

演習: vector

- C++作業フォルダ内に**PracVector**フォルダを作成
`mkdir PracVector`
`cd PracVector`
- **PracVector**フォルダ内に `main.cpp` を作成する
`copy nul main.cpp`
- `main.cpp` をVisualStudioで開く

演習: vector

- 最大値や最小値の取得

int値を格納できるvectorコンテナクラスの
インスタンスを `vec` として宣言し、初期値として

`20, 11, 9, 33, 40, 25`

を与える

これらの数値の中から最大値`max`と最小値`min`を
みつけて画面上に表示したい

演習: vector

- 最大値や最小値の取得

同じプログラム(main.cpp)内で以下の**3通りのループ処理**で最大値と最小値を取得しなさい

- ① 配列の**添え字番号**を変更しながら最大値と最小値を探す
- ② **イテレータ**を使って、イテレータを進めながら最大値と最小値を探す
- ③ **範囲for**を使って最大値と最小値を探す

演習:vector

PracVector(main.cpp)

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<int> vec{ 20,11,9,33,40,25 };
    int max, min;
    /** 添え字を使ったループ **
    max = min = vec[0]; //仮の最大値最小値
    for (int i = 1; i < vec.size(); i++) {
        if (max < vec[i]) {
            max = vec[i];
        }
        if (min > vec[i]) {
            min = vec[i];
        }
    }
    cout << "最大値:" << max << " 最小値:" << min << endl;
```

演習:vector

PracVector(main.cpp)

```
/** イテレータを使ったループ **/  
max = min = vec.at(0); //vec[0]とvec.at(0)は同じ動作  
for (auto itr = vec.begin()+1; itr != vec.end(); ++itr) {  
    if (max < *itr) {  
        max = *itr;  
    }  
    if (min > *itr) {  
        min = *itr;  
    }  
}  
cout << "最大値:" << max << " 最小値:" << min << endl;
```

演習:vector

PracVector(main.cpp)

```
/** 範囲forを使ったループ **/  
max = min = vec.front(); //front()関数は先頭要素を取得可  
for (auto d : vec) {  
    if (max < d) {  
        max = d;  
    }  
    if (min > d) {  
        min = d;  
    }  
}  
cout << "最大値:" << max << " 最小値:" << min << endl;
```

演習: vector

PracVector(main.cpp)

/** 範囲forを使ったループ **/

max = min = vec[0]; //仮の最大値最小値

for (auto d : vec) {

if (max < d) {
 max = d;

}

if (min > d) {
 min = d

}

}

cout << "最大値:" << max << "最小値:" << min << endl;

どのループ処理をするときでも
比較する対象が異なるだけで
似た処理を記述する必要がある
ため、ここを関数化する

vec[i]
*itr

演習: vector

PracVector(main.cpp)

```
/** 範囲forを使ったループ **/
```

```
max = min = vec[0]; //仮の最大値最小値
```

```
for (auto d : vec) {
```

```
    compare( max, min, d );
```

```
}
```

```
cout << "最大
```

```
void compare(int& max, int& min, int data ){
```

```
    if (max < data) {  
        max = data;
```

```
    }
```

```
    if (min > data) {  
        min = data;
```

```
    }
```

```
}
```

引数の参照渡しを用いた関数

演習：vector

- 関数への引数の参照渡し

C言語では、

- 値渡し(コピー)
- ポインタ渡し(アドレス)

の2種類の方法で関数に引数を渡していたが、
C++ではあらたに

- 参照渡し

という方法が追加された

演習: vector

• 関数の参照渡し

```
void kansu(int a, int& b){  
    cout << "a:" << a << endl  
        << "b:" << b << endl;  
    a = 0;  
    b = 1;  
}  
int main(){  
    int a = 100, b = 200;  
    kansu(a, b);  
    cout << "a:" << a << endl  
        << "b:" << b << endl;  
    return 0;  
}
```

実行結果

a:100

b:200

a:100

b:1

演習: vector

• 関数の参照渡し

引数リストの変数名の前に
&を付けると参照渡しとなる



```
void kansu(int a, int& b){  
    cout << "a:" << a << endl  
         << "b:" << b << endl;  
    a = 0;  
    b = 1;  
}  
int main(){  
    int a = 100, b = 200;  
    kansu(a, b);  
    cout << "a:" << a << endl  
         << "b:" << b << endl;  
    return 0;  
}
```

関数の引数に&を付けるだけで
普通の変数のように使用可能

実行結果

a:100

b:200

a:100

b:1

演習: vector

• 関数のポインタ渡し

```
void kansu(int a, int* b){  
    cout << "a:" << a << endl  
        << "b:" << *b << endl;  
    a = 0;  
    *b = 1; ポインタになるので  
            間接参照演算子*が必要  
}  
int main(){  
    int a = 100, b = 200;  
    kansu(a, &b); 関数実行時に&が必要  
    cout << "a:" << a << endl  
        << "b:" << b << endl;  
    return 0;  
}
```

実行結果

a:100

b:200

a:100

b:1

演習: vector

- 関数の参照渡し of の仕組み

```
void kansu(int a, int& b){  
    cout << "a:" << a << endl  
    <<  
    a = 0  
    b = 1;  
}
```

&を付けることで、引数となる変数のアドレスを関数側で受け取り、そのアドレスに別の変数名を割り当てて、関数の中で使えるようになる仕組み

演習: vector

• 関数の参照渡し of の仕組み

```
void kansu(int a, int& b){  
    cout << "a:" << a << endl  
         << "b:" << b << endl;  
    a = 0;  
    b = 1;  
}  
int main(){  
    int a = 100, b = 200;  
    kansu(a, b);  
    cout << "a:" << a << endl  
         << "b:" << b << endl;  
    return 0;  
}
```

main関数の変数b
のアドレス情報
0x01A304

アドレス	変数名
0x01A300	a(main)
0x01A304	b(main) b(kansu)
0x01A308	a(kansu)
0x01A30C	

main関数の b と
kansu関数の b は
同じアドレスになるので
変更すると両方に反映

演習: vector

- 関数の参照渡しを行う理由

値のコピー渡しを行う際、例えば巨大な構造体データを引数に使用すると**データのコピーに時間がかかる...**

そのため、参照渡しでデータが格納されている**アドレスを指定することでコピーにかかる時間を省略できる**
(データのあるアドレスだけを伝えて直でアクセス)

※参照渡しの際に、**const**を付けると関数側で変更が禁止されるため、値渡しと等価な使用が可能

演習: vector

PracVector(main.cpp)

```
/** 範囲forを使ったループ **/
```

```
max = min = vec[0]; //仮の最大値最小値
```

```
for (auto d : vec) {
```

```
    compare( max, min, d );
```

```
}
```

```
cout
```

```
void compare(int& max, int& min, int data ){
```

```
    if (max < data) {  
        max = data;
```

```
    }
```

```
    if (min > data) {  
        min = data;
```

```
    }
```

```
}
```

引数の参照渡しを用いた関数

演習: vector

PracVector(main.cpp)

```
/** 範囲forを使ったループ **/
```

```
max = min = vec[0]; //仮の最大値最小値
```

```
for (auto d : vec) {
```

```
    compare( max, min, d );
```

```
}
```

```
cout
```

```
void compare(int& max, int& min, const int& data ){
```

```
    if (max < data) {  
        max = data;
```

```
    }
```

```
    if (min > data) {  
        min = data;
```

```
    }
```

```
}
```

参照渡しする変数に"const"を
付けると"定数"と同じ扱いになり
関数内で変更ができなくなる

演習:vector

PracVector(main.cpp)

```
/** 範囲forを使ったループ **  
max = min = vec[0]; //仮の最大値最小値  
for (auto d : vec) {  
    compare( max, min, d );  
}  
cout << "最大値:" << max << " 最小値:" << min << endl;
```

```
void compare(int& max, int& min, const int& data ){  
    max = max < data ? data : max;  
    min = min > data ? data : min;  
} //条件演算子を使った書き方
```

演習: vector(おまけ)

- `std::max_element`(イテレータ1, イテレータ2)
`std::min_element`(イテレータ1, イテレータ2)

イテレータ1～イテレータ2までの範囲で最大値・最小値のあるイテレータを返す標準関数がC++では実装済み

これらを使用するとループ処理がそもそも不要になる

```
max = *max_element(vec.begin(), vec.end());  
min = *min_element(vec.begin(), vec.end());  
cout << "最大値:" << max << " 最小値:" << min << endl;
```

演習: vector

PracVector(main.cpp)

```
auto compare
= [](int& max, int& min, const int& data ){
    max = max < data ? data : max;
    min = min > data ? data : min;
};
/** 範囲forを使ったループ **/
max = min = vec[0]; //仮の最大値最小値
for (auto d : vec) {
    compare( max, min, d );
}
cout << "最大値:" << max << " 最小値:" << min << endl;
```

ラムダ式を用いた
関数記法[](){}
(関数の中に関数を
実装可能)