        return sockfd;
    }
#endif

#if RLIB_OS_ID != OS_WINDOWS
    // POSIX-only fdIO
    class fdIO
    {
    public:
        static ssize_t readn(fd_t fd, void *vptr, size_t n) noexcept //Return -1 on error, read bytes on success, blocks until nbytes done.
        {
            size_t   nleft;
            ssize_t nread;
            char     *ptr;

            ptr = (char *)vptr;
            nleft = n;
            while (nleft > 0) {
                if ( (nread = read(fd, ptr, nleft)) < 0) {
                    if (errno == EINTR)
                        nread = 0;          /* and call read() again */
                    else
```

```
                        return (-1);
                    } else if (nread == 0)
                        return (-1);                    /* EOF */

                    nleft -= nread;
                    ptr += nread;
                }
                return (n);             /* return success */
            }
            static ssize_t writen(fd_t fd, const void *vptr, size_t n) noexcept //Return -1 on error, read bytes on success, blocks until nbytes done.
            {
                size_t nleft;
                ssize_t nwritten;
                const char *ptr;

                ptr = (const char *)vptr;
                nleft = n;
                while (nleft > 0) {
                    if ( (nwritten = write(fd, ptr, nleft)) <= 0) {
                        if (nwritten < 0 && errno == EINTR)
                            nwritten = 0;     /* and call write() again */
                        else
                            return (-1);      /* error */
                     }

                     nleft -= nwritten;
                     ptr += nwritten;
                }
                return (n);
            }
            static ssize_t readall(fd_t fd, void **pvptr, size_t initSize) noexcept //Return -1 on error, read bytes on success. pvptr must be a malloc/calloced
buffer, I'll malloc one if *pvptr is NULL.
            {
                size_t current = initSize ? initSize : 1024;
                void *vptr = *pvptr;
                if(vptr == NULL)
                    vptr = malloc(current);
                void *currvptr = vptr;

                {
                    ssize_t ret = read(fd, currvptr, current / 2);
                    if(ret == -1) return -1;
                    if(ret < current / 2)
                    {
                        *pvptr = vptr;
                        return ret;
                    }
                    currvptr = (char *)vptr + current / 2;
                }

                while(true)
                {
                    ssize_t ret = read(fd, currvptr, current / 2);
                    if(ret == -1) return -1;
                    if(ret < current)
                    {
                        *pvptr = vptr;
                        return ret + current / 2;
                    }

                    current *= 2;
                    void *vptrBackup = vptr;
                    if((vptr = realloc(vptr, current)) == NULL) {
                        free(vptrBackup);
                        errno = EMSGSIZE;
                        return -1;
                    }
                    currvptr = (char *)vptr + current / 2;
                }
            }
            static void readn_ex(fd_t fd, void *vptr, size_t n) //never return error.
            {
                auto ret = readn(fd, vptr, n);
                if(ret == -1) throw std::runtime_error("readn failed. errno = {}"_format(strerror(errno)));
            }
            static void writen_ex(fd_t fd, const void *vptr, size_t n)
            {
                auto ret = writen(fd, vptr, n);
                if(ret == -1) throw std::runtime_error("writen failed. errno = {}"_format(strerror(errno)));
            }
            static ssize_t readall_ex(fd_t fd, void **pvptr, size_t initSize) //never return -1
            {
                auto ret = readall(fd, pvptr, initSize);
                if(ret == -1) throw std::runtime_error("readall failed. errno = {}"_format(strerror(errno)));
                return ret;
            }
            static std::string quick_readn(fd_t fd, size_t n) {
                std::string res(n, '0');
                readn_ex(fd, (void *)res.data(), n);
                return res;
            }
            static std::string quick_readall(fd_t fd) {
                void *ptr;
                auto size = readall_ex(fd, &ptr, 0);
                return std::string((char *)ptr, size);
            }
            static void quick_write(fd_t fd, const std::string &data) {
                writen_ex(fd, data.data(), data.size());
            }
        };
#endif

        // Win32 sockIO
        class sockIO
        {
#if RLIB_OS_ID == OS_WINDOWS
        private:
            static int WSASafeGetLastError()
            {
                int i;
                WSASetLastError(i = WSAGetLastError());
                return i;
            }
        public:
            static ssize_t recvn(sockfd_t fd, void *vptr, size_t n, int flags) noexcept //Return -1 on error, read bytes on success, blocks until nbytes done.
            {
                size_t   nleft;
                ssize_t nread;
                char     *ptr;

                ptr = (char *)vptr;
                nleft = n;
                while (nleft > 0) {
                    if ( (nread = recv(fd, ptr, nleft, flags)) == SOCKET_ERROR) {
                        if (WSASafeGetLastError() == WSAEINTR)
                            nread = 0;          /* and call read() again */
                        else
                            return (-1);
                    } else if (nread == 0)
                        return (-1);                    /* EOF */

                    nleft -= nread;
                    ptr += nread;
                }
                return (n);             /* return >= 0 */
            }
            static ssize_t sendn(sockfd_t fd, const void *vptr, size_t n, int flags) noexcept //Return -1 on error, read bytes on success, blocks until nbytes
done.
            {
```

```
		size_t nleft;
		ssize_t nwritten;
		const char *ptr;

		ptr = (const char *)vptr;
		nleft = n;
		while (nleft > 0) {
			if ( (nwritten = send(fd, ptr, nleft, flags)) <= 0) {
				if (nwritten == SOCKET_ERROR && WSASafeGetLastError() == WSAEINTR)
					nwritten = 0;	/* and call write() again */
				else
					return (-1);	/* error */
			 }

			 nleft -= nwritten;
			 ptr += nwritten;
		}
		return (n);
	}
	static ssize_t recvall(sockfd_t fd, void **pvptr, size_t initSize, int flags) noexcept //Return -1 on error, read bytes on success. pvptr must be a
malloc/calloced buffer, I'll malloc one if *pvptr is NULL.
	{
		size_t current = initSize ? initSize : 1024;
		void *vptr = *pvptr;
		if(vptr == NULL)
			vptr = malloc(current);
		void *currvptr = vptr;

		{
		_retry_1:
			ssize_t ret = recv(fd, (char *)currvptr, current / 2, flags);
			if(ret == SOCKET_ERROR) {
				if(WSASafeGetLastError() == WSAEINTR)
					goto _retry_1;
				return SOCKET_ERROR;
			}
			if(ret < current / 2)
			{
				*pvptr = vptr;
				return ret;
			}
			currvptr = (char *)vptr + current / 2;
		}

		while(true)
		{
			ssize_t ret = recv(fd, (char *)currvptr, current / 2, flags);
			if(ret == SOCKET_ERROR) {
				if(WSASafeGetLastError() == WSAEINTR)
					continue; //retry
				return SOCKET_ERROR;
			}
			if(ret < current)
			{
				*pvptr = vptr;
				return ret + current / 2;
			}

			current *= 2;
			void *vptrBackup = vptr;
			if((vptr = realloc(vptr, current)) == NULL) {
				free(vptrBackup);
				WSASetLastError(WSAEMSGSIZE);
				return SOCKET_ERROR;
			}
			currvptr = (char *)vptr + current / 2;
		}
	}
#else
	// POSIX sockIO
	public:
	static ssize_t recvn(sockfd_t fd, void *vptr, size_t n, int flags) noexcept //Return -1 on error, read bytes on success, blocks until nbytes done.
	{
		size_t	nleft;
		ssize_t nread;
		char	*ptr;

		ptr = (char *)vptr;
		nleft = n;
		while (nleft > 0) {
			if ( (nread = recv(fd, ptr, nleft, flags)) < 0) {
				if (errno == EINTR)
					nread = 0;		/* and call read() again */
				else
					return (-1);
			} else if (nread == 0)
				return -1;			/* EOF */

			nleft -= nread;
			ptr += nread;
		}
		return (n);		/* return success */
	}
	static ssize_t sendn(sockfd_t fd, const void *vptr, size_t n, int flags) noexcept //Return -1 on error, read bytes on success, blocks until nbytes
done.
	{
		size_t nleft;
		ssize_t nwritten;
		const char *ptr;

		ptr = (const char *)vptr;
		nleft = n;
		while (nleft > 0) {
			if ( (nwritten = send(fd, ptr, nleft, flags)) <= 0) {
				if (nwritten < 0 && errno == EINTR)
					nwritten = 0;	/* and call write() again */
				else
					return (-1);	/* error */
			 }

			 nleft -= nwritten;
			 ptr += nwritten;
		}
		return (n);
	}
	static ssize_t recvall(sockfd_t fd, void **pvptr, size_t initSize, int flags) noexcept //Return -1 on error, read bytes on success. pvptr must be a
malloc/calloced buffer, I'll malloc one if *pvptr is NULL.
	{
		size_t current = initSize ? initSize : 1024;
		void *vptr = *pvptr;
		if(vptr == NULL)
			vptr = malloc(current);
		void *currvptr = vptr;

		{
			ssize_t ret = recv(fd, currvptr, current / 2, flags);
			if(ret == -1) return -1;
			if(ret < current / 2)
			{
				*pvptr = vptr;
				return ret;
			}
			currvptr = (char *)vptr + current / 2;
		}

		while(true)
		{
			ssize_t ret = recv(fd, currvptr, current / 2, flags);
			if(ret == -1) return -1;
```

```
                if(ret < current)
                {
                    *pvptr = vptr;
                    return ret + current / 2;
                }

                current *= 2;
                void *vptrBackup = vptr;
                if((vptr = realloc(vptr, current)) == NULL) {
                    free(vptrBackup);
                    errno = EMSGSIZE;
                    return -1;
                }
                currvptr = (char *)vptr + current / 2;
            }
        }
#endif

#ifndef MSG_NOSIGNAL
    private:
        static constexpr int MSG_NOSIGNAL = 0;
#endif
        // both POSIX and Win32
    public:
        static void recvn_ex(sockfd_t fd, void *vptr, size_t n, int flags) //return read bytes.
        {
            auto ret = recvn(fd, vptr, n, flags);
            if(ret == -1) throw std::runtime_error("recvn failed. {}"_format(strerror(errno)));
        }
        static void sendn_ex(sockfd_t fd, const void *vptr, size_t n, int flags)
        {
            auto ret = sendn(fd, vptr, n, flags);
            if(ret == -1) throw std::runtime_error("sendn failed. {}"_format(strerror(errno)));
        }
        static ssize_t recvall_ex(sockfd_t fd, void **pvptr, size_t initSize, int flags) //never return -1
        {
            auto ret = recvall(fd, pvptr, initSize, flags);
            if(ret == -1) throw std::runtime_error("recvall failed. {}"_format(strerror(errno)));
            return ret;
        }

        static std::string quick_recvn(sockfd_t fd, size_t n) {
            std::string res(n, '0');
            recvn_ex(fd, (void *)res.data(), n, MSG_NOSIGNAL);
            return res;
        }
        static std::string quick_recvall(sockfd_t fd) {
            void *ptr = NULL;
            auto size = recvall_ex(fd, &ptr, 0, MSG_NOSIGNAL);
            auto result = std::string((char *)ptr, size);
            free(ptr);
            return result;
        }
        static void quick_send(sockfd_t fd, const std::string &data) {
            sendn_ex(fd, data.data(), data.size(), MSG_NOSIGNAL);
        }

        // practical message with head
    private:
#pragma pack(push, 1)
        struct packed_msg_head {
            uint32_t magic = 0x19980427;
            uint64_t len;
        };
#pragma pack(pop)
        static_assert(sizeof(packed_msg_head) == sizeof(uint32_t) + sizeof(uint64_t), "Compiler doesn't compile the struct `packed_msg_head` as
packed.");
    public:
        static std::string recv_msg(sockfd_t fd) {
            packed_msg_head head;
            recvn_ex(fd, &head, sizeof(head), MSG_NOSIGNAL);
            if(head.magic != 0x19980427)
                throw std::runtime_error("Invalid magic received.");
            if(head.len > 1024ull*1024*1024*2)
                throw std::runtime_error("Message len is greater than 2GiB. Refuse to alloc space.");
            std::string dat(head.len, '\0');
            recvn_ex(fd, (void *)dat.data(), head.len, MSG_NOSIGNAL);
            return dat;
        }
        static void send_msg(sockfd_t fd, const std::string &dat) {
            packed_msg_head head;
            head.len = dat.size();
            sendn_ex(fd, &head, sizeof(head), MSG_NOSIGNAL);
            sendn_ex(fd, dat.data(), head.len, MSG_NOSIGNAL);
        }

    };


} // namespace rlib


#endif
/******* ./class_decorator.hpp ****************/
#ifndef RLIB_CLASS_DECO_HPP
#define RLIB_CLASS_DECO_HPP_

#include <rlib/require/cxx11>

namespace rlib {
    namespace _noncp_ {
        class noncopyable
        {
        public:
            noncopyable() = default;
            ~noncopyable() = default;
            noncopyable(const noncopyable &) = delete;
            noncopyable &operator=(const noncopyable &) = delete;
        };
    }
    typedef _noncp_::noncopyable noncopyable;
}

namespace rlib {
    namespace _nonmv_ {
        class nonmovable : private noncopyable
        {
        public:
            nonmovable() = default;
            ~nonmovable() = default;
            nonmovable(const nonmovable &&) = delete;
            nonmovable &operator=(const nonmovable &&) = delete;
        };
    }
    typedef _nonmv_::nonmovable nonmovable;
}

namespace rlib {
    namespace _nonconstructible_ {
        class nonconstructible : private rlib::nonmovable
        {
        public:
            nonconstructible() = delete;
            ~nonconstructible() = delete;
        };
    }
    typedef _nonconstructible_::nonconstructible nonconstructible;
    typedef nonconstructible static_class;
}
```

```
#endif/******* ./opt.hpp ****************/
/*
This opt_parser works well for correct cmd args,
but not guaranteed to works well in all condition
(for example, some ill formed argument).

It's possible to read wrong information rather than
raise an exception on some rare ill formed arguments.
*/
#ifndef R_OPT_HPP
#define R_OPT_HPP

#include <rlib/require/cxx14>
#include <rlib/class_decorator.hpp>
#include <rlib/string.hpp>
#include <rlib/scope_guard.hpp>

#include <string>
#include <vector>
#include <algorithm>
#include <stdexcept>

namespace rlib {
	class opt_parser : private noncopyable
	{
	public:
		opt_parser() = delete;
		opt_parser(size_t arglen, char **argv) {
			if(argv[0] == nullptr)
				throw std::runtime_error("Invalid argv passed to rlib::opt_parser. argv[0] is nullptr.");
			arg0 = argv[0];
			for(size_t cter = 1; cter < arglen; ++cter)
				args.push_back(std::string(argv[cter]));
		}

		rlib::string getSubCommand() {
			if(args.empty())
				throw std::runtime_error("No sub-command available.");
			auto cmd = std::move(args[0]);
			args.erase(args.begin());
			return std::move(cmd);
		}

		rlib::string getSubCommand(const std::string &def) {
			if(args.empty())
				return def;
			auto cmd = std::move(args[0]);
			args.erase(args.begin());
			return std::move(cmd);
		}

		rlib::string getSelf() {
			return arg0;
		}

		rlib::string getValueArg(const std::string &argName, bool required = true, const std::string &def = std::string())
		{ //If required argument not exist, I'll throw. Else, return "" if arg is not read.
			using rlib::literals::operator ""_format;
			bool useEqualSym = false;
			auto pos = std::find_if(args.cbegin(), args.cend(), [&](auto &ele)->bool{
				if(ele == argName) return true;
				if(ele.size() > argName.size() && ele.substr(0, argName.size()+1) == argName + "=") {
					useEqualSym = true;
					return true;
				}
				return false;
			});
			if(required && pos == args.cend())
				throw std::invalid_argument("Required argument '{}' not provided."_format(argName));
			if(pos == args.cend())
				return std::move(def);
			rlib_defer([&, pos]{if(!useEqualSym) args.erase(pos+1); args.erase(pos);});
			if(useEqualSym)
				return pos->substr(argName.size() + 1);
			else
			{
				if(++pos == args.cend())
					throw std::invalid_argument("Argument '{}' must provide value."_format(argName));
				return *pos;
			}
		}

		rlib::string getValueArg(const std::string &longName, const char *shortName)
		{ //getValueArg("--long", "-l") may be converted to getValueArg("--long", true).
			return getValueArg(longName, shortName, true);
		}

		bool getBoolArg(const std::string &argName)
		{ //Return if it's defined.
			auto pos = std::find(args.cbegin(), args.cend(), argName);
			if(pos == args.cend()) return false;
			args.erase(pos);
			return true;
		}

		rlib::string getValueArg(const std::string &longName, const std::string &shortName, bool required = true, const std::string &def = std::string())
		{
			using rlib::literals::operator ""_format;
			std::string valueL = getValueArg(longName, false);
			std::string valueS = getValueArg(shortName, false);

			const std::string &value = valueL.empty() ? valueS : valueL;
			if(value.empty()) {
				if(required)
					throw std::invalid_argument("Required argument '{}/{}' not provided."_format(longName, shortName));
				else
					return def;
			}
			return std::move(value);
		}

		bool getBoolArg(const std::string &longName, const std::string &shortName)
		{
			return getBoolArg(longName) || getBoolArg(shortName);
		}

		bool allArgDone() const
		{
			return args.empty();
		}
	private:
		std::vector<std::string> args;
		std::string arg0;
	};
}

#endif
/******* ./meta.hpp ****************/
#ifndef RLIB_MEelement_typeA_HPP
#define RLIB_MEelement_typeA_HPP_

#include <rlib/sys/os.hpp>
#include <cstddef> // size_t
#include <tuple>

namespace rlib {
	#if RLIB_CXX_STD >= 2017
	namespace impl {
```

```
        template <auto first_ele, auto... _>
        struct array_first_ele_type_impl { using type = decltype(first_ele); };
    }
    template <auto... arr>
    struct meta_array {
        using this_type = typename ::rlib::meta_array<arr ...>;
        using element_type = typename ::rlib::impl::array_first_ele_type_impl<arr...>::type;

        template <size_t index>
        struct at_last {
            static constexpr auto value() noexcept {
                return at_last_impl<index, arr ...>::value();
            }
            constexpr operator element_type() {
                return at_last<index>::value();
            }
        };

        template <size_t index>
        struct at {
            static constexpr auto value() {
                return at_last<sizeof...(arr) - index - 1>::value();
            }
            constexpr operator element_type() {
                return at<index>::value();
            }
        };

        static constexpr auto to_tuple() {
            return std::make_tuple(arr ...);
        }

    private:
        template <size_t index, auto first_ele, auto... _arr>
        struct at_last_impl {
            static constexpr auto value() {
                if constexpr(sizeof...(_arr) == index)
                    return first_ele;
                else
                    return at_last_impl<index, _arr ...>::value();
            }
        };
    };
    #endif

    template <size_t... forwardedArgs> struct argForwarder {};
}

#endif
/******* ./pool.hpp ****************/
#ifndef _RLIB_OBJ_POOL_HPP
#define _RLIB_OBJ_POOL_HPP 1

#include <rlib/impl/traceable_list.hpp>
#include <rlib/class_decorator.hpp>
#include <utility>
#include <tuple>
#include <functional>
#include <algorithm>

#ifdef RLIB_SWITCH_USE_MINGW_THREAD_FIX
#include <mingw.mutex.h>
#include <mingw.thread.h>
#include <mingw.condition_variable.h>
#else
#include <thread>
#include <mutex>
#include <condition_variable>
#endif

namespace rlib {
    /*
     * Multi-threaded object_pool. It will block current thread and wait if
     *       borrow_one() starves, until some other threads release their obj.
     */
    template<typename obj_t, typename... _bound_construct_args_t>
    class fixed_object_pool : rlib::nonmovable {
    protected:
        using element_t = obj_t;
        using buffer_t = impl::traceable_list<obj_t, bool>;
        using this_type = fixed_object_pool<obj_t, _bound_construct_args_t ...>;
    public:
        explicit fixed_object_pool(size_t max_size, _bound_construct_args_t ... _args)
                : max_size(max_size), _bound_args(std::forward<_bound_construct_args_t>(_args) ...)
        {}

        void fill_full() {
            for (size_t cter = 0; cter < max_size; ++cter) {
                new_obj_to_buffer();
                free_list.push_back(&*--buffer.end());
            }
        }

        // `new` an object. Return nullptr if pool is full.
        obj_t *try_borrow_one() {
            std::lock_guard<std::mutex> _l(buffer_mutex);
            return do_try_borrow_one();
        }
        obj_t *borrow_one() {
            auto result = try_borrow_one();
            if(result)
                return result;
            // Not available. Wait for release_one.
            std::unique_lock<std::mutex> lk(buffer_mutex);

            borrow_cv.wait(lk, [this]{return this->new_obj_ready;});

            result = do_try_borrow_one();
            lk.unlock();
            if(!result)
                throw std::logic_error("unknown par error.");
            return result;
        }
        void release_one(obj_t *which) {
            {
                std::lock_guard<std::mutex> _l(buffer_mutex);
                free_list.push_front(which);
                typename buffer_t::iterator elem_iter(which);
                elem_iter.get_extra_info() = true; // mark as free.
                new_obj_ready = true;
            } // lock released.
            borrow_cv.notify_one();
        }

        void reconstruct_one(obj_t *which) {
            reconstruct_impl(which, std::make_index_sequence<sizeof...(_bound_construct_args_t)>());
        }

    protected:
        buffer_t buffer; // list<obj_t obj, bool is_free>
    private:
        std::tuple<_bound_construct_args_t ...> _bound_args;

        size_t max_size;
        std::list<obj_t *> free_list;
        std::mutex buffer_mutex;
        std::condition_variable borrow_cv;
        volatile bool new_obj_ready = false;
```

```
		// try_borrow_one without lock.
		obj_t *do_try_borrow_one() {
			// Optimize here if is performance bottleneck (lockless list... etc...)
			borrow_again:
			if (free_list.size() > 0) {
				// Some object is free. Just return one.
				obj_t *result = *free_list.begin();
				free_list.pop_front();

				typename buffer_t::iterator elem_iter(result);
				elem_iter.get_extra_info() = false; // mark as busy.
				new_obj_ready = false;
				return result;
			}
			if (buffer.size() < max_size) {
				new_obj_to_buffer();
				free_list.push_back(&*--buffer.end());
				goto borrow_again;
			}
			return nullptr;
		}

		// fake emplace_back
		template<size_t ... index_seq>
		inline void new_obj_to_buffer_impl(std::index_sequence<index_seq ...>) {
			buffer.emplace_one(buffer.end(), true, std::get<index_seq>(_bound_args) ...);
		}
		template<size_t ... index_seq>
		inline void reconstruct_impl(obj_t *which, std::index_sequence<index_seq ...>) {
			which->~obj_t();
			new(which) obj_t(std::get<index_seq>(_bound_args) ...);
		}

		inline void new_obj_to_buffer() {
			new_obj_to_buffer_impl(std::make_index_sequence<sizeof...(_bound_construct_args_t)>());
		}
	};
}

#endif
/******* ./libr.cc ****************/
#include <rlib/log.hpp> //log_level_t
#include <rlib/stream.hpp>
#include <sstream>
namespace rlib {
	namespace impl {
//#if RLIB_CXX_STD < 2017
		bool enable_endl_flush = true;
		int max_predefined_log_level = (int)log_level_t::DEBUG;
		NullStreamBuf null_streambuf;
//#endif
//#ifndef RLIB_MINGW_DISABLE_TLS
//#if RLIB_CXX_STD < 2017
		thread_local std::stringstream _format_string_helper_ss;
		thread_local std::stringstream to_string_by_sstream_ss;
//#endif
//#endif
	}
//#if RLIB_CXX_STD < 2017
	std::ostream null_stream(&impl::null_streambuf);
//#endif
}
/******* ./stdio.hpp ****************/
/*
 *
 * stdio wrapper for modern c++: python like print/println/printf/printfln
 *                               print_iter println_iter
 * Recolic Keghart <root@recolic.net>
 * MIT License
 *
 */

#ifndef R_STDIO_HPP
#define R_STDIO_HPP

#include <rlib/require/cxx11> // Use fold expression if cxx17 is available.
#include <rlib/sys/os.hpp> // Enable inline variable if cxx17 is available.
#include <string>
#include <iostream>
#include <rlib/string.hpp> // format_string

#if RLIB_OS_ID == OS_WINDOWS
#define RLIB_IMPL_ENDLINE "\r\n"
#elif RLIB_OS_ID == OS_MACOS
#define RLIB_IMPL_ENDLINE "\r"
#else
#define RLIB_IMPL_ENDLINE "\n"
#endif

namespace rlib {
// print to custom stream
	template <typename PrintFinalT>
	void print(std::ostream &os, PrintFinalT reqArg);
	template <typename Required, typename... Optional>
	void print(std::ostream &os, Required reqArgs, Optional... optiArgs);
	template <typename... Optional>
	void println(std::ostream &os, Optional... optiArgs);
	template <>
	void println(std::ostream &os);

	template <typename Iterable, typename Printable>
	void print_iter(std::ostream &os, Iterable arg, Printable spliter);
	template <typename Iterable, typename Printable>
	void println_iter(std::ostream &os, Iterable arg, Printable spliter);
	template <typename Iterable>
	void print_iter(std::ostream &os, Iterable arg);
	template <typename Iterable>
	void println_iter(std::ostream &os, Iterable arg);

	template <typename... Args>
	size_t printf(std::ostream &os, const std::string &fmt, Args... args);
	template <typename... Args>
	size_t printfln(std::ostream &os, const std::string &fmt, Args... args);

	inline rlib::string scanln(std::istream &is = std::cin, char delimiter = '\n') noexcept {
		std::string line;
		std::getline(is, line, delimiter);
		return std::move(line);
	}

// print to stdout
	template <typename PrintFinalT>
	void print(PrintFinalT reqArg);
	template <typename Required, typename... Optional>
	void print(Required reqArgs, Optional... optiArgs);
	template <typename... Optional>
	void println(Optional... optiArgs);
	template <>
	void println();

	template <typename Iterable, typename Printable>
	void print_iter(Iterable arg, Printable spliter);
	template <typename Iterable, typename Printable>
	void println_iter(Iterable arg, Printable spliter);
	template <typename Iterable>
	void print_iter(Iterable arg);
	template <typename Iterable>
	void println_iter(Iterable arg);
```

```
	template <typename... Args>
	size_t printf(const std::string &fmt, Args... args);
	template <typename... Args>
	size_t printfln(const std::string &fmt, Args... args);

	namespace impl {
//#if RLIB_CXX_STD < 2017
		extern bool enable_endl_flush;
//#else
//		inline bool enable_endl_flush = true;
//#endif
	}

	inline bool sync_with_stdio(bool sync = true) noexcept {
		return std::ios::sync_with_stdio(sync);
	}
	inline bool enable_endl_flush(bool enable = true) noexcept {
		return impl::enable_endl_flush = enable;
	}

// Implements.
	template < class CharT, class Traits >
	inline std::basic_ostream<CharT, Traits>& endl(std::basic_ostream<CharT, Traits>& os) {
		os << RLIB_IMPL_ENDLINE;
		if(impl::enable_endl_flush)
			os.flush();
		return os;
	}

	template <typename PrintFinalT>
	void print(PrintFinalT reqArg)
	{
		std::cout << reqArg;
	}
	template <typename Required, typename... Optional>
	void print(Required reqArgs, Optional... optiArgs)
	{
		std::cout << reqArgs << ' ';
		print(optiArgs ...);
	}
	template <typename... Optional>
	void println(Optional... optiArgs)
	{
		print(optiArgs ...);
		println();
	}
	template <>
	inline void println()
	{
		std::cout << rlib::endl;
	}

	template <typename Iterable, typename Printable>
	void print_iter(Iterable arg, Printable spliter)
	{
		for(const auto & i : arg)
			std::cout << i << spliter;
	}
	template <typename Iterable, typename Printable>
	void println_iter(Iterable arg, Printable spliter)
	{
		print_iter(arg, spliter);
		std::cout << rlib::endl;
	}
	template <typename Iterable>
	void print_iter(Iterable arg)
	{
		for(const auto & i : arg)
			std::cout << i << ' ';
	}
	template <typename Iterable>
	void println_iter(Iterable arg)
	{
		print_iter(arg);
		std::cout << rlib::endl;
	}

	template <typename... Args>
	size_t printf(const std::string &fmt, Args... args)
	{
		std::string to_print = impl::format_string(fmt, args...);
		std::cout << to_print;
		return to_print.size();
	}
	template <typename... Args>
	size_t printfln(const std::string &fmt, Args... args)
	{
		size_t len = rlib::printf(fmt, args...);
		std::cout << rlib::endl;
		return len + 1;
	}

// With custom os
	template <typename PrintFinalT>
	void print(std::ostream &os, PrintFinalT reqArg)
	{
		os << reqArg;
	}
	template <typename Required, typename... Optional>
	void print(std::ostream &os, Required reqArgs, Optional... optiArgs)
	{
		os << reqArgs << ' ';
		print(os, optiArgs ...);
	}
	template <typename... Optional>
	void println(std::ostream &os, Optional... optiArgs)
	{
		print(os, optiArgs ...);
		println();
	}
	template <>
	inline void println(std::ostream &os)
	{
		os << rlib::endl;
	}

	template <typename Iterable, typename Printable>
	void print_iter(std::ostream &os, Iterable arg, Printable spliter)
	{
		for(const auto & i : arg)
			os << i << spliter;
	}
	template <typename Iterable, typename Printable>
	void println_iter(std::ostream &os, Iterable arg, Printable spliter)
	{
		print_iter(os, arg, spliter);
		os << rlib::endl;
	}
	template <typename Iterable>
	void print_iter(std::ostream &os, Iterable arg)
	{
		for(const auto & i : arg)
			os << i << ' ';
	}
	template <typename Iterable>
	void println_iter(std::ostream &os, Iterable arg)
	{
		print_iter(os, arg);
		os << rlib::endl;
```

```
	}

	template <typename... Args>
	size_t printf(std::ostream &os, const std::string &fmt, Args... args)
	{
		std::string to_print = format_string(fmt, args...);
		os << to_print;
		return to_print.size();
	}
	template <typename... Args>
	size_t printfln(std::ostream &os, const std::string &fmt, Args... args)
	{
		size_t len = rlib::printf(fmt, args...);
		os << rlib::endl;
		return len + 1;
	}
}


#endif
/******* ./README.md ****************/
# rlib

Here is recolic's private library...

# TODO

```c++
RETEST rlib::noncopyable
rlib::stdio fd control is still a problem.

rlib::meta::print_constexpr
rlib::meta::array
rlib::meta::array::to_tuple
rlib::meta::string

#DONE#rlib::logger
```

/******* ./terminal.hpp ****************/
/*
 *
 * terminal.hpp: unix terminal font/color wrapper for modern c++
 * by Recolic Keghart <root@recolic.net>
 * MIT License
 *
 * */

#ifndef R_STD_COLOR_HPP
#define R_STD_COLOR_HPP

#include <rlib/require/cxx11>
#include <rlib/sys/os.hpp>

#include <iostream>
#include <string>
#include <stdexcept>
#include <exception>
using std::string;
using std::basic_ostream;

namespace rlib::terminal {
	enum class color_t {color_unset = 10, black = 0, red, green, brown, blue, magenta, cyan, lightgray};
	enum class font_t {font_unset = 0, bold = 1, underline = 4, dark = 2, background = 7, striked = 9}; //Edit line53 if (int)font_t may >= 10 !!
	class clear_t {} clear;

	class fontInfo
	{
	public:
		fontInfo(color_t text_color) : textColor(text_color) {}
		fontInfo(font_t font_type) : fontType(font_type) {}
		fontInfo(color_t text_color, font_t font_type) : textColor(text_color), fontType(font_type) {}
		fontInfo(const clear_t &) : clear(true) {}
		fontInfo() = default;
		string toString() const
		{
			if(rlib::os_info::os == rlib::os_info::os_t::WINDOWS)
				return std::move(std::string());
			else
				return std::move(clear ? std::string("\033[0m") : (color_to_string() + font_to_string()));
		}
	private:
		color_t textColor = color_t::color_unset;
		font_t fontType = font_t::font_unset;
		bool clear = false;
	private:
		constexpr static int color_to_int(const color_t &_ct)
		{
			return static_cast<int>(_ct);
		}
		constexpr static int font_to_int(const font_t &_ft)
		{
			return static_cast<int>(_ft);
		}
		constexpr static char color_to_char(const color_t &_ct)
		{
			return _ct == color_t::color_unset ? '\0' : '0' + color_to_int(_ct); //Return '\0' if unset.
		}
		constexpr static char font_to_char(const font_t &_ft)
		{
			return _ft == font_t::font_unset ? '\0' :'0' + font_to_int(_ft);
		}
		string color_to_string() const
		{
			if(textColor == color_t::color_unset)
				return std::move(std::string());
			char toret[] = "\033[3?m";
			toret[3] = color_to_char(textColor);
			return std::move(std::string(toret));
		}
		string font_to_string() const
		{
			if(fontType == font_t::font_unset)
				return std::move(std::string());
			char toret[] = "\033[?m";
			toret[2] = font_to_char(fontType);
			return std::move(std::string(toret));
		}
	};

	struct _rosi_font {_rosi_font(const fontInfo &_ref_fi) : _ref_fi(_ref_fi) {} const fontInfo &_ref_fi;};
	inline _rosi_font setfont(const fontInfo &__fi) {return _rosi_font(__fi);}

	template<typename _CharT, typename _Traits>
		inline basic_ostream<_CharT, _Traits>&
		operator<<(basic_ostream<_CharT, _Traits>& __os, const fontInfo &__f)
		{
			__os << __f.toString();
			return __os;
		}

	template<typename _CharT, typename _Traits>
		inline basic_ostream<_CharT, _Traits>&
		operator<<(basic_ostream<_CharT, _Traits>& __os, _rosi_font __rosi_f)
		{
			const fontInfo &__f = __rosi_f._ref_fi;
			return operator<<<_CharT, _Traits>(__os, __f);
		}
}
#endif
```

```
/******* ./test/Makefile ****************/
# Compile and run tests for rlib
# CopyRight (C) 2017-2018 Recolic Keghart <root@recolic.net>
#
# Tests may fail on both compile-time(traits and meta-lib) and run-time(return non-zero).
# Use `make <module>` to build and run a module,
#         and `make` to build and run all modules.
#
# ready-to-use modules:
#         string          rlib/string.hpp
#         meta            rlib/meta.hpp
#         trait           rlib/traits.hpp
#         stdio           rlib/stdio.hpp
#         scope_guard rlib/scope_guard.hpp rlib/scope_guard_buffer.hpp
#         opt             rlib/opt.hpp
#         sio             rlib/sys/sio.hpp
#         os              rlib/sys/os.hpp
#         require         rlib/require/*.hpp

MODULES=string meta trait stdio sio scope_guard

def:


/******* ./test/src/fstr.cc ****************/
#include <rlib/string.hpp>
#include <rlib/stdio.hpp>

using namespace rlib;
using namespace rlib::literals;

int main()
{
	println("fuck {} at {} a.m."_format("hust", 8));
	auto s = "shit {}/{}."_rs .format("???", 1.234);
	println(s);
	println("\\{} will be replaced but \\\\{} will be preserved like {}."_format("you"));
	return 0;
}

/******* ./test/src/color.cc ****************/
#include <iostream>
#include <rlib/terminal.hpp>
using namespace rlib::terminal;
int main()
{
		std::cout << color_t::red << font_t::bold << "hello world" << font_t::dark << "qaq..." << clear << "66666" << std::endl;
		std::cout << color_t::green << "Hi~" << font_t::underline << "Miaow" << std::endl;
		std::cout << "Continue!" << clear << std::endl;
		return 0;
}
/******* ./test/src/print.cc ****************/
#include <rlib/stdio.hpp>
#include <rlib/terminal.hpp>
using namespace rlib;
using namespace rlib::terminal;

int main() {
	auto cter = printfln("{}Hello, {}={}, miao{}.{}", color_t::red, 6.6, 7, "www", clear);
	printfln("cter={}.", cter);
	println("test");
	return 0;
}
/******* ./test/src/str.cc ****************/
#include <rlib/string.hpp>
using namespace rlib;
using namespace rlib::literals;

#include <rlib/stdio.hpp>

void test(const string &s)
{
	println_iter(s.split());
	println_iter(s.split("w"));
	println(">>>", string("|").join(s.split()));
}

int main()
{
	test("dsaf wefew fwef we ");
	test("sfwaef wef wef wefew fwef eg") ;
	test("");
	test(">PAQ P<DSP<")	;

	println(string(" 87sd6 8s7d5 8	8 75	").strip() + "|");
	string s = "fuck		you		r mom 34qwo0 ghwerf 0832 ";
	s.replace("ghw", "fuck you");
	println(s);
	s.replace("	", "	");
	println(s);

	println("{} are {} shits."_format("hust and hust", 2));
	println("?"_rs .join("	shit !! ..."_rs .split()));
	return 0;
}

/******* ./test/src/call.cc ****************/
#include <rlib/traits.hpp>

using namespace rlib;

void f(int);

class c{
public:
	auto operator()(int a){
		return a+1;
	}
};

int main(){
	static_assert(is_callable<decltype(f)>(), "a");
	static_assert(is_callable<c>(), "b");
}
/******* ./test/src/print_performance.cc ****************/
#include <rlib/functional.hpp>
#include <rlib/stdio.hpp>
#include <iostream>
using namespace rlib;
#include <cstdio>

#define test_str "This is some test string."
#define test_times 1000000

int main() {
	rlib::sync_with_stdio(false);
	println(std::cerr, timeof(repeat(test_times, []{
				print(test_str "\n");
				})));
	std::cerr << (timeof(repeat(test_times, []{
				std::cout << test_str << "\n";
				}))) << std::endl;
	std::cerr << (timeof(repeat(test_times, []{
				std::printf(test_str);
				std::printf("\n");
				}))) << std::endl;
}
/******* ./test/src/_fd.cc ****************/
#include <iostream>
```

```
#include <cstdio>
#include <unistd.h>

#define IOSTREAM_PRINT(str) std::cout << str "\n"
//#define STDIO_PRINT(str) std::printf(str "\n")
#define STDIO_PRINT(str)
#define UNISTD_PRINT(str) write(1,str "\n",sizeof(str)+1)

int main() {
	std::ios::sync_with_stdio(false);
	IOSTREAM_PRINT("1");
	STDIO_PRINT("2");
	UNISTD_PRINT("3");
	IOSTREAM_PRINT("4");
	STDIO_PRINT("5");
	UNISTD_PRINT("6");
	IOSTREAM_PRINT("7");
	STDIO_PRINT("8");
	UNISTD_PRINT("9");
	IOSTREAM_PRINT("10");
	STDIO_PRINT("11");
	UNISTD_PRINT("12");
}
/******* ./test/src/cclass.c ****************/
#include <rlib/c-with-class.h>
#include <stdio.h>

RCPP_CLASS_DECL(vector)
RCPP_CLASS_METHOD_DECL_1(vector, push_back, void, int)
RCPP_CLASS_METHOD_DECL_1(vector, at, int, int)

RCPP_CLASS_BEGIN(vector)
RCPP_CLASS_METHOD_DECL_2(vector, push_back)
RCPP_CLASS_METHOD_DECL_2(vector, at)
RCPP_CLASS_END()

RCPP_CLASS_METHOD_IMPL(vector, push_back, void, int data) {
	printf("pushing back %d\n", data);
}
RCPP_CLASS_METHOD_IMPL(vector, at, int, int index) {
	int element = index * index;
	return element;
}

RCPP_CLASS_CONSTRUCTOR_IMPL(vector) {
	RCPP_CLASS_METHOD_REGISTER(vector, push_back)
	RCPP_CLASS_METHOD_REGISTER(vector, at)
	printf("constructor called\n");
}
RCPP_CLASS_DESTRUCTOR_IMPL(vector) {
	printf("destructor called\n");
}
int main(){
	RCPP_NEW(vector, vct, NULL);
	RCPP_CALL(vct, push_back, 333);
	vct.push_back(&vct, 666);

	printf("Element at index %d is %d.\n", 5, vct.at(&vct, 5));
	return 123;
}
/******* ./test/src/meta.cc ****************/
#include <rlib/meta.hpp>
#include <string>

int main() {
	static_assert(rlib::meta_array<33,34,35,36,37>::at<1>() == 34);
	static_assert(rlib::meta_array<'f','u','c','k'>::at<2>() == 'c');
	static_assert(std::get<1>(rlib::meta_array<'f','u','c','k'>::to_tuple()) == 'u');
}
/******* ./test/src/functional.cc ****************/
#include <rlib/functional.hpp>
#include <rlib/stdio.hpp>


int main() {
//	auto f = rlib::repeat(4, [](int i){
//		rlib::println("i is", i);
//		}, 777);
//	f();
	auto m = [](int i){
		rlib::println("i is", i);
	};
	std::function<void(int)> b (m);
	auto f = rlib::repeat(4, m, 777);
//	f(4,m,444);
//	auto ff = std::bind(&decltype(f)::operator(), &f, 4, m, 444);
	//std::function<void(size_t, decltype(m), int)> goodf(f);

//	auto ff = std::bind(f, 4,m,444);
	f();

	rlib::println("time of f is", rlib::timeof(f));

}

/******* ./test/src/log.cc ****************/
#include <rlib/log.hpp>

using namespace rlib;
int main() {
	// logging to stdout.
	logger stdout_logger(std::cout);
	stdout_logger.info("test");
	stdout_logger.debug("running shit.");
	stdout_logger.info("{} is {} {}."_format("hust asm", 1, "shit"));
	// logging to file.
	logger file_logger("/tmp/rlib.test.log");
	file_logger.info("shit here.");
	file_logger.verbose("???");

	log_level_t my_level = file_logger.register_log_level("MyLogLev");
	file_logger.log(my_level, "my info.... whit's a fuck");
	return 0;
}
/******* ./test/src/opt.cc ****************/
#include <rlib/opt.hpp>
#include <rlib/stdio.hpp>
using rlib::println;
using rlib::print;

#include <iomanip>
#include <cassert>
int main(int argl, char **argv)
{
	rlib::opt_parser opt(argl, argv);
	print(std::boolalpha);
	println(opt.getValueArg("--fuck", false), opt.getValueArg("--shit", "-s"), opt.getBoolArg("--boolt", "-b"));
	println("ALldone:", opt.allArgDone());
	return 0;
}

/******* ./test/src/os.cc ****************/
#include <rlib/sys/os.hpp>

#if RLIB_OS_ID != OS_LINUX
#error fuck
#endif

static_assert(rlib::os_info::os == rlib::os_info::os_t::LINUX);
```

```
static_assert(rlib::os_info::compiler == rlib::os_info::compiler_t::GCC);

int main(){

}
/******* ./test/src/macro.c ****************/
#define a b
#include <rlib/macro.hpp>
#include <rlib/print.hpp>

#define ma 'c'
#define mb a
using rlib::println;

#define b 'c'
int main()
{
	if(MACRO_EQL(ma, mb))
	{
		println("Hello world.");
	}
	println(MACRO_TO_CSTR(a));
	println("done");

	return 0;

}
/******* ./test/src/c.fish ****************/
for fl in (ls *.cc)
	g++ $fl -g -o /tmp/$fl.ex -std=c++17 -lr
end
/******* ./3rdparty/prettyprint.hpp ****************/
//		Copyright Louis Delacroix 2010 - 2014.
// Distributed under the Boost Software License, Version 1.0.
//	(See accompanying file LICENSE_1_0.txt or copy at
//		http://www.boost.org/LICENSE_1_0.txt)
//
// A pretty printing library for C++
//
// Usage:
// Include this header, and operator<< will "just work".

#ifndef H_PRETTY_PRINT
#define H_PRETTY_PRINT

// Recolic add
#if __GLIBCXX__OSTREAM
#include <rlib/sys/os.hpp>
#if RLIB_COMPILER_ID == CC_CLANG
#error in clang, you must include prettyprint.hpp before STD ostream or rlib/stdio.hpp.
#endif
#endif

#include <cstddef>
#include <iterator>
#include <memory>
#include <ostream>
#include <set>
#include <tuple>
#include <type_traits>
#include <unordered_set>
#include <utility>
#include <valarray>

#ifndef RLIB_3RD_ENABLE_PRETTYPRINT
namespace rlib {
	namespace _3rdparty {
#endif
namespace pretty_print
{
	namespace detail
	{
		// SFINAE type trait to detect whether T::const_iterator exists.

		struct sfinae_base
		{
			using yes = char;
			using no   = yes[2];
		};

		template <typename T>
		struct has_const_iterator : private sfinae_base
		{
		private:
			template <typename C> static yes & test(typename C::const_iterator*);
			template <typename C> static no   & test(...);
		public:
			static const bool value = sizeof(test<T>(nullptr)) == sizeof(yes);
			using type =   T;
		};

		template <typename T>
		struct has_begin_end : private sfinae_base
		{
		private:
			template <typename C>
			static yes & f(typename std::enable_if<
				std::is_same<decltype(static_cast<typename C::const_iterator(C::*)() const>(&C::begin)),
						typename C::const_iterator(C::*)() const>::value>::type *);

			template <typename C> static no & f(...);

			template <typename C>
			static yes & g(typename std::enable_if<
				std::is_same<decltype(static_cast<typename C::const_iterator(C::*)() const>(&C::end)),
						typename C::const_iterator(C::*)() const>::value, void>::type*);

			template <typename C> static no & g(...);

		public:
			static bool const beg_value = sizeof(f<T>(nullptr)) == sizeof(yes);
			static bool const end_value = sizeof(g<T>(nullptr)) == sizeof(yes);
		};

	}   // namespace detail


	// Holds the delimiter values for a specific character type

	template <typename TChar>
	struct delimiters_values
	{
		using char_type = TChar;
		const char_type * prefix;
		const char_type * delimiter;
		const char_type * postfix;
	};


	// Defines the delimiter values for a specific container and character type

	template <typename T, typename TChar>
	struct delimiters
	{
		using type = delimiters_values<TChar>;
		static const type values;
	};
```

```
// Functor to print containers. You can use this directly if you want
// to specificy a non-default delimiters type. The printing logic can
// be customized by specializing the nested template.

template <typename T,
          typename TChar = char,
          typename TCharTraits = ::std::char_traits<TChar>,
          typename TDelimiters = delimiters<T, TChar>>
struct print_container_helper
{
    using delimiters_type = TDelimiters;
    using ostream_type = std::basic_ostream<TChar, TCharTraits>;

    template <typename U>
    struct printer
    {
        static void print_body(const U & c, ostream_type & stream)
        {
            using std::begin;
            using std::end;

            auto it = begin(c);
            const auto the_end = end(c);

            if (it != the_end)
            {
                for ( ; ; )
                {
                    stream << *it;

                if (++it == the_end) break;

                if (delimiters_type::values.delimiter != NULL)
                    stream << delimiters_type::values.delimiter;
                }
            }
        }
    };

    print_container_helper(const T & container)
    : container_(container)
    { }

    inline void operator()(ostream_type & stream) const
    {
        if (delimiters_type::values.prefix != NULL)
            stream << delimiters_type::values.prefix;

        printer<T>::print_body(container_, stream);

        if (delimiters_type::values.postfix != NULL)
            stream << delimiters_type::values.postfix;
    }

private:
    const T & container_;
};

// Specialization for pairs

template <typename T, typename TChar, typename TCharTraits, typename TDelimiters>
template <typename T1, typename T2>
struct print_container_helper<T, TChar, TCharTraits, TDelimiters>::printer<std::pair<T1, T2>>
{
    using ostream_type = typename print_container_helper<T, TChar, TCharTraits, TDelimiters>::ostream_type;

    static void print_body(const std::pair<T1, T2> & c, ostream_type & stream)
    {
        stream << c.first;
        if (print_container_helper<T, TChar, TCharTraits, TDelimiters>::delimiters_type::values.delimiter != NULL)
            stream << print_container_helper<T, TChar, TCharTraits, TDelimiters>::delimiters_type::values.delimiter;
        stream << c.second;
    }
};

// Specialization for tuples

template <typename T, typename TChar, typename TCharTraits, typename TDelimiters>
template <typename ...Args>
struct print_container_helper<T, TChar, TCharTraits, TDelimiters>::printer<std::tuple<Args...>>
{
    using ostream_type = typename print_container_helper<T, TChar, TCharTraits, TDelimiters>::ostream_type;
    using element_type = std::tuple<Args...>;

    template <std::size_t I> struct Int { };

    static void print_body(const element_type & c, ostream_type & stream)
    {
        tuple_print(c, stream, Int<0>());
    }

    static void tuple_print(const element_type &, ostream_type &, Int<sizeof...(Args)>)
    {
    }

    static void tuple_print(const element_type & c, ostream_type & stream,
                            typename std::conditional<sizeof...(Args) != 0, Int<0>, std::nullptr_t>::type)
    {
        stream << std::get<0>(c);
        tuple_print(c, stream, Int<1>());
    }

    template <std::size_t N>
    static void tuple_print(const element_type & c, ostream_type & stream, Int<N>)
    {
        if (print_container_helper<T, TChar, TCharTraits, TDelimiters>::delimiters_type::values.delimiter != NULL)
            stream << print_container_helper<T, TChar, TCharTraits, TDelimiters>::delimiters_type::values.delimiter;

        stream << std::get<N>(c);

        tuple_print(c, stream, Int<N + 1>());
    }
};

// Prints a print_container_helper to the specified stream.

template<typename T, typename TChar, typename TCharTraits, typename TDelimiters>
inline std::basic_ostream<TChar, TCharTraits> & operator<<(
    std::basic_ostream<TChar, TCharTraits> & stream,
    const print_container_helper<T, TChar, TCharTraits, TDelimiters> & helper)
{
    helper(stream);
    return stream;
}


// Basic is_container template; specialize to derive from std::true_type for all desired container types

template <typename T>
struct is_container : public std::integral_constant<bool,
                                                    detail::has_const_iterator<T>::value &&
                                                    detail::has_begin_end<T>::beg_value   &&
                                                    detail::has_begin_end<T>::end_value> { };

template <typename T, std::size_t N>
struct is_container<T[N]> : std::true_type { };

template <std::size_t N>
struct is_container<char[N]> : std::false_type { };
```

```
template <typename T>
struct is_container<std::valarray<T>> : std::true_type { };

template <typename T1, typename T2>
struct is_container<std::pair<T1, T2>> : std::true_type { };

template <typename ...Args>
struct is_container<std::tuple<Args...>> : std::true_type { };


// Default delimiters

template <typename T> struct delimiters<T, char> { static const delimiters_values<char> values; };
template <typename T> const delimiters_values<char> delimiters<T, char>::values = { "[", ", ", "]" };
template <typename T> struct delimiters<T, wchar_t> { static const delimiters_values<wchar_t> values; };
template <typename T> const delimiters_values<wchar_t> delimiters<T, wchar_t>::values = { L"[", L", ", L"]" };


// Delimiters for (multi)set and unordered_(multi)set

template <typename T, typename TComp, typename TAllocator>
struct delimiters< ::std::set<T, TComp, TAllocator>, char> { static const delimiters_values<char> values; };

template <typename T, typename TComp, typename TAllocator>
const delimiters_values<char> delimiters< ::std::set<T, TComp, TAllocator>, char>::values = { "{", ", ", "}" };

template <typename T, typename TComp, typename TAllocator>
struct delimiters< ::std::set<T, TComp, TAllocator>, wchar_t> { static const delimiters_values<wchar_t> values; };

template <typename T, typename TComp, typename TAllocator>
const delimiters_values<wchar_t> delimiters< ::std::set<T, TComp, TAllocator>, wchar_t>::values = { L"{", L", ", L"}" };

template <typename T, typename TComp, typename TAllocator>
struct delimiters< ::std::multiset<T, TComp, TAllocator>, char> { static const delimiters_values<char> values; };

template <typename T, typename TComp, typename TAllocator>
const delimiters_values<char> delimiters< ::std::multiset<T, TComp, TAllocator>, char>::values = { "{", ", ", "}" };

template <typename T, typename TComp, typename TAllocator>
struct delimiters< ::std::multiset<T, TComp, TAllocator>, wchar_t> { static const delimiters_values<wchar_t> values; };

template <typename T, typename TComp, typename TAllocator>
const delimiters_values<wchar_t> delimiters< ::std::multiset<T, TComp, TAllocator>, wchar_t>::values = { L"{", L", ", L"}" };

template <typename T, typename THash, typename TEqual, typename TAllocator>
struct delimiters< ::std::unordered_set<T, THash, TEqual, TAllocator>, char> { static const delimiters_values<char> values; };

template <typename T, typename THash, typename TEqual, typename TAllocator>
const delimiters_values<char> delimiters< ::std::unordered_set<T, THash, TEqual, TAllocator>, char>::values = { "{", ", ", "}" };

template <typename T, typename THash, typename TEqual, typename TAllocator>
struct delimiters< ::std::unordered_set<T, THash, TEqual, TAllocator>, wchar_t> { static const delimiters_values<wchar_t> values; };

template <typename T, typename THash, typename TEqual, typename TAllocator>
const delimiters_values<wchar_t> delimiters< ::std::unordered_set<T, THash, TEqual, TAllocator>, wchar_t>::values = { L"{", L", ", L"}" };

template <typename T, typename THash, typename TEqual, typename TAllocator>
struct delimiters< ::std::unordered_multiset<T, THash, TEqual, TAllocator>, char> { static const delimiters_values<char> values; };

template <typename T, typename THash, typename TEqual, typename TAllocator>
const delimiters_values<char> delimiters< ::std::unordered_multiset<T, THash, TEqual, TAllocator>, char>::values = { "{", ", ", "}" };

template <typename T, typename THash, typename TEqual, typename TAllocator>
struct delimiters< ::std::unordered_multiset<T, THash, TEqual, TAllocator>, wchar_t> { static const delimiters_values<wchar_t> values; };

template <typename T, typename THash, typename TEqual, typename TAllocator>
const delimiters_values<wchar_t> delimiters< ::std::unordered_multiset<T, THash, TEqual, TAllocator>, wchar_t>::values = { L"{", L", ", L"}" };


// Delimiters for pair and tuple

template <typename T1, typename T2> struct delimiters<std::pair<T1, T2>, char> { static const delimiters_values<char> values; };
template <typename T1, typename T2> const delimiters_values<char> delimiters<std::pair<T1, T2>, char>::values = { "(", ", ", ")" };
template <typename T1, typename T2> struct delimiters< ::std::pair<T1, T2>, wchar_t> { static const delimiters_values<wchar_t> values; };
template <typename T1, typename T2> const delimiters_values<wchar_t> delimiters< ::std::pair<T1, T2>, wchar_t>::values = { L"(", L", ", L")" };

template <typename ...Args> struct delimiters<std::tuple<Args...>, char> { static const delimiters_values<char> values; };
template <typename ...Args> const delimiters_values<char> delimiters<std::tuple<Args...>, char>::values = { "(", ", ", ")" };
template <typename ...Args> struct delimiters< ::std::tuple<Args...>, wchar_t> { static const delimiters_values<wchar_t> values; };
template <typename ...Args> const delimiters_values<wchar_t> delimiters< ::std::tuple<Args...>, wchar_t>::values = { L"(", L", ", L")" };


// Type-erasing helper class for easy use of custom delimiters.
// Requires TCharTraits = std::char_traits<TChar> and TChar = char or wchar_t, and MyDelims needs to be defined for TChar.
// Usage: "cout << pretty_print::custom_delims<MyDelims>(x)".

struct custom_delims_base
{
    virtual ~custom_delims_base() { }
    virtual std::ostream & stream(::std::ostream &) = 0;
    virtual std::wostream & stream(::std::wostream &) = 0;
};

template <typename T, typename Delims>
struct custom_delims_wrapper : custom_delims_base
{
    custom_delims_wrapper(const T & t_) : t(t_) { }

    std::ostream & stream(std::ostream & s)
    {
        return s << print_container_helper<T, char, std::char_traits<char>, Delims>(t);
    }

    std::wostream & stream(std::wostream & s)
    {
        return s << print_container_helper<T, wchar_t, std::char_traits<wchar_t>, Delims>(t);
    }

private:
    const T & t;
};

template <typename Delims>
struct custom_delims
{
    template <typename Container>
    custom_delims(const Container & c) : base(new custom_delims_wrapper<Container, Delims>(c)) { }

    std::unique_ptr<custom_delims_base> base;
};

template <typename TChar, typename TCharTraits, typename Delims>
inline std::basic_ostream<TChar, TCharTraits> & operator<<(std::basic_ostream<TChar, TCharTraits> & s, const custom_delims<Delims> & p)
{
    return p.base->stream(s);
}


// A wrapper for a C-style array given as pointer-plus-size.
// Usage: std::cout << pretty_print_array(arr, n) << std::endl;

template<typename T>
struct array_wrapper_n
{
    typedef const T * const_iterator;
    typedef T value_type;

    array_wrapper_n(const T * const a, size_t n) : _array(a), _n(n) { }
```

```
            inline const_iterator begin() const { return _array; }
            inline const_iterator end() const { return _array + _n; }

      private:
            const T * const _array;
            size_t _n;
      };


      // A wrapper for hash-table based containers that offer local iterators to each bucket.
      // Usage: std::cout << bucket_print(m, 4) << std::endl;   (Prints bucket 5 of container m.)

      template <typename T>
      struct bucket_print_wrapper
      {
            typedef typename T::const_local_iterator const_iterator;
            typedef typename T::size_type size_type;

            const_iterator begin() const
            {
                  return m_map.cbegin(n);
            }

            const_iterator end() const
            {
                  return m_map.cend(n);
            }

            bucket_print_wrapper(const T & m, size_type bucket) : m_map(m), n(bucket) { }

      private:
            const T & m_map;
            const size_type n;
      };

}     // namespace pretty_print


// Global accessor functions for the convenience wrappers

template<typename T>
inline pretty_print::array_wrapper_n<T> pretty_print_array(const T * const a, size_t n)
{
      return pretty_print::array_wrapper_n<T>(a, n);
}

template <typename T> pretty_print::bucket_print_wrapper<T>
bucket_print(const T & m, typename T::size_type n)
{
      return pretty_print::bucket_print_wrapper<T>(m, n);
}


// Main magic entry point: An overload snuck into namespace std.
// Can we do better?

namespace std
{
      // Prints a container to the stream using default delimiters

      template<typename T, typename TChar, typename TCharTraits>
      inline typename ::std::enable_if< pretty_print::is_container<T>::value,
                                        ::std::basic_ostream<TChar, TCharTraits> &>::type
      operator<<(::std::basic_ostream<TChar, TCharTraits> & stream, const T & container)
      {
            return stream << pretty_print::print_container_helper<T, TChar, TCharTraits>(container);
      }
}
#ifndef RLIB_3RD_ENABLE_PRETTYPRINT
      }
} // end namespace rlib::3rdparty
#endif // RLIB_3RD_ENABLE_PRETTYPRINT



#endif   // H_PRETTY_PRINT
/******* ./3rdparty/prettyprint98.hpp ****************/
//              Copyright Louis Delacroix 2010 - 2014.
// Distributed under the Boost Software License, Version 1.0.
//      (See accompanying file LICENSE_1_0.txt or copy at
//              http://www.boost.org/LICENSE_1_0.txt)


#ifndef H_PRETTY_PRINT
#define H_PRETTY_PRINT


#include <ostream>
#include <utility>
#include <iterator>
#include <set>

#ifndef NO_TR1
#   include <tr1/tuple>
#   include <tr1/unordered_set>
#endif

namespace pretty_print
{

      template <bool, typename S, typename T> struct conditional { };
      template <typename S, typename T> struct conditional<true,   S, T> { typedef S type; };
      template <typename S, typename T> struct conditional<false, S, T> { typedef T type; };

      template <bool, typename T> struct enable_if { };
      template <typename T> struct enable_if<true, T> { typedef T type; };

      // SFINAE type trait to detect whether T::const_iterator exists.

      template<typename T>
      struct has_const_iterator
      {
      private:
            typedef char                              yes;
            typedef struct { char array[2]; } no;

            template<typename C> static yes test(typename C::const_iterator*);
            template<typename C> static no   test(...);
      public:
            static const bool value = sizeof(test<T>(0)) == sizeof(yes);
            typedef T type;
      };

      // SFINAE type trait to detect whether "T::const_iterator T::begin/end() const" exist.

      template <typename T>
      struct has_begin_end
      {
            struct Dummy { typedef void const_iterator; };
            typedef typename conditional<has_const_iterator<T>::value, T, Dummy>::type TType;
            typedef typename TType::const_iterator iter;

            struct Fallback { iter begin() const; iter end() const; };
            struct Derived : TType, Fallback { };

            template<typename C, C> struct ChT;

            template<typename C> static char (&f(ChT<iter (Fallback::*)() const, &C::begin>*))[1];
            template<typename C> static char (&f(...))[2];
```

```
        template<typename C> static char (&g(ChT<iter (Fallback::*)() const, &C::end>*))[1];
        template<typename C> static char (&g(...))[2];

        static bool const beg_value = sizeof(f<Derived>(0)) == 2;
        static bool const end_value = sizeof(g<Derived>(0)) == 2;
    };

    // Basic is_container template; specialize to have value "true" for all desired container types

    template<typename T> struct is_container { static const bool value = has_const_iterator<T>::value && has_begin_end<T>::beg_value &&
has_begin_end<T>::end_value; };

    template<typename T, std::size_t N> struct is_container<T[N]> { static const bool value = true; };

    template<std::size_t N> struct is_container<char[N]> { static const bool value = false; };


    // Holds the delimiter values for a specific character type

    template<typename TChar>
    struct delimiters_values
    {
        typedef TChar char_type;
        const TChar * prefix;
        const TChar * delimiter;
        const TChar * postfix;
    };


    // Defines the delimiter values for a specific container and character type

    template<typename T, typename TChar>
    struct delimiters
    {
        typedef delimiters_values<TChar> type;
        static const type values;
    };


    // Default delimiters

    template<typename T> struct delimiters<T, char> { static const delimiters_values<char> values; };
    template<typename T> const delimiters_values<char> delimiters<T, char>::values = { "[", ", ", "]" };
    template<typename T> struct delimiters<T, wchar_t> { static const delimiters_values<wchar_t> values; };
    template<typename T> const delimiters_values<wchar_t> delimiters<T, wchar_t>::values = { L"[", L", ", L"]" };


    // Delimiters for (multi)set and unordered_(multi)set

    template<typename T, typename TComp, typename TAllocator>
    struct delimiters< ::std::set<T, TComp, TAllocator>, char> { static const delimiters_values<char> values; };

    template<typename T, typename TComp, typename TAllocator>
    const delimiters_values<char> delimiters< ::std::set<T, TComp, TAllocator>, char>::values = { "{", ", ", "}" };

    template<typename T, typename TComp, typename TAllocator>
    struct delimiters< ::std::set<T, TComp, TAllocator>, wchar_t> { static const delimiters_values<wchar_t> values; };

    template<typename T, typename TComp, typename TAllocator>
    const delimiters_values<wchar_t> delimiters< ::std::set<T, TComp, TAllocator>, wchar_t>::values = { L"{", L", ", L"}" };

    template<typename T, typename TComp, typename TAllocator>
    struct delimiters< ::std::multiset<T, TComp, TAllocator>, char> { static const delimiters_values<char> values; };

    template<typename T, typename TComp, typename TAllocator>
    const delimiters_values<char> delimiters< ::std::multiset<T, TComp, TAllocator>, char>::values = { "{", ", ", "}" };

    template<typename T, typename TComp, typename TAllocator>
    struct delimiters< ::std::multiset<T, TComp, TAllocator>, wchar_t> { static const delimiters_values<wchar_t> values; };

    template<typename T, typename TComp, typename TAllocator>
    const delimiters_values<wchar_t> delimiters< ::std::multiset<T, TComp, TAllocator>, wchar_t>::values = { L"{", L", ", L"}" };

#ifndef NO_TR1
    template<typename T, typename THash, typename TEqual, typename TAllocator>
    struct delimiters< ::std::tr1::unordered_set<T, THash, TEqual, TAllocator>, char> { static const delimiters_values<char> values; };

    template<typename T, typename THash, typename TEqual, typename TAllocator>
    const delimiters_values<char> delimiters< ::std::tr1::unordered_set<T, THash, TEqual, TAllocator>, char>::values = { "{", ", ", "}" };

    template<typename T, typename THash, typename TEqual, typename TAllocator>
    struct delimiters< ::std::tr1::unordered_set<T, THash, TEqual, TAllocator>, wchar_t> { static const delimiters_values<wchar_t> values; };

    template<typename T, typename THash, typename TEqual, typename TAllocator>
    const delimiters_values<wchar_t> delimiters< ::std::tr1::unordered_set<T, THash, TEqual, TAllocator>, wchar_t>::values = { L"{", L", ", L"}" };

    template<typename T, typename THash, typename TEqual, typename TAllocator>
    struct delimiters< ::std::tr1::unordered_multiset<T, THash, TEqual, TAllocator>, char> { static const delimiters_values<char> values; };

    template<typename T, typename THash, typename TEqual, typename TAllocator>
    const delimiters_values<char> delimiters< ::std::tr1::unordered_multiset<T, THash, TEqual, TAllocator>, char>::values = { "{", ", ", "}" };

    template<typename T, typename THash, typename TEqual, typename TAllocator>
    struct delimiters< ::std::tr1::unordered_multiset<T, THash, TEqual, TAllocator>, wchar_t> { static const delimiters_values<wchar_t> values; };

    template<typename T, typename THash, typename TEqual, typename TAllocator>
    const delimiters_values<wchar_t> delimiters< ::std::tr1::unordered_multiset<T, THash, TEqual, TAllocator>, wchar_t>::values = { L"{", L", ",
L"}" };
#endif


    // Delimiters for pair (reused for tuple, see below)

    template<typename T1, typename T2> struct delimiters< ::std::pair<T1, T2>, char> { static const delimiters_values<char> values; };
    template<typename T1, typename T2> const delimiters_values<char> delimiters< ::std::pair<T1, T2>, char>::values = { "(", ", ", ")" };
    template<typename T1, typename T2> struct delimiters< ::std::pair<T1, T2>, wchar_t> { static const delimiters_values<wchar_t> values; };
    template<typename T1, typename T2> const delimiters_values<wchar_t> delimiters< ::std::pair<T1, T2>, wchar_t>::values = { L"(", L", ", L")" };


    // Iterator microtrait class to handle C arrays uniformly

    template <typename T> struct get_iterator { typedef typename T::const_iterator iter; };
    template <typename T, std::size_t N> struct get_iterator<T[N]> { typedef const T * iter; };

    template <typename T> typename enable_if<has_const_iterator<T>::value, typename T::const_iterator>::type begin(const T & c) { return
c.begin(); }
    template <typename T> typename enable_if<has_const_iterator<T>::value, typename T::const_iterator>::type end(const T & c) { return c.end(); }
    template <typename T, size_t N> const T * begin(const T(&x)[N]) { return &x[0];       }
    template <typename T, size_t N> const T * end   (const T(&x)[N]) { return &x[0] + N; }

    // Functor to print containers. You can use this directly if you want to specificy a non-default delimiters type.

    template<typename T, typename TChar = char, typename TCharTraits = ::std::char_traits<TChar>, typename TDelimiters = delimiters<T,
TChar> >
    struct print_container_helper
    {
        typedef TChar char_type;
        typedef TDelimiters delimiters_type;
        typedef std::basic_ostream<TChar, TCharTraits> ostream_type;
        typedef typename get_iterator<T>::iter TIter;

        print_container_helper(const T & container)
        : _container(container)
        {
        }

        inline void operator()(ostream_type & stream) const
        {
            if (delimiters_type::values.prefix != NULL)
```

```
                stream << delimiters_type::values.prefix;

            if (begin(_container) != end(_container))
            for (TIter it = begin(_container), it_end = end(_container); ; )
            {
                stream << *it;

                if (++it == it_end) break;

                if (delimiters_type::values.delimiter != NULL)
                    stream << delimiters_type::values.delimiter;
            }

            if (delimiters_type::values.postfix != NULL)
                stream << delimiters_type::values.postfix;
        }

    private:
        const T & _container;
    };


    // Type-erasing helper class for easy use of custom delimiters.
    // Requires TCharTraits = std::char_traits<TChar> and TChar = char or wchar_t, and MyDelims needs to be defined for TChar.
    // Usage: "cout << pretty_print::custom_delims<MyDelims>(x)".

    struct custom_delims_base
    {
        virtual ~custom_delims_base() { }
        virtual ::std::ostream & stream(::std::ostream &) = 0;
        virtual ::std::wostream & stream(::std::wostream &) = 0;
    };

    template<typename T, typename Delims>
    struct custom_delims_wrapper : public custom_delims_base
    {
        custom_delims_wrapper(const T & t_) : t(t_) { }

        ::std::ostream & stream(::std::ostream & s)
        {
            return s << ::pretty_print::print_container_helper<T, char, ::std::char_traits<char>, Delims>(t);
        }
        ::std::wostream & stream(::std::wostream & s)
        {
            return s << ::pretty_print::print_container_helper<T, wchar_t, ::std::char_traits<wchar_t>, Delims>(t);
        }

    private:
        const T & t;
    };

    template<typename Delims>
    struct custom_delims
    {
        template<typename Container> custom_delims(const Container & c) : base(new custom_delims_wrapper<Container, Delims>(c)) { }
        ~custom_delims() { delete base; }
        custom_delims_base * base;
    };

} // namespace pretty_print

template<typename TChar, typename TCharTraits, typename Delims>
inline std::basic_ostream<TChar, TCharTraits> & operator<<(std::basic_ostream<TChar, TCharTraits> & s, const pretty_print::custom_delims<Delims>
& p)
{
    return p.base->stream(s);
}

// Template aliases for char and wchar_t delimiters
// Enable these if you have compiler support
//
// Implement as "template<T, C, A> const sdelims::type sdelims<std::set<T,C,A>>::values = { ... }."

//template<typename T> using pp_sdelims = pretty_print::delimiters<T, char>;
//template<typename T> using pp_wsdelims = pretty_print::delimiters<T, wchar_t>;


namespace std
{
    // Prints a print_container_helper to the specified stream.

    template<typename T, typename TChar, typename TCharTraits, typename TDelimiters>
    inline basic_ostream<TChar, TCharTraits> & operator<<(basic_ostream<TChar, TCharTraits> & stream,
                                                          const ::pretty_print::print_container_helper<T, TChar, TCharTraits, TDelimiters> & helper)
    {
        helper(stream);
        return stream;
    }

    // Prints a container to the stream using default delimiters

    template<typename T, typename TChar, typename TCharTraits>
    inline typename ::pretty_print::enable_if< ::pretty_print::is_container<T>::value, basic_ostream<TChar, TCharTraits>&>::type
    operator<<(basic_ostream<TChar, TCharTraits> & stream, const T & container)
    {
        return stream << ::pretty_print::print_container_helper<T, TChar, TCharTraits>(container);
    }

    // Prints a pair to the stream using delimiters from delimiters<std::pair<T1, T2>>.
    template<typename T1, typename T2, typename TChar, typename TCharTraits>
    inline basic_ostream<TChar, TCharTraits> & operator<<(basic_ostream<TChar, TCharTraits> & stream, const pair<T1, T2> & value)
    {
        if (::pretty_print::delimiters<pair<T1, T2>, TChar>::values.prefix != NULL)
            stream << ::pretty_print::delimiters<pair<T1, T2>, TChar>::values.prefix;

        stream << value.first;

        if (::pretty_print::delimiters<pair<T1, T2>, TChar>::values.delimiter != NULL)
            stream << ::pretty_print::delimiters<pair<T1, T2>, TChar>::values.delimiter;

        stream << value.second;

        if (::pretty_print::delimiters<pair<T1, T2>, TChar>::values.postfix != NULL)
            stream << ::pretty_print::delimiters<pair<T1, T2>, TChar>::values.postfix;

        return stream;
    }
} // namespace std


#ifndef NO_TR1

// Prints a tuple to the stream using delimiters from delimiters<std::pair<tuple_dummy_t, tuple_dummy_t>>.

namespace pretty_print
{
    struct tuple_dummy_t { }; // Just if you want special delimiters for tuples.

    typedef std::pair<tuple_dummy_t, tuple_dummy_t> tuple_dummy_pair;

    template<typename Tuple, size_t N, typename TChar, typename TCharTraits>
    struct pretty_tuple_helper
    {
        static inline void print(::std::basic_ostream<TChar, TCharTraits> & stream, const Tuple & value)
        {
            pretty_tuple_helper<Tuple, N - 1, TChar, TCharTraits>::print(stream, value);

            if (delimiters<tuple_dummy_pair, TChar>::values.delimiter != NULL)
                stream << delimiters<tuple_dummy_pair, TChar>::values.delimiter;
```

```
                stream << std::tr1::get<N - 1>(value);
            }
        };

        template<typename Tuple, typename TChar, typename TCharTraits>
        struct pretty_tuple_helper<Tuple, 1, TChar, TCharTraits>
        {
            static inline void print(::std::basic_ostream<TChar, TCharTraits> & stream, const Tuple & value)
            {
                stream << ::std::tr1::get<0>(value);
            }
        };
} // namespace pretty_print


/* The following macros allow us to write "template <TUPLE_PARAMAS> std::tuple<TUPLE_ARGS>"
 * uniformly in C++0x compilers and in MS Visual Studio 2010.
 * Credits to STL: http://channel9.msdn.com/Shows/Going+Deep/C9-Lectures-Stephan-T-Lavavej-Advanced-STL-6-of-n
 */

#define TUPLE_PARAMS \
        typename T0, typename T1, typename T2, typename T3, typename T4,          \
        typename T5, typename T6, typename T7, typename T8, typename T9
#define TUPLE_ARGS T0, T1, T2, T3, T4, T5, T6, T7, T8, T9


namespace std
{
        template<typename TChar, typename TCharTraits, TUPLE_PARAMS>
        inline basic_ostream<TChar, TCharTraits> & operator<<(basic_ostream<TChar, TCharTraits> & stream, const tr1::tuple<TUPLE_ARGS> & value)
        {
            if (::pretty_print::delimiters< ::pretty_print::tuple_dummy_pair, TChar>::values.prefix != NULL)
                stream << ::pretty_print::delimiters< ::pretty_print::tuple_dummy_pair, TChar>::values.prefix;

            ::pretty_print::pretty_tuple_helper<const      tr1::tuple<TUPLE_ARGS>      &,      tr1::tuple_size<tr1::tuple<TUPLE_ARGS>      >::value,      TChar,
TCharTraits>::print(stream, value);

            if (::pretty_print::delimiters< ::pretty_print::tuple_dummy_pair, TChar>::values.postfix != NULL)
                stream << ::pretty_print::delimiters< ::pretty_print::tuple_dummy_pair, TChar>::values.postfix;

            return stream;
        }
} // namespace std

#endif // NO_TR1


// A wrapper for raw C-style arrays. Usage: int arr[] = { 1, 2, 4, 8, 16 };   std::cout << wrap_array(arr) << ...

namespace pretty_print
{
        template<typename T>
        struct array_wrapper_n
        {
            typedef const T * const_iterator;
            typedef T value_type;

            array_wrapper_n(const T * const a, size_t n) : _array(a), _n(n) { }
            inline const_iterator begin() const { return _array; }
            inline const_iterator end() const { return _array + _n; }

        private:
            const T * const _array;
            size_t _n;
        };
} // namespace pretty_print

template<typename T>
inline pretty_print::array_wrapper_n<T> pretty_print_array(const T * const a, size_t n)
{
    return pretty_print::array_wrapper_n<T>(a, n);
}


#endif
/******* ./3rdparty/test.cc ****************/
#include "prettyprint.hpp"
#include <rlib/stdio.hpp>
#include <list>
using namespace rlib;

int main() {
        std::list ls {1,3,2};
        _3rdparty::std::operator<<(std::cout, ls);
}
/******* ./macro.hpp ****************/
#ifndef R_MACRO_HPP
#define R_MACRO_HPP

#ifdef RLIB_EMPTY_MACRO
#undef RLIB_EMPTY_MACRO
#endif
#define RLIB_EMPTY_MACRO

#ifndef RLIB_MACRO_DECAY
#define RLIB_MACRO_DECAY(m) (m)
#endif

#ifndef _RLIB_MACRO_ENSTRING
#define _RLIB_MACRO_ENSTRING(_s) #_s
#endif

#ifndef RLIB_MACRO_TO_CSTR
#define RLIB_MACRO_TO_CSTR(m) _RLIB_MACRO_ENSTRING(m)
#endif

//#ifndef RLIB_MACRO_EQL
//#define RLIB_MACRO_EQL(a, b) (RLIB_MACRO_TO_CSTR(a) == RLIB_MACRO_TO_CSTR(b))
//#endif

#ifndef RLIB_MACRO_CAT
#define RLIB_MACRO_CAT(a, b)   _RLIB_MACRO_CAT_I(a, b)
#define _RLIB_MACRO_CAT_I(a, b)   _RLIB_MACRO_CAT_II(~, a ## b)
#define _RLIB_MACRO_CAT_II(p, res) res
#endif
#ifndef RLIB_MAKE_UNIQUE_NAME
#define RLIB_MAKE_UNIQUE_NAME(base) RLIB_MACRO_CAT(base, __COUNTER__)
#endif

#endif
/******* ./stream.hpp ****************/
#include <rlib/require/cxx11>
#include <streambuf>
#include <iostream>
#include <rlib/sys/os.hpp>

namespace rlib {
        namespace impl {
            class NullStreamBuf : public std::streambuf
            {
            public:
                int overflow(int c) { return c; }
            };
            // stdc++ 17 removed
            extern NullStreamBuf null_streambuf;
        }

//#if RLIB_CXX_STD < 2017
```

```
	extern std::ostream null_stream;
//#else
//	inline std::ostream null_stream(&impl::null_streambuf);
//#endif
}
/******* ./functional.hpp ****************/
#ifndef RLIB_FUNCTIONAL_HPP
#define RLIB_FUNCTIONAL_HPP_

#include <rlib/require/cxx17>
#include <rlib/class_decorator.hpp>
#include <rlib/sys/os.hpp>

#include <type_traits>
#include <list>
#include <functional>
#include <chrono>

namespace rlib {
	namespace impl {
		template <typename Func, typename... Args>
		struct repeated_func {
			using return_type = typename std::invoke_result<Func, Args ...>::type;
			auto operator ()(size_t count, Func f, Args ... args) {
				for(size_t cter = 0; cter < count - 1; ++cter)
					f(std::forward<Args>(args) ...);
				return f(std::forward<Args>(args) ...);
			}
		};
		template <typename Func, typename... Args>
		struct repeated_func_return_list {
			using return_type = typename std::invoke_result<Func, Args ...>::type;
			auto operator ()(size_t count, Func f, Args ... args) {
				std::list<return_type> ret;
				for(size_t cter = 0; cter < count; ++cter)
					ret.push_back(std::move(f(std::forward<Args>(args) ...)));
				return std::move(ret);
			}
		};

	}
}

namespace rlib {
	template <class Func, typename... Args>
	constexpr static inline double timeof(Func && f, Args && ... args)
	{
		auto begin = std::chrono::high_resolution_clock::now();
		f(std::forward<Args>(args) ...);
		auto end = std::chrono::high_resolution_clock::now();
		return ::std::chrono::duration<double>(end - begin).count();
	}

	template <class Func, typename... Args>
	constexpr static inline auto repeat(size_t count, Func && f, Args && ... args)
	{
		// Unnecessary asserts for debugging.
		using return_type = typename std::invoke_result<Func, Args ...>::type;
		using return_type2 = typename std::invoke_result<typename impl::repeated_func<Func, Args ...>, size_t, Func, Args ...>::type;
		using return_type3 = decltype(impl::repeated_func<Func, Args ...>()(count, f, args ...));
		static_assert(std::is_same<return_type, return_type2>::value);
		static_assert(std::is_same<return_type, return_type3>::value);

		return std::bind(impl::repeated_func<Func, Args ...>(), count, std::forward<Func>(f), std::forward<Args>(args) ...);
	}
	template <class Func, typename... Args>
	constexpr static inline auto repeat_and_return_list(size_t count, Func f, Args... args)
	{
		return std::bind(impl::repeated_func_return_list<Func, Args ...>(), count, std::forward<Func>(f), std::forward<Args>(args) ...);
	}
}

namespace std {
#if RLIB_CXX_STD >= 2017
	class execution;
#endif
}

// functools here.
#if 0 // not finished
#include <algorithm>
namespace rlib {
	template <Iterable buffer_t>
	class wrappedIterable : public buffer_t {
	public:
		using buffer_t::buffer_t;
		using buffer_type = buffer_t;
		using value_type = buffer_t::value_type;
		using this_type = wrappedIterable<buffer_t>;
		wrappedIterable(const buffer_t &b) : buffer_t(b) {}
		wrappedIterable(buffer_t &&b) : buffer_t(std::forward<buffer_t>(b)) {}

#if RLIB_CXX_STD >= 2017
		// std::foreach wrapper
		this_type &map(std::execution &&policy, std::function<value_type(const value_type &)> mapper_func) {
			std::for_each(std::forward<std::execution>(policy), begin, end, [&mapper_func](value_type &v){v = mapper_func(v);});
		}
		this_type &map(std::execution &&policy, std::function<void(value_type &)> mapper_func) {
			std::for_each(std::forward<std::execution>(policy), begin, end, mapper_func);
		}


#endif
		// std::foreach wrapper
		this_type &map(std::function<value_type(const value_type &)> mapper_func) {
			std::for_each(begin, end, [&mapper_func](value_type &v){v = mapper_func(v);});
			return *this;
		}
		this_type &map(std::function<void(value_type &)> mapper_func) {
			std::for_each(begin, end, mapper_func);
			return *this;
		}

		this_type &filter(std::function<bool(const value_type &)> filter_func) {
			std::remove_if(begin, end, [&filter_func](const value_type &v) -> bool {return !filter_func(v);});
			return *this;
		}

		this_type &flat_map(std::function<buffer_type<value_type>(const value_type &)>) {

		}

	};
}
#endif

#endif
/******* ./scope_guard.hpp ****************/
/* Exception safe usage:
 *
 * reinforce_scope_begin(_gname, [](){do_sth();})
 * do_something();
 * reinforce_scope_end(_gname)
 *
 */

#ifndef R_SCOPE_GUARD
#define R_SCOPE_GUARD
```

```
#include <rlib/require/cxx11>
#include <functional>
#include <rlib/class_decorator.hpp>

namespace rlib {
	class scope_guard : private noncopyable
	{
	public:
		template<class Callable>
		scope_guard(Callable && undo_func) : f(std::forward<Callable>(undo_func)) {}

		scope_guard(scope_guard && other) : f(std::move(other.f)) {
			other.f = nullptr;
		}

		~scope_guard() {
			if(f) f(); // must not throw
		}

		void dismiss() noexcept {
			f = nullptr;
		}

		void force_call() noexcept {
			if(f) f();
			dismiss();
		}

	private:
		std::function<void()> f;
	};
}


#ifndef rlib_defer
#include <rlib/macro.hpp>
#define rlib_defer(callable) ::rlib::scope_guard RLIB_MAKE_UNIQUE_NAME(_guarder_id_) (callable)
#endif

#define RLIB_reinforce_scope_begin(guarderName, callable) scope_guard guarderName = callable; try{
#define RLIB_reinforce_scope_end(guarderName) } catch(...) { guarderName.force_call(); throw;}

/*
scope_guards scope_exit, scope_fail;

action1();
scope_exit += []{ cleanup1(); };
scope_fail += []{ rollback1(); };

action2();
scope_exit += []{ cleanup2(); };
scope_fail += []{ rollback2(); };

do_something();

scope_fail.dismiss();
*/

#include <deque>

namespace rlib {
	class scope_guards : private noncopyable
	{
	public:
		template<class Callable>
		scope_guards& operator += (Callable && undo_func) {
			fbuf.emplace_front(std::forward<Callable>(undo_func));
			return *this;
		}

		~scope_guards() {
			force_call();
		}

		void dismiss() noexcept {
			fbuf.clear();
		}

		void force_call() noexcept {
			for(auto &f : fbuf) f();
			dismiss();
		}
	private:
		std::deque<std::function<void(void)> > fbuf;
	};
}


#endif
/******* ./cmake/cmake.include ****************/
if ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "Clang")
	# using clang
elseif ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "GNU")
	set(rlib_CXX_FLAGS "-Wno-terminate")
elseif ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "Intel")
	add_definitions(-DRLIB_MINGW_DISABLE_TLS)
elseif ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "MSVC")
	# using Visual Studio C++
endif()

if(MSYS OR MINGW)
	add_definitions(-DRLIB_MINGW_DISABLE_TLS)
endif()


/******* ./cmake/CMakeLists.txt ****************/
# Use this file if you want to include rlib as a subdirectory

if ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "Clang")
	# using clang
elseif ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "GNU")
	set(rlib_CXX_FLAGS "-Wno-terminate")
elseif ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "Intel")
	add_definitions(-DRLIB_MINGW_DISABLE_TLS)
elseif ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "MSVC")
	# using Visual Studio C++
endif()

if(MSYS OR MINGW)
	add_definitions(-DRLIB_MINGW_DISABLE_TLS)
endif()

add_library(r STATIC ../libr.cc)
include_directories(..)

/******* ./cmake/rlib-config.cmake ****************/
# use this file if you want to install rlib

set(rlib_INCLUDE_DIRS ${PREFIX}/include)
set(rlib_LIBRARIES ${PREFIX}/lib/libr.a)

if ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "Clang")
	# using clang
elseif ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "GNU")
	set(rlib_CXX_FLAGS "-Wno-terminate")
elseif ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "Intel")
	add_definitions(-DRLIB_MINGW_DISABLE_TLS)
elseif ("${CMAKE_CXX_COMPILER_ID}" STREQUAL "MSVC")
	# using Visual Studio C++
```

```
endif()

if(MINGW)
	add_definitions(-DRLIB_MINGW_DISABLE_TLS)
endif()
/******* ./impl/traceable_list.hpp ****************/
#ifndef _RLIB_IMPL_TRACEABLE_LIST_HPP
#define _RLIB_IMPL_TRACEABLE_LIST_HPP 1

#include <list>
#include <stdexcept>
#include <utility>

namespace rlib {
	namespace impl {
		// A double linked list with ability to get iterator from data_t*. Designed for *_object_pool.
		// Only implement few necessary functionalibities.
		template<typename T, typename extra_info_t>
		class traceable_list {
			struct node {
				/*
				 * You may not manage data ownness here. Or you must carefully check if (node*)&data works.
				 */
				T data; // Able to get iterator from T*
				node *prev;
				node *next;
				extra_info_t extra_info; // bool flag. specially designed for object_pool.
				uint32_t magic = 0x19980427;
				template <typename... TConstructArgs>
				node(node *prev, node *next, const extra_info_t &extra_info, TConstructArgs... args)
					: data(std::forward<TConstructArgs>(args) ...), prev(prev), next(next), extra_info(extra_info)
				{}
			};

		public:
			class iterator : std::bidirectional_iterator_tag {
				friend class traceable_list;

			public:
				using pointer = T *;
				using reference = T &;
				explicit iterator(node *ptr) : ptr(ptr) {}

				explicit iterator(T *data_pointer) : ptr(reinterpret_cast<node *>(data_pointer)) {
					if (ptr->magic != 0x19980427)
						throw std::invalid_argument(
							"magic_num verification failed. invalid data_pointer passed or ruined memory?");
				}

				T &operator*() {
					// If this is an iterator to empty_list.begin(), then nullptr->data throws.
					return ptr->data;
				}

				T *operator->() {
					// If this is an iterator to empty_list.begin(), then nullptr->data throws.
					return &ptr->data;
				}

				extra_info_t &get_extra_info() {
					return ptr->extra_info;
				}

				const T &operator*() const {
					// If this is an iterator to empty_list.begin(), then nullptr->data throws.
					return ptr->data;
				}

				const T *operator->() const {
					// If this is an iterator to empty_list.begin(), then nullptr->data throws.
					return &ptr->data;
				}

				const extra_info_t &get_extra_info() const {
					return ptr->extra_info;
				}

				iterator &operator++() {
					ptr = ptr->next;
					return *this;
				}

				const iterator operator++(int) {
					iterator backup(ptr);
					operator++();
					return std::move(backup);
				}

				iterator &operator--() {
					if (!ptr && _impl_tail)
						ptr = _impl_tail;
					else
						ptr = ptr->prev;
					return *this;
				}

				const iterator operator--(int) {
					iterator backup(ptr);
					operator--();
					return std::move(backup);
				}

				bool operator==(const iterator &another) const {
					return ptr == another.ptr;
				}

				bool operator!=(const iterator &another) const {
					return !operator==(another);
				}

			private:
				node *ptr;

				iterator(node *ptr, node *tail) : ptr(ptr), _impl_tail(tail) {}

				node *_impl_tail = nullptr; // If this iter is created by begin() or end(), it must set this ptr to support end().operator--(), to indicate that this
iterator is not invalid.
			};

			~traceable_list() {
				for (auto iter = begin(); iter != end();) {
					auto to_release = iter++;
					delete to_release.ptr;
				}
			}

			iterator begin() {
				return iterator(head, tail);
			}

			iterator end() {
				return iterator((node *) nullptr, tail);
			}

			template <typename... ConstructArgs>
			void emplace_one(const iterator &where, const extra_info_t &extra_info, ConstructArgs... constructArgs) {
				auto new_node = new node(nullptr, where.ptr, extra_info, std::forward<ConstructArgs>(constructArgs) ...);
				++m_size;
				if (!head) {
```

```
                tail = head = new_node;
                return;
            };
            auto ptr = where.ptr;
            if (!ptr) {
                // is end();
                tail->next = new_node;
                new_node->prev = tail;

                tail = new_node;
                return;
            }
            auto left = ptr->prev, right = ptr;
            new_node->prev = right->prev;
            if (left) left->next = new_node;
            right->prev = new_node;

            if (head == ptr)
                head = new_node;
        }

        void push_one(const iterator &where, T &&data, const extra_info_t &extra_info) {
            emplace_one(where, extra_info, std::forward<T>(data));
        }

        void push_one(const iterator &where, const T &data, const extra_info_t &extra_info) {
            T _data(data);
            push_one(where, std::move(_data), extra_info);
        }

        void push_back(T &&data, const extra_info_t &extra_info) {
            push_one(end(), std::forward<T>(data), extra_info);
        }

        void push_back(const T &data, const extra_info_t &extra_info) {
            push_one(end(), std::forward<T>(data), extra_info);
        }

        void push_front(T &&data, const extra_info_t &extra_info) {
            push_one(begin(), std::forward<T>(data), extra_info);
        }

        void push_front(const T &data, const extra_info_t &extra_info) {
            push_one(begin(), std::forward<T>(data), extra_info);
        }

        void pop_one(const iterator &which) {
            if (!head)
                throw std::invalid_argument("nothing to pop.");
            auto ptr = which.ptr;
            if (!ptr) {
                // end()
                throw std::invalid_argument("you may not pop end().");
            }
            auto left = ptr->prev, right = ptr->next;

            if (left) left->next = right;
            if (right) right->prev = left;
            if (head == ptr)
                head = right;
            if (tail == ptr)
                tail = left;
            --m_size;
            delete which.ptr;
        }

        void pop_front() {
            pop_one(begin());
        }

        void pop_back() {
            pop_one(--end());
        }

        void pop_some(const iterator &from, const iterator &to) {
            for (auto iter = from; iter != to;) {
                auto to_pop = iter++;
                pop_one(to_pop);
            }
        }

        size_t size() {
            return m_size;
        }

    private:
        node *head = nullptr;
        node *tail = nullptr;
        size_t m_size = 0;
    };
  }
}
#endif // _RLIB_IMPL_TRACEABLE_LIST_HPP
/******* ./c-with-class.h ****************/
#ifndef RLIB_CWITHCLASS_H_
#define RLIB_CWITHCLASS_H_

//TODO: clean namespace.
//TODO: use macro to type class_name only once.
//#error c_with_class not completed yet

#ifdef __cplusplus
#error You should not use c-with-class.h in real C++.
#endif

#define RCPP_NEW(type,name,constructor_arg) struct type name __attribute__((cleanup(type##_rcpp_destructor))); type##_rcpp_constructor(&name,constructor_arg)
#define RCPP_CALL(i_objectname,i_funcname, ...) i_objectname.i_funcname(&i_objectname, ##__VA_ARGS__) //ONLY static public function can be called directly!!!
#define RCPP_PCALL(p_objectname,i_funcname, ...) p_objectname->i_funcname(p_objectname, ##__VA_ARGS__)

#define RCPP_CLASS_DECL(class_name) struct class_name;
#define RCPP_CLASS_METHOD_DECL_1(class_name, method_name, return_type, ...) typedef return_type (*class_name##method_name##_rcpp_t)(struct class_name *this, ##__VA_ARGS__); //VAARGS is `int arg1, float arg2, ...`
#define RCPP_CLASS_BEGIN(class_name) struct class_name {
#define RCPP_CLASS_METHOD_DECL_2(class_name, method_name) RCPP_CLASS_MEMBER_DECL(class_name##method_name##_rcpp_t, method_name)
#define RCPP_CLASS_MEMBER_DECL(type, name) type name;
#define RCPP_CLASS_END() };
#define RCPP_CLASS_METHOD_IMPL(class_name, method_name, return_type, ...) return_type class_name##method_name##_rcpp_impl(struct class_name *this, ##__VA_ARGS__) //VAARGS is `int arg1, float arg2, ...`
#define RCPP_CLASS_CONSTRUCTOR_IMPL(class_name) void class_name##_rcpp_constructor(struct class_name *this, void *arg) //TODO: Register all methods.
#define RCPP_CLASS_METHOD_REGISTER(class_name, method_name) this->method_name = &class_name##method_name##_rcpp_impl;
#define RCPP_CLASS_DESTRUCTOR_IMPL(class_name) void class_name##_rcpp_destructor(struct class_name *this)

#endif
/******* ./log.hpp ****************/
#ifndef RLIB_LOG_HPP
#define RLIB_LOG_HPP_ 1

#include <string>
#include <fstream>
#include <list>
#include <climits>
#include <rlib/sys/os.hpp>
#include <rlib/stdio.hpp>
#include <rlib/sys/time.hpp>
#include <rlib/class_decorator.hpp>
```

```
// currently disable this error-prone shit.
#define RLIB_IMPL_ENABLE_LOGGER_FROM_FD 0

#ifndef RLIB_IMPL_ENABLE_LOGGER_FROM_FD
#include <rlib/sys7fd.hpp>
#if RLIB_OS_ID != OS_UNKNOWN
#     if RLIB_COMPILER_ID == CC_GCC
#          include <ext/stdio_filebuf.h>
#          define RLIB_IMPL_ENABLE_LOGGER_FROM_FD 1
#     elif RLIB_COMPILER_ID == CC_MSVC
#          define RLIB_IMPL_ENABLE_LOGGER_FROM_FD 1
#     endif
#endif
#endif

#ifdef ERROR
#pragma message (": warning MSVC_Macro_pollution: You MUST NOT define the macro `ERROR`. I've undefined it here.")
#undef ERROR
#endif

namespace rlib {
     using namespace rlib::literals;

     // Allow extension.
     enum class log_level_t : int { FATAL = 1, ERROR, WARNING, INFO, VERBOSE, DEBUG };
     namespace impl {
//#if RLIB_CXX_STD < 2017
          extern int max_predefined_log_level;
//#else
//          inline int max_predefined_log_level = (int)log_level_t::DEBUG;
//#endif
     }
     /*
     How to update log_level_t:
          Extend `enum log_level_t ...`
          Modify libr.cc:`max_predefined_log_level ...`
          Add an RLIB_IMPL_MACRO_LOG_ADD_SHORTHAND
          Append logger::predefined_log_level_name
     */

     class logger : rlib::noncopyable {
     public:
          logger() = delete;
          logger(std::ostream &stream) : pstream(&stream) {}
          logger(const std::string &file_name) : pstream(new std::ofstream(file_name, std::ios::out)),
               must_delete_stream_as_ofstream(true) {
                    if(!dynamic_cast<std::ofstream &>(*pstream))
                         throw std::runtime_error("Failed to open file {}."_format(file_name));
               }
          logger(logger &&another) : pstream(another.pstream),
               custom_log_level_names(std::move(another.custom_log_level_names)),
               log_level(another.log_level),
               must_delete_stream_as_ofstream(another.must_delete_stream_as_ofstream),
               enable_flush(another.enable_flush)
               {another.must_delete_stream_as_ofstream = false;}
          ~logger() {
               if(must_delete_stream_as_ofstream)
                    delete dynamic_cast<std::ofstream *>(pstream);
          }

          logger &operator=(logger &&another) {
               pstream = another.pstream;
               enable_flush = another.enable_flush;
               must_delete_stream_as_ofstream = another.must_delete_stream_as_ofstream;
               log_level = another.log_level;
               custom_log_level_names = std::move(another.custom_log_level_names);
               another.must_delete_stream_as_ofstream = false;
               return *this;
          }

#if RLIB_IMPL_ENABLE_LOGGER_FROM_FD == 1
#if RLIB_OS_ID != OS_UNKNOWN
          logger(fd file_descriptor_or_handle)
#     if RLIB_COMPILER_ID == CC_GCC
          : _gcc_filebuf(file_descriptor_or_handle, std::ios::out), _gcc_real_stream(&_gcc_filebuf),
               stream(_gcc_real_stream) {}
#     elif RLIB_COMPILER_ID == CC_MSVC
          : _msvc_real_stream(::_fdopen(file_descriptor_or_handle, "w")),
               stream(_msvc_real_stream) {}
#     endif
#endif
#endif

          void set_log_level(log_level_t max_level) {
               this->log_level = max_level;
          }
          void set_flush(bool enable_flush) noexcept {
               this->enable_flush = enable_flush;
          }
          template <typename ... Args>
          void log(log_level_t level, const std::string &info, Args ... extra_args) const {
               if(is_predefined_log_level(level) && level > this->log_level)
                    return;
               (*pstream)          <<          "[{}|{}]          {}"_format(get_current_time_str(),          log_level_name(level),          impl::format_string(info,
std::forward<Args>(extra_args) ...)) << RLIB_IMPL_ENDLINE;
               if(enable_flush)
                    pstream->flush();
          }
          // Warning: this method is not thread-safe.
          log_level_t register_log_level(const std::string &name) {
               if(impl::max_predefined_log_level == INT_MAX)
                    throw std::overflow_error("At most {}(INT_MAX) log_level is allowed."_format(INT_MAX));
               ++ impl::max_predefined_log_level;
               log_level_t new_level = (log_level_t)impl::max_predefined_log_level;
               custom_log_level_names.push_back({new_level, name});
               return new_level;
          }

#define RLIB_IMPL_MACRO_LOG_ADD_SHORTHAND(_name, _enum_name) template <typename ... Args> void _name(const std::string &info, Args ...
extra) const {\
               log(log_level_t::_enum_name, info, std::forward<Args>(extra) ...); }

          RLIB_IMPL_MACRO_LOG_ADD_SHORTHAND(fatal, FATAL)
          RLIB_IMPL_MACRO_LOG_ADD_SHORTHAND(error, ERROR)
          RLIB_IMPL_MACRO_LOG_ADD_SHORTHAND(warning, WARNING)
          RLIB_IMPL_MACRO_LOG_ADD_SHORTHAND(info, INFO)
          RLIB_IMPL_MACRO_LOG_ADD_SHORTHAND(verbose, VERBOSE)
          RLIB_IMPL_MACRO_LOG_ADD_SHORTHAND(debug, DEBUG)

#undef RLIB_IMPL_MACRO_LOG_ADD_SHORTHAND

     private:
          static constexpr const char * predefined_log_level_name(log_level_t level) noexcept {
               switch(level) {
               case log_level_t::FATAL:
                    return "FATAL";
               case log_level_t::ERROR:
                    return "ERROR";
               case log_level_t::WARNING:
                    return "WARNING";
               case log_level_t::INFO:
                    return "INFO";
               case log_level_t::VERBOSE:
                    return "VERBOSE";
               case log_level_t::DEBUG:
                    return "DEBUG";
               default:
                    return "";
```

```
            }
        }
        static constexpr bool is_predefined_log_level(log_level_t level) noexcept {
            return predefined_log_level_name(level)[0] != '\0';
        }
        std::string log_level_name(log_level_t level) const noexcept {
            std::string name = predefined_log_level_name(level);
            if(!name.empty())
                return name;
            for(const auto &level_and_name : custom_log_level_names) {
                if(level == level_and_name.first) {
                    name = level_and_name.second;
                    break;
                }
            }
            if(!name.empty())
                return name;
            name = "LEVEL-";
            name += std::to_string((int)level);
            return name;
        }

        std::ostream *pstream;

        std::list<std::pair<log_level_t, std::string> > custom_log_level_names;
        log_level_t log_level = log_level_t::INFO; // `Ignore` deadline.

        bool must_delete_stream_as_ofstream = false;
        bool enable_flush = true;
#if RLIB_IMPL_ENABLE_LOGGER_FROM_FD == 1
#     if RLIB_COMPILER_ID == CC_GCC
        __gnu_cxx::stdio_filebuf<char> _gcc_filebuf;
        std::ostream _gcc_real_stream;
#     elif RLIB_COMPILER_ID == CC_MSVC
        std::ofstream _msvc_real_stream;
#     endif
#endif
    };
}

#endif
/******* ./require/cxx17 ****************/
#ifndef R_CXX17_REQUIRED
#define R_CXX17_REQUIRED

#include <rlib/sys/os.hpp>

#if RLIB_CXX_STD <= 2014
#error This file requires compiler and library support \
for the ISO C++ 2017 standard. This support must be enabled \
with the -std=c++17, -std=gnu++17, -std=c++1z, -std=gnu++1z \
compiler options.
#endif

#endif
/******* ./require/cxx14 ****************/
#ifndef R_CXX14_REQUIRED
#define R_CXX14_REQUIRED

#include <rlib/sys/os.hpp>

#if RLIB_CXX_STD <= 2011
#error This file requires compiler and library support \
for the ISO C++ 2014 standard. This support must be enabled \
with the -std=c++14 or -std=gnu++14 compiler options.
#endif

#endif
/******* ./require/cxx11 ****************/
#ifndef R_CXX11_REQUIRED
#define R_CXX11_REQUIRED

#include <rlib/sys/os.hpp>

#if RLIB_CXX_STD <= 1997
#error This file requires compiler and library support \
for the ISO C++ 2011 standard. This support must be enabled \
with the -std=c++11 or -std=gnu++11 compiler options.
#endif

#endif
```