テンプレート

- **関数テンプレート**
- C++作業フォルダ内の**Sample503**フォルダを**Sample503t**フォルダとしてコピー

```
robocopy Sample503 Sample503t
cd Sample503t
```

- calc.h, calc.cpp, main.cppを変更

テンプレート

- calc.h (Sample503t)

```
#pragma once
class Calc {
private:
   int m_a, m_b;
public:
   Calc();
   Calc(int a, int b);
   int add();
   int add(int a, int b);
   double add(double a, double b);
   string add(string a, string b);
   void setValue(int a, int b);
```

2つのメンバ関数を追加

テンプレート

- calc.cpp (Sample503t)

```
            (略)
int Calc::add() {
   return m_a + m_b;
}
int Calc::add(int a, int b) {
   return a + b;
}
double Calc::add(double a, double b) {
   return a + b;
}
string Calc::add(string a, string b) {
   return a + b;
}
```

2つのメンバ関数の処理内容を追加

テンプレート

- main.cpp (Sample503t)

```
      (略)
int main() {
    Calc* pC1, * pC2;
    pC1 = new Calc();
    pC2 = new Calc(1, 2);
    cout << 3 << "+" << 4 << "=" << pC1->add(3, 4) << endl;
    cout << pC2->getA() << “+”<< pC2->getB()
      << "=“ << pC2->add() << endl;
    cout << 1.1 << “+” << 2.5 << “=“ << pC1->add1.1, 2.5)
          << endl;
    cout << “ABC” << “+” << “DEF” << “=“
          << pC1->add(“ABC”, “DEF”) << endl;
    delete pC1;
    delete pC2;
    return 0;
}
```

テンプレート

- **関数テンプレート**

メンバ関数のオーバーロード(多重定義)を行うときに引数の数は同じだが、引数の型が異なると、似たような記述を何回もしないといけない

```
double Calc::add(double a,double b) {
   return a + b;
}
int Calc::add(int a, int b) {
   return a + b;
}
string Calc::add(string a, string b) {
  return a + b;
}
```

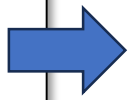
型名が異なるだけで
処理内容はほぼ同じ…

テンプレート

- **関数テンプレート**

メンバ型名のところをテンプレート機能を使って、
別の文字に置き換えて関数を定義することで
ひとつにまとめることが可能な仕組み

```
double Calc::add(double a,double b) {
    return a + b;
}
int Calc::add(int a, int b) {
    return a + b;
}
string Calc::add(string a, string b) {
  return a + b;
}
```



```
template <typename T>
T Calc::add(T a, T b) {
  return a + b;
}
```

テンプレート

- calc.h (Sample503t)

```
#pragma once
class Calc {
private:
      (略)
public:
   int add();
   template <typename T>
   T add(T a, T b) {
      return a + b;
   };
      (略)
};
```

add関数(引数あり)をテンプレートを用いて書き換える
メンバ関数の場合は、関数の記述もヘッダファイルで行う

テンプレート

- calc.cpp (Sample503t)

```
Calc::Calc(int a, int b) : m_a(a), m_b(b) {}
int Cal
   retur
}
int Calc::add(int a, int b) {
   return a + b;
}
double Calc::add(double a, double b) {
   return a + b;
}
string Calc::add(string a, string b) {
   return a + b;
}
```

ヘッダファイルで処理を記述したので
従来の関数処理を削除

テンプレート

- main.cpp (Sample503t)

```
      (略)
int main() {
    Calc* pC1, * pC2;
    pC1 = new Calc();
    pC2 = new Calc(1, 2);
    cout << 3 << "+" << 4 << "=" << pC1->add<int>(3, 4) << endl;
    cout << pC2->getA() << “+”<< pC2->getB()
      << "=“ << pC2->add() << endl;
    cout << 1.1 << “+” << 2.5 << “=“ << pC1->add<double>(1.1, 2.5)
          << endl;
    cout << “ABC” << “+” << “DEF” << “=“
          << pC1->add<string>(“ABC”, “DEF”) << endl;
    delete pC1;
    delete pC2;
    return 0;
}
```

テンプレート

•関数テンプレート

- テンプレートを用いることで、引数の数が同じだが引数の戻り値や型だけが異なる関数の記述をまとめることができる

【宣言方法】

```
template <typename T>
T 関数名(T 引数1, T 引数2,･･･)
```

【実行方法】

```
関数名<型名>(引数1, 引数2,･･･)
```

テンプレート

- **関数テンプレート**
- 引数の型を複数もつテンプレート関数も定義可能

【宣言方法】

```
template <typename T, typename U>
auto 関数名(T 引数1, U 引数2)
```

【実行方法】

```
関数名<型名,型名>(引数1, 引数2)
```

テンプレート

- calc.h (Sample503t)

```
template <typename T, typename U>
auto add(T a, U b) {
    return a + b;
};
```

- main.cpp (Sample503t)

```
cout << 5 << “+” << 3.2 << “=“
  << pC1->add(5, 3.2) << endl;
                //整数と実数の足し算
```

テンプレート

- **テンプレートクラス**
- クラス定義についてもテンプレートを用いることができる

【宣言方法】

```
template <typename T>
class クラス名 {
   メンバ…
};
```

テンプレート

- テンプレートクラス
- 【例】

```
template <typename T>
class Kurasu {
private:
    T m_a;
public:
    T func(T a, T b){
        return a + b;
    }
};
```

```
int main()
{
  Kurasu<int> k1;
  k1.m_a = 2;
  k1.func(3,5);
  Kurasu<string> k2;
  k2.m_a = “ABC”;
  .....
}
```

テンプレート

- まとめ

  テンプレートを用いると、“型”に囚われないプログラミングが可能になる

- 関数のオーバーロード（多重定義）の処理を簡略化することができる

- クラスにテンプレートを用いることで、クラス内のメンバの型を自由に変更することが可能になる