

# プログラミング講座 宿題

---

Day3 (2019/3/12)実施分

小林篤史

# 宿題1.

---

- ポインタ・参照とconst について，以下の3つの違い

1. `const int &ref = val;`

2. `const int *ptr1 = val;`


3. `int *const ptr2 = val;`

# 宿題1.1 const int &

- `const int &ref = val;`

refの値の変更は不可, 参照先valの値の変更は可

```
int main() {  
    int val = 5;  
    const int &ref = val;  
    std::cout << "ref=" << ref << ", val=" << val << "¥n";  
    ref = 10;  
    std::cout << "ref=" << ref << ", val=" << val << "¥n";  
    system("pause");  
    return 0;  
}
```

→   
(エラー)

```
int main() {  
    int val = 5;  
    const int &ref = val;  
    std::cout << "ref=" << ref << ", val=" << val << "¥n";  
    val = 10;  
    std::cout << "ref=" << ref << ", val=" << val << "¥n";  
    system("pause");  
    return 0;  
}
```

→  
ref=5, val=5  
ref=10, val=10

const int val を参照できる(constなし参照型では参照不可)

注：参照型では、参照先（アドレス）の変更はそもそも不可

# 宿題1.2 const int \*

- `const int *ptr = val;`

\*ptrを用いたvalの値の変更が不可, ただし直接valの書き換えは可

```
int main() {  
    int val = 5;  
    const int *ptr = &val;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << "\n";  
    *ptr = 10;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << "\n";  
  
    system("pause");  
    return 0;  
}
```



```
int main() {  
    int val = 5;  
    const int *ptr = &val;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << "\n";  
    val = 10;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << "\n";  
  
    system("pause");  
    return 0;  
}
```



```
*ptr=5, val=5  
*ptr=10, val=10
```

あとから別の値を参照するのは可能

```
int main() {  
  
    int val = 5;  
    int val1 = 8;  
    const int *ptr = &val;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << ", val1=" << val1 << "\n";  
    ptr = &val1;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << ", val1=" << val1 << "\n";  
    system("pause");  
    return 0;  
}
```



```
*ptr=5, val=5, val1=8  
*ptr=8, val=5, val1=8
```

# 宿題1.3 int \*const

- `int *const ptr = val;`

参照する先(アドレス)をあとで変更できない, 値の変更は可能

```
int main() {  
    int val = 5;  
    int val1 = 8;  
    int *const ptr = &val;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << ", val1=" << val1 << "\n";  
    ptr = &val1;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << ", val1=" << val1 << "\n";  
    system("pause");  
    return 0;  
}
```



```
int main() {  
    int val = 5;  
    int *const ptr = &val;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << "\n";  
    *ptr = 10;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << "\n";  
    system("pause");  
    return 0;  
}
```



```
*ptr=5, val=5  
*ptr=10, val=10
```

定数を指すことはできない("const int \*ptr"は可能)

```
int main() {  
    const int val = 5;  
    int *const ptr = &val;  
    std::cout << "*ptr=" << *ptr << ", val=" << val << "\n";  
    system("pause");  
    return 0;  
}
```



## 宿題2.

---

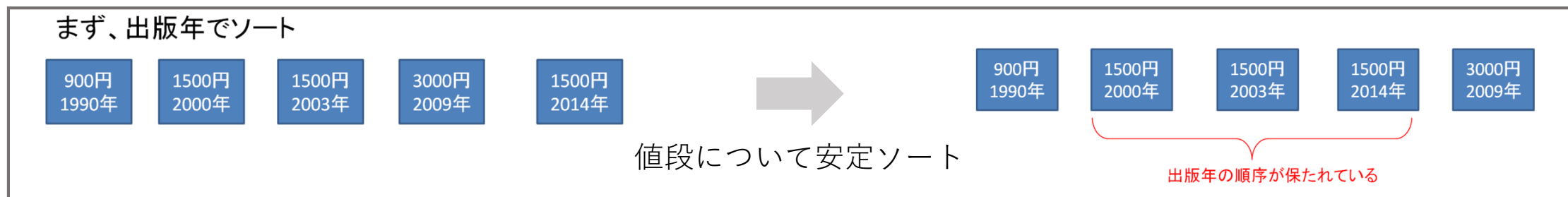
- ソートアルゴリズムについて，以下の3つの実装  
および計算量などの違いの考察

1. バブルソート
2. クイックソート
3. 挿入ソート

(※実装は別ファイル)

# 宿題2.1 バブルソート

- バブルソート
- 安定ソートの一種  
同じ値をもつデータの並び順がソート実行前の順序を保持する



## バブルソートの計算量

	比較回数	交換回数	
最小の場合	$\frac{n(n-1)}{2}$ (固定)	0	計算量は $O(n^2)$
平均・最大の場合		$O(n^2)$	

# 宿題2.2 クイックソート

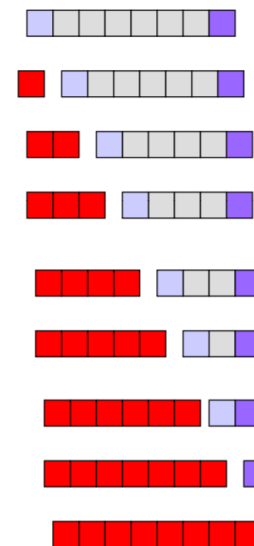
- クイックソート: 不安定ソート的一种

クイックソートの計算量

	計算量
最良の場合・平均	$O(n \log_2 n)$
最大の場合	$O(n^2)$

- 計算量が最大になる状況

毎回、領域内の最大（最小）値をピボットに選んでしまう  
→毎回の分割が「ピボットのみ」と「それ以外」となる





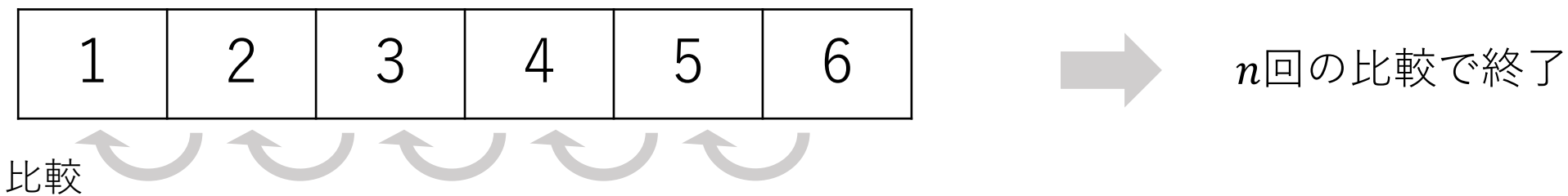
# 宿題2.3 挿入ソート

- 挿入ソート：安定ソート

挿入ソートの計算量

	計算量
最小の場合	$O(n)$
平均・最大の場合	$O(n^2)$

- 計算量の最小になる場合：整列済みのデータに対し実行するとき  
← 整列済みに近いデータほど高速にソートできる



# 宿題2 使い分けなど

---

- クイックソート：大規模なデータでは平均計算量が比較的少ない
- 計算量が最大になるのを防ぐための対処法
  - データ（全体もしくは一部）の中央値をピボットにする
  - あらかじめ入力データ列をランダムに入れ替える
- 挿入ソート
- 整列済みデータに新しいデータを加えるときに高速
- 大規模なデータに対し，まずクイックソートを実行し分割が進んだデータに挿入ソートを行うことで高速化