C++ Exercises
Set 7

Author(s): Olivier Gelling, Leon Lan, Mohammad Habibi

13:41

October 21, 2025

49

Exercise 49:

Time a program that has the failbit set as it sends a debug message into an outstream object.

Since the outcome wasn't very distinct yet when using argv[1] = 100'000'000 we elected to use 1'000'000'000 instead:

Without the if clause:
49$ time noIf 1000000000

```
real    0m29,766s
user    0m29,759s
sys     0m0,004s
```

With:
49$ time yeIf 1000000000

```
real    0m0,478s
user    0m0,473s
sys     0m0,004s
```

Clearly the if clause is guarding a bunch of extra evaluations from being done. The case where the stream itself is evaluating the failbit takes almost sixty times as long. It seems that when the ostream object itself evaluates whether the failbit is set it takes much longer than the if statement does.

A fitting rule of thumb here would probably be to not let a stream do computations that can easily be checked elsewhere. Or more simply: don't rely on the stream object to evaluate and suppress output if you know there will be things to suppress.

Listing 1: `main.ih`

```
#include <iostream>
#include <iomanip>

#ifndef SPCH_
using namespace std;
#endif
```

Listing 2: `main.cc`

```
#include "main.ih"

int main(int argc, char *argv[])
{
    if (argc == 1)
        return 1;
```

```
    ostream out(cout.rdbuf());

    out.setstate(ios::failbit);

    size_t const count = stoull(argv[1]);
    for (size_t index = 0; index != count; ++index)
    {
        //if (out.good())
        out << "Nr. of command line arguments " << argc << '\n';
    }
}
```

50

Exercise 50:

Use the given program and fix it such that it properly writes to out2 as well.

We copy this program and amend it to work properly. Running it initially shows that only the first file is being created, opened, and written, while the second is not. We can even add a debug statement showing that the second file has failed to open:

```
    fstream out2{ "./tmp/out2" };
    if (!out2.is_open()
        cout << "Error\n";
```

This will indeed print "Error" to our screen. The reason for this is that this second stream, out2, is not an *o*fstream, but just an *f*stream. As such it is not sure what to do (whether to send or receive this text). We can amend this by either making this an ofstream:

```
ofstream out2{ "/tmp/out2" }
```

or by adding an extra parameter to this constructor:

```
fstream out2{ "/tmp/out2", ios::out };
```

such that it now knows that it is supposed to be an ostream!

Listing 3: main.cc

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

void hello(ostream &out)
{
    out << "hello world\n";
}

int main()
{
    ofstream out1{ "/tmp/out1" };
    if (out1.is_open())
        cout << "opened out1\n";
    hello(out1);

    fstream out2{ "/tmp/out2", ios::out };
    if (out2.is_open())
        cout << "opened out2\n";
    hello(out2);
}
```

51

Exercise 51:

Write a simple program that can clean up capitalised letters from email addresses.

Our program has convert() check if the file could be opened, and shows an error message and returns if not. Doing this in a separate function seems excessive for the aim of the exercise but it does mean the convert method becomes slightly lengthy. We elect to keep it this way as it is still easy to uderstand.

We note that even though getline strips the currently read line of its '\n', our lines still nicely align beneath each other. this due to the fact that the newline character that was still there from the original line is not being overwritten. If we had added a single extra character into the line it would overwrite the newline and cause the next one to continue right after it. Since we only change contents that are already in the string we do not have to worry about this though.

Listing 4: main.ih

```
#include <string>
#include <fstream>
#include <cctype>          // for lowercase
#include <iostream>        // for debugging


void convert(char *filename);
std::string lowercase(std::string &input);

using namespace std;
```

Listing 5: convert.cc

```
#include "main.ih"

void convert(char *filename)
{
    fstream file(filename, ios::in | ios::out);
    if (!file)                                  // test file presence
    {
        cerr << "Unable to open file\n";
        return;
    }

    size_t setPos = file.tellg();               // set initial position
    string storeData;
    while (getline(file, storeData))
    {
        if (storeData.find("email:") != string::npos)
        {
            file.seekp(setPos);                 // back to start of line
            file << lowercase(storeData);       // write line in lowercase
        }
        setPos = file.tellg();                  // store next line position
    }
}
```

Listing 6: lowercase.cc

```
#include "main.ih"

string lowercase(string &input)
{
    for (char &ch : input)
       ch = tolower(ch);

    return input;
}
```

Listing 7: main.cc

```
#include "main.ih"

int main(int argc, char *argv[])
{
    if (argc == 1)                  // verify input
        return 1;

    convert(argv[1]);               //  verifies file and corrects UC
}
```

52

Exercise 52:

Write a manipulator that uses put_time and outputs the current time and date like asctime does, but without the newline.

We do exactly this, showing off our understanding of the format string by recreating the easy direct option (%c) by hand with all the available specifiers.

Listing 8: main.cc

```
#include <iostream>
#include <iomanip>
#include <chrono>

using namespace std;

ostream &now(ostream &os)                   // free function, no ns needed
{
    time_t time = chrono::system_clock::to_time_t(
                                            chrono::system_clock{}.now() );

    char const *asctimeformat = "%a %b %e %H:%M:%S %Y";
    // char const *format = "%c"; // the easy way

    return os << put_time(localtime(&time), asctimeformat);
}

int main()
{
    cout << now << '\n';
}
```

53

Exercise 53:

Format the output of a single concatenated cout statement to show a multitude of options.

We do this exactly. We have left out the push operators where possible. We considered adding " | ios::showpoint" to the resetflags(ios::fixed) function's argument, as showpoint persists after being set. Since it would not show any change to value's output we have elected to keep it out, but do understand that it means not all special formatting rules have been undone.

```
std display:          '          12.04'
left aligned:         '12.04          '
right aligned:        '          12.04'
3 digits:             '           12.0'
4 digit fraction:     '       12.0400'
std display:          '          12.04'
```

Listing 9: main.cc

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    double const value = 12.04;

    cout << "std display:      '" << setw(15) << value << "'\n"
         "left aligned:     '" << setw(15) << left << value << "'\n"
         "right aligned:    '" << setw(15) << right << value << "'\n"
         "3 digits:         '" << setw(15) << defaultfloat << showpoint
             << setprecision(3) << value << "'\n"
         "4-digit fraction: '" << setw(15) << fixed << setprecision(4)
             << value << "'\n"
         "std display:      '" << setw(15) << resetiosflags(ios::fixed)
             << value << "'\n";
}
```

8

54

Exercise 54:

Make a program that can read through a file and can interactively change and output its lines.

We do this exactly, choosing to use an array of pointers to member functions inside process(), as we have learned how to use them and it seemed a nice fit! We do use a switch inside ask, as that makes plenty of sense there. We tried to keep is compact and readable, using a fallthrough for actions that could be combined, but that doesn't really save much space anyways. We keep it as it is part of the covered materials and a nice little detail (I learned abot it so why not use it!).

When an incorrect command is given to the ask() function, it is interpreted as a simple 'n'. We considered adding functionality that either resets the d_location variable so that user error can be amended, but do not think it necessary to implement it at this time. It could be done without much issue by adding a
d_location = d_target.length();
statement to the default case.

We also note that the exercise names an istream, where we used an ifstream. Not exactly the same but they are both istreams of one sort or another! It seemed proper to use the file version for this.

We also note that in our implementation the changeAll() function isn'tneeded, as we can simply call modify() in our array of pointers to functions. Adding it to uor implementation would make it merely an alias of modify, executing it and then nothing else. As such we have elected to slightly deviate from the examinators' implementation and hope that that is ok with the framework of ours.

We add a short passage from a favourite book of mine and a version amended by this program (target: He, replacement: Paul Muad'dib)

"There was a man so wise,
He jumped into
A sandy place
And burnt out both his eyes!
And when he knew his eyes were gone,
He offered no complaint.
He summoned up a vision
And made himself a saint.
-Children's Verse

;-) Nice!

from History of Muad'dib"

"There was a man so wise,
Paul Muad'dib jumped into
A sandy place
And burnt out both his eyes!
And when he knew his eyes were gone,
Paul Muad'dib offered no complaint.
Paul Muad'dib summoned up a vision
And made himself a saint.
Children's Verse
from History of Muad'dib"

Listing 10: enum/enum.h

```
#ifndef INCLUDED_ENUM_H_
#define INCLUDED_ENUM_H_

enum Action
{
    ASK,
    CHANGE_ALL,
    NO_CHANGES,
};


#endif
```

OK, but consider defining it in Fch, since it's not used elsewhere

Listing 11: fch/fch.h

```
#ifndef INCLUDED_FCH_
#define INCLUDED_FCH_

#include "../enum/enum.h"
#include <iosfwd>
#include <fstream>
#include <string>

class Fch
{
    std::fstream    d_input;
    std::string     d_target;
    std::string     d_replacement;
    std::string     d_line;
    size_t          d_location = 0;

    Action d_action = ASK;

    static void (Fch::*s_action[])();

    public:
        Fch(char const *fname);
        int run();                  // reads all lines from d_input
                                    // processes changes
    private:
        void ask();
        //void changeAll();
        bool findTarget();
        void modify();
        bool openInput(char const *fname);      // Called by constructor
        void process();
        char request() const;
        bool requestedN();
        void searchReplace();
        void showModification() const;
        void insertLine() const;
};

#endif
```

Listing 12: fch/fch.ih

```
#include "fch.h"

#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>

using namespace std;
```

Listing 13: fch/s_action.cc

```
#include "fch.ih"

    // by searchreplace.cc

void (Fch::*Fch::s_action[])()
=
{
    &Fch::ask,
    &Fch::modify
    // &Fch::changeAll
};
```

Listing 14: fch/fch1.cc

```
#include "fch.ih"

    // by main.cc

Fch::Fch(char const *fname)
{
    if (openInput(fname))
        searchReplace();
}
```

Listing 15: fch/run.cc

```
#include "fch.ih"

    // by main.cc

int Fch::run()
{
    if (!d_input.is_open())
        return 1;

    while(getline(d_input, d_line))
        process();

    return 0;
}
```

Listing 16: fch/ask.cc

```
#include "fch.ih"

    // by process.cc via s_action.cc

void Fch::ask()
{
    showModification();
    switch (request())                          // requests change decision
    {
        case 'Y':
            d_action = CHANGE_ALL;
        [[fallthrough]];
        case 'y':
```

```
            modify();
        break;
        case 'N':
            d_action = NO_CHANGES;
        [[fallthrough]];
        case 'n':
        break;
        default:
            cerr << "incorrect command\n";
    }
    d_location += d_target.size();
}   // could *not* advance location on wrong input?
```

Listing 17: fch/changeall.cc

```
#include "fch.ih"

    // by

//void Fch::changeAll()
//{                                   // This is just an alias. We could have
//    modify();                       // called modify directly, but it's in
//}                                   // the header so I guess we're using it
```

Listing 18: fch/findtarget.cc

```
#include "fch.ih"

    // by process.cc

bool Fch::findTarget()
{
    d_location = d_line.find(d_target, d_location);

    return d_location != string::npos;
}
```

Listing 19: fch/modify.cc

```
#include "fch.ih"

    // by

void Fch::modify()
{
    string const prefix = d_line.substr(0, d_location);
    string const postfix = d_line.substr(d_location + d_target.length(),
                                         d_line.length());

    d_line = prefix + d_replacement + postfix;
}
```

Listing 20: fch/openinput.cc

```
#include "fch.ih"

    // by fch1.cc

bool Fch::openInput(char const *fname)
{
    d_input.open(fname, ios::in);
    return d_input.is_open();
}
```

Listing 21: fch/process.cc

```
#include "fch.ih"

    // by
```

```
void Fch::process()
{
    d_location = 0;

    while (!requestedN() && findTarget())
        (this->*s_action[d_action])();

    insertLine();
}
```

Listing 22: fch/request.cc

```
#include "fch.ih"

    // by

char Fch::request() const
{
    char command;
    cerr << "change [ynYN]? ";
    cin >> command;
    return command;
}
```

Listing 23: fch/requestedn.cc

```
#include "fch.ih"

    // by

bool Fch::requestedN()
{
    return d_action == NO_CHANGES;
}
```

Listing 24: fch/searchreplace.cc

```
#include "fch.ih"

    // by fch1.cc

void Fch::searchReplace()
{
    cerr << "target: ";
    getline(cin, d_target);
    cerr << "replacement: ";
    getline(cin, d_replacement);
}
```

Listing 25: fch/showmodification.cc

```
#include "fch.ih"

    // by

void Fch::showModification() const
{
    string const underline(d_target.length(), '^');

    cerr << d_line << '\n'
        << setw(d_location) << "" << underline << '\n';
}
```

Listing 26: fch/insertline.cc

```
#include "fch.ih"
```

```
    // by
void Fch::insertLine() const
{
    cout << d_line << '\n';
}
```

Listing 27: main.ih

```
#include "fch/fch.h"

using namespace std;
```

Listing 28: main.cc

```
#include "main.ih"

int main(int argc, char *argv[])
{
    if (argc == 1)
        return 1;

    Fch file(argv[1]);

    return file.run();
}
```

BS!

55

Exercise 55:

Write the Lock class and a program that locks a file and appends a line of text to it.

We do exactly this, choosing not to refactor subroutines like copying the string into a char *, as the exercise specifies that no more members are needed than the ones mentioned. We had some trouble juggling the conversions from char *s to strings and back and forth, and as such hope this works. There might be room for improvements and optimisation by either leaning more on char * and less on strings, but that is a matter for another time.

We note that checking for memory leaks with valgrind makes the program fail to open the lock file! This might be due to the way valgrind runs the input commands? Maybe it runs from a different directory, as we specifically see our validate function output the debug message that there was an issue with the lock file itself. We do not see any memory leaks luckily!

// Not in our implementation. Interesting...

We have chosen to remove the debug messages, but the "lock successful" and "lock failed" will remain as part of the program's feedback.

We have left the option to add a second commandline argument to select the directory in which the .lck file will be stored out. Adding it in would require some extra shenanigans with memory allocation that seems irrelevant to this exercise's scope, but the code works and can be un commented for testing!

As for writing a library:

We make Lock's library by first compiling all the source files into object files, we can do this with:
g++ -std=c++26 -Wall -Werror -c *.cc

Or by using make or icmbuild library. We can then make a library from this by running:
ar rcs liblock.a */*/*.o

(since our object files were placed in tmp/o/). We can now either link directly to this library, or place it somewhere and link to it from there.

Our class is made within the lock/ directory, separating it from the main program. We could install the library in /usr/lib, and the header in /usr/include, but since it will be growing over time that is likely a bit premature. It would greatly simplify including and linking it though. Instead, for now, we will place it somewhere else high up the file structure. My projects are usually in a directory like so:
~/Projects/Olivier/FBC++/set7/55

We will make a new directory:
~/orgutility
which has subdirectories lib/ and include/. We will store our headers and libraries inside here for now, and move them over to /usr/ when they are ready!

```
orgutility$ tree
.
        include
                lock.h
        lib
            liborgutils.a
```

We include the header in angled brackets inside main.ih, and then call the compiler to link the project like so:

```
g++ -I$HOME/orgutility/include main.cc -L$HOME/orgutility/lib -lorgutility -o
lock.out
```

This compiles on our end and gives us the program exactly as such. We will turn the angled brackets back into double quotes to satisfy the mailhandler though, as it likely won't pick up on our lock.h otherwise.

We will henceforth expand on this library.

Listing 29: lock/lock.h

```
#ifndef INCLUDED_LOCK_
#define INCLUDED_LOCK_

//#include <iosfwd>
#include <string>
#include <filesystem>

class Lock
{
    int d_filedesc = -1;
    public:
        Lock(std::string const &path);
        Lock(std::string const &path, std::string lockDir);
        ~Lock();

        int open(std::filesystem::path path);
        bool valid();

    private:
        static std::string stringName(std::string const &path1,
                                      char *(*name)(char* pathPH));
        std::string lockPath(std::string const &path,
                             std::string const &lockDir); // const ref?
};

#endif
```

Listing 30: lock/lock.ih

```
#include "lock.h"
#include <libgen.h>          // for basename/dirname
#include <fstream>
```

```
#include <fcntl.h>          // for open()
#include <sys/file.h>       // for flock()
#include <unistd.h>         // for close()
#include <iostream>
#include <cstring>          // for strcpy

using namespace std;
```

Listing 31: lock/lock1.cc

```
#include "lock.ih"

    // by

Lock::Lock(string const &path)
:
    Lock(path, stringName(path, dirname))
{}
```

Listing 32: lock/lock2.cc

```
#include "lock.ih"

    // by

Lock::Lock(string const &path, string lockDir)
{
    string fullName = lockPath(path, lockDir);
    cerr << fullName << '\n';
    filesystem::path pathObj(fullName);

    d_filedesc = open(pathObj);

    if(valid())
        cerr << "lock successful\n";
    else
        cerr << "lock failed\n";
}
```

Listing 33: lock/lock3.cc

```
#include "lock.ih"

    // by

Lock::~Lock()
{
    if (valid())
        ::close(d_filedesc);
}
```

Listing 34: lock/open.cc

```
#include "lock.ih"

    // by

int Lock::open(filesystem::path path)
{
    string pathStr = path.string();
    char const *pathC = pathStr.c_str();
    int filedesc;

    if (filesystem::exists(path))
        filedesc = ::open(pathC, O_RDWR);
    else
        filedesc = ::open(pathC, O_CREAT | O_TRUNC | O_RDWR, 0600);

    return filedesc;
}
```

Listing 35: lock/valid.cc

```
#include "lock.ih"

    // by

bool Lock::valid()
{
    if (d_filedesc == -1)
        return false;

    if (flock(d_filedesc, LOCK_EX | LOCK_NB) == -1)
        return false;

    return true;
}
```

13

CTR

Listing 36: lock/stringname.cc

```
#include "lock.ih"

    // by

string Lock::stringName(string const &path1, char *(*name)(char *pathPH))
{
    size_t length = path1.length();
    char *pathHandle = new char[length + 1];

    for (size_t index = 0; index != length; ++index)
        pathHandle[index] = path1[index];   // No refactoring on purpose
    pathHandle[length] = '\0';              // see info.txt for reason

    char *ret = name(pathHandle);           // get ret val
    string result(ret);

    delete [] pathHandle;
    return result;
}
```

VC

// don't make life hard on yourself; avoid allocations if 'string' can also be used

Listing 37: lock/lockpath.cc

```
#include "lock.ih"

    // by

string Lock::lockPath(string const &path, string const &lockDir)
{
    string baseName = stringName(path, basename);
    string fullName = lockDir + "/" + baseName + ".lck";

    return fullName;
}
```

CTR

Listing 38: main.ih

```
#include "lock/lock.h"
//#include <lock.h>      // when compiling with a static library and header
                         // placed in our orgutility directory
#include <fstream>
#include <iostream>

using namespace std;
```

Listing 39: main.cc

```
#include "main.ih"

int main(int argc, char *argv[])
```

BS.

```
    if (argc == 1)
        return 1;

    // For testing manually handing the .lck file's directory:
    // string directory = argv[2];
    // Lock fileLock(argv[1], directory);

    Lock fileLock(argv[1]);

    if (fileLock.valid())
    {
        fstream writeFile(argv[1], ios::app);

        string appendText;
        cerr << "? ";

        getline(cin, appendText);
        writeFile << appendText << '\n';
    }
    else
    {
        cerr << "failed to open file\n";
        return 1;
    }
}
```

14

swap, and you don't need the 'else'

56

Exercise 56:

Write a program that can output the process accounting log in readable form, either in full or just what wasn't exited properly

We do exactly this, adding some notes about what is going on:

We found that our command names are 16 bits from the ACCT.COMM definition inside the acct.h header. As such process names longer than that are simply truncated. Since it is thus already defined inside our program we use it to read the max length, and then also cut off any possible trailing nulls to keep things nice and tidy.

After some researching (and a bit of arguing with copilot, which we don't actually trust that well on matters like these) we noticed that the signal codes from signum.h shown in the exercise go up to 31, and we can have our computer show us the rest by running:
kill -l, which gives me:

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7
42) SIGRTMIN+8  43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11
46) SIGRTMIN+12 47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15
50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11
54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2  63) SIGRTMAX-1
64) SIGRTMAX
```

A bunch of the error signals shown in the exercise's example go way higher than that. Copilot might be onto something when it tells us that the ac_exitcode member of the acct_v3 struct stores both a SIG type kill code, and an exit code that simply represents how the program ended. If the first 8 bits are stripped off of the ac_exitcode member and used as our d_signal to then run the program we get a nice set of programs that actually ended in a

faulty way, testig this by manually killing and terminating processes inside a terminal, like:

sleep 1000 &
and then executing
kill <PID>
where PID is the process id that shows up after running the sleep command. We can add a -9 or -15 after kill as well to specify SIGKILL or SIGTERM. We then indeed see:
'sleep' KILL
'sleep' TERM

We made the program print debug info, like the (interpreted) value of ac_exitcode, and it seems to align with clipping the first 8 bits off for the signal, and the next 8 for possible exitcodes if a process was not killed by SIG. We have decided, however, to stick with the exercise's convention, and only use the ac_exitcode member in full, as the exercise shows codes that do not simply fit into 8 bits. We thus comment out the second int datamember and any lines that are used for debugging and filtering/showing the exitcode and signal separately. Learning about this process and the data structure was rather interesting!

We do run into an issue that shows we might be doing something wrong: when a program terminates due to a segfault we should see an exitcode 11, for SIGSEGV, yet when we purposely make a program segfault our stored error code is 139, with a hex value: 8b. This does not seem to align properly with the SIGnals and exitcode, and instead seems to represent the shell's exit status is 139, which we can see by executing
echo $?
right after the segfaulting program runs. Apparently in this case the process accounting is storing the shell's exit status rather than the segfault code? We aren't entirely sure why this is.

Listing 40: procacc/procacc.h

```
#ifndef INCLUDED_PROCACC_
#define INCLUDED_PROCACC_

#include <fstream>
#include <string>
#include <linux/acct.h>

class ProcAcc
{
    std::ifstream d_file;
    acct_v3 d_record;
    std::string d_processName;

    //int d_exitcode;
    int d_signal;

    static void (ProcAcc::*s_output[])() const;

    public:
        ProcAcc(std::string const &fileName);
        // ~ProcAcc();                                  // Needed?
        int process(bool flag);

    private:
        void selectOutput() const;
        void show() const;
        bool isOpen() const;
        void setData();
};

#endif
```

Listing 41: procacc/procacc.ih

```
#include "procacc.h"
// #include <fstream>
#include <iostream>
// #include <linux/acct.h>
#include <csignal>
#include <cstring> // for strnlen()

using namespace std;
```

Listing 42: procacc/s_output.cc

```
#include "procacc.ih"

    // by

void (ProcAcc::*ProcAcc::s_output[])() const
=
{
    &ProcAcc::selectOutput,
    &ProcAcc::show
};
```

Listing 43: procacc/procacc1.cc

```
#include "procacc.ih"

    // by

ProcAcc::ProcAcc(string const &fileName)
:
    d_file(fileName, ios::binary)
{
    if (isOpen())
        cerr << "reading from: " << fileName << '\n';
}
```

Listing 44: procacc/process.cc

```
#include "procacc.ih"

    // by

int ProcAcc::process(bool showAll)
{
    if (!isOpen())
    {
        cerr << "failed to open file\n";
        return 1;
    }

    while (d_file.read(reinterpret_cast<char *>(&d_record), sizeof(d_record)))
    {
        setData();
        (this->*s_output[showAll])();
    }
    return 0;
}
```

Listing 45: procacc/selectoutput.cc

```
#include "procacc.ih"

    // by

void ProcAcc::selectOutput() const
{
    if (d_signal != 0) // || d_exitcode != 0)
```

```
        show();
}
```

Listing 46: procacc/show.cc


```
#include "procacc.ih"

    // by

void ProcAcc::show() const
{
    cout << "'" + d_processName + "' ";
    switch (d_signal)
    {
        case SIGKILL:
            cout << "KILL";
            //cout << " " << hex << d_record.ac_exitcode << dec;
        break;
        case SIGTERM:
            cout << "TERM";
            //cout << " " << hex << d_record.ac_exitcode << dec;
        break;
        default:
            cout << d_signal; // << " " << d_exitcode;
            //cout << (d_signal != 0 ? d_signal : d_exitcode);
            //cout << " " << hex << d_record.ac_exitcode << dec;
    }
    cout << '\n';
}
```

Listing 47: procacc/isopen.cc

```
#include "procacc.ih"

    // by

bool ProcAcc::isOpen() const
{
    if (d_file.is_open())
        return true;

    return false;
}
```

// CGR

Listing 48: procacc/setdata.cc

```
#include "procacc.ih"

    // by

namespace {
    enum {
        LOWEST_8_BITS = 0b11111111,
    };
}

void ProcAcc::setData()
{                                           // We cut off trailing '\0's
    size_t length = strnlen(d_record.ac_comm, ACCT_COMM);
    d_processName.assign(d_record.ac_comm, d_record.ac_comm + length);

    //d_signal = d_record.ac_exitcode & LOWEST_8_BITS;       // lowest 8 bits
    //d_exitcode = (d_record.ac_exitcode >> 8) & LOWEST_8_BITS;     // next 8

    d_signal = d_record.ac_exitcode;
}
```

Listing 49: main.ih

```
#include "procacc/procacc.h"
#include <fstream>
#include <iostream>

#include <csignal>

using namespace std;
```

18

Listing 50: main.cc

```
#include "main.ih"

int main(int argc, char *argv[])
{
    // if -a flag output all processes

    bool showAll = false;
    string fileName = "/var/log/account/pacct";
    size_t argcount = argc;
    for (size_t index = 1; index != argcount; ++index)
    {
        string arg = argv[index];

        if (arg == "-a")
            showAll = true;
        else
            fileName = arg;
    }

    ProcAcc logs(fileName);

    logs.process(showAll);
}
```

TC