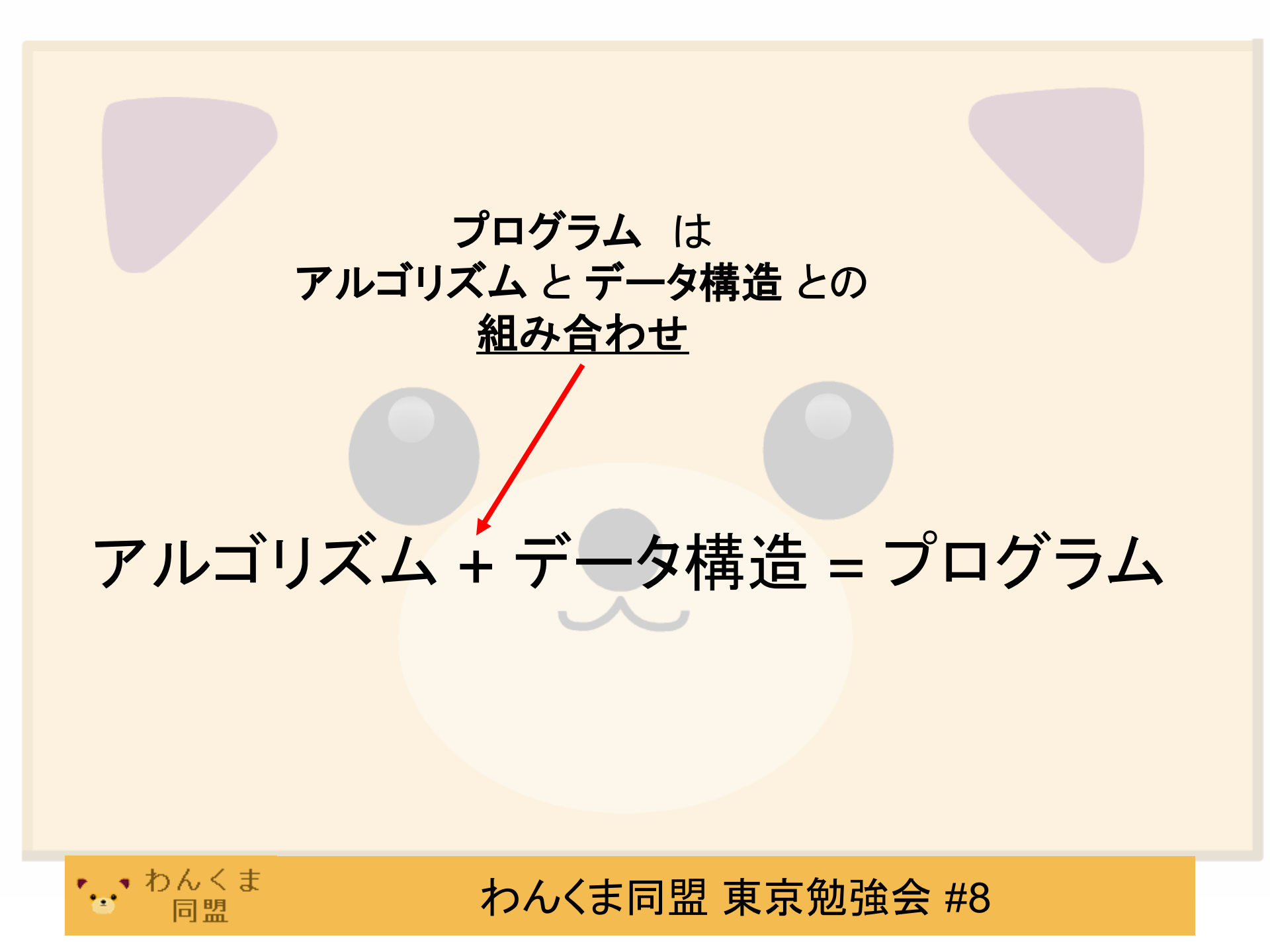


STL/CLR による Generic Programming

わんくま同盟
MVP for Visual C++

ΕΠΙΣΤΗΜΗ



プログラム は
アルゴリズムとデータ構造 との
組み合わせ

アルゴリズム + データ構造 = プログラム

手続き型

アルゴリズムにデータ構造を
食わす

アルゴリズム + データ構造 = プログラム

関数

構造体

オブジェクト指向

アルゴリズムでデータ構造を
包み込む

アルゴリズム + データ構造 = プログラム

メソッド

メンバ変数

クラス

Generic Programming

アルゴリズムとデータ構造を
繋ぐ

進化した手続き型

アルゴリズム + データ構造 = プログラム

イテレータ

コンテナ

- i 種のデータ (int, long, string, etc.)
- j 種のデータ構造 (配列, リスト, 二分木, etc.)
- k 種のアルゴリズム (列挙, 検索, 変換, etc.)

$i \times j \times k$ 個の実装を必要とする



$i + j + k$ 個の実装を目指す

→ **Generic Programming**

first 以上 last 未満 の範囲から target を探す (見つからなければlastを返す)

```
int* find(int* first, int* last, int target) {  
    while ( first != last ) {  
        if ( *first == target ) {  
            break;  
        }  
        ++first;  
    }  
    return first;  
}
```

intを要素とする配列に
対してのみ適用可能

first 以上 last 未満 の範囲から target を探す (見つからなければlastを返す)

```
template<typename T>
T* find(T* first, T* last, T target) {
    while ( first != last ) {
        if ( *first == target ) {
            break;
        }
        ++first;
    }
    return first;
}
```

アルゴリズムは
データ型に対して自由になった!

first 以上 last 未満 の範囲から target を探す (見つからなければlastを返す)

```
template<typename Itertator, typename T>
Iterator find(Iterator first, Iterator last, T target) {
    while ( first != last ) {
        if ( *first == target ) {
            break;
        }
        ++first;
    }
    return first;
}
```

アルゴリズムは
データ型とデータ構造に対して自由になった!

```
template<typename Itertator, typename T>
Iterator find(Iterator first, Iterator last, T target) {
    while ( first != last ) {
        if ( *first == target ) {
            break;
        }
        ++first;
    }
    return first;
}
```

Iteratorは:

- != で比較できる
- * で要素を取り出せる
- ++ で次に進めることができる

ならどんなものでもかまわない

Iteratorは:

- != で比較できる
- * で要素を取り出せる
- ++ で次に進めることができる

ならどんなものでもかまわない ...ってことは

数々のデータ構造(要素の集合)それぞれがこれを満たす
Iteratorを返してくれるなら、
そしてそのデータ構造がいかなるデータも格納できるなら
関数findはひとつ書けばいい!

→ **そこが Generic (総称的)**

```
#include <cliext/vector>
#include <cliext/list>
#include <cliext/algorithm>

using namespace cliext;

vector<int> iv; // intを要素とする可変長配列
list<String^> sl; // String^を要素とする双方向リスト

// データ/コンテナにかかわらず find できる♪
vector<int>::iterator iiv =
    find(iv.begin(), iv.end(), 5);

list<String^>::iterator isl =
    find(sl.begin(), sl.end(), "five");
```

C# で Generic Programming

IEnumerable<T> で、データ/コレクションにかかわらず列挙できる。

```
List<int> il;  
LinkedList<string> sll;  
char[] ca;  
  
foreach ( int item in il ) { ... }  
foreach (string item in sll ) { ... }  
foreach ( char item in ca ) { ... }
```

それぞれの IEnumerable<T>
に対して

- bool MoveNext()
- T Current { get; }

できるから。

C# で Generic Programming

```
public class Enums {  
    public static IEnumerable<T>  
        Select(IEnumerable<T> e, Predicate<T> p)  
    {  
        foreach ( T item in e ) {  
            if ( p(item) ) {  
                yield return item;  
            }  
        }  
    }  
}
```

なんてのを定義すれば...

C# で Generic Programming

```
Static bool IsEven(int n) {  
    return n % 2 == 0;  
}
```

```
List<int> il;  
LinkedList<int> ill;  
Int[] ia;
```

// 偶数のみを列挙する

```
foreach ( int item in Enums.Select<int>(il, IsEven) {  
    Console.WriteLine(item);  
}
```

ill でも ia でも、IEnumerable<int>でさえあれば。

// 偶数のみを抽出する

```
List<int> evens =  
    new List<int>(Enums.Select<int>(il, IsEven))
```