1　オーバーロードの説明と目的

関数のオーバーロードとは、同じ名前の関数を使用し、引数の型や数によって区別して適切な処理を柔軟に行うための仕組みである。引数リストが異なる場合にのみ成立し、戻り値の型だけでは区別できない。目的としては、関数を使う側が引数の違いを意識せずに直感的に使えるようにすることや、似た処理を共通の名前で表現することでコードの可読性を向上させることにある。

2　静的メンバ変数の説明

静的メンバ変数というほかの変数、特に非静的メンバ変数と違ってすべてのクラスのインスタンスで同じ実体を持ち共有している。クラスからもインスタンスからもアクセスすることができる。また、グローバルスコープで一度だけ宣言する必要がある。

3　コンパイルできない原因

1. 純粋仮想関数が入っていると，その純粋仮想関数に関する記述がそのクラスにあったとしてもそのクラスは抽象クラスになってそのクラスのインスタンスを生成することはできない.
   しかし, 抽象クラスの中に純粋仮想関数の中身を書いた場合それはデフォルト実装となって, 継承したクラスでオーバーライドしない場合の動作になる.
2. クラスの定義の中に書いた関数の中身がどこかに書かれていないとエラーになる

4　型の注意点

double 型の計算の時は

```
average = (double)Math + (double)Jap + (double)Sci  +(double)Soc + (double)Eng;
average /= 5;
```

のように double 型をそろえて計算しないと丸め込まれたりしちゃうので注意.

C 言語の printf では小数点以下の連続する 0 は表示される．double では小数点以下 6 桁．Cpp の cout ではそれらは省略されて表示される.

5　継承の際の注意点

新しいメンバ変数を追加した場合は，Disp や Average の計算の関数が基底クラスのままにならないようにちゃんとオーバーライドする.

6　オブジェクトをポインタで定義する方法とメリット

```
#include <iostream>
using namespace std;

class MyClass {
public:
    void Display() { cout << "Hello, Pointer!" << endl; }
};

int main() {
    MyClass* obj = new MyClass(); // 動的メモリ確保
    obj->Display();
    delete obj; // メモリ解放
    return 0;
}
```

このように動的にメモリ管理をすることでリソースを節約することができる.

また, 大規模なデータを参照渡しすることで無駄なコピーを避けることができる.

基底クラスのポインタ型から仮想関数を使用することで異なる派生クラスの関数を呼び出すことができる.

7　多重継承 (特にひし形継承)

Python では多重継承特にひし形継承) を解決するために MRO(Method Resolution Order) と super() が用意されている.

その順番は A を B と C が継承し, D が B と C を継承しているとき, D, B, C, A となる. super() は MRO に従って順に関数を呼び出していく.

8　オブジェクト指向プログラミングの利点

1. コードの再利用性が向上する.

   継承やポリモーフィズムを利用することで共通の関数, 共通のメンバ変数をまとめることができる. 例えば, 図形の辺の長さの合計を計算するというような処理をする場合, 辺に値を入れる, 辺をすべて合計するという処理は同一のため基底クラスにまとめて記述することで結果的にエラーの発生の減少, コードの簡略化につながる.
2. コードの可読性と保守性を向上できる. グローバル変数, 関数が増えるとコードが複雑になり

修正が難しくなる．クラス内にメンバ変数，メンバ関数を実装することで修正が簡単になる．非破壊的なアップデートが可能になる．メンバ変数，関数のセキュリティが向上するなどのメリットがある．

3. 拡張性が高くなる．新しい商品の追加など，既存のプログラムを継承して新たな商品を追加するだけでよい．ショッピングサイトでの新たな支払方法の追加など

Python のインスタンスを引数にする，クラスを引数にするといった処理について調べてからオブジェクト指向の勉強終わり．