Week 7

Exercise 58

Type	*Time*
real	0m2,944s
user	0m2,924s
sys	0m0,020s

Table 1: Not prefixed

Type	*Time*
real	0m0,033s
user	0m0,033s
sys	0m0,000s

Table 2: If-prefixed

Without the if check in every iteration the string has to be stored in the buffer. Afterwards when it tries to pass it to out, it wont be printed since the state is set to failbit. With the if check there is no need to store things in the buffer, which makes it much faster. So a general rule should be to perform checks before storing things in a buffer when possible.

../58/main.ih

```
1  #include <iostream>
2
3  using namespace std;
```

../58/main.cc

```
1  #include "main.ih"
2
3  int main(int argc, char **argv)
4  {
5    ostream out(cout.rdbuf()); // Initialise ostream out using buffer cout
6
7    out.setstate(ios::failbit);     // Set the failbit of out
8
9    size_t its = atoi(argv[1]);  // Convert command line argument to its
10
11   for (size_t index = 0; index != its; ++index) // Loop through its
12     if (out.good()) // If failbit is not set, comment out for other version
13       out << "Nr. of command line arguments " << argc << '\n';  // Output
14 }
```

Exercise 59

The code does work as intended for the first operation as the default open mode of `ofstream` is `ios_base::out`. This means that it immediately tries to write to the file, and create said file if it is not available. Conversely, `fstream` is meant for both reading and writing, and thus its default open mode is `ios_base::in | ios_base::out`: first open the file, then write to it. Since it does not exist, it can't open it, and won't continue.

The code can be fixed in a number of ways. First, the second operation could simply make use of `ofstream` as well. In this case, this is probably the best solution as the use of `ofstream` indicates that this concerns a 'write-only' application. Secondly, the default open mode of `fstream` can be overridden using the following parameter `out2{ "./tmp/out2", ios_base::out };`. Third, the file could be created first, before running the operation; either manually or by some other means in the code itself.

Exercise 60

As before, istr has error state flags. After it has been assigned to something for the first time, istr is now fully read until its end, and its `eofbit` has been set / toggled accordingly. In order to utilise istr again, its error state flags must be cleared, using `std::ios::clear` (i.e. `istr.clear();`).

```
1  ...
2  cout << "extracted first number: " << no1 << '\n';
3
4  istr.clear();        // Clear error state flags
5  istr.str(argv[2]);   // Sets istr to a copy of argv[2]
6  size_t no2 = 0;
7  istr >> no2;         // Assign contents of istr to no2
8  ...
```

Exercise 61

../61/main.cc

```
1  #include <iostream>
2  #include <iomanip>
3  #include <ctime>
4
5  using namespace std;
6
7  ostream &now(std::ostream &stream)
8  {
9    time_t currentTime = time(0);                  // Get caledar time
10   tm lTime = *localtime(&currentTime);           // Convert to tm
11   return stream << std::put_time(&lTime, "%c"); // %c refers to std. time/date string
12 }
13
14 int main()
15 {
16   cout << now << '\n';                           // Print
17 }
```

Exercise 62

../62/main.cc

```
1  void fun(...)
2  {
3  }
4
5  int main()
6  {
7      fun();
8      fun("with functions");
9      fun(1, 2, 3);
10 }
```

Exercise 63

../63/main.cc

```
1  #include <iomanip>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8    double value = 12.04;
9
10   cout  << setw(15)                                        << value << '\n'
11         << setw(15) << left                                << value << '\n'
12         << setw(15) << right                               << value << '\n'
13         << setw(15) << fixed << setprecision(1)            << value << '\n'
14         << setw(15) << fixed << setprecision(4)            << value << '\n'
15         << setw(15) << resetiosflags(std::cout.flags()) << value << '\n';
16 }
```

Exercise 65

../65/main.ih

```
#include <iostream>
#include <asm/types.h>
#include <sys/acct.h>
#include <fstream>
#include <csignal>
#include <iomanip>
#include <cstring>

using namespace std;

struct clOptions
{
  char const **filePaths = 0;
  bool dispAllVars = 0;
};

void printAccts(size_t idx, clOptions runOptions);
void destroy(struct clOptions);
string exitcode(__u32 exitcode);
struct clOptions processArgv(int argc, char **argv);
size_t numStructs(const char *filePath);
void populateAcct(acct_v3 &acct, ifstream &stream);
```

../65/main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
  clOptions runOptions = processArgv(argc, argv);

  for (int idx = 0; runOptions.filePaths[idx] != 0; ++idx)
  {
    printAccts(idx, runOptions);
  }
  destroy(runOptions);
}
```

../65/destroy.cc

```
#include "main.ih"

void destroy(clOptions toBeDeleted)
{
  delete[] toBeDeleted.filePaths; // Release the memory
}
```

../65/exitcode.cc

```
#include "main.ih"

string exitcode(__u32 exitcode) // Formats the exitcode print statements
{
  switch (exitcode)
  {
    case SIGTERM:               // Since these are already defined as ints,
      return "TERM";            // they can be used in this switch as-is
      break;
    case SIGKILL:
      return "KILL";
```

```
12        break;
13      default:
14        return to_string(exitcode);
15        break;
16    }
17  }
```

../65/numStructs.cc

```
1  #include "main.ih"
2
3  size_t numStructs(const char *filePath)
4  {
5    std::ifstream dFile(filePath, std::ios::binary);  // Open the file
6    dFile.seekg(0, ios_base::end);  // Go to last position in stream
7    size_t size = dFile.tellg();    // Get that position, assign to size
8    dFile.close();                  // Dissassociate file from stream
9    return size / sizeof(acct_v3);  // Return position divided by struct size,
10 }                                 // i.e. how many structs the file contains
```

../65/populateAcct.cc

```
1  #include "main.ih"
2
3  void populateAcct(acct_v3 &acct, ifstream &stream)
4  {
5    stream.read(reinterpret_cast<char*>(&acct), sizeof(acct_v3)); // Read in one struct
6  }
```

../65/printAcct.cc

```
1  #include "main.ih"
2
3  void printAccts(size_t idx, clOptions runOptions)
4  {
5    std::ifstream dFile(runOptions.filePaths[idx], std::ios::binary); // Open bin file
6    cout << runOptions.filePaths[idx] << '\n';  // Display filename
7
8    for (size_t index = 0; index != numStructs(runOptions.filePaths[idx]); ++index)
9    {                                  // Loop through bin file
10     struct acct_v3 acct;  // Define new struct
11     populateAcct(acct, dFile);  // Populate that struct from the ifstream file
12     if (runOptions.dispAllVars || acct.ac_exitcode) // If exitcode = 0 or if -a
13     {       // Print the processes
14       cout  << setw(20) << left << acct.ac_comm
15             << setw(10) << left << exitcode(acct.ac_exitcode) << '\n';
16     }
17   }
18 }
```

../65/processArgv.cc

```
1  #include "main.ih"
2
3  void enlarge(size_t fp, clOptions &currentStruct)  // Helper function: enlarge
4  {
5    char const **ret = new const char*[fp + 1];
6
7    for (size_t idx = 0; idx != fp; ++idx)
8        ret[idx] = currentStruct.filePaths[idx];
9
10   destroy(currentStruct);
11   currentStruct.filePaths = ret;
12 }
```

```
13
14  struct clOptions processArgv(int argc, char** argv)
15  {
16    clOptions runOptions;
17    size_t fp = 0;
18    if (argc > 1)
19    {
20      for (int idx = 1; idx != argc; ++idx)
21      {
22        if (strcmp(argv[idx], "-a") == 0)
23        {
24          runOptions.dispAllVars = 1;
25          break;
26        }
27        else
28        {
29          enlarge(fp, runOptions);
30          runOptions.filePaths[fp] = argv[idx];
31          ++fp;
32        }
33      }
34    }
35    if (!fp)
36    {
37      enlarge(fp, runOptions);
38      runOptions.filePaths[0] = "./pacct.bin";
39    }
40    return runOptions;
41  }
```

Exercise 66

For reference, using this program cuts down the size necessary to store the nucleobases of the first human chromosome down from `242mb` to `60.5mb`. This is a very intuitive result, as normally a character will take up a full byte to be stored, and now four characters are stored in a single byte.