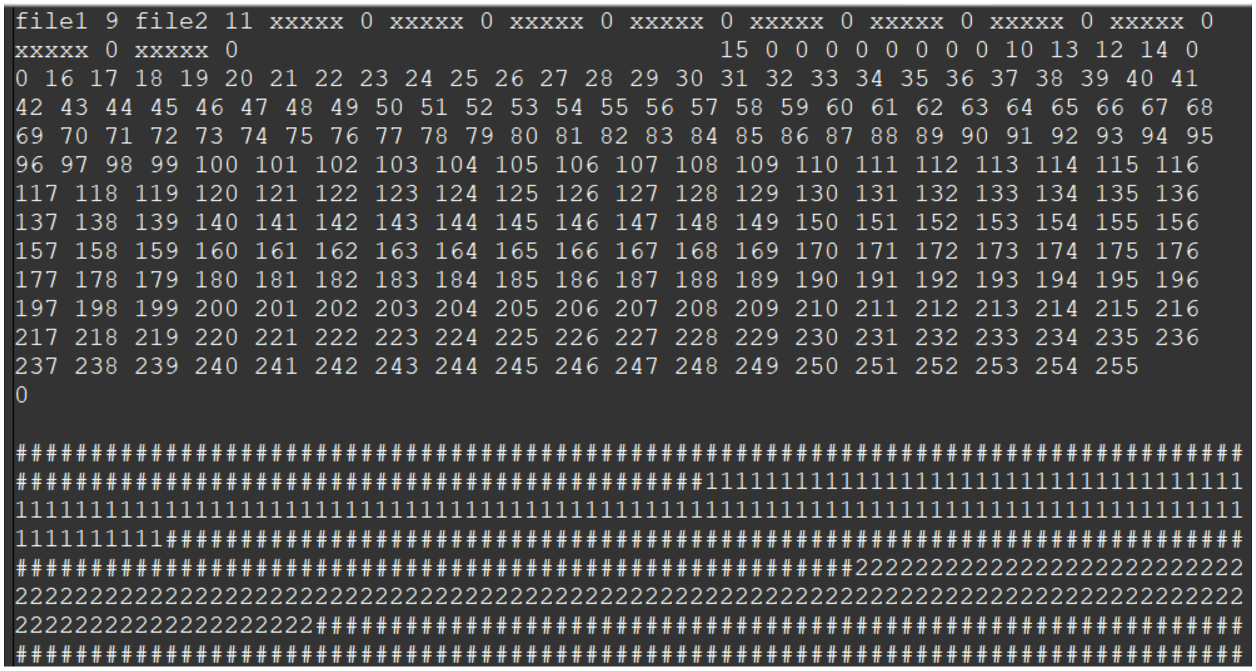
Zoe Veale
Lab 3 and 4



```
file1 9 file2 11 xxxxx 0 xxxxx 0 xxxxx 0 xxxxx 0 xxxxx 0 xxxxx 0 xxxxx 0 xxxxx 0
xxxxx 0 xxxxx 0                                 15 0 0 0 0 0 0 0 0 10 13 12 14 0
0 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68
69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116
117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156
157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176
177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196
197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236
237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
0

###################################################################################
##############################################1111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111#########################################################################
########################################################222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222###############################################################
###################################################################################
```

```
#include "fileSystem.h"

int main(int argc, char* argv) {
  Sdisk diskA("diskA", 256, 128);
  FileSystem fsys("diskA", 256, 128);

  fsys.NewFile("file1");
  fsys.NewFile("file2");


  std::string bfile1;
  for (int i = 1; i <= 1024; i++) {
    bfile1 += "1";
  }
  fsys.AddBlock("file1", bfile1);
  //fsys.RemoveFile("file1");
  //fsys.RemoveFile("file2");

  std::string bfile2;
  std::string block;



  int blocknumber = 0;




  for (int i = 1; i <= 2048; i++) {
    bfile2 += "2";
  }
  blocknumber = fsys.AddBlock("file2", bfile2);
  //fsys.DeleteBlock("file2", blocknumber);
  return 0;
}


#ifndef FILESYSTEM_H
#define FILESYSTEM_H

#include <vector>
#include <sstream>
#include <iomanip>
#include <utility>

#include "sDisk.h"

class FileSystem : public Sdisk {
public:
  FileSystem(std::string diskname, int numberofblocks, int blocksize);
  int FileSystemClose();
  int FileSystemSynch();
  int NewFile(std::string file);
  int RemoveFile(std::string file);
  std::pair<int, int> GetFirstBlock(std::string file);
  int AddBlock(std::string file, std::string blockBuffer);
  int DeleteBlock(std::string file, int blocknumber);
  int ReadBlock(std::string file, int blocknumber, std::string& buffer);
  int WriteBlock(std::string file, int blocknumber, std::string buffer);
  int NextBlock(std::string file, int blocknumber);
  //std::vector<std::string> Block(std::string buffer, int blockSize);
private:
  const int FILE_ALLOCATION_SIZE = 4;
  const int FILE_NAME_SIZE = 10;
  const int FAT_MEMORY_POSITION = 1;
  const int ROOT_MEMORY_POSITION = 0;
  const int FREE_BLOCK = 0;
  int FreeBlock();
  int rootSize;           // maximum number of entries in ROOT
  int fatSize;            // number of blocks occupied by FAT
  std::vector<std::string> filename;   // filenames in ROOT
  std::vector<int> firstBlock; // firstblocks in ROOT
  std::vector<int> fat;             // FAT

};
#endif // !FILESYSTEM_H

#include "fileSystem.h"

FileSystem::FileSystem(
  std::string diskname, int numberofblocks, int blocksize) :
  Sdisk(diskname, numberofblocks, blocksize),
  rootSize(GetBlockSize() / FILE_NAME_SIZE),
  fatSize(GetNumberOfBlocks()* FILE_ALLOCATION_SIZE / GetBlockSize()) {


  std::string buffer;
  GetBlock(1, buffer);
  if (buffer[1] == '#') {
    //new disk
    for (int i = 0; i < rootSize; i++) {
      //TODO:: preprocessor defines for no file and first block zero, NULL maybe
      filename.push_back("xxxxx");
      firstBlock.push_back(0);
      ///
    }
    for (int i = 0; i < GetNumberOfBlocks(); i++) {
      if (i < fatSize + FAT_MEMORY_POSITION) {
        fat.push_back(0);
      }
      else {
        fat.push_back(i + 1);
      }
    }
    //fat.resize(GetNumberOfBlocks(), -1);
    fat[0] = fatSize + FAT_MEMORY_POSITION;
    fat[GetNumberOfBlocks() - FAT_MEMORY_POSITION] = 0;
    FileSystemSynch();
  }
  else {
    //load disk
    buffer.clear();
    GetBlock(0, buffer);
    std::stringstream diskData(buffer);
    for (unsigned int i = 0; i < rootSize; i++) {
      std::string name;
      int blockNumber;
      diskData >> name >> blockNumber;
      filename.push_back(name);
      firstBlock.push_back(blockNumber);
    }

    diskData.str("");
    diskData.clear();
    for (int i = FAT_MEMORY_POSITION; i < fatSize + FAT_MEMORY_POSITION; i++) {
      GetBlock(i, buffer);
      diskData.str(diskData.str() + buffer);
    }

    for (unsigned int i = 0; i < GetNumberOfBlocks(); i++) {
      int blockNumber;
      diskData >> blockNumber;
      fat.push_back(blockNumber);
    }
  }


}

int FileSystem::FileSystemClose() {
  FileSystemSynch();
  return 1;
}

int FileSystem::FileSystemSynch() {

  std::string syncBuffer;
  for (int i = 0; i < filename.size(); i++) {
    syncBuffer += filename[i] + " " + std::to_string(firstBlock[i]) + " ";
  }
  for (int i = syncBuffer.size(); i < GetBlockSize(); i++) {
    syncBuffer += " ";
  }
  PutBlock(ROOT_MEMORY_POSITION, syncBuffer);

  syncBuffer.clear();
  //TODO:: convert int to hex
  for (unsigned i = 0; i < fat.size(); ++i) {
    syncBuffer += std::to_string(fat[i]) + " ";
  }
  for (int i = syncBuffer.size(); i < GetBlockSize() * fatSize; i++) {
    syncBuffer += " ";
  }
  for (int i = 0; i < fatSize; i++) {
    PutBlock(i + 1, syncBuffer.substr(i * GetBlockSize(), GetBlockSize()));
  }



  return 0;
}

int FileSystem::NewFile(std::string file) {

  for (std::string i : filename) {
    if (i == file)
      return 0; // file exists already
  }

  int freeFileIndex = 0;
  for (int i : firstBlock) {
    if (i == 0) {
      break;
    }
    ++freeFileIndex;
  }
  if (freeFileIndex >= rootSize) return 0;

  //TODO:: fat index following new file needs to = 0
  filename[freeFileIndex] = file;
  firstBlock[freeFileIndex] = fat[FREE_BLOCK];
  fat[FREE_BLOCK] = fat[fat[fat[FREE_BLOCK]]];
  fat[fat[firstBlock[freeFileIndex]]] = 0;



  FileSystemSynch();
  return 1;
}

int FileSystem::RemoveFile(std::string file) {
  std::pair<int, int> firstBlockPair = GetFirstBlock(file);
  if (firstBlockPair.first < fatSize + FAT_MEMORY_POSITION) return 0;

  DeleteBlock(file, firstBlockPair.first);
  filename[firstBlockPair.second] = "xxxxx";
  fat[FREE_BLOCK] = firstBlock[firstBlockPair.second];
  firstBlock[firstBlockPair.second] = 0;
  FileSystemSynch();
  return 1;
}

//first element = firstBlock[index], second element = index
std::pair<int, int> FileSystem::GetFirstBlock(std::string file) {
  std::pair<int, int> firstBlockPair;
  int freeFileIndex = 0;
  for (std::string i : filename) {
    if (i == file)
      break;
    else
      ++freeFileIndex;
  }
  //TODO >= or just >??
  if (freeFileIndex > rootSize) return std::pair<int, int>(0, 0);


  firstBlockPair.first = firstBlock[freeFileIndex];
  firstBlockPair.second = freeFileIndex;
  FileSystemSynch();
  return std::pair<int, int>(firstBlock[freeFileIndex], freeFileIndex);
}

int FileSystem::AddBlock(std::string file, std::string blockBuffer) {
  int endBlock = 0;
  int firstBlock = GetFirstBlock(file).first;
  if (firstBlock < fatSize + FAT_MEMORY_POSITION) return 0;
  for (int i = firstBlock; i < fat.size(); i = fat[i]) {
    if (fat[i] == 0) {
      endBlock = i;
      break;
    }
  }
  PutBlock(endBlock, blockBuffer);
  fat[endBlock] = fat[FREE_BLOCK];
  fat[FREE_BLOCK] = fat[fat[FREE_BLOCK]];
  fat[fat[endBlock]] = 0;
  FileSystemSynch();
  return 0;
}

int FileSystem::DeleteBlock(std::string file, int blocknumber) {
  int startBlock = GetFirstBlock(file).first;
  for (int i = startBlock; i < fat.size(); i = fat[i]) {
    if (i == blocknumber) {
      startBlock = blocknumber;
      break;
    }
    if (fat[i] == 0) {
      return 0;
    }
  }
  for (int i = startBlock; i < fat.size(); i = fat[i]) {
    if (fat[i] == 0) {
```

```cpp
      fat[i] = fat[FREE_BLOCK];
      fat[FREE_BLOCK] = fat[startBlock];
      return 1;
    }
  }
  return 0;
}

int FileSystem::ReadBlock(std::string file, int blocknumber, std::string& blockBuffer) {
  int startBlock = GetFirstBlock(file).first;
  for (int i = startBlock; i < fat.size(); i = fat[i]) {
    if (i == blocknumber) {
      GetBlock(blocknumber, blockBuffer);
      return 1;
    }
  }
  return 0;
}



int FileSystem::WriteBlock(std::string file, int blocknumber, std::string blockBuffer) {
  int startBlock = GetFirstBlock(file).first;
  for (int i = startBlock; i < fat.size(); i = fat[i]) {
    if (i == blocknumber) {
      PutBlock(blocknumber, blockBuffer);
      return 1;
    }
  }
  return 0;
}

int FileSystem::NextBlock(std::string file, int blocknumber) {

  return 0;
}

int FileSystem::FreeBlock() {
  int index = 0;
  for (int i : fat) {
    if (i == -1) {
      return index;
    }
    ++index;
  }
  return 0;
}
```