Fileクラス

ファイルの読み書きを行うクラスの作り方、バイナリで読み込みそのまま保持する。

※細かいエラー制御などは考えない

利用するC++標準ヘッダ

ファイルの読み書きには、C++標準の`<fstream>`を使う

File.h

```
class File {
public:
    // コンストラクタ、ファイル名からファイルをロード
    File(const char* filename);
    // デストラクタ
    ~File();

    // ファイルサイズを返す
    int size() const;

    // 読み込んだデータの先頭アドレスを返す
    const char* data() const;

    // 指定した場所にあるデータをunsignedとして取り出す
    unsigned getUnsigned(int position) const;

    // ファイルを書き込む
    static void write(const char* filename, const char* data, int size);
    void write(const char*) const;

private:
    // ファイルサイズ
    int mSize;

    // 読み込んだデータの先頭を指すポインタ
    char* mData;
};
```

File.cpp

準備

`File.cpp`を用意したらとりあえず使うヘッダと使う`namespace`を記述

```
#include <fstream>
using namespace std;
```

ファイル読み込み

ファイルの読み込みはコンストラクタで行う。

```
File::File(const char* filename)
    : mSize(0)
    , mData(0)
{
    ifstream in(filename, ifstream::binary);

    if (in) {
        in.seekg(0, ifstream::end);
        mSize = static_cast<int>(in.tellg());
        in.seekg(0, ifstream::beg);
        mData = new char[mSize];
        in.read(mData, mSize);
    }
}
```

1. `ifstream`でファイルを開く

```
ifstream in(filename, ifstream::binary);
```

第一引数は開くファイル名(パス)、第二引数はモードで、バイナリモードを指定

2.ファイルサイズを取得する

ファイルサイズを取得するには`seekg`でファイルの末尾まで移動し、移動した先を`tellg`で取得する。

サイズを取得したらファイルの先頭に移動しておく。

```
in.seekg(0, ifstream::end);           // ファイル末尾に移動
mSize = static_cast<int>(in.tellg()); // 移動先の位置をファイルサイズとして保存
in.seekg(0, ifstream::beg);           // ファイルの先頭へ移動
```

3. ファイルを読み込む

必要なメモリを確保してファイルの中身を読み込む

```
mData = new char[mSize]; // ファイルサイズ分のメモリを確保
in.read(mData, mSize);   // 読み込み先と読み込みサイズを指定して読み込む
```

デストラクタ

デストラクタでは確保しておいたデータ用のメモリをちゃんと破棄すること

```
File::~File() {
    delete[] mData;
}
```

getter

ファイルサイズとファイルのデータにアクセスできるようにgetterを用意しておく

```
int File::size() const {
    return mSize;
}

const char* File::data() const {
    return mData;
}
```

ファイルを読み込んだデータが勝手に書き換えられたら困るので、`mData`は`const`を付けたポインタを返している。

ファイルのデータを取得するときも`const`のポインタじゃないと受け取れないので、呼び出す側に`const`を強制できる。

```
// const char*じゃないと受け取れない
const char* data = file.data();
```

任意のデータを符号なし整数(4byte)で取得する

引数で指定された場所から4byte分のデータを論理和で結合して、4byte整数として取得する。

```
unsigned File::getUnsigned(int p) const {
    const unsigned char* up;
    up = reinterpret_cast<const unsigned char*>(mData);
    unsigned r = up[p];
    r |= up[p + 1] << 8;
    r |= up[p + 2] << 16;
    r |= up[p + 3] << 24;
    return r;
}
```

ファイル書き込み

バイナリデータをそのままファイルに書き込む機能、どこからでも使えるようにstatic版を用意しておき、楽をするためにFileクラス用にも実装しておく

```
// static版、どこからでも使える
void File::write(const char* filename, const char* data, int size){
    ofstream out(filename, ofstream::binary);
    if (out) {
        out.write(data, size);
    }
}

// インスタンスメソッド、mDataの内容をファイルに保存する
void File::write(const char* filename) const{
    File::write(filename, mData, mSize);
}
```

使い方

```
// file読み込み
File file("hoge.txt");

// サイズを取得
file.size();

// データの先頭のポインタを取得
file.data();

// 先頭1byte目のデータが欲しければ
char c = file.data()[0];

// データの任意の場所を符号なし整数として取得
unsigned u = file.getUnsigned(10);

// ファイルに書き込む
file.write("hoge2.txt");

// ファイルに書き込む static版
File::write("static.txt", "static", 6);
```