Zoe Veale
CSE 461
Lab3


```
#include "fileSystem.h"

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

}
```

```
#ifndef FILESYSTEM_H
#define FILESYSTEM_H

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

#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);
  int GetFirstBlock(std::string file);
  int AddBlock(std::string file, std::string block);
  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);
private:
  const int FILE_ALLOCATION_SIZE = 4;
  const int FILE_NAME_SIZE = 7;
  const int FAT_MEMORY_POSITION = 1;
  const int ROOT_MEMORY_POSITION = 0;
```

```
  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("xxxx");
      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);
      }
    }
    //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::ostringstream rootStringStream;
  for (int i = 0; i < filename.size(); i++) {
    rootStringStream << filename[i] << " " << firstBlock[i] << " ";
  }
  for (int k = rootStringStream.tellp(); k < GetBlockSize(); k++) {
    rootStringStream << " ";
  }
  PutBlock(ROOT_MEMORY_POSITION, rootStringStream.str());

  std::ostringstream fileTableAllocationStringStream;

  for (int i = 0; i < fatSize; i++) {

    for (int k = GetBlockSize() * (i) / FILE_ALLOCATION_SIZE;
             k < GetBlockSize() * (i + 1) / FILE_ALLOCATION_SIZE; k++) {
      fileTableAllocationStringStream << fat[k] << " ";
    }
    for (int k = fileTableAllocationStringStream.tellp(); k < GetBlockSize(); k++) {
      fileTableAllocationStringStream << " ";
    }
    PutBlock(FAT_MEMORY_POSITION + i, fileTableAllocationStringStream.str());
    fileTableAllocationStringStream.str("");
    fileTableAllocationStringStream.clear();
  }

  return 0;
}
```