**Baimakanov Arsen   
CS-2113**

**Report:**This code is a C program for securely deleting a file on Windows. It uses the file system to write random data over the clusters of a file before deleting it. Here is a breakdown of the main components of the code:

- `securelyDeleteFile`: This function takes a filename as an argument and is responsible for securely deleting it. It first attempts to open the file with write access. If it cannot be opened, the function returns. Then it retrieves the file size, cluster size, and disk capacity information using the `GetFileSizeEx` and `GetDiskFreeSpace` functions. Using that information, the function calculates the number of random clusters required to overwrite the file's contents. It then generates an array of random cluster numbers using the `CryptAcquireContext`, `CryptGenRandom`, and `CryptReleaseContext` functions. Finally, it writes random data to each of the selected clusters using the `WriteFile` function, frees memory, closes the file handle, and deletes the file using the `DeleteFile` function.

- `main`: This function is the entry point of the program. It checks if a filename has been provided as a command line argument. If not, it prints the program usage and returns an error code. If a filename has been provided, it calls the `securelyDeleteFile` function with that filename as an argument.

Overall, this code is an example of how one can create a program that securely deletes files by overwriting their contents with random data. Please note that, while this is an effective way to make it very difficult to recover deleted files, there is no guaranteed way to securely delete data. (by using Windows API)

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <windows.h>

void securelyDeleteFile(char\* fileName) {

HANDLE fileHandle = CreateFile(fileName, GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (fileHandle == INVALID\_HANDLE\_VALUE) {

return;

}

LARGE\_INTEGER fileSize;

GetFileSizeEx(fileHandle, &fileSize);

DWORD clusterSize;

GetDiskFreeSpace(NULL, NULL, &clusterSize, NULL, NULL);

DWORD sectorsPerCluster;

DWORD bytesPerSector;

DWORD numberOfFreeClusters;

DWORD totalNumberOfClusters;

GetDiskFreeSpace(NULL, &sectorsPerCluster, &bytesPerSector, &numberOfFreeClusters, &totalNumberOfClusters);

DWORD bytesToWipe = fileSize.QuadPart;

DWORD bytesPerCluster = clusterSize \* sectorsPerCluster;

DWORD clusterCount = bytesToWipe / bytesPerCluster;

if (bytesToWipe % bytesPerCluster) {

++clusterCount;

}

DWORD\* clusters = malloc(sizeof(DWORD) \* clusterCount);

if (clusters == NULL) {

return;

}

for (DWORD i = 0; i < clusterCount; i++) {

DWORD randomCluster;

do {

if (!CryptAcquireContext(NULL, NULL, NULL, PROV\_RSA\_FULL, 0)) {

break;

}

if (!CryptGenRandom(NULL, sizeof(DWORD), (BYTE\*)&randomCluster)) {

break;

}

if (!CryptReleaseContext(NULL, 0)) {

break;

}

randomCluster = randomCluster % totalNumberOfClusters;

} while (randomCluster < 2);

clusters[i] = randomCluster;

}

for (DWORD i = 0; i < clusterCount; i++) {

DWORD sectorCount = bytesPerCluster / bytesPerSector;

BYTE\* buffer = malloc(bytesPerCluster);

if (buffer == NULL) {

continue;

}

DWORD sectorNumber = clusters[i] \* sectorsPerCluster;

SetFilePointer(fileHandle, sectorNumber \* bytesPerSector, NULL, FILE\_BEGIN);

for (DWORD j = 0; j < sectorCount; j++) {

DWORD bytesWritten;

WriteFile(fileHandle, buffer, bytesPerSector, &bytesWritten, NULL);

}

free(buffer);

}

free(clusters);

CloseHandle(fileHandle);

DeleteFile(fileName);

}

int main(int argc, char\* argv[]) {

if (argc < 2) {

printf("Usage: %s <filename>", argv[0]);

return 1;

}

securelyDeleteFile(argv[1]);

return 0;

}