

mapクラス

- **連想配列**と呼ばれるデータ構造
- 配列を添え字番号でなく、**【キー(key)】**というデータでアクセス
- 配列の要素は**【値(value)】**
- **【キー】**と**【値】**のデータ型は、基本データ型(intやstring等)が使用可能
- **【キー】**と**【値】**は一対一で対応

```
std::map <データ型, データ型> インスタンス名
```

mapクラス

- mapクラスのメンバ関数

- `size()` : 全要素をカウント
- `clear()` : 配列要素を全消去
- `empty()` : 配列が空かどうかをチェック
- `erase()` : 指定のデータを削除
- `insert()` : キーと値のペアを追加
- `emplace()` : キーと値のペアを追加
- `find()` : 指定したキーのイテレータを返す
- `count()` : 指定したキーにマッチする要素数を返す
- `at()` : 指定したキーの値を返す

mapクラス

- 教科書P241~242 Sample608
- C++作業フォルダ内にSample608フォルダを作成
`mkdir Sample608`
`cd Sample608`
- main.cppを作成してVisualStudioで編集
`copy nul main.cpp`

mapクラス

main.cpp (Sample608)

```
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
    map <string, int> score; //キーがstring、値がintのmapを宣言
                             //キーがTom、値が100
    score["Tom"] = 100;
    score["Bob"] = 80;
    score["Mike"] = 76;
    cout << "Tomの点数は" << score["Tom"] << "点" << endl;
    cout << "Bobの点数は" << score["Bob"] << "点" << endl;
    cout << "Mikeの点数は" << score["Mike"] << "点" << endl;
    return 0;
}
```

mapクラス

main.cpp (Sample608)

```
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
    map <string, int> score;
    score["Tom"] = 100;
    score["Bob"] = 80;
    score["Mike"] = 76;
    score.erase("Mike"); //キーを指定して削除
    cout << "Tomの点数は" << score["Tom"] << "点" << endl;
    cout << "Bobの点数は" << score["Bob"] << "点" << endl;
    cout << "Mikeの点数は" << score["Mike"] << "点" << endl;
    return 0;
}
```



mapクラス

main.cpp (Sample608)

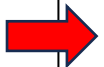
```
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
    map <string, int> score;
    score["Tom"] = 100;
    score["Bob"] = 80;
    score["Mike"] = 76;
    score.erase("Mike");
    cout << "Tomの点数は" << score["Tom"] << "点" << endl;
    cout << "Bobの点数は" << score["Bob"] << "点" << endl;
    if(score.count("Mike")) { //指定キーの要素数から値の有無をチェック
        cout << "Mikeの点数は" << score["Mike"] << "点" << endl;
    }
    return 0;
}
```

mapクラス

main.cpp (Sample608)

```
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
    map <string, int> score;
    score["Tom"] = 100;
    score["Mike"] = 90;
    score.insert(make_pair("John", 88)); //キー:John、値:88のペアを追加
    cout << "Tomの点数は" << score["Tom"] << "点" << endl;
    cout << "Bobの点数は" << score["Bob"] << "点" << endl;
    if(score.count("Mike")) {
        cout << "Mikeの点数は" << score["Mike"] << "点" << endl;
    }
    return 0;
}
```

ペアを定義(make_pair)して挿入
イテレータ不要(キー値により自動ソート)



mapクラス

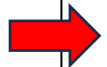
main.cpp (Sample608)

```
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
    map <string, int> score;
    score["Tom"] = 100;
    score["Bob"] = 80;
    score["Mike"] = 76;
    score.erase("Mike");
    score.insert(make_pair("John", 88));
    auto itr = score.find("John"); //キー:Johnがあればイテレータを返す
    cout << "Tomの点数は" << score["Tom"] << "点" << endl;
    cout << "Bobの点数は" << score["Bob"] << "点" << endl;
    if(score.count("Mike")) {
        cout << "Mikeの点数は" << score["Mike"] << "点" << endl;
    }
    return 0;
}
```


mapクラス

main.cpp (Sample608)

```
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
    map <string, int> score;
    score["Tom"] = 100;
    score["Bob"] = 80;
    score["Mike"] = 76;
    score.erase("Mike");
    score.insert(make_pair("John", 88));
    auto itr = score.find("John");
    cout << itr->first << "の点数は" << itr->second << "点" << endl;
    cout << "Tomの点数は" << score["Tom"] << "点" << endl;
    cout << "Bobの点数は" << score["Bob"] << "点" << endl;
    if(score.count("Mike"))
        cout << "Mikeの点数は" << score["Mike"] << "点" << endl;
}
```



イテレータの**first**メンバがキー
secondメンバが値を表す

mapクラス

main.cpp (Sample608)


```
#include <iostream>
#include <string>
#include <map>
using namespace std;
int main() {
    map <string, int> score;
    score["Tom"] = 100;
    score["Bob"] = 80;
    score["Mike"] = 76;
    score.erase("Mike");
    score.insert(make_pair("John", 88));
    score.emplace("David", 70);
    auto itr = score.find("John");
    cout << itr->second << "点" << endl;
    cout << score["Tom"] << "点" << endl;
    cout << "Bobの点数は" << score["Bob"] << "点" << endl;
    if(score.count("Mike")) {
        cout << "Mikeの点数は" << score["Mike"] << "点" << endl;
    }
}
```

emplaceで追加

→この場合はペアの定義は不要

mapクラス

main.cpp (Sample608)



```
cout << "Tomの点数は" << score["Tom"] << "点" << endl;  
cout << "Bobの点数は" << score["Bob"] << "点" << endl;  
for (auto it = score.begin(); it != score.end(); it++){  
    cout << "Key:" << it->first  
        << " Value:" << it->second << endl;  
}
```

実行すると、キーが昇順ソートされていることがわかる

mapクラスではキーが昇順ソートされていることが確認できる

mapクラス

main.cpp (Sample608)

```
cout << "Tomの点数は" << score["Tom"] << "点" << endl;
cout << "Bobの点数は" << score["Bob"] << "点" << endl;
//for (auto it = score.begin(); it != score.end(); it++){
//    cout << "Key:" << it->first
//        << " Value:" << it->second << endl;
//}
for (auto p : score){
    cout << "Key:" << p.first
        << " Value:" << p.second << endl;
}
```

イテレータのかわりに範囲forでも記述可能

mapクラス

main.cpp (Sample608)

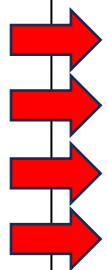
```
cout << "Tomの点数は" << score["Tom"] << "点" << endl;
cout << "Bobの点数は" << score["Bob"] << "点" << endl;
//for (auto it = score.begin(); it != score.end(); it++){
//    cout << "Key:" << it->first
//        << " Value:" << it->second << endl;
//}
//for (auto it = score.begin(); it != score.end(); it++){
//    cout << "Key:" << it->first
//        << " Value:" << it->second << endl;
//}
```

C++17からできるようになった記述方法

cl /EHsc /std:c++17 main.cpp

でコンパイルが可能

```
for (const auto& [key, value] : score){
    cout << "Key:" << key
        << " Value:" << value << endl;
}
```



mapクラス

【mapクラスの例】

```
map<string, string>
    email{ {“kd1234567@st.kobedenshi.ac.jp”, “青木一郎”},
           {“kd2345678@st.kobedenshi.ac.jp”, “赤城次郎”},
           {“kd3456789@st.kobedenshi.ac.jp”, “緑川三郎”} };
// メールアドレスと氏名の対応表
```

```
map<int, double>
    root{ { 1, 1.00000000 }, { 2, 1.1421356 },
          { 3, 1.7320504 }, { 4, 2.00000000 },
          { 5, 2.2360679 } };
// 自然数と平方根の対応表
```

何かのペアをデータとして利用したいときに
mapクラスは適している

mapクラス

【mapクラスの例】

```
map<int, map<string, int>>
    tMap{ { 1, { { "abc", 3 } } }, { 2, { { "abc", 1 } } },
          { 3, { { "def", 9 } } }, { 4, { { "def", 7 } } }
    };
cout << tMap[1]["abc"] << endl; // 3
cout << tMap[2]["abc"] << endl; // 1
cout << tMap[3]["def"] << endl; // 9
cout << tMap[4]["def"] << endl; // 7      }
// Mapの二次元配列
```

mapクラス

- mapまとめ

- **連想配列**を実現するコンテナクラス
- 添え字番号でなく、**【キー】**を使ってアクセス可能
- **【キー(key)】**と**【値(value)】**がペアになる
- map内部のデータは、キーの値によって昇順でソートされている
- mapにデータを追加する際はキーと値のペアで追加
- イテレータを使ってキーや値を取得可能