```
#include<bits/stdc++.h>
#include <conio.h>
#include <direct.h>
#include <stdio.h>
#include <sys/types.h>
#include <dirent-master/include/dirent.h>
#include <stdlib.h>
using namespace std;
struct File {
        string name;
        char* date;
        char* buffer;
        long long size = 0;
        void getNameFile(string  path) {
                while (path.back() != '\\')
                        name += path.back(), path.pop_back();
                reverse(name.begin(), name.end());
        }
        bool read(string s) {

                char* path = new char[s.size() + 1];
                strcpy(path, s.c_str());
                FILE* pFile;
                size_t result;
                pFile = fopen(path, "rb");
                if (pFile == NULL) { fputs("The path is Error\n", stderr); return 0; }
                time_t now = time(0);
                date = ctime(&now);
                getNameFile(s);
```

```
		// obtain file size:
		fseek(pFile, 0, SEEK_END);
		size = ftell(pFile);
		rewind(pFile);

		// allocate memory to contain the whole file:
		buffer = new char[size];
		if (buffer == NULL) { fputs("Memory error", stderr); exit(2); }

		// copy the file into the buffer:
		result = fread(buffer, 1, size, pFile);
		if (result != size) {
			fputs("Reading error", stderr); exit(3);
		}

		// the whole file is now loaded in the memory buffer.
		fclose(pFile);
		return 1;
	}
	void write(string s, string newName) {
		s += "\\" + newName;
		char* path = new char[s.size() + 1];
		strcpy(path, s.c_str());
		FILE* pFile1;
		cout << path << endl;
		pFile1 = fopen(path, "wb");
		if (pFile1 == NULL)
		{
			cout << "path Error" << endl;
```

```
                return;
            }
            fwrite(buffer, sizeof(char), size, pFile1);
            fclose(pFile1);
        }
        void print() {
            //string sp1(70 - name.size(), ' '), sp2(60- to_string(size / 1024.0).size(), ' ');
            cout << "    " << name << "              " << "TYPE:FILE            " << "SIZE:" << size /
1024.0 << " KB            " << "date:" << date;
        }
};
long long totalSize, currentSize = 0;
struct Folder {
        vector<File> files;
        vector<Folder> folders;
        string name;
        long long size;
        Folder(string n) {
            size = 0;
            name = n;
        }
        void copyFromMeToMe(int idx, vector<string> path2, Folder F) {
            if (idx + 1 == path2.size()) {
                for (int i = 0; i < folders.size(); i++)
                    if (F.name == folders[i].name) {
                        cout << "This Folder is Exist" << endl;
                        return;
                    }
                folders.push_back(F);
                return;
        }


        cout << "The path not found" << endl;


}
bool copyFolderTo(vector<string> v) {
        string path = v[1];
        v.push_back("\\" + name);
        for (int i = 2; i < v.size(); i++)
                path += v[i];
        if (_mkdir(path.c_str()) == -1)return 0;
        for (File f : files)
                f.write(path, f.name);
        for (Folder f : folders)
                if (!f.copyFolderTo(v))return 0;
        return 1;
}
pair<bool, Folder> openChild(vector<string> v) {
        string s = v[1];
        for (int i = 2; i < v.size(); i++)
                s += " " + v[i];
        for (Folder i : folders)
                if (i.name == s) {
                        return{ 1, i };
                }
        return make_pair(0, Folder("Error"));
}
void makeDir(vector<string> v) {
```

```
                string s = v[1];
                for (int i = 2; i < v.size(); i++)
                        s += " " + v[i];
                for (Folder i : folders)
                        if (i.name == s) {
                                cout << "A subdirectory or file " << s << " already exists." <<
endl;
                                return;
                        }
                folders.push_back(Folder(s));
        }
        void deleteFile(vector<string> v) {
                string s = v[1];
                for (int i = 2; i < v.size(); i++)
                        s += " " + v[i];
                for (int i = 0; i < files.size(); i++)
                        if (files[i].name == s) {
                                size -= files[i].size;
                                currentSize -= files[i].size;
                                for (int j = i + 1; j < files.size(); j++)
                                        swap(files[j], files[j - 1]);
                                files.pop_back();
                                return;
                        }
                cout << "The File Not Found " << s << endl;


        }
        void deleteFolder(vector<string> v) {
                string s = v[1];
                for (int i = 2; i < v.size(); i++)
                        s += " " + v[i];
                for (int i = 0; i < folders.size(); i++)
                        if (folders[i].name == s) {
                                size -= folders[i].size;
                                currentSize -= folders[i].size;
                                for (int j = i + 1; j < folders.size(); j++)
                                        swap(folders[j], folders[j - 1]);
                                folders.pop_back();
                                return;
                        }
                cout << "The Folder Not Found " << s << endl;


        }
        void addFolder(string n, int idx, vector<string> path) {
                if (idx + 1 == path.size()) {
                        folders.push_back(Folder(n));
                        return;
                }
                for (int i = 0; i < folders.size(); i++)
                        if (path[idx + 1] == folders[i].name) {
                                folders[i].addFolder(n, idx + 1, path);
                                return;
                        }
                cout << "The path not found" << endl;
        }

        void listFilesRecursively(char* basePath)
        {
                char path[1000];
                struct dirent* dp;
                DIR* dir = opendir(basePath);
                // Unable to open directory stream
                if (!dir) {
                        //cout << "Path is not correct" << endl;
                        return;
                }
                while ((dp = readdir(dir)) != NULL)
                {
                        if (strcmp(dp->d_name, ".") != 0 && strcmp(dp->d_name, "..") != 0)
                        {
                                //cout << dp->d_name << endl;
                                strcpy(path, basePath);// Construct new path from our base
        path
                                strcat(path, "\\");
                                strcat(path, dp->d_name);

                                if (dp->d_type != 16384) {
                                        File f;
                                        f.read(path);
                                        files.push_back(f);
                                        size += f.size;
                                }
                                else {
                                        folders.push_back(Folder(dp->d_name));
                                        folders.back().listFilesRecursively(path);
                                        size += folders.back().size;
                                }
                        }
                }
                closedir(dir);
            }
            string getNameFolder(string  path) {
                string s;
                while (path.back() != '\\')
                        s += path.back(), path.pop_back();
                reverse(s.begin(), s.end());
                return s;
            }
            void copyfolder(vector<string> v) {
                string s = v[1];
                for (int i = 2; i < v.size(); i++)
                        s += " " + v[i];
                char* path = new char[s.size() + 1];
                strcpy(path, s.c_str());
                string z;

                if (path[1] == ':' && path[2] == '\\') {
                        z = getNameFolder(path);
                        if (z.size() == 0) {
                                cout << "path is Erorr" << endl;
                                return;
                        }
                }

                else
                        z = path;
                for (Folder i : folders)
                        if (i.name == z) {
                                cout << "The FOlder is already Exist" << endl;
                                return;
                        }
                struct dirent* dp;
                DIR* dir = opendir(path);
                // Unable to open directory stream
                if (!dir) {
                        cout << "Path is not correct" << endl;
                        return;
                }

                Folder F(z);
                F.listFilesRecursively(path);
                if (F.size + currentSize > totalSize) {
                        cout << "Not Exist Free space" << endl;
                        return;
                }
                size += F.size;
                currentSize += F.size;
                folders.push_back(F);
            }
            void print() {
                printf("----------------------------------------------------------------\n");
                printf("----------------------------------------------------------------\n");
                cout << "folder total size : " << size / 1024.0 << " KB" << endl << endl;
                for (Folder i : folders)
            				cout << i.name << "  " << "type:<DIR>" << "   size: " << i.size / 1024.0 << "
            KB" << endl;
            			cout << endl;
            			for (File i : files)
            				i.print();
            			cout << endl;
            			printf("--------------------------------------------------------------\n");
            			printf("--------------------------------------------------------------\n");
            			cout << endl;


            		}
            		void copyFile(vector<string> v) {
            			string s = v[1];
            			for (int i = 2; i < v.size(); i++)
            				s += " " + v[i];
            			char* path = new char[s.size() + 1];
            			strcpy(path, s.c_str());
            			File f;
            			if (!f.read(path))return;
            			for (File F : files)
            				if (F.name == f.name) {
            					cout << "The file is aready exist" << endl;
            					return;
            				}
            			if (currentSize + f.size > totalSize) {
            				cout << "Not Exist Free space" << endl;
            				return;
            			}
            			size += f.size;
```

```
        currentSize += f.size;

        files.push_back(f);


}

void WriteFile(vector<string> v) {

        string newName = v[2];

        string path = v[3];

        string fileName = v[1];

        for (int i = 4; i < v.size(); i++)

                path += " " + v[i];

        for (File i : files)

                if (i.name == fileName) {

                        i.write(path, newName);

                        return;

                }

}

void getFolder(string n, bool& ok, Folder& ch) {

        ok = 0;

        for (Folder f : folders)

                if (f.name == n) {

                        ok = 1;

                        ch = f;

                        return;

                }

}

void getFile(string n, bool& ok, File& ch) {

        ok = 0;

        for (File f : files)

                if (f.name == n) {
                        ok = 1;
                        ch = f;
                        return;
                    }
        }
        bool pestFolder(Folder f) {
                for (Folder ch : folders)
                        if (ch.name == f.name)
                                return 0;
                folders.push_back(f);
                size += f.size;
                currentSize += f.size;
                return 1;
        }
        bool pestFile(File f) {
                for (File ch : files)
                        if (ch.name == f.name)
                                return 0;
                files.push_back(f);
                size += f.size;
                currentSize += f.size;
                return 1;
        }
};
void PrintPath(vector<Folder> Mypartion) {
        for (int i = 0; i < Mypartion.size(); i++) {
                if (i)cout << "\\";
                cout << Mypartion[i].name;
                if (!i)cout << ":";
        }
        cout << "\\";
}
vector<string> splitComand(string s) {
        stringstream ss;
        ss << s;
        vector<string> v;
        while (ss >> s)v.push_back(s);
        for (int i = 0; i < v[0].size(); i++)
                v[0][i] = tolower(v[0][i]);
        return v;
}

int main() {
        bool x = true;
        while (x == true) {
                try {
                        cout << "Please Enter the size of your pation in MB : ";
                        cin >> totalSize;
                        x = false;
                        if (cin.fail())
                                throw totalSize;



                        if (totalSize < 0)
                                throw  x;
                }
                catch (bool e) {
                        x = true;
                }
                catch (long long x) {
                        cin.clear();
                        cin.ignore(132, '\n');
                        x = true;
                }
        }
        cin.ignore();


        totalSize *= 1024 * 1024;
        Folder newFolder("New Folder");
        File newFile;
        bool isCopeyf = 0, isCopeyd = 0;
        vector<Folder> Mypartion(1, Folder("Mypartion"));
        while (true) {
                PrintPath(Mypartion);
                string s;
                getline(cin, s);
                reverse(s.begin(), s.end());
                while (s.size() && s.back() == ' ')s.pop_back();
                reverse(s.begin(), s.end());
                if (s.empty())continue;


                vector<string> v = splitComand(s);
                if (v[0] == "cd") {
                        if (v.size() != 2) {
                                cout << '\'' << v[0] << '\'' << " is not recognized as an internal or
external command" << endl;
                        }
                        else {
                                pair<bool, Folder> p = Mypartion.back().openChild(v);
                                if (p.first)
                                        Mypartion.push_back(p.second);
                                else cout << "The system cannot find the path specified." <<
endl;
                        }
                }
                else if (v[0] == "cls") {
                        system("cls");
                }
                else if (v[0] == "dir") {
                        Mypartion.back().print();
                }
                else if (v[0] == "md")
                {
                        if (v.size() == 1) {
                                cout << '\'' << v[0] << '\'' << " is not recognized as an internal or
external command" << endl;
                                continue;
                        }
                        Mypartion.back().makeDir(v);
                }
                else if (v.size() == 1 && v[0] == "cd..") {
                        if (Mypartion.size() > 1) {
                                for (int f = 0; f < Mypartion[Mypartion.size() - 2].folders.size();
f++)
                                        if (Mypartion[Mypartion.size() - 2].folders[f].name ==
Mypartion.back().name) {
```

```
                                                Mypartion[Mypartion.size() - 2].size -=
Mypartion[Mypartion.size() - 2].folders[f].size;

                                                Mypartion[Mypartion.size() - 2].folders[f] =
Mypartion.back();

                                                Mypartion[Mypartion.size() - 2].size +=
Mypartion[Mypartion.size() - 2].folders[f].size;

                                                Mypartion.pop_back();

                                                break;

                                        }


                                }

                        }

                        else if (v[0] == "delf") {

                                Mypartion.back().deleteFile(v);

                        }

                        else if (v[0] == "deld")

                        {

                                Mypartion.back().deleteFolder(v);

                        }

                        else if (v[0] == "copyd")

                        {

                                if (v.size() == 1) {


                                        cout << '\"' << v[0] << '\"' << " is not recognized as an internal or
external command" << endl;

                                        continue;

                                }

                                Mypartion.back().copyfolder(v);

                        }

                        else if (v[0] == "import")
```

```
		{
			if (v.size() == 1) {
				cout << '\"' << v[0] << '\"' << " is not recognized as an internal or
external command" << endl;
				continue;
			}
			Mypartion.back().copyFile(v);
		}
		else if (v[0] == "mem")
		{
			cout << "Total : " << totalSize / 1024.0 << " KB" << endl << "Current Size :
" << currentSize / 1024.0 << " KB" << endl << "Free Space : " << (totalSize - currentSize) / 1024.0
<< " KB" << endl;
		}
		else if (v[0] == "copydto")
		{
			if (v.size() == 1) {
				cout << '\"' << v[0] << '\"' << " is not recognized as an internal or
external command" << endl;
				continue;
			}
			if (!Mypartion.back().copyFolderTo(v)) {
				cout << "Unable To Create the Folder" << endl;
			}


		}
		else if (v[0] == "export")
		{
			if (v.size() < 4) {
```

```
                cout << '\"' << v[0] << '\"' << " is not recognized as an internal or
external command" << endl;
                continue;
            }
            Mypartion.back().WriteFile(v);
        }
        else if (v[0] == "copyfme") {
            if (v.size() == 1) {
                cout << '\"' << v[0] << '\"' << " is not recognized as an internal or
external command" << endl;
                continue;
            }
            string s;
            for (int i = 1; i < v.size(); i++)
                s += v[i];
            Mypartion.back().getFile(s, isCopeyf, newFile);
            if (!isCopeyf)cout << "The file not exist" << endl;
            else if (currentSize + newFile.size > totalSize) {
                isCopeyf = 0;
                cout << "Not Exist Free space" << endl;
            }
        }
        else if (v[0] == "copydme") {
            if (v.size() == 1) {
                cout << '\"' << v[0] << '\"' << " is not recognized as an internal or
external command" << endl;
                continue;
            }
            string s;
            for (int i = 1; i < v.size(); i++)
```

```
				s += v[i];
			Mypartion.back().getFolder(s, isCopeyd, newFolder);
			if (!isCopeyd)cout << "The folder not exist" << endl;
			else if (currentSize + newFolder.size > totalSize) {
				isCopeyd = 0;
				cout << "Not Exist Free space" << endl;
			}
		}
		else if (v[0] == "pastf") {
			if (!isCopeyf)cout << "NO File Copyed" << endl;
			else if (!Mypartion.back().pestFile(newFile))
				cout << "The File is aready exist" << endl;
		}
		else if (v[0] == "pastd") {
			if (!isCopeyd)cout << "NO Folder Copyed" << endl;
			else if (!Mypartion.back().pestFolder(newFolder))
				cout << "The Folder is aready exist" << endl;
		}
		else if (v[0] == "help")
		{
			cout << "CD  /cd..      Displays the name of or changes the current
directory." << endl;
			cout << "CLS           Clears the screen." << endl;
			cout << "DELf           Deletes one  files. " << endl;
			cout << "DELd           Deletes one  Folder." << endl;
			cout << "DIR           Displays a list of files and subdirectories in a
directory." << endl;
			cout << "import         Copies one files to mypartion." << endl;
			cout << "copyd         Copies one folder to mypartion." << endl;
```

```
                        cout << "mem          Display use space." << endl;
                        cout << "md          Create new folder" << endl;
                        cout << "pastf          past file" << endl;
                        cout << "copyfme          copy file" << endl;
                        cout << "copydme          copy  Folder" << endl;
                        cout << "pastd          past Folder" << endl;
                        cout << "export          copy file Outside my partion syntax filename +new
name + path" << endl;
                        cout << "copydto          copy folder Outside my partion syntax new name
+path" << endl;


                }
                else cout << '\'' << v[0] << '\'' << " is not recognized as an internal or external
command" << endl;








        }
        _getch();
        system("pause");
        return 0;
}
```