Week 7: II

Exercise 62

../62/main.cc

```
1  void fun(...);
2  // This is a variadic function declaration. It does not 'do' anything, per se,
3  // but it does allow the program to compile.
4
5  int main()
6  {
7      fun();
8      fun("with functions");
9      fun(1, 2, 3);
10 }
```

Exercise 65

This accompanying text was already checked, but is included again for consistency
In general, tailing a file would be easier in `C` than it would be in `C++` as `C` is a system programming language, and `C++` is not, at least not without calling `C`-functions. This implies that there is usually one or more more level(s) of abstraction between a file and the program in `C++`. In this case, this abstraction is in the form of a buffer. When buffering, or reading, a file is blocked from being accessed by another program, possibly the one that may be adding the additional information that we are interested in to the file. Hence, more low-level access to a file (through a more low-level language) would definitely better facilitate such functionality.

../65–2/main.ih

```
using namespace std;

#include "accth/accth.h"
```

../65–2/main.cc

```
#include "main.ih"

int main(int argc, char const *argv[])
{
  AcctH defaultAcct(argc, argv);  // Pass argc/argv to constructor
  defaultAcct.processFiles();     // Process the accounting files
}
```

../65–2/accth/accth.h

```
// Acct handler

#ifndef INCLUDED_ACCTH_
#define INCLUDED_ACCTH_

#include <iosfwd>

#include <asm/types.h>  // __u32, unsigned int
#include <sys/acct.h>   // acct_v3

class AcctH
{

  public:
    char const *defaultPath = "/var/log/account/pacct";  // Default file to use

    AcctH(int argc, char const **argv); // Constructor
    void processFiles();                // Process input files

  private:
    struct clPars                       // Command-line option struct
    {
      bool allExits = 0;                // Display all exits (-a)
      size_t nFiles = 0;                // Number of input files
      char const **filePaths = 0;       // Array of filepaths
    };

    clPars d_clPars;                                    // Struct data member
    void processArgv(size_t argc);                      // Process argc/argv
    void processBin(std::ifstream &stream);             // Process binary file
    bool popAcct(acct_v3 &acct, std::ifstream &stream); // Populate acct struct
    void printAcct(acct_v3 const &acct);                // Print acct struct
    std::string formExit(__u32 exitcode);               // Format exit code
};

#endif
```

../65–2/accth/accth.ih

```
#include "accth.h"

#include <csignal>  // Exit code definitions
#include <cstddef>  // size_t
#include <cstring>  // strcmp
#include <iostream>
#include <fstream>  // Streams
#include <iomanip>  // Output formatting

using namespace std;
```

../65–2/accth/c_accth.cc

```
#include "accth.ih"

AcctH::AcctH(int argc, char const **argv)
:
  d_clPars( clPars{0, static_cast<size_t>(argc), argv} )  // Initialise struct
{
  processArgv( static_cast<size_t>(argc) );  // Process argc/argv
  // Note that argc is passed again instead of nFiles, because the latter is
  // altered, and would therefore interfere with the loop in processArgv

  d_clPars.filePaths[0] = defaultPath;       // Re-use program name element for
}                                            // default file
```

../65–2/accth/formExit.cc

```
#include "accth.ih"

string AcctH::formExit(__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";
      break;
    default:
      return to_string(exitcode);
      break;
  }
}
```

../65–2/accth/popAcct.cc

```
#include "accth.ih"

bool AcctH::popAcct(acct_v3 &acct, ifstream &stream)
{
  if ( stream.read(reinterpret_cast<char*>(&acct), sizeof(acct_v3)) )
    return 1;                                          // Read in one struct
  else
    return 0;
  // Returns boolean according to whether a struct was read
}
```

../65–2/accth/printAcct.cc

```
#include "accth.ih"
```

```
2
3  void AcctH::printAcct(acct_v3 const &acct)
4  {                                                  // Loop through bin file
5      if (d_clPars.allExits || acct.ac_exitcode) // If exitcode != 0 or if -a,
6      {                                              // print the process exits
7        std::cout << setw(20) << left << acct.ac_comm
8                  << setw(10) << left << formExit(acct.ac_exitcode) << '\n';
9      }
10 }
```

../65–2/accth/processArgv.cc

```
1  #include "accth.ih"
2
3  void AcctH::processArgv(size_t argc)
4  {
5    for (size_t idx = 0; idx != argc; ++idx) // Iterate over arguments
6    {
7      if ( strcmp(d_clPars.filePaths[idx], "-a") == 0 ) // If -a is found
8      {
9        d_clPars.allExits = 1;  // Set bool to display all all exits
10       --d_clPars.nFiles;      // Number of files are therefore one less than argc
11       d_clPars.filePaths[idx] = d_clPars.filePaths[d_clPars.nFiles];
12     }                                  // Replace -a with last filename in list
13   }
14 }
```

../65–2/accth/processBin.cc

```
1  #include "accth.ih"
2
3  void AcctH::processBin(ifstream &stream)
4  {
5    struct acct_v3 acct;  // Define new struct
6    while( popAcct(acct, stream) )  // While structs can still be populated
7      printAcct(acct);  // Print populated acct struct
8  }
```

../65–2/accth/processFiles.cc

```
1  #include "accth.ih"
2
3  void AcctH::processFiles()
4  {
5    size_t defaultFile = ( (d_clPars.nFiles == 1) ? 0 : 1 );
6    // If no file specified, use default at position 0
7
8    for (size_t idx = defaultFile; idx != d_clPars.nFiles; ++idx) // Iterate over files
9    {
10     std::ifstream accFile(d_clPars.filePaths[idx], std::ios::binary); // Open bin
11     cout << '\n' << d_clPars.filePaths[idx] << '\n';  // Display filename
12     processBin(accFile);  // Process binary file
13   }
14 }
```

Exercise 66

../66–tko–Sharknado/main.ih

```
#include <iostream>
#include <string>
#include <ostream>
#include <fstream>
#include "data/data.h"

using namespace std;
```

../66–tko–Sharknado/main.cc

```
#include "main.ih"

int main(int argc, char const *argv[])
{
  string inputLoc = argv[1];
  string outputLoc = argv[2];
  string optionb;
  if (argv[3])
    optionb = argv[3];

  Data goodName(inputLoc, outputLoc);

  goodName.writeFile(optionb);

}
```

../66–tko–Sharknado/data/data.h

```
#ifndef INCLUDED_DATA_
#define INCLUDED_DATA_

#include <iosfwd>
#include <string>
#include <fstream>


class Data
{
    std::string d_inputLoc;
    std::string d_outputLoc;
    int d_filesize;
    char d_firstchar;

    struct nucleobase         // Struct that can hold four ints of 2 bits each.
    {                         // Hence, they can hold the numbers 0 through 3,
      unsigned char nb1 : 2; // or 00, 01, 10, 11 in binary, i.e.
      unsigned char nb2 : 2; // the four nucleobase options
      unsigned char nb3 : 2;
      unsigned char nb4 : 2;
    };

    enum nucleoInts           // enums that represent the nucleobase options.
    {                         // '= 0' is SF in this case but added for clarity.
      A = 0,
      C,
      T,
      G
    };

    public:

```

```
34        Data(std::string inputLoc, std::string outputLoc);
35
36        int writeFile(std::string optionb);
37
38      private:
39
40        char enumToChar(int nucEnum);
41
42        enum nucleoInts charToEnum(char ch);
43
44        char interpretStruct(nucleobase &nB, size_t idx);
45
46        void popStruct(nucleobase &nB, char ch, size_t idx);
47
48        std::ifstream::pos_type filesize(std::string filename);
49
50        int chartobin(std::ifstream &iF);
51
52        void bintochar(std::ifstream &iF);
53
54        int chartochar(std::ifstream &iF);
55
56        void bintobin(std::ifstream &iF);
57
58        bool isItaBinaryFile(std::ifstream &iF);
59
60        void binTo(std::ifstream &iF, std::string optionb);
61
62        int charTo(std::ifstream &iF, std::string optionb);
63  };
64
65  #endif
```

../66–tko–Sharknado/data/data.ih

```
1  #include "data.h"
2  #include <ostream>
3
4
5  using namespace std;
```

../66–tko–Sharknado/data/binTo.cc

```
1  #include "data.ih"
2
3  void Data::binTo(ifstream &iF, string optionb)
4  {
5    if (optionb == "-b")
6      bintobin(iF);
7    else
8      bintochar(iF);
9  }
```

../66–tko–Sharknado/data/binToBin.cc

```
1  #include "data.ih"
2
3  void Data::bintobin(ifstream &iF)
4  {
5    ofstream oF( d_outputLoc, std::ofstream::out | std::ofstream::trunc);
6
7    nucleobase oNB;
8    int8_t nrinlastbyte = d_firstchar;
9
10   oF.write(reinterpret_cast<char*>(&nrinlastbyte), sizeof(char));
```

```
11    //reading the first byte which is an idicator of howmany characters are
12    //stored in the last byte. And then writing this file to the new bin file.
13
14    while(!iF.eof())
15    {
16      iF.read(reinterpret_cast<char*>(&oNB), sizeof(nucleobase));
17      oF.write(reinterpret_cast<char*>(&oNB), sizeof(nucleobase));
18    }
19    //reading the other bytes and writing them to the new binary file.
20
21  }
```

../66–tko–Sharknado/data/binToChar.cc

```
1   #include "data.ih"
2
3   void Data::bintochar(ifstream &iF)
4   {
5     ofstream oF( d_outputLoc, std::ofstream::out | std::ofstream::trunc);
6
7     nucleobase oNB;
8     int8_t nrinlastbase = d_firstchar;
9
10    //reading howmany characters are in the last byte and also increasing the
11    //position in the file by one.
12    size_t nrbytes = filesize( d_inputLoc ) - 1; //first byte is size of last byte
13
14    size_t charcounter = 0;
15
16    while (iF.read(reinterpret_cast<char*>(&oNB), sizeof(nucleobase)))
17    {
18      for (size_t idx = 0; idx != 4; ++idx)
19        if (charcounter < 4 * nrbytes - (4 - nrinlastbase))
20        {
21            oF << interpretStruct(oNB, idx);
22            ++charcounter;
23        }
24    }
25    //reading the structs and writing their contained information into the new
26    //character file. Up until the required number of characters are printed.
27    //Since every byte contains 4 characters except the last one we need to do
28    //this 4 times the nr of bytes minus the number of empty missing values
29    //that are contained in the last byte.
30  }
```

../66–tko–Sharknado/data/charTo.cc

```
1   #include "data.ih"
2
3   int Data::charTo(ifstream &iF, string optionb)
4   {
5     if (optionb == "-b")
6       return chartobin(iF);
7     else
8       return chartochar(iF);
9   }
```

../66–tko–Sharknado/data/charToBin.cc

```
1   #include "data.ih"
2
3
4   int Data::chartobin(ifstream &iF)
5   {
6
```

```
7       ofstream oF( d_outputLoc, ios::binary | std::ofstream::out |
8          std::ofstream::trunc);
9
10      char ch = d_firstchar;
11
12      int8_t nrinlastbase = d_filesize % 4 - 1;
13      //nr of chars in the last base, -1 since its starts counting at 1 instead
14      // of 0
15
16      oF.write(reinterpret_cast<char*>(&nrinlastbase), sizeof(char));
17
18      while (!iF.eof())
19      {
20
21        nucleobase nB; //initialising nB and its values
22        nB.nb1 = 0;
23        nB.nb2 = 0;
24        nB.nb3 = 0;
25        nB.nb4 = 0;
26
27
28        for (size_t idx = 0; idx != 4; ++idx)
29        {
30          if (ch != 'A' && ch != 'C' && ch != 'G' && ch != 'T' && ch != '\n')
31            return 1;
32       //not for the new line which often occurs at the end of
33       //files. and must be a valid base character
34
35            popStruct(nB, ch, idx); //putting the characters in the struct
36            iF.get(ch);             //so that we only have to use 1 byte for
37                                    //4 chars.
38        }
39
40        oF.write(reinterpret_cast<char*>(&nB), sizeof(nucleobase));
41      }
42      oF.close();
43    return 0;
44  }
```

../66–tko–Sharknado/data/charToChar.cc

```
1   #include "data.ih"
2
3   int Data::chartochar(ifstream &iF)
4   {
5       ofstream oF( d_outputLoc, std::ofstream::out | std::ofstream::trunc);
6
7       oF << d_firstchar;
8
9       char ch;
10
11      while (iF.get(ch))
12      {
13        if (ch != 'A' && ch != 'C' && ch != 'G' && ch != 'T')
14          return 1;
15
16        oF << ch;
17      }
18
19      oF.close();
20      //writing the characters in the input file to the output file.
21    return 0;
22  }
```

../66–tko–Sharknado/data/charToEnum.cc

```
1  #include "data.ih"
2
3  enum Data::nucleoInts Data::charToEnum(char ch)
4  { // Returns the enum representation of the four nucleobase options
5    switch (ch)
6      {
7        case 'A':
8          return A;
9          break;
10       case 'C':
11         return C;
12         break;
13       case 'T':
14         return T;
15         break;
16       case 'G':
17         return G;
18         break;
19       default:
20         break;
21     }
22 }
```

../66–tko–Sharknado/data/data.cc

```
1  #include "data.ih"
2
3  Data::Data(string inputLoc, string outputLoc)
4  {
5    d_inputLoc = inputLoc;
6    d_outputLoc = outputLoc;
7
8    d_filesize = filesize ( d_inputLoc );
9
10 }
```

../66–tko–Sharknado/data/enumToChar.cc

```
1  #include "data.ih"
2
3  char Data::enumToChar(int nucEnum)  // Take the passed enum (int),
4  {                                   // and return its associated character
5    switch (nucEnum)
6      {
7        case A:
8          return 'A';
9          break;
10       case C:
11         return 'C';
12         break;
13       case T:
14         return 'T';
15         break;
16       case G:
17         return 'G';
18         break;
19       default:
20         break;
21     }
22 }
```

../66–tko–Sharknado/data/filesize.cc

```
1  #include "data.ih"
```

```
2
3  std::ifstream::pos_type Data::filesize(const string filename)
4  {
5    std::ifstream input(filename, std::ifstream::ate | std:: ifstream::binary);
6    return input.tellg();
7    input.close();
8  }
```

../66–tko–Sharknado/data/interpretStruct.cc

```
1  #include "data.ih"
2
3  char Data::interpretStruct(nucleobase &nB, size_t idx)
4  {                 //From nB, take element n and have enumToChar
5    switch(idx)     //convert them to char and return that char
6    {
7      case 0:
8        return enumToChar(nB.nb1);
9        break;
10     case 1:
11       return enumToChar(nB.nb2);
12       break;
13     case 2:
14       return enumToChar(nB.nb3);
15       break;
16     case 3:
17       return enumToChar(nB.nb4);
18       break;
19     default:
20       break;
21   }
22 }
```

../66–tko–Sharknado/data/isItaBinaryFile.cc

```
1  #include "data.ih"
2
3  bool Data::isItaBinaryFile(ifstream &iF)
4  {
5    char ch;
6
7    iF.get(ch);
8
9    if (ch < 4) //first ch in bin file gives nr of bits stored in last char
10     return true; //which has to be lower than 4
11
12   d_firstchar = ch;
13
14   return false;
15
16 }
```

../66–tko–Sharknado/data/popStruct.cc

```
1  #include "data.ih"
2
3  void Data::popStruct(nucleobase &nB, char ch, size_t idx)
4  {                    // Populate element n within struct nB, calling enumChar
5    switch(idx)        // on the passed char first
6    {
7      case 0:
8        nB.nb1 = charToEnum(ch);
9        break;
10     case 1:
11       nB.nb2 = charToEnum(ch);
```

```
12        break;
13      case 2:
14        nB.nb3 = charToEnum(ch);
15        break;
16      case 3:
17        nB.nb4 = charToEnum(ch);
18        break;
19      default:
20        break;
21    }
22  }
```

../66–tko–Sharknado/data/writeFile.cc

```
1  #include "data.ih"
2
3
4  int Data::writeFile(string optionb)
5  {
6    ifstream iF( d_inputLoc );
7
8    char ch;
9    iF.get(ch);
10   d_firstchar = ch;
11
12   if (ch < 4) //first ch in bin file gives nr of bits stored in last char
13     binTo(iF, optionb); //which has to be lower than 4
14   else
15     return charTo(iF, optionb);
16
17   return false;
18 }
```