Programming in C/C++
Exercises set six: Basic Input/Output

Christiaan Steenkist
Diego Ribas Gomes
Jaime Betancor Valado
Remco Bos

October 26, 2016

Exercise 50, creating a 'one-size-fits-all' function

The exercise gives a main function that won't compile because it lacks a function declaration. The declaration must allow for different types and number of arguments and is added in the following source.

Listing 1: variadic.cc

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

Exercise 51, understanding the behaviour of istreams

A piece of code is presented which is expected to take and output two numerical unsigned inputs, but outputs 0 instead of the second value. The reason for this is that the stream's data hasn't been activated by a seek operation after receiving the value by the << operator. The fix for this behaviour is added to the code.

Listing 2: Fix is on line 21

```
1 #include <iostream>
2 #include <sstream>
3 #include <string>
4
5 using namespace std;
6
7 int main(int argc, char **argv)
```

```
{
    if (argc > 3)
    {
        cout << "Require two numerical unsigned inputs" << '\n';
        return 1;
    }

    istringstream istr(argv[1]);
    size_t no1 = 0;
    istr >> no1;

    cout << "extracted first number: " << no1 << '\n';

    istr.clear();
    istr.str(argv[2]);
    size_t no2 = 0;
    istr >> no2;

    cout << "extracted second number: " << no2 << '\n';
}
```

Exercise 52, defining a manipulator

A manipulator is implemented to insert the current date and time as produced by `asctime` but without the trailing newline it automatically appends.

Listing 3: nowManip.cc

```
#include <iostream>
#include <ctime>
#include <cstring>

using namespace std;

ostream &now(ostream &stream)
{
  time_t ltime = time(NULL);

  string *currentTime = asctime(localtime(&ltime));
  currentTime.erase(currentTime.find_last_of("\n"), 1);

  return stream << currentTime;
}

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

```
20 }
```

Exercise 53, displaying floating point numbers using modifiers

A program is built that defines a double variable and displays it in different requested formats using a single `cout` statement.

Listing 4: floatModif.cc

```
1  #include <iostream>
2  #include <iomanip>
3
4  using namespace std;
5
6  int main()
7  {
8      double value = 12.04;
9
10     cout
11     << setw(15) << value << "\n"
12     << setw(15) << left << value << "\n"
13     << setw(15) << right << value << "\n"
14     << setw(15) << setprecision(3) << showpoint << value << "\n"
15     << setw(15) << setprecision(4) << fixed << value << "\n"
16     << resetiosflags(ios::fixed) << setw(15) << value << "\n";
17 }
```

Exercise 54, using binary files

A program is built reading by default the binary file /var/log/account/pacct or another specified command-line argument and outputs the names of all processes that didn't exit properly. The inclusion of a '-a' option should make it output information about all exited processes and if a process was killed with SIGKILL or SIGTERM, it should mention the name of the signal instead of the number.

Listing 5: main.cc

```
1  #include "main.h"
2
3  int main(int argc, char **argv)
4  {
5    Vars vars;
6    arguments(vars, argc, argv);
7    process(vars, std::cout);
8  }
```

Listing 6: main.h

```
1  #ifndef MAIN_H_
```

```
2  #define MAIN_H_
3
4  #include <iostream>
5
6  enum OPTION
7  {
8    ABRIDGED, VERBOSE
9  };
10
11 struct Vars
12 {
13   OPTION method = ABRIDGED;
14   std::string path = "/var/log/account/pacct";
15 };
16
17 void arguments(Vars &vars, int argc, char **argv);
18 void process(Vars &vars, std::ostream &output);
19 void printProcess(std::ostream &output, char *name,
20   size_t exitCode);
21
22 #endif
```

Listing 7: main.ih

```
1  #include "main.h"
2
3  #include <iostream>
4  #include <fstream>
5  #include <unistd.h>
6  #include </usr/include/linux/acct.h>
7  #include <csignal>
8
9  using namespace std;
```

Listing 8: arguments.cc

```
1  #include "main.ih"
2
3  void arguments(Vars &vars, int argc, char **argv)
4  {
5    // If there are no arguments do nothing.
6    if (argc == 1)
7      return;
8
9    // If an option -a is found then the method is set to verbose.
10   int opt;
11   while((opt = getopt(argc, argv, "a")) != EOF)
12   {
```

```
13      switch (opt)
14          {
15        case 'a':
16          vars.method = VERBOSE;
17        break;
18        default:
19          cerr << "Invalid option " << opt << '\n';
20          return;
21      }
22    }
23
24    if (vars.method == VERBOSE)
25    {
26      if (argc >= 3)
27        vars.path = argv[2];
28    }
29    else
30      vars.path = argv[1];
31  }
```

Listing 9: process.cc

```
1   #include "main.ih"
2
3   // Size of the acct_v3 struct
4   // as well as relative positions
5   // of the name and exit code.
6
7   void process(Vars &vars, ostream &output)
8   {
9     ifstream is(vars.path, ifstream::binary);
10
11    if (!is)
12    {
13      cerr << "File not found or not openable.\n";
14      cerr << vars.path << '\n';
15      return;
16    }
17
18    // Get the length of the file and
19    // return to the beginning.
20    is.seekg(0, is.end);
21    size_t length = is.tellg();
22    is.seekg(0, is.beg);
23
24    // The reading process reads acct_v3 structs from the file.
25    // It then reads the important information directly
26    // from the struct.
```

```
27    acct_v3 log;
28
29    size_t process = 0;
30    size_t place = 0;
31    size_t stepSize = sizeof(log);
32    while ((place = process * stepSize) < length)
33    {
34      is.seekg(place, is.beg);
35      is.read(reinterpret_cast<char *>(&log), sizeof(log));
36
37      if (log.ac_exitcode != 0 || vars.method == VERBOSE)
38        printProcess(output, log.ac_comm, log.ac_exitcode);
39
40      ++process;
41    }
42  }
```

Listing 10: printProcess.cc

```
1  #include "main.ih"
2
3  void printProcess(ostream &output, char *name, size_t exitCode)
4  {
5    output << '\'' << name << "\' ";
6    switch(exitCode)
7    {
8      case(SIGKILL):
9        output << "KILL";
10     break;
11     case(SIGTERM):
12       output << "TERM";
13     break;
14     default:
15       output << exitCode;
16     break;
17   }
18   output << '\n';
19 }
```