CPSC 457 - Assignment 5 – Part 2

Q7 - Programming question (30 marks)

For this question you will implement a program that will check the consistency of a file allocattion table with respect to the entries of a directory. Your program will read input from standard input, and will output results to standard output.

Input

The input will contain a simplistic representation of the filesystem. It will contain:

- block size – an integer in range [1, 1024]
- number of files / entries in the directory – an integer in range [0, 50]
- number of entries in FAT – an integer in range [1, 200000]
- the entries in the directory – an array, each entry containg:
    - filename – a string of up to 16 characters, any chars allowed except white space
    - first block pointer – an index into the FAT, an integer in range [-1, 200000) -1 denotes 'null' pointer
    - actual files size, integer in range [0, 2^{30}]
- the entries in the FAT – an array of integers
    - each entry represents a pointer to the next entry in the FAT
    - each entry is an integer in a range [-1, 200000)
    - -1 denotes a 'null' pointer (end of chain)
    - the entries in FAT are numbered starting from 0

Here is an example input:

```
10 3 11
A 0 31
B 6 23
C -1 0
5 9 5 3 -1 1 8 0 6 -1 0
```

The above input describes a filesystem that has a block size of 10, contains FAT with 11 entries, and holds 3 files: A, B and C.

File A contains 31 bytes, and it is stored in blocks: {0, 5, 1, 9}. It has the correct number of blocks, contains no cycles, and does not share blocks with any other file.

File B has 23 bytes, and it is stored on blocks {6, 8}. The blocks belonging to file B form a cycle, which is a problem that your program will need to detect. File B also has an incorrect number of blocks for its size, which is another problem your program needs to report. File B does not share blocks with any other file. If it did, you would need to detect and report that as well.

File C is empty. No blocks are allocated to this file. There are no problems with this file.

Finally, the total number of unused blocks on the filesystem is 5. This is a number you will need to calculate and report.

Output

After reading in the input, your program will check the consistency of the filesystem and report its findings to standard output. For every file you need to determine whether thera are any issues with the file. You need to check for the following issues:

- Does the file contain the right number of blocks, or are there too many or too few?
- Do the blocks allocated to the file contain a cycle?
- Does the file share its blocks with any other file?

You then need to report all issues you found for every file.

You will also determine how many of the blocks are unused in the filesystem. Unused blocks are the ones not allocated to any file.

Sample output:

Here is an output that your program should produce for the above sample input:

```
Issues with files:
  A:
  B: not enough blocks, contains cycle
  C:
Number of free blocks: 5
```

Submission

You need to submit your implementation in a file called impl.cpp or impl.c to D2L.

Appendix 1 – Skeleton for Q7

The following C++ skeleton program impl.cpp is provided for you to make parsing and output easier. All you need to do is to reimplement the function checkConsistency().

```
// CPSC457 Fall 2017, University of Calgary
// Skeleton C++11 program for Q7 of Assignment 5.
//
// The program reads in the input, then calls the (wrongly implemented) checkConsistency()
// function, and finally formats the output.
//
// You only need to reimplement the checkConsistency() function.
//
// Author: Pavol Federl (pfederl@ucalgary.ca or federl@gmail.com)
// Date: November 29, 2017
// Version: 1

#include <stdio.h>
#include <string>
#include <vector>

typedef std::string SS;
typedef std::vector<SS> VS;

struct DEntry {
    SS fname = SS( 4096, 0);
    int size = 0;
    int ind = 0;
    bool tooManyBlocks = true;
    bool tooFewBlocks = false;
    bool hasCycle = true;
    bool sharesBlocks = true;
};

static SS join( const VS & toks, const SS & sep) {
    SS res;
    bool first = true;
    for( auto & t : toks) { res += (first ? "" : sep) + t; first = false;}
    return res;
}

// re-implement this function
//
// Parameters:
//   blockSize - contains block size as read in from input
//   files - array containing the entries as read in from input
//   fat - array representing the FAT as read in from input
// Return value:
//   the function should return the number of free blocks
//   also, for ever entry in the files[] array, you need to set the appropriate flags:
//      i.e. tooManyBlocks, tooFewBlocks, hasCycle and sharesBlocks
int checkConsistency( int blockSize, std::vector<DEntry> & files, std::vector<int> & fat)
{
    // make the first entry contain no errors
    if( files.size() > 0) {
        files[0].hasCycle = false;
        files[0].tooFewBlocks = false;
        files[0].tooManyBlocks = false;
        files[0].sharesBlocks = false;
    }

    // make the 2nd entry contain one error
    if( files.size() > 1) {
        files[1].hasCycle = true;
        files[1].tooFewBlocks = false;
        files[1].tooManyBlocks = false;
        files[1].sharesBlocks = false;
    }

    // make the 3rd entry contain two errors
    if( files.size() > 2) {
        files[2].hasCycle = false;
        files[2].tooFewBlocks = false;
        files[2].tooManyBlocks = true;
        files[2].sharesBlocks = true;
    }

    // finally, return the number of free blocks
    return 0;
}

int main()
{
    try {
        // read in blockSize, nFiles, fatSize
        int blockSize, nFiles, fatSize;
        if( 3 != scanf( "%d %d %d", & blockSize, & nFiles, & fatSize))
            throw "cannot read blockSize, nFiles and fatSize";
        if( blockSize < 1 || blockSize > 1024) throw "bad block size";
        if( nFiles < 0 || nFiles > 50) throw "bad number of files";
        if( fatSize < 1 || fatSize > 200000) throw "bad FAT size";
        // read in the entries
        std::vector<DEntry> entries;
        for( int i = 0 ; i < nFiles ; i ++ ) {
            DEntry e;
            if( 3 != scanf( "%s %d %d", (char *) e.fname.c_str(), & e.ind, & e.size))
                throw "bad file entry";
            e.fname = e.fname.c_str();
            if( e.fname.size() < 1 || e.fname.size() > 16)
                throw "bad filename in file entry";
            if( e.ind < -1 || e.ind >= fatSize) throw "bad first block in fille entry";
            if( e.size < 0 || e.size > 1073741824) throw "bad file size in file entry";
            entries.push_back( e);
        }
        // read in the FAT
        std::vector<int> fat( fatSize);
        for( int i = 0 ; i < fatSize ; i ++ ) {
            if( 1 != scanf( "%d", & fat[i])) throw "could not read FAT entry";
            if( fat[i] < -1 || fat[i] >= fatSize) throw "bad FAT entry";
        }

        // run the consistency check
        int nFreeBlocks = checkConsistency( blockSize, entries, fat);

        // format the output
        size_t maxflen = 0;
        for( auto & e : entries ) maxflen = std::max( maxflen, e.fname.size());
        SS fmt = "  %" + std::to_string( maxflen) + "s: %s\n";

        printf( "Issues with files:\n");
        for( auto & e : entries ) {
            VS issues;
            if( e.tooFewBlocks) issues.push_back( "not enough blocks");
            if( e.tooManyBlocks) issues.push_back( "too many blocks");
            if( e.hasCycle) issues.push_back( "contains cycle");
            if( e.sharesBlocks) issues.push_back( "shares blocks");
            printf( fmt.c_str(), e.fname.c_str(), join( issues, ", ").c_str());
        }
        printf( "Number of free blocks: %d\n", nFreeBlocks);
    }
    catch( const char * err) {
        fprintf( stderr, "Error: %s\n", err);
    }
    catch( ... ) {
        fprintf( stderr, "Errro: unknown.\n");
    }
    return 0;
}
```