

# CSVファイル

- Comma Separated Valuesの略
- データを「,(コンマ)」で項目ごとに区切って、管理するデータファイル形式
- ひとつの行で、あるひとまとまりのデータを取り扱い、それらのデータを複数行で管理する

Name	HP	SP	Atk	Def	Gold	Exp	...
AAA	100	50	80	50	1000	0	
BBB	200	10	100	110	2500	800	
CCC	80	90	10	300	9000	9999	

# CSVファイル

- ゲームやアプリのデータを簡易的に管理すること利用されている(本格的な管理はデータベースが使用されている)
- CSVファイルは表計算アプリ(Excel)やメモ張等で読み書き可能なテキストファイル
- C++でCSVファイルを扱う専用の関数はないため、ファイルの内容をコンマを基準に分解して読み込む機能を自前で実装する必要がある

# CSVファイル

- C++作業フォルダ内に**CSV**フォルダを作成

```
mkdir CSV  
cd CSV
```

- **main.cpp**を作成する

```
copy nul main.cpp
```

# CSVファイル

- CSVファイルの作り方  
以下の2通りの方法がある

① メモ帳などのテキストエディタで作成

② Excelを使って表形式で入力したものをCSV形式で保存する

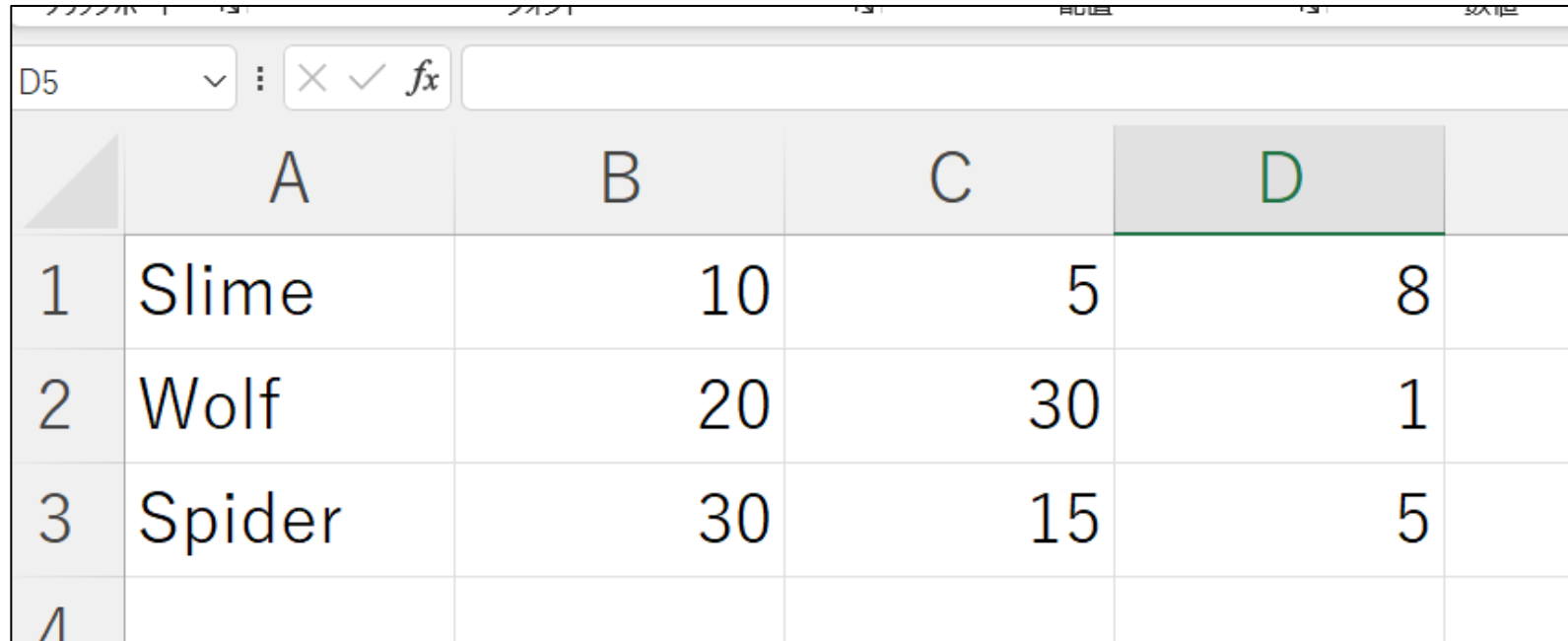
今回は②の方法で行う

# CSVファイル

- CSVファイルの作り方

① Excelを起動する

② セルにデータを入力



The screenshot shows an Excel spreadsheet with a table containing 5 columns and 4 rows of data. The columns are labeled A, B, C, D, and E. The rows are numbered 1, 2, 3, and 4. The data is as follows:

	A	B	C	D	E
1	Slime	10	5	8	
2	Wolf	20	30	1	
3	Spider	30	15	5	
4					

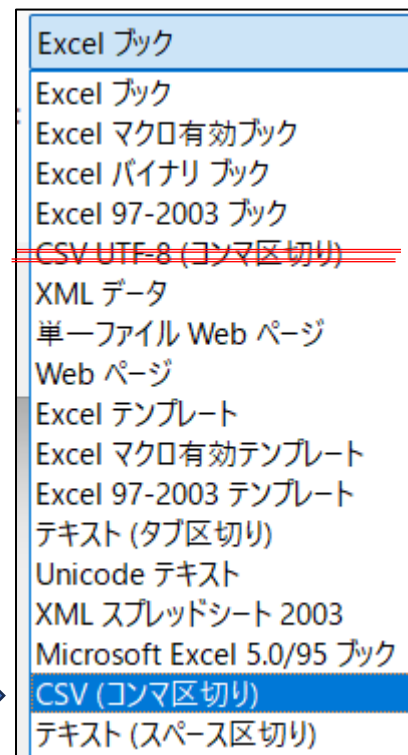
# CSVファイル

## • CSVファイルの作り方

③ [ファイル] > [名前をつけて保存] > [参照]  
から、C++作業フォルダ内の「**CSV**」フォルダ  
を選択する

④ ファイル名を「**enemy\_list**」と入力

⑤ ファイルの種類を「**CSV(コンマ区切り)**」にする  
※**CSV UTF-8**ではない



# CSVファイル

- CSVファイルの作り方

- ⑥ コマンドプロンプトから

`type enemy_list.csv`

と入力して、CSVファイルの内容を表示してコンマ区切りのファイルになっていればOK

# CSVファイル

- CSVファイルの読み込み方法

- ① CSVファイルをオープンする
- ② ファイル末尾に到達するまで一行ずつ読み込む
- ③ 読み込んだ一行をコンマを基準に項目分けする
- ④ 各項目を配列に格納する



# CSVファイル

- ファイルを開く方法 その①  
インスタンス生成と同時にファイルを開く

```
#include <fstream>  
std::ifstream インスタンス名(ファイル名);
```

```
例) std::ifstream ifs( "C:¥test.csv" );
```

# CSVファイル

- ファイルを開く方法 その②  
インスタンス生成後にopen関数でファイルを開く

```
#include <fstream>
std::ifstream インスタンス名;
インスタンス名.open(ファイル名);
```

```
例) std::ifstream ifs;
    ifs.open("C:¥test.csv");
```

# CSVファイル

- エラーチェック

ファイルを開く際に、ファイルが存在しない、ファイルを開く権限がないといったことが生じた際は、以降の処理を停止する必要がある

```
if(ifs.fail()) {  
    //エラーが生じたときの処理を記述  
} else {  
    //ファイルオープンに成功したときの処理  
}
```

# CSVファイル

main.cpp (CSV)

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;
int main() {
    string filename = "enemy_list.txt";
    ifstream ifs( filename ); //間違ったファイル名の場合
    if (ifs.fail()) {
        cout << "ファイルを開けません!!" << endl;
        return -1; //エラーの場合、戻り値を-1にして即時終了
    }
    return 0;
}
```

# CSVファイル

- プログラム終了コードの確認方法

コマンドプロンプトから

`echo %ERRORLEVEL%`

と入力すれば、直前に実行したプログラムの終了コード  
を取得可能

# CSVファイル

- ファイルからの読み出し処理

`std::getline(ストリーム, 文字列変数)`

ストリームから、改行までの一文を読み込んで、  
文字列変数に格納する

例) `string text;`  
`getline(ifs, text);`  
`//読み込んだ内容がtext内に格納`

# CSVファイル

main.cpp (CSV)

```
int main() {  
    string filename = "enemy_list.csv";  
    ifstream ifs( filename );  
    if (ifs.fail()) {  
        cout << "ファイルを開けません!!" << endl;  
        return -1;  
    }  
    → string text;  
    → getline(ifs, text); //一行だけファイルから読み込む  
    → cout << text << endl;  
    → ifs.close(); //ファイルを閉じる  
    return 0;  
}
```

# CSVファイル

main.cpp (CSV)

```
int main() {  
    string filename = "enemy_list.csv";  
    ifstream ifs( filename );  
    if (ifs.fail()) {  
        cout << "ファイルを開けません!!" << endl;  
        return -1;  
    }  
    string text;  
    while(getline(ifs, text)) { //ファイル末尾まで  
        cout << text << endl; //一行ずつ読み込む  
    }  
    ifs.close();  
    return 0;  
}
```



# CSVファイル

- ファイルからの読み出し処理

`std::getline(ストリーム, 変数, 区切り文字)`

ストリームから、区切り文字(もしくは改行)で分割しながら読み込んで、文字列変数に格納する

例) `string text;`  
`getline(ifs, text, ',');`

# CSVファイル

main.cpp (CSV)

```
int main() {  
    string filename = "enemy_list.csv";  
    ifstream ifs( filename );  
    if (ifs.fail()) {  
        cout << "ファイルを開けません!!" << endl;  
        return -1;  
    }  
    string text;  
    while(getline(ifs, text, ',')) { //コンマで分割  
        cout << text << endl;      //して文字列を読み込む  
    }  
    ifs.close();  
    return 0;  
}
```

# CSVファイル

- 実行結果を見ると、CSVファイルの先頭からコンマを区切りとして文字列を読み込むことができる！
- しかし、プログラムからはCSVファイルがどちらになっているかはわからない

```
Slime,10,5,8  
Wolf,20,30,1  
Spider,30,15,5
```

or

```
Slime,10,5,8,Wolf,20,30,1,Spider,30,15,5
```

```
while(getline(ifs, text, ','))
```

## 実行結果

Slime

10

5

8

Wolf

20

30

1

Spider

30

15

5

# CSVファイル

- CSVファイルに項目を追加する際は、各Enemyが一行ずつわかれていたほうが見やすくて、管理もしやすい

```
Slime,10,5,8,5,5  
Wolf,20,30,1,5,2  
Spider,30,15,5,2,3
```

赤字が新規追加パラメータ

データが各行にわかれていると  
データの末尾に追加するだけでよい！

```
Slime,10,5,8,5,5,Wolf,20,30,1,5,2,Spider,3...
```

データが横一線に並んでいると途中途中でデータを挿入する手間が必要・・・

# CSVファイル

- CSVファイルの読み込み方法を改良する

① CSV(ファイルストリーム)から一行丸ごと読み込む

```
Slime,10,5,8
```

② 読み込んだ一行を文字列ストリームに変換して  
getlineを使いコンマで区切って項目に分割する

③ 分割してできた文字列を二次元配列の各要素へ  
順次格納していく。一行分終われば①へもどる

# CSVファイル

- CSVファイルの読み込み方法を改良する

① CSV(ファイルストリーム)から一行丸ごと読み込む

② 読み込んだ一行を文字列ストリームに変換して  
getlineを使いコンマで区切って項目に分割する

`Slime, 10, 5, 8` → `istringstream`  
(`#include <sstream>`)

③ 分割してできた文字列を二次元配列の各要素へ  
順次格納していく。一行分終われば①へもどる

# CSVファイル

- CSVファイルの読み込み方法を改良する

① CSV(ファイルストリーム)から一行丸ごと読み込む

② 読み込んだ一行を文字列ストリームに変換して  
getlineを使いコンマで区切って項目に分割する

③ 分割してできた文字列を二次元配列の各要素へ  
順次格納していく。一行分終われば①へもどる

Slime 10 5 8 → vector配列  
(#include <vector>)

# CSVファイル

main.cpp (CSV)

```
int main() {  
    string filename = "enemy_list.csv";  
    ifstream ifs( filename );  
    if (ifs.fail()) {  
        cout << "ファイルを開けません!!" << endl;  
        return -1;  
    }  
    string text;  
    ➡ getline(ifs, text); //textの内容: Slime,10,5,8  
    ➡ istringstream iss(text); //文字列をストリームに変換  
    ➡ getline(iss, text, ','); //ストリームを分割してtextへ  
    cout << text << endl; //Slime
```



# CSVファイル

main.cpp (CSV)

```
string filename = "enemy_list.csv";
ifstream ifs( filename );
if (ifs.fail()) {
    cout << "ファイルを開けません!!" << endl;
    return -1;
}
string text;
getline(ifs, text); //textの内容: Slime,10,5,8
istringstream iss(text); //文字列をストリームに変換
while(getline(iss, text, ',')) { //ストリーム末尾まで
    cout << text << endl;      //繰り返し処理する
}
```

# CSVファイル

## main.cpp (CSV)

```
→ vector<string> vEne{};
   string text;
   getline(ifs, text); //textの内容: Slime,10,5,8
   istream<string> iss(text); //文字列をストリームに変換
   while(getline(iss, text, ',')) { //ストリーム末尾まで
                                   //繰り返し処理する
                                   //配列に各項目を格納
→     vEne.push_back(text);
   }
→ for (const auto& d: vEne) {
→     cout << d << endl;
→ }
```

# CSVファイル

## main.cpp (CSV)

```
→ vector<vector<string>> vEne{}; //二次元配列化
   string text;
   getline(ifs, text); //textの内容: Slime,10,5,8
   istream<string> iss(text); //文字列をストリームに変換
→ vEne.resize(1); //データを格納できる行を増やす
   while(getline(iss, text, ',')) { //ストリーム末尾まで
                                   //繰り返し処理する
→     vEne[0].push_back(text);      //配列に各項目を格納
   }
→ for (const auto& d: vEne[0]) {
   cout << d << endl;
}
```

# CSVファイル

## main.cpp (CSV)

```
vector<vector<string>> vEne{}; //二次元配列化
string text;
→ int j = 0; //行番号を管理する変数
while(getline(ifs, text)) { //ファイル末尾まで読出し
    istream iss(text);
    → vEne.resize(j+1); //データを格納できる行を増やす
    while(getline(iss, text, ',')) {
        → vEne[j].push_back(text); //配列に各項目を格納
    }
    → for (const auto& d: vEne[j]) { //各行の内容を表示
        cout << d << endl;
    }
    → j++;
}
```

# CSVファイル

- 複数行にわたるCSVファイルを読み込んで配列に格納することができた

・・・が、すべての要素が文字列(数字も文字列)として格納されている

```
vector<vector<string>> vEne
```

vEne	[][0]	[][1]	[][2]	[][3]
[0][ ]	Slime	10	5	8
[1][ ]	Wolf	20	30	1
[2][ ]	Spider	30	15	5

# CSVファイル

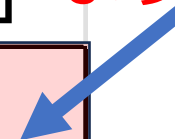
- 複数行にわたるCSVファイルを読み込んで配列に格納することができた

・・・が、すべての要素が文字列(数字も文字列)として格納されている

```
vector<vector<string>> vEne
```

vEne	[][0]	[][1]	[][2]	[][3]
[0][]	Slime	10	5	8
[1][]	Wolf	20	30	1
[2][]	Spider	30	15	5

整数として扱える  
ようにする



# CSVファイル

- そこで文字列は`string`、数値は`int`値で格納できるように変更を行う
- `Enemy`クラスを定義して、そのメンバ変数にCSVファイルからデータを格納していく
- 複数体のEnemy情報に対応できるように、Enemyクラスは`vector`で配列化する

# CSVファイル

## main.cpp (CSV)

```
class Enemy {
private:
    string m_name;           //メンバ変数
    int m_hp, m_atk, m_def;
public: //コンストラクタのイニシャライザでメンバ変数を初期化
    Enemy(): m_name(0), m_hp(0), m_atk(0), m_def(0) {};
    Enemy(string name, int hp, int atk, int def)
        : m_name(name), m_hp(hp), m_atk(atk), m_def(def) {};
    ~Enemy() = default; //デフォルトデストラクタ
    string getName() { return m_name; } //各メンバのゲッター
    int getHp() { return m_hp; }
    int getAtk() { return m_atk; }
    int getDef() { return m_def; }
};
```

main関数の前にクラス定義を追加する



# CSVファイル

## main.cpp (CSV)

```
→ vector<string> vEne{}; //一次元配列にもどす
→ vector<Enemy*> pEne{}; //Enemyクラスを格納する配列
string text;
int j = 0; //行番号を管理する変数
while(getline(ifs, text)) { //ファイル末尾まで読出し
    istringstream iss(text);
    → vEne.resize(j+1); //この行を削除
    while(getline(iss, text, ',')) {
        → vEne[j].push_back(text); //[j]を削除
    }
    → pEne.push_back(new Enemy(vEne[0], stoi(vEne[1]),
        stoi(vEne[2]), stoi(vEne[3]))); //整数に変換
    → vEne.clear(); //一行分格納したので全要素を削除
    j++;
}
```

# CSVファイル

## main.cpp (CSV)

```
    pEne.push_back(new Enemy(vEne[0], stoi(vEne[1]),  
        stoi(vEne[2]), stoi(vEne[3]))); // 整数に変換  
    vEne.clear(); // 一行分格納したので全要素を削除  
    j++;  
}  
→ for (const auto& p : pEne) { // pはインスタンスのアドレス  
→     cout << p->getName() << '¥t' << p->getHp()  
→     << '¥t' << p->getAtk() << '¥t' << p->getDef()  
→     << endl;  
→ }  
ifs.close();  
return 0;  
}
```

# CSVファイル

- ファイルへの書き込み

`ofstream` インスタンス名(ファイル名,モード);

`ios::app`は上書きモード

例) `ofstream ofs("aaa.csv", ios::out);`

`aaa.csv`という名前ファイルを新規作成する

# CSVファイル

- ファイルへの書き込み

```
ofstream インスタンス名;  
インスタンス名.open(ファイル名, モード);
```

ios::appは追記モード

```
例) ofstream ofs;  
    ofs.open("aaa.csv", ios::app);
```

aaa.csvというファイルの末尾にデータ追加

# CSVファイル

## main.cpp (CSV)

```
ifs.close();  
→ string ofilename = "enemy_list2.csv"; //出力ファイル名  
→ ofstream ofs(ofilename, ios::out); //上書モードで開く  
→ if (ofs.fail()) {  
→     cout << "ファイルを開けません!!" << endl;  
→     return -1; //エラー時は強制終了  
→ }  
→ for (const auto& p : pEne) {  
→     ofs << p->getDef() << ',' << p->getAtk() << ','  
→         << p->getHp() << ',' << p->getName() << endl;  
→ } //項目をDef,Atk,Hp,Nameの順で出力  
→ ofs.close();
```

# CSVファイル

## main.cpp (CSV)

```
for (const auto& p : pEne) {  
    ofs << p->getDef() << ',' << p->getAtk() << ','  
        << p->getHp() << ',' << p->getName() << endl;  
} //項目をDef, Atk, Hp, Nameの順で出力  
ofs.close(); //ファイルを閉じる  
→ ofs.open(ofilename, ios::app); //追記モードで開く  
→ ofs << "10,10,15,Rat" << endl; //ファイル末尾に追加  
→ ofs.close(); //ファイルを閉じる
```

書き込みモードを切り替える場合は、一旦閉じる(close)してから再度開く(open)する必要がある

# CSVファイル

## • CSVまとめ

- データを簡易的に管理することのできるファイル形式
- Excelやメモ帳等を使って簡単に編集可能
- 一般的に「,(コンマ)」で各項目を区切る
- C++ではCSVを読み込む関数がないため、自分で処理を実装する必要がある
- CSVは文字列として記録されているため、実際に値を利用するためには型変換する必要がある
- CSVを出力するのはファイルストリームに対して、  
出力したいデータ + ', ' をつけてあげるだけでよい