ASSIGNMENT 2 FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	Unit 20: Advanced Programming		
Submission date	25/6/2020	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Student Name	LE ANH HIEU	**Student ID**	GCH18106
Class	GCH0714	**Assessor name**	DOAN TRUNG TUNG

Student declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.

Student's signature	HIEU

Grading grid

P3	P4	M3	M4	D3	D4

☼ **Summative Feedback:**		☼ **Resubmission Feedback:**
Grade:	**Assessor Signature:**	**Date:**
Lecturer Signature:		

Table of Content

Table of Figure

1. Introduction

OOP is one of the programming paradigms. Perhaps in the Programming Paradigm at the present time OOP is the most famous. It is so famous that in the programming community, few people mention other models. OOP enables programmers to view a program as a collection of interactive objects.

Design Patterns are a technique in object-oriented programming, it's quite important and every good programmer must know. Design patterns are frequently used in OOP languages. It gives you "designs", solutions to solve common problems, common in programming. The problems you encounter may come up with a solution yourself, but it may not be optimal. Design Pattern helps you solve problems in the most optimal way, providing you with solutions in OOP programming.

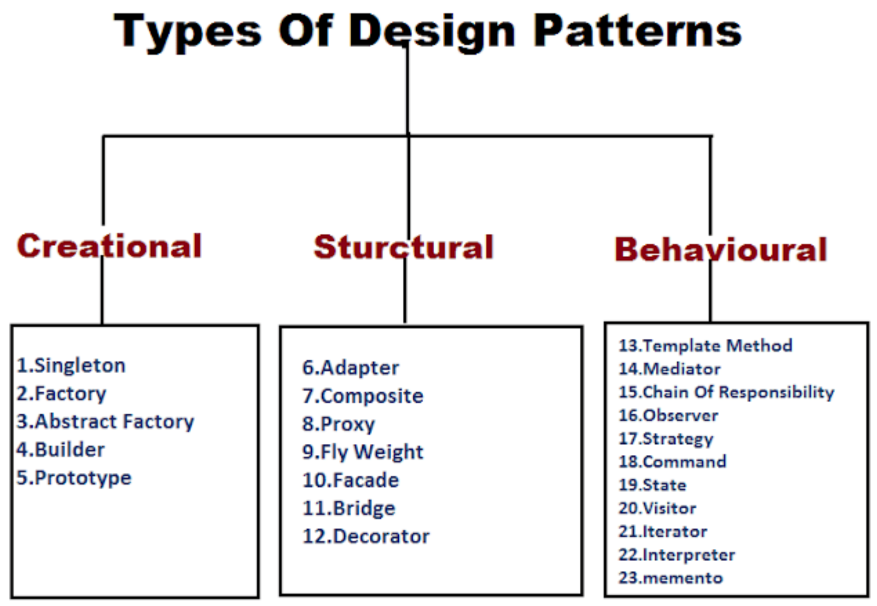


Figure 1: Types of Design Patterns

2. Scenario analysis

2.1. Screnario

VietHoaGame is a team specializing in localizing foreign language games. They are very interested in the localization project of Pearl Abyss's famous Black Desert Online game, but this is an online game, too much text, and the game is updated weekly making it impossible continue to use the old translation file. So they need a software for both players and translators to support updating translations into new files every week.

The technical team had an algorithm to decrypt the game, but in the process of coding, they didn't know if the input file was encrypted or decrypted, so they needed to build a file handling solution for easy code maintenance and expansion.

2.2. Diagram

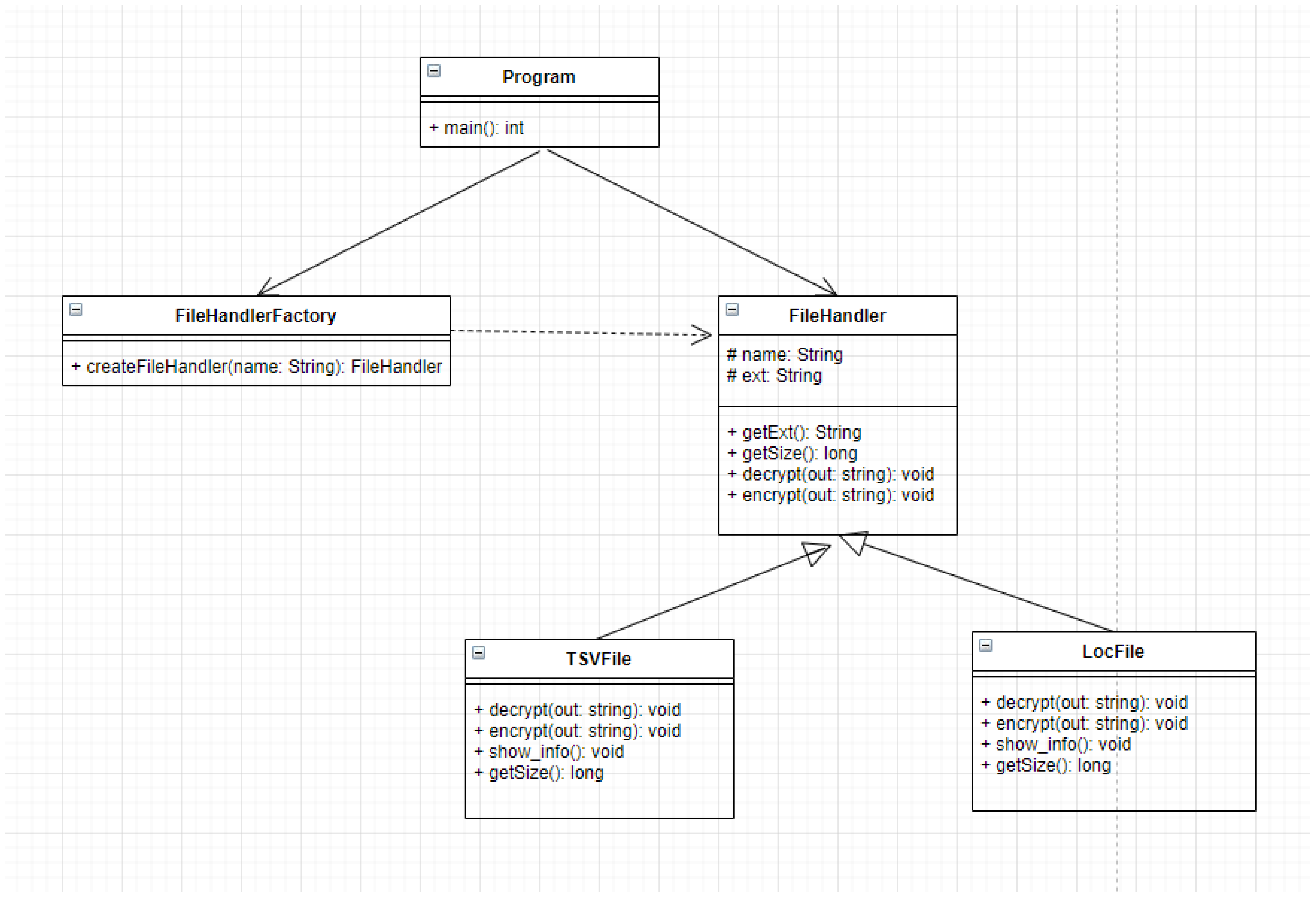

Program
+ main(): int
FileHandlerFactory
+ createFileHandler(name: String): FileHandler
FileHandler
# name: String
# ext: String
+ getExt(): String
+ getSize(): long
+ decrypt(out: string): void
+ encrypt(out: string): void
TSVFile
+ decrypt(out: string): void
+ encrypt(out: string): void
+ show_info(): void
+ getSize(): long
LocFile
+ decrypt(out: string): void
+ encrypt(out: string): void
+ show_info(): void
+ getSize(): long

Figure 2: Class Diagram

Solution to design file handling function using **Factory Method Pattern** (belongs to Creational Patterns group).

By definition, the factory method pattern is a creational pattern that uses factory methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. This is done by creating objects by calling a factory method — either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes — rather than by calling a constructor.

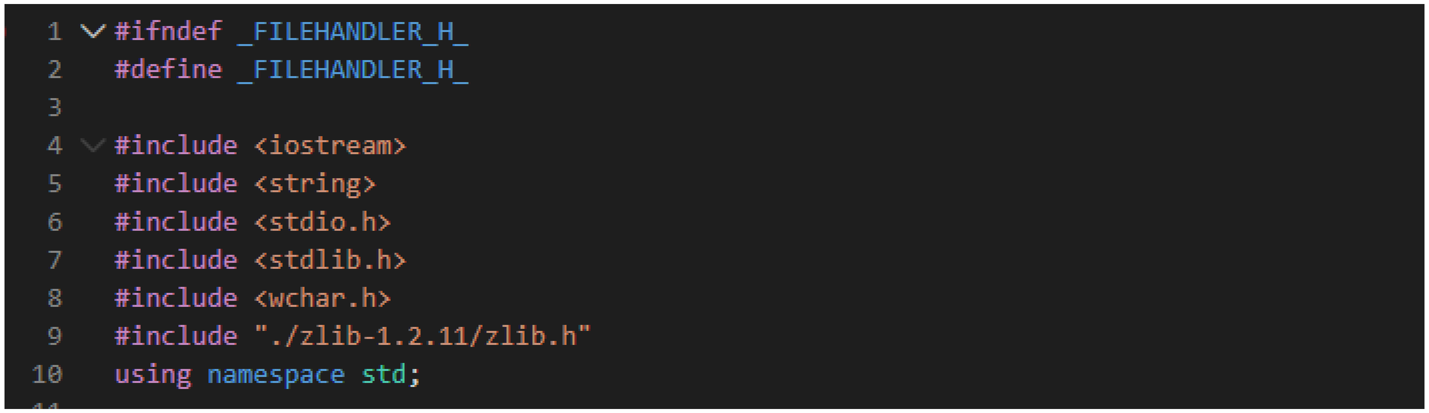
In class-diagram, the library is now responsible to decide which object type to create based on an input (fileName). Programer just needs to make call to library's factory createFileHandler method and pass the file name it wants without worrying about the actual implementation of creation of objects.

Currently the program only has two subclasses TSVFile and LocFile inheriting from the FileHandler parent class, but later if the programer want to expand more types of files, he just add subclasses inheriting from this FileHandler class.

3. Implementation

3.1. Code

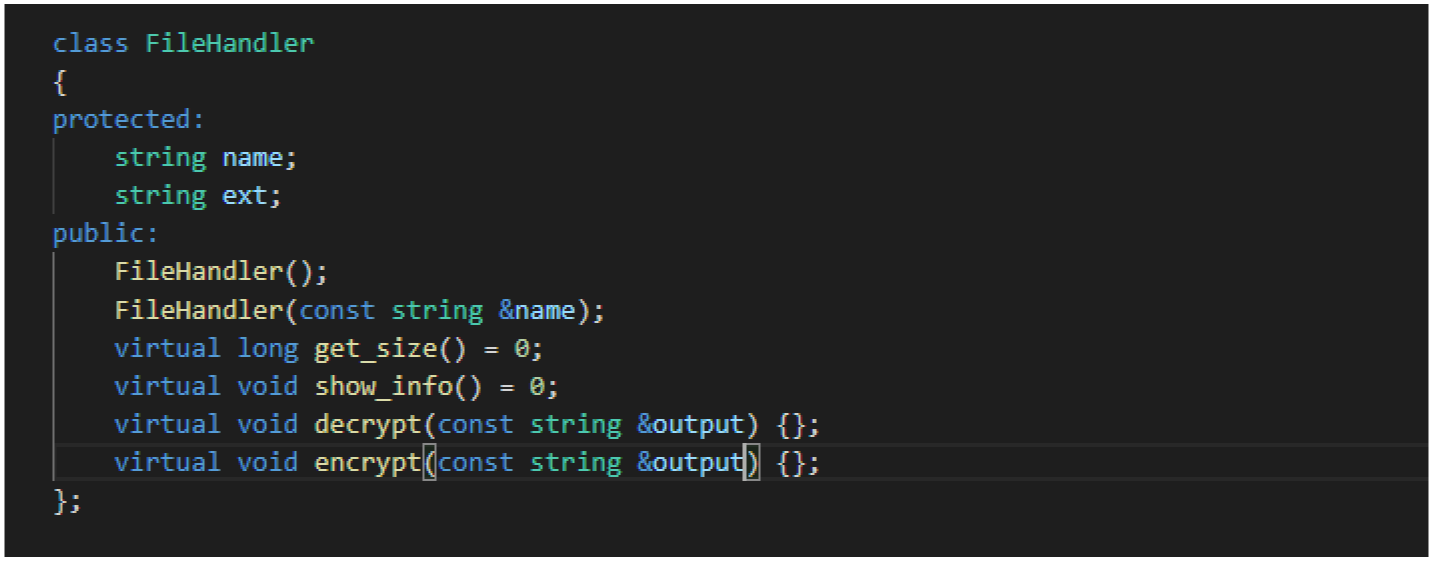
The first is the FileHandler.h header file, we declare some libraries to use:



```
#ifndef _FILEHANDLER_H_
#define _FILEHANDLER_H_

#include <iostream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include "./zlib-1.2.11/zlib.h"
using namespace std;
```

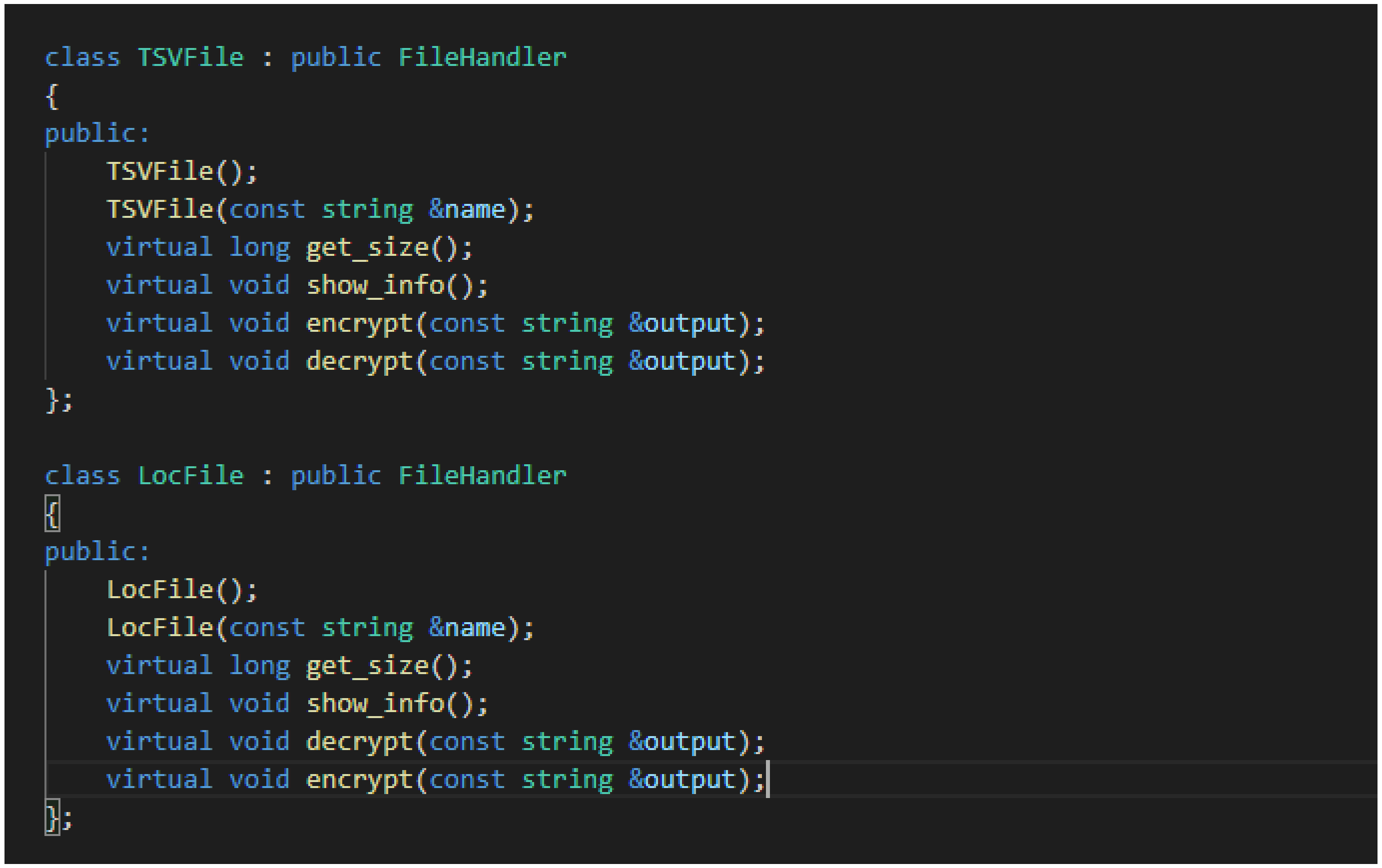
Declare class FileHandler, including 2 attribute name and ext are of type string. These two properties are protected members so that the outside cannot be accessed and subclasses can inherit.

The public methods include FileHandler() (create constructor), and virtual methods: get_size() (return long), void show_info(), void decrypt(), void encrypt() so that subclasses can inherit and override.



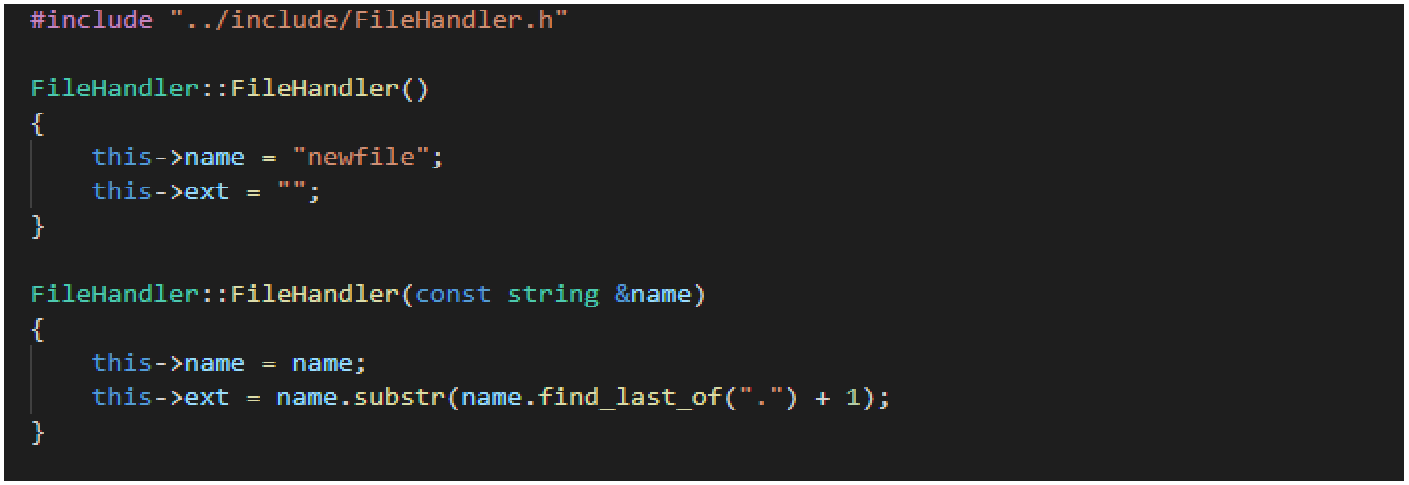
```
class FileHandler
{
protected:
    string name;
    string ext;
public:
    FileHandler();
    FileHandler(const string &name);
    virtual long get_size() = 0;
    virtual void show_info() = 0;
    virtual void decrypt(const string &output) {};
    virtual void encrypt(const string &output) {};
};
```

The TSVFile and Loc class inherits the FileHandler class:



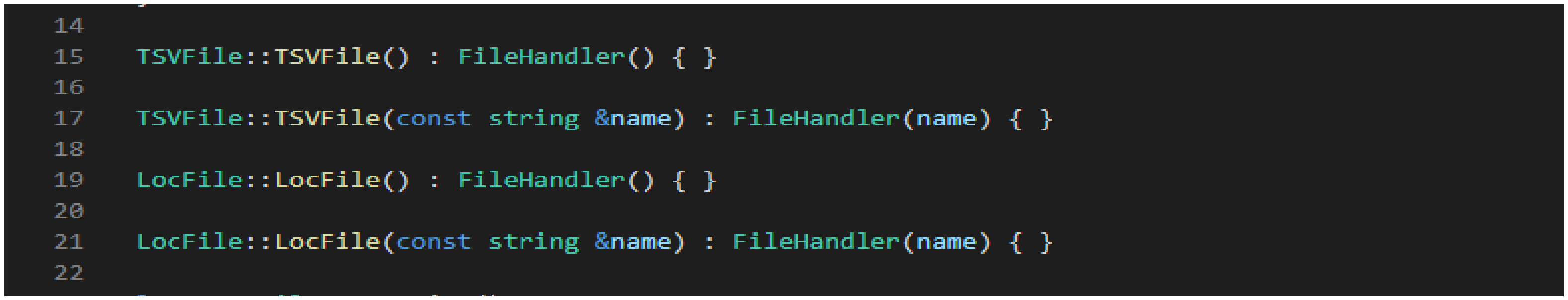
```
class TSVFile : public FileHandler
{
public:
    TSVFile();
    TSVFile(const string &name);
    virtual long get_size();
    virtual void show_info();
    virtual void encrypt(const string &output);
    virtual void decrypt(const string &output);
};

class LocFile : public FileHandler
{
public:
    LocFile();
    LocFile(const string &name);
    virtual long get_size();
    virtual void show_info();
    virtual void decrypt(const string &output);
    virtual void encrypt(const string &output);
};
```

In the cpp file, include the header file and initialize the constructor for the FileHandler class:



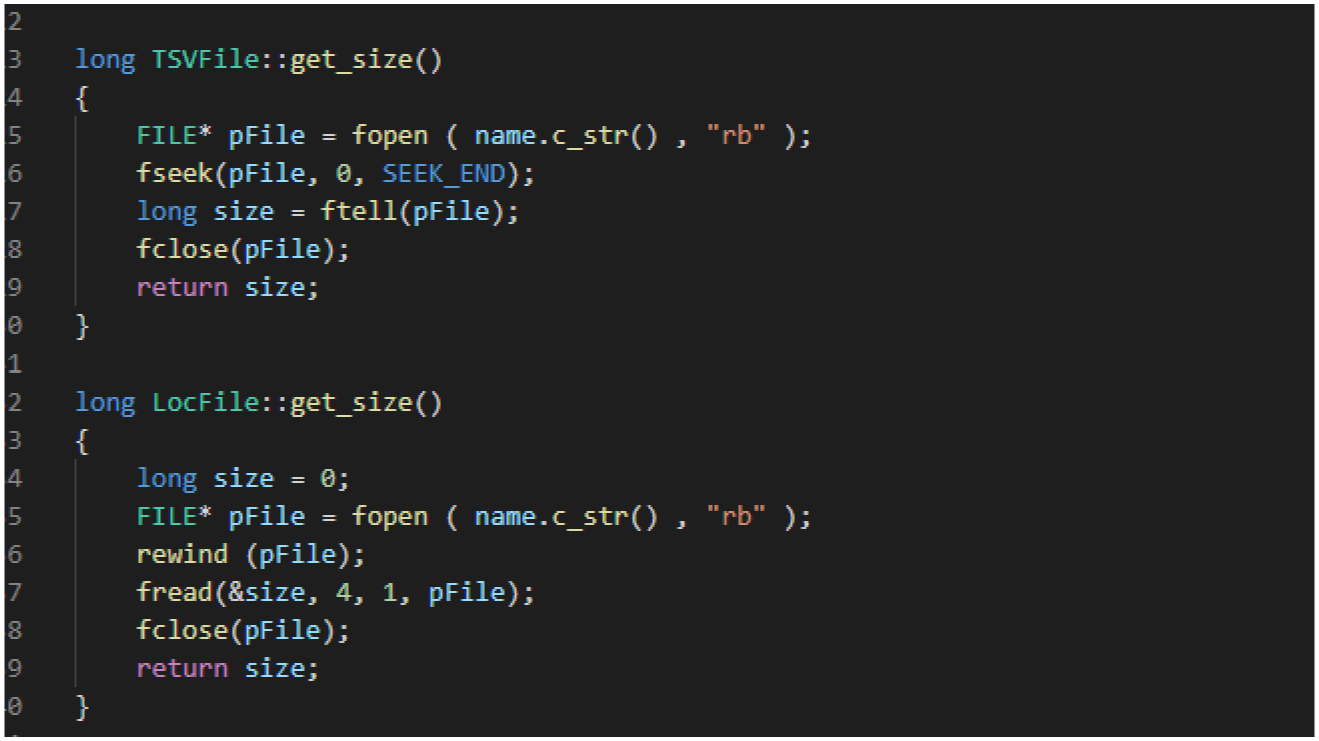
```cpp
#include "../include/FileHandler.h"

FileHandler::FileHandler()
{
    this->name = "newfile";
    this->ext = "";
}

FileHandler::FileHandler(const string &name)
{
    this->name = name;
    this->ext = name.substr(name.find_last_of(".") + 1);
}
```

Create constructors for two classes that inherit FileHandler, TSVFile and LocFile:



```cpp

TSVFile::TSVFile() : FileHandler() { }

TSVFile::TSVFile(const string &name) : FileHandler(name) { }

LocFile::LocFile() : FileHandler() { }

LocFile::LocFile(const string &name) : FileHandler(name) { }

```

get_size() methods in subclasses:

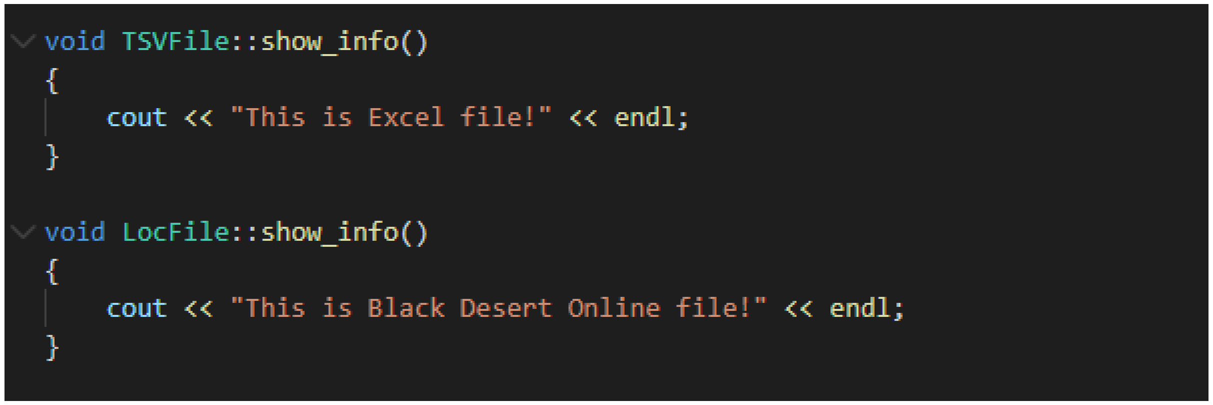


```
long TSVFile::get_size()
{
    FILE* pFile = fopen ( name.c_str() , "rb" );
    fseek(pFile, 0, SEEK_END);
    long size = ftell(pFile);
    fclose(pFile);
    return size;
}

long LocFile::get_size()
{
    long size = 0;
    FILE* pFile = fopen ( name.c_str() , "rb" );
    rewind (pFile);
    fread(&size, 4, 1, pFile);
    fclose(pFile);
    return size;
}
```

Because TSV is plain text file, its size is the file size.

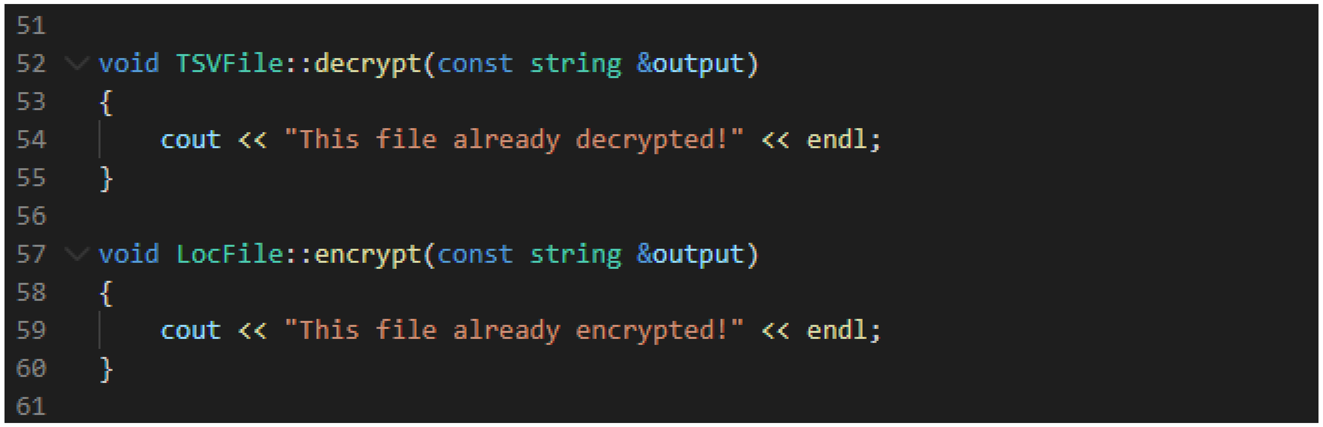
Loc is an encrypted and compressed file, so the uncompressed size is stored in the first 4 bytes.

Show_info() methods in subclasses will print the file information.



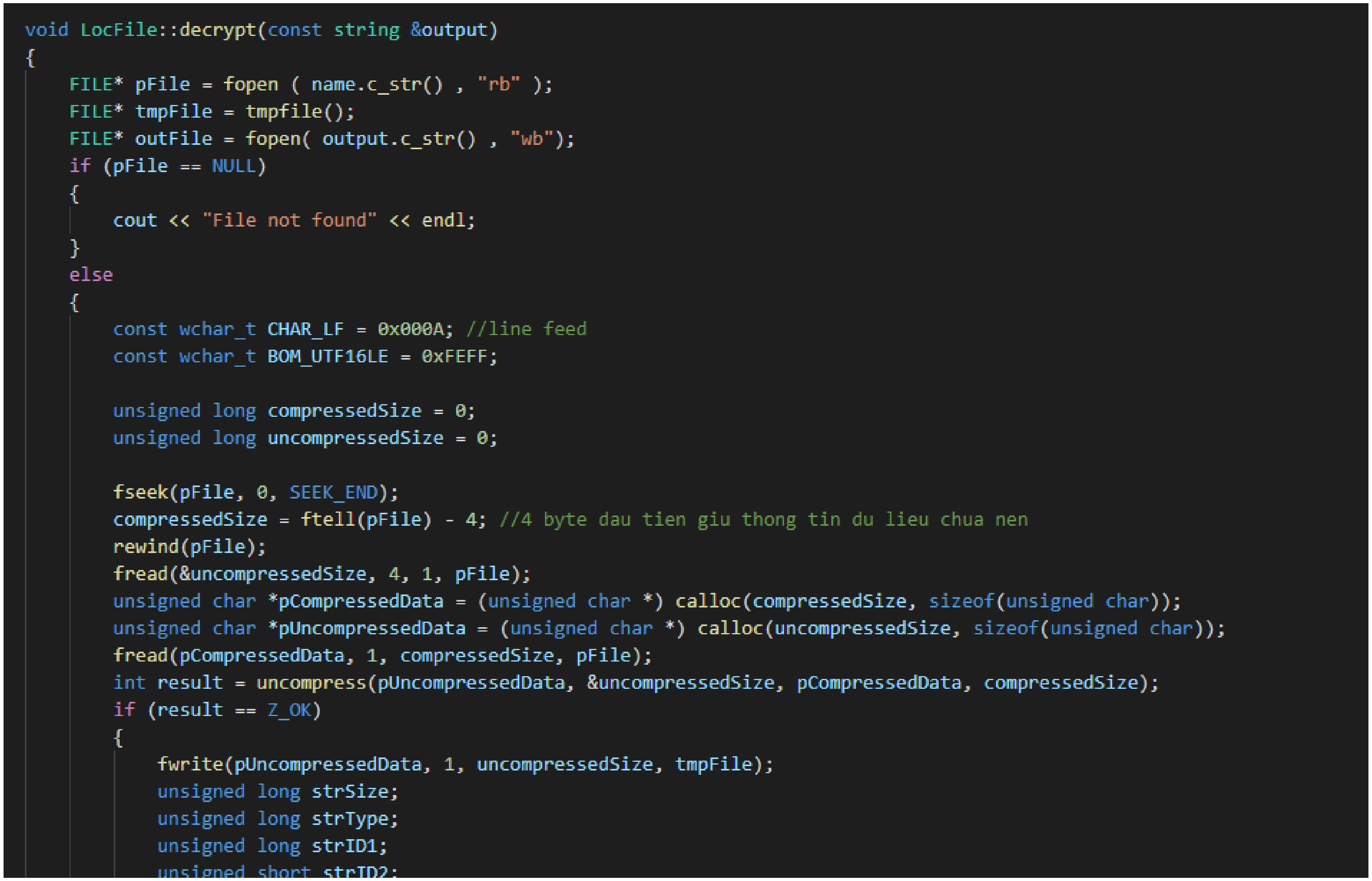
```cpp
void TSVFile::show_info()
{
    cout << "This is Excel file!" << endl;
}

void LocFile::show_info()
{
    cout << "This is Black Desert Online file!" << endl;
}
```

Because the TSV is an decrypted file, and the Loc file is an encrypted file, it cannot be decrypt / encrypt again.



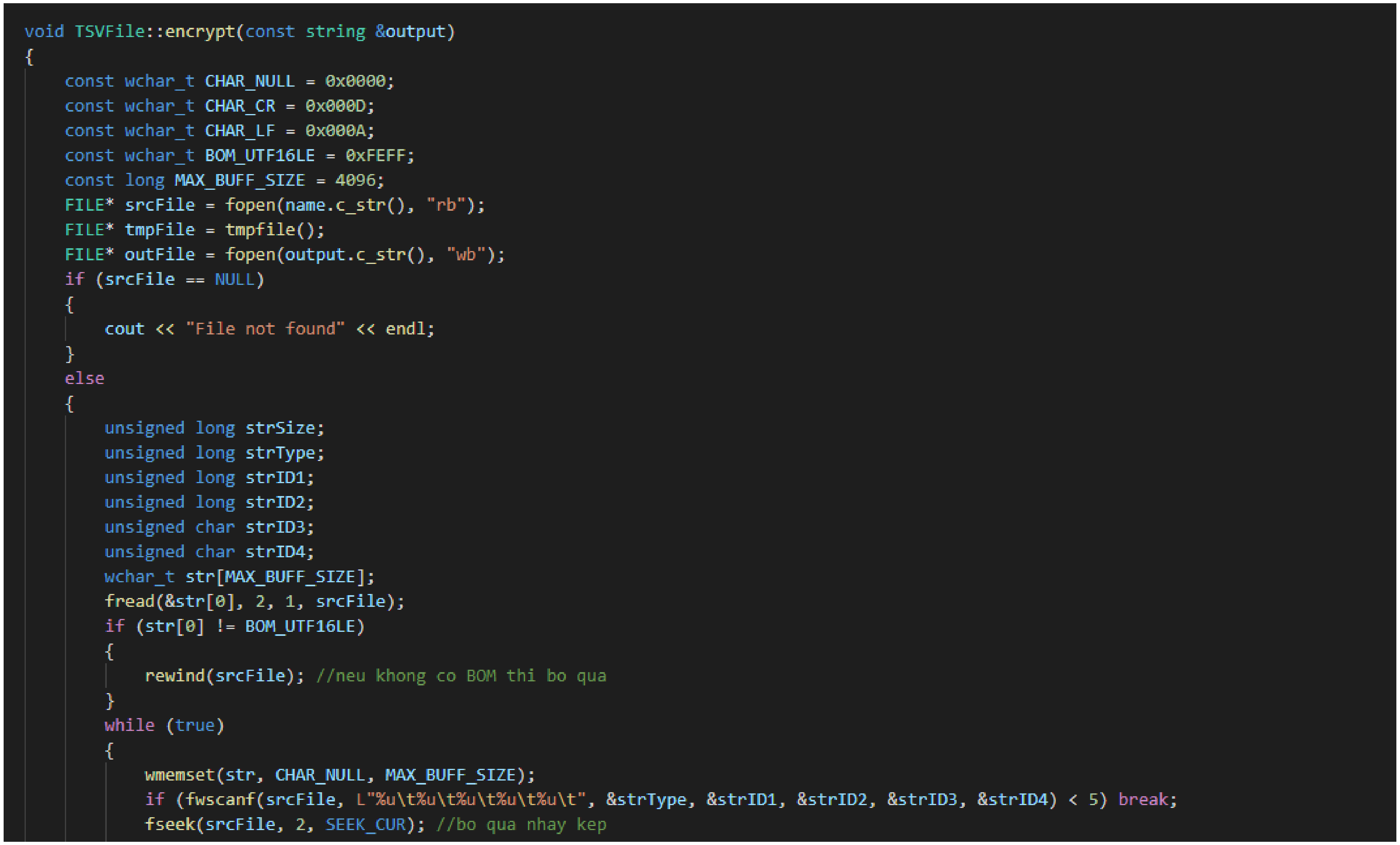
```cpp

void TSVFile::decrypt(const string &output)
{
    cout << "This file already decrypted!" << endl;
}

void LocFile::encrypt(const string &output)
{
    cout << "This file already encrypted!" << endl;
}

```

Decrypt() method in LocFile class, is responsible for decompressing (using the zlib library) and decrypt into plain text.



```
void LocFile::decrypt(const string &output)
{
    FILE* pFile = fopen ( name.c_str() , "rb" );
    FILE* tmpFile = tmpfile();
    FILE* outFile = fopen( output.c_str() , "wb");
    if (pFile == NULL)
    {
        cout << "File not found" << endl;
    }
    else
    {
        const wchar_t CHAR_LF = 0x000A; //line feed
        const wchar_t BOM_UTF16LE = 0xFEFF;

        unsigned long compressedSize = 0;
        unsigned long uncompressedSize = 0;

        fseek(pFile, 0, SEEK_END);
        compressedSize = ftell(pFile) - 4; //4 byte dau tien giu thong tin du lieu chua nen
        rewind(pFile);
        fread(&uncompressedSize, 4, 1, pFile);
        unsigned char *pCompressedData = (unsigned char *) calloc(compressedSize, sizeof(unsigned char));
        unsigned char *pUncompressedData = (unsigned char *) calloc(uncompressedSize, sizeof(unsigned char));
        fread(pCompressedData, 1, compressedSize, pFile);
        int result = uncompress(pUncompressedData, &uncompressedSize, pCompressedData, compressedSize);
        if (result == Z_OK)
        {
            fwrite(pUncompressedData, 1, uncompressedSize, tmpFile);
            unsigned long strSize;
            unsigned long strType;
            unsigned long strID1;
            unsigned short strID2;
```

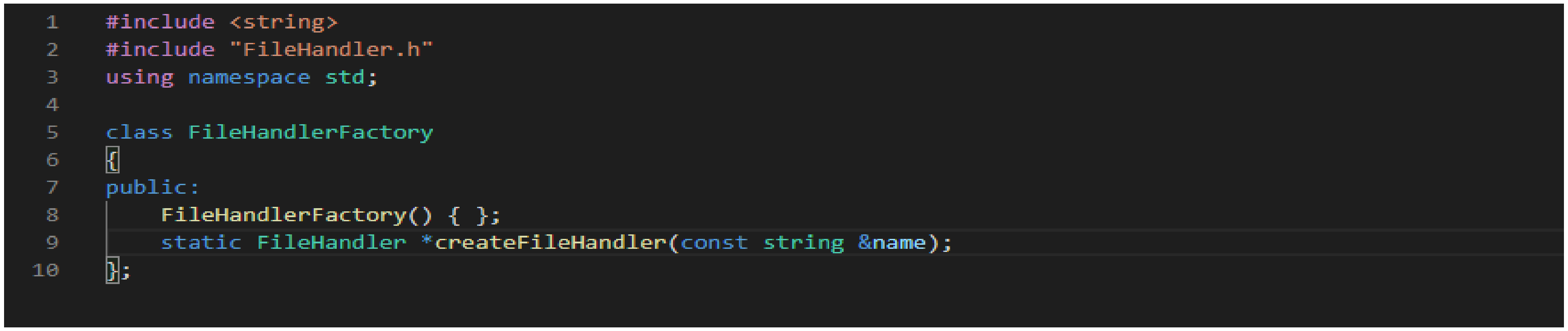
Encrypt() method in TSVFile class, is responsible for encrypt and compressing as the original file of the game.



```
void TSVFile::encrypt(const string &output)
{
    const wchar_t CHAR_NULL = 0x0000;
    const wchar_t CHAR_CR = 0x000D;
    const wchar_t CHAR_LF = 0x000A;
    const wchar_t BOM_UTF16LE = 0xFEFF;
    const long MAX_BUFF_SIZE = 4096;
    FILE* srcFile = fopen(name.c_str(), "rb");
    FILE* tmpFile = tmpfile();
    FILE* outFile = fopen(output.c_str(), "wb");
    if (srcFile == NULL)
    {
        cout << "File not found" << endl;
    }
    else
    {
        unsigned long strSize;
        unsigned long strType;
        unsigned long strID1;
        unsigned long strID2;
        unsigned char strID3;
        unsigned char strID4;
        wchar_t str[MAX_BUFF_SIZE];
        fread(&str[0], 2, 1, srcFile);
        if (str[0] != BOM_UTF16LE)
        {
            rewind(srcFile); //neu khong co BOM thi bo qua
        }
        while (true)
        {
            wmemset(str, CHAR_NULL, MAX_BUFF_SIZE);
            if (fwscanf(srcFile, L"%u\t%u\t%u\t%u\t%u\t", &strType, &strID1, &strID2, &strID3, &strID4) < 5) break;
            fseek(srcFile, 2, SEEK_CUR); //bo qua nhay kep
```

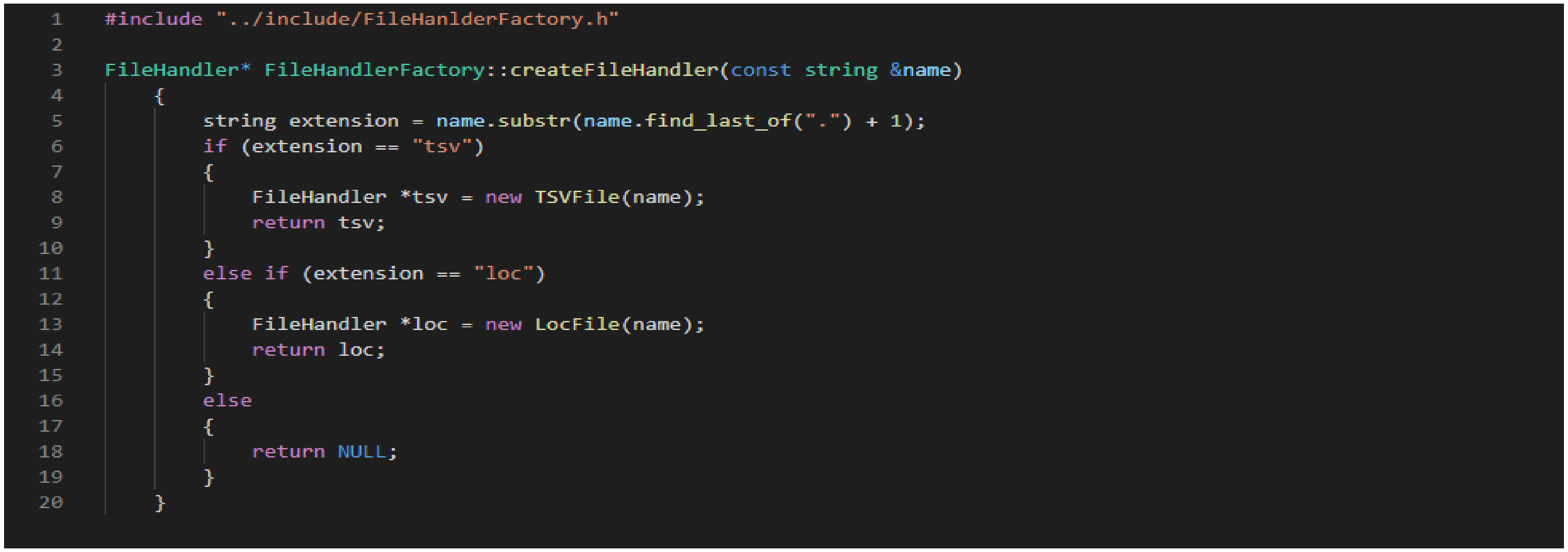
Next is the FileHandlerFactory class, which is responsible for creating subclasses according to the file extension, temporarily the program has only two file types: “.tsv” and “.loc”.

FileHandlerFactory.h



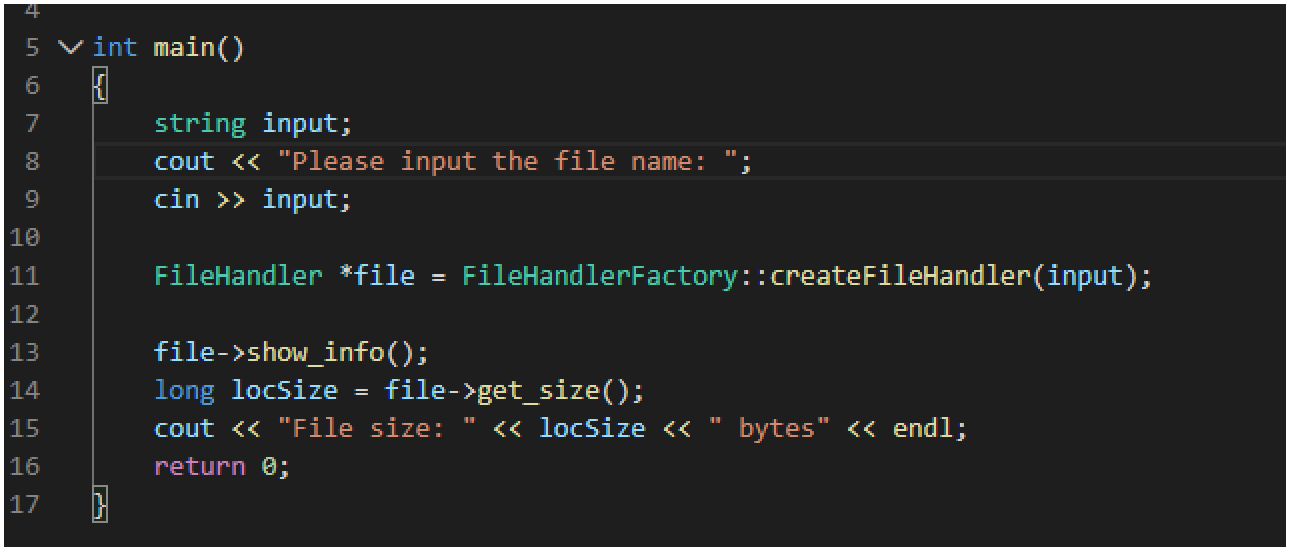
```cpp
#include <string>
#include "FileHandler.h"
using namespace std;

class FileHandlerFactory
{
public:
    FileHandlerFactory() { };
    static FileHandler *createFileHandler(const string &name);
};
```

FileHandlerFactory.cpp



```cpp
#include "../include/FileHanlderFactory.h"

FileHandler* FileHandlerFactory::createFileHandler(const string &name)
    {
        string extension = name.substr(name.find_last_of(".") + 1);
        if (extension == "tsv")
        {
            FileHandler *tsv = new TSVFile(name);
            return tsv;
        }
        else if (extension == "loc")
        {
            FileHandler *loc = new LocFile(name);
            return loc;
        }
        else
        {
            return NULL;
        }
    }
```

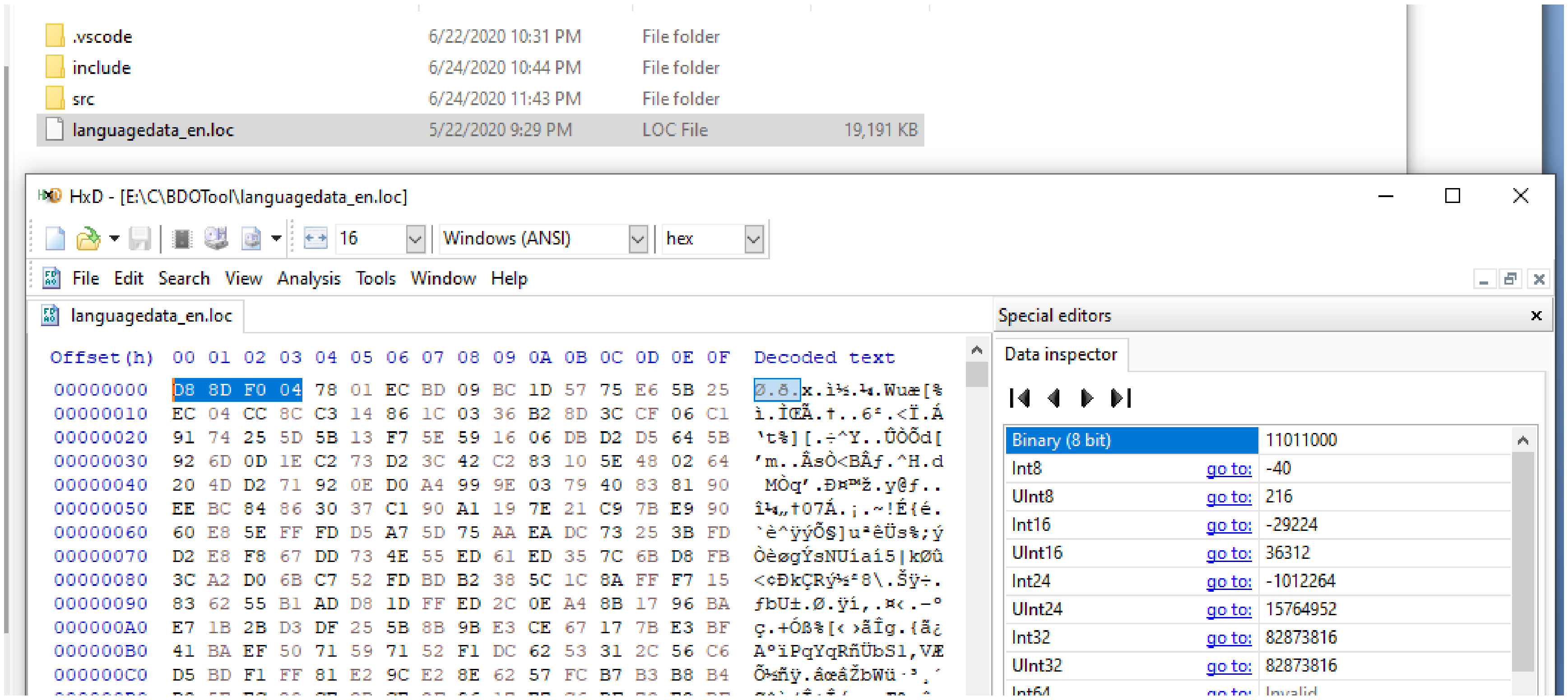
Main program:



```
int main()
{
    string input;
    cout << "Please input the file name: ";
    cin >> input;

    FileHandler *file = FileHandlerFactory::createFileHandler(input);

    file->show_info();
    long locSize = file->get_size();
    cout << "File size: " << locSize << " bytes" << endl;
    return 0;
}
```

3.2. Program screenshots

To test the program, we have the game's .loc file and the first 4 bytes are 828738816, indicating the uncompressed size is 828738816.



```
.vscode                 6/22/2020 10:31 PM   File folder
include                 6/24/2020 10:44 PM   File folder
src                     6/24/2020 11:43 PM   File folder
languagedata_en.loc     5/22/2020 9:29 PM    LOC File      19,191 KB
```

HxD - [E:\C\BDOTool\languagedata_en.loc]

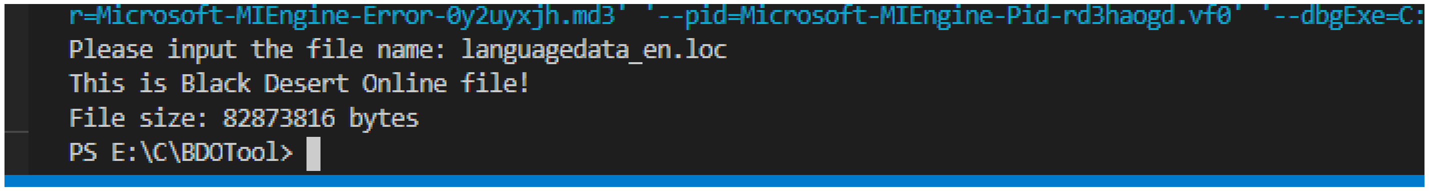
16 | Windows (ANSI) | hex

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  D8 8D F0 04 78 01 EC BD 09 BC 1D 57 75 E6 5B 25  Ø.ð.x.ì½.¼.Wuæ[%
00000010  EC 04 CC 8C C3 14 86 1C 03 36 B2 8D 3C CF 06 C1  ì.ÌŒÃ.†..6².<Ï.Á
00000020  91 74 25 5D 5B 13 F7 5E 59 16 06 DB D2 D5 64 5B  't%][.÷^Y..ÛÒÕd[
00000030  92 6D 0D 1E C2 73 D2 3C 42 C2 83 10 5E 48 02 64  'm..ÂsÒ<BÂƒ.^H.d
00000040  20 4D D2 71 92 0E D0 A4 99 9E 03 79 40 83 81 90   MÒq'.Ð¤™ž.y@ƒ..
00000050  EE BC 84 86 30 37 C1 90 A1 19 7E 21 C9 7B E9 90  î¼„†07Á.¡.~!É{é.
00000060  60 E8 5E FF FD D5 A7 5D 75 AA EA DC 73 25 3B FD  `è^ÿýÕ§]uªêÜs%;ý
00000070  D2 E8 F8 67 DD 73 4E 55 ED 61 ED 35 7C 6B D8 FB  ÒèøgÝsNUíaí5|kØû
00000080  3C A2 D0 6B C7 52 FD BD B2 38 5C 1C 8A FF F7 15  <¢ÐkÇRý½²8\.Šÿ÷.
00000090  83 62 55 B1 AD D8 1D FF ED 2C 0E A4 8B 17 96 BA  ƒbU±.Ø.ÿí,.¤‹.–º
000000A0  E7 1B 2B D3 DF 25 5B 8B 9B E3 CE 67 17 7B E3 BF  ç.+Óß%[‹›ãÎg.{ã¿
000000B0  41 BA EF 50 71 59 71 52 F1 DC 62 53 31 2C 56 C6  AºïPqYqRñÜbS1,VÆ
000000C0  D5 BD F1 FF 81 E2 9C E2 8E 62 57 FC B7 B3 B8 B4  Õ½ñÿ.âœâŽbWü·³¸´
```

Special editors — Data inspector

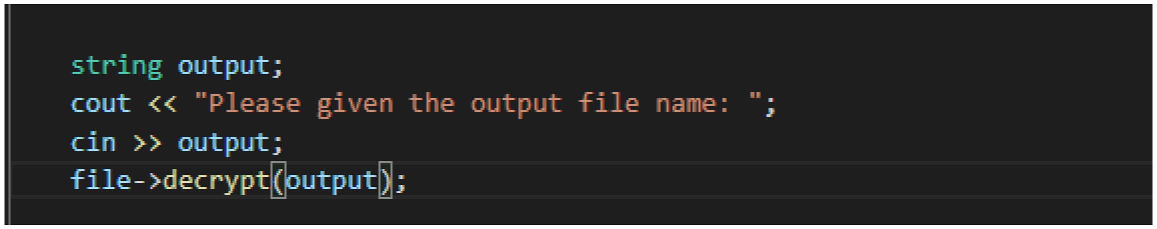
Binary (8 bit)		11011000
Int8	go to:	-40
UInt8	go to:	216
Int16	go to:	-29224
UInt16	go to:	36312
Int24	go to:	-1012264
UInt24	go to:	15764952
Int32	go to:	82873816
UInt32	go to:	82873816
Int64	go to:	Invalid

The program produces the expected results:



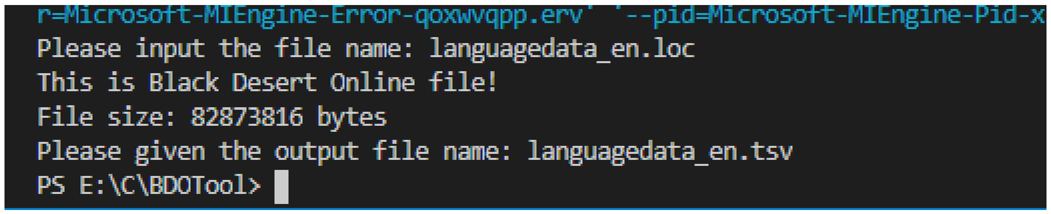
```
r=Microsoft-MIEngine-Error-0y2uyxjh.md3' '--pid=Microsoft-MIEngine-Pid-rd3haogd.vf0' '--dbgExe=C:
Please input the file name: languagedata_en.loc
This is Black Desert Online file!
File size: 82873816 bytes
PS E:\C\BDOTool>
```

Add some code to decrypt this file:



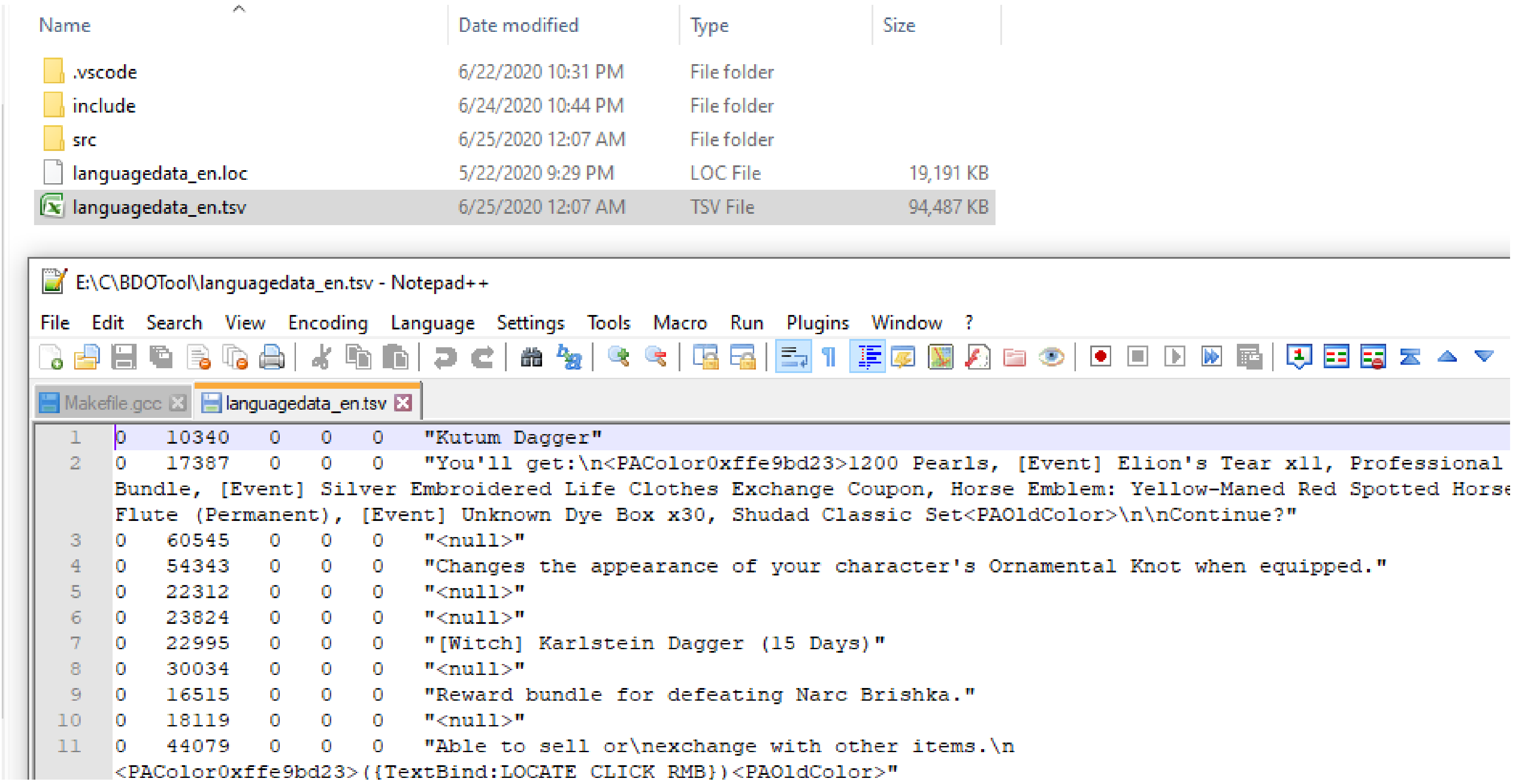
```
string output;
cout << "Please given the output file name: ";
cin >> output;
file->decrypt(output);
```

Then run the program again, we have a TSV file contains plain text (languagedata_en.tsv):



```
r=Microsoft-MIEngine-Error-qoxwvqpp.erv' '--pid=Microsoft-MIEngine-Pid-x
Please input the file name: languagedata_en.loc
This is Black Desert Online file!
File size: 82873816 bytes
Please given the output file name: languagedata_en.tsv
PS E:\C\BDOTool>
```

Decrypted file contains plain text:



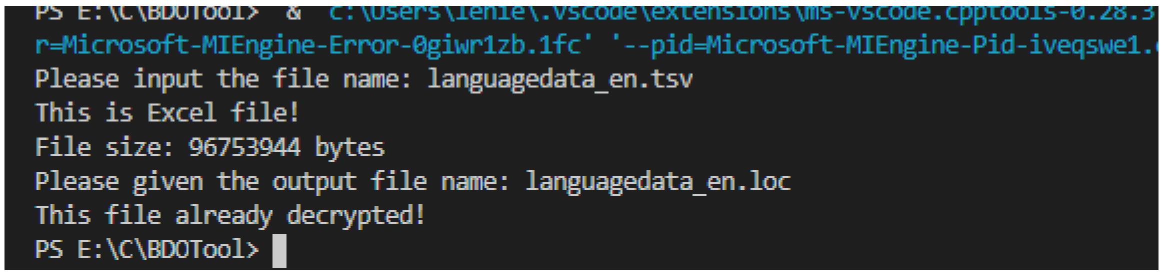
Name	Date modified	Type	Size
.vscode	6/22/2020 10:31 PM	File folder	
include	6/24/2020 10:44 PM	File folder	
src	6/25/2020 12:07 AM	File folder	
languagedata_en.loc	5/22/2020 9:29 PM	LOC File	19,191 KB
languagedata_en.tsv	6/25/2020 12:07 AM	TSV File	94,487 KB

E:\C\BDOTool\languagedata_en.tsv - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Makefile.gcc
languagedata_en.tsv

```
0    10340    0    0    0    "Kutum Dagger"
0    17387    0    0    0    "You'll get:\n<PAColor0xffe9bd23>1200 Pearls, [Event] Elion's Tear x11, Professional
Bundle, [Event] Silver Embroidered Life Clothes Exchange Coupon, Horse Emblem: Yellow-Maned Red Spotted Horse
Flute (Permanent), [Event] Unknown Dye Box x30, Shudad Classic Set<PAOldColor>\n\nContinue?"
0    60545    0    0    0    "<null>"
0    54343    0    0    0    "Changes the appearance of your character's Ornamental Knot when equipped."
0    22312    0    0    0    "<null>"
0    23824    0    0    0    "<null>"
0    22995    0    0    0    "[Witch] Karlstein Dagger (15 Days)"
0    30034    0    0    0    "<null>"
0    16515    0    0    0    "Reward bundle for defeating Narc Brishka."
0    18119    0    0    0    "<null>"
0    44079    0    0    0    "Able to sell or\nexchange with other items.\n
<PAColor0xffe9bd23>({TextBind:LOCATE CLICK RMB})<PAOldColor>"
```

Rerun the program with input "languagedata_en.tsv" (above decrypted file), the program recognized the extension tsv and called the TSVFile subclass, and the file could not continue decrypt.



```
PS E:\C\BDOTool>  & 'c:\Users\lenle\.vscode\extensions\ms-vscode.cpptools-0.28.3
r=Microsoft-MIEngine-Error-0giwr1zb.1fc' '--pid=Microsoft-MIEngine-Pid-iveqswe1.
Please input the file name: languagedata_en.tsv
This is Excel file!
File size: 96753944 bytes
Please given the output file name: languagedata_en.loc
This file already decrypted!
PS E:\C\BDOTool>
```

4. Discussion

4.1. Range of similar patterns

Some similar design patterns:

4.1.1. Builder pattern

The builder pattern is a design pattern designed to provide a flexible solution to various object creation problems in object-oriented programming. The intent of the Builder design pattern is to separate the construction of a complex object from its representation.

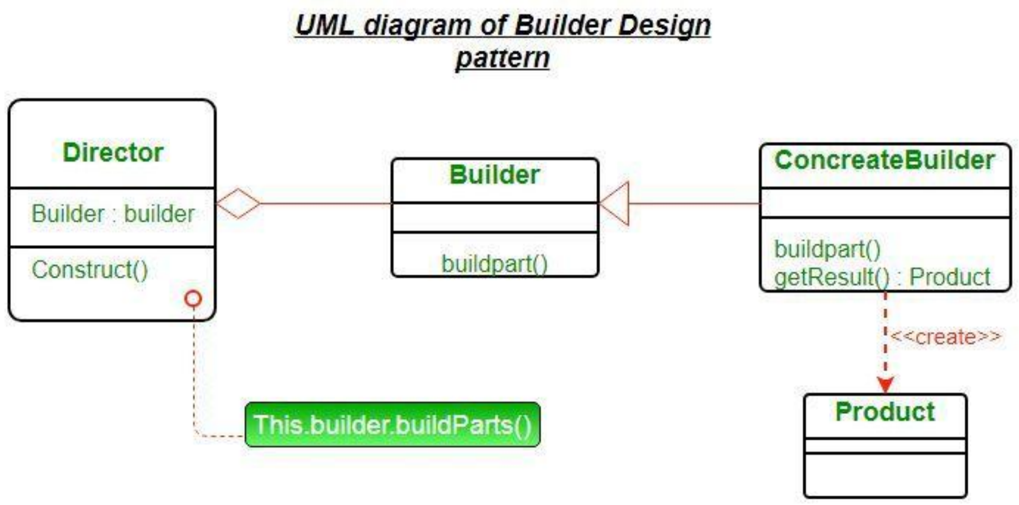


Figure 3: UML diagram of Builder pattern

The reason the Builder pattern does not match this scenario:

- Classes do not have many properties (only name and ext properties).
- The main purpose is to rely on file extensions that call subclasses with methods that are appropriate for each file type. Builder pattern does not meet this.

4.1.2. Singleton pattern

Singleton Pattern is a design pattern used to ensure that each class has only one instance and every interaction is through this instance.

Singleton Pattern provides a private constructor, which maintains a static property to refer to an instance of this Singleton class. It provides an additional static method that returns this static property.

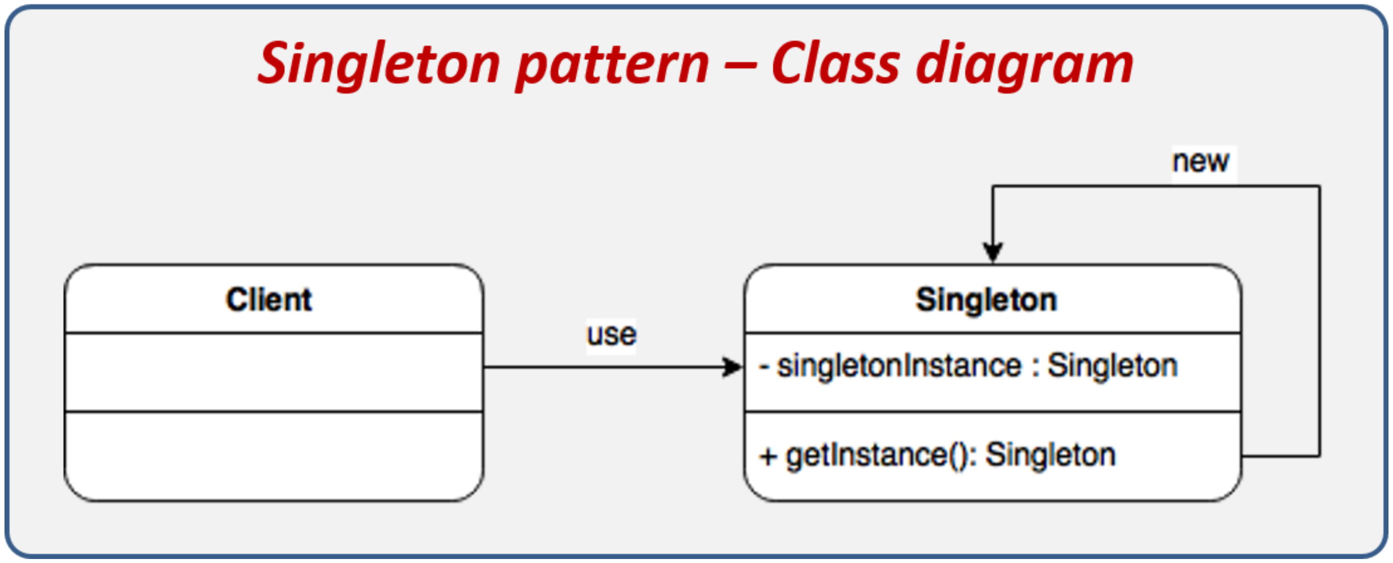

Singleton pattern – Class diagram
new
Client
use
Singleton
- singletonInstance : Singleton
+ getInstance(): Singleton

Figure 4: Singleton pattern class diagram

The reason the Singleton pattern does not match this scenario:

- The Singleton design pattern ensures that a class has only one instance. This purpose is not suitable for the scenario, because the program may have to work with many types of files at the same time.
- In multithreading mode, the Singleton design pattern may also work poorly: since getInstance() is unsafe thread, two threads can call the method to generate the object at the same time and two instances will be created.

4.2. Usage of pattern

Usage of Factory Method Pattern:

- When a class doesn't know what sub-classes will be required to create.
- When a class wants that its sub-classes specify the objects to be created.
- When the parent classes choose the creation of objects to its sub-classes.

Because the user can import different types of files, the program cannot know which subclass will be called, so using the Factory Method Pattern to create a class that manages the subclass creation is best suited to the scenario.

Advantages of Factory Method Pattern:

- Factory Method Pattern helps limit the dependency between creator and concrete products.
- The Factory Method Pattern helps gather product creation code into one place in the program, thus helping to track and manipulate.
- With Factory Method Pattern, we can freely add many new types of products to the program without changing the previous code base.

- For example: When we want the program to be able to work with some other file types such as Bin, ISO, Dat,... We just need to create additional subclasses to inherit FileHandler class.

Disadvantages of Factory Method Pattern:

- The code of the program can become more and more complex due to the need to use multiple classes to implement this pattern.