typedef struct

{

Int nLife; メンバ

・構造体とクラス

Int nAttack; 構造体が表す「もの」の

Int nSpeed; 固有情報性質

}PLAYER;

↓

class CPlayer //CPlayerクラス

{

int m\_nLife; メンバ

int m\_nSpeed; 構造体と同様に

int m\_nAttack; 「もの」の情報性質を表す

};

・オブジェクト指向

PLAYER----プレイヤー一人

PATTICLE---パーティクル粒子１個

「もの」をデータに表す→同じものや似たもの

＝ １個１個を「部品」として扱うことができる

オブジェクト（もの）指向

共通部分、固有部分を切り分けて

プログラム化していく＿クラス設計

・アクセス指定子

（構造体）

外部からメンバを自由に参照・操作（アクセス）

することができる

（クラス）

外部からのメンバへのアクセスに

アクセス指定を使って制限を設けられる

・アクセス指定子の種類

・public(公の、公的な)

クラス外部から内部メンバに

自由にアクセスできる。

設計者が予期しないアクセス

によってメンバが破壊される恐れもある。

・private(私的な、個人的な)

クラス外部からメンバへのアクセスを

禁止する。クラス内部（自分自身）

でのみ利用可能。

protected(保護された)

自分自身とは静クラスからのみアクセスできる

・アクセス指定子の利用例

Class CPlayer クラス定義の開始

{

public: 次の指定子が記述されるまで有効

publicメンバ

publicメンバ

private:

メンバ

}public: 同じ指定を不空数宣言してもよい（順番不要）

・カプセル化

アクセス指定子によってクラス内部のメンバへの

想定外のアクセスを防ぐ。

複数人数による開発において

クラス設計者の意図と異なる操作は

バグの原因となる。

・外部からアクセスされてもよい

アクセスが必要ないメンバ→bublic

・外部に見せる必要がない

アクセスされると困るメンバ→private

・メンバ関数

Class CPlayer

{

Public : //アクセス指定子

Void InputData(void);

Void OutputDate(void); //プロトタイプ宣言の形で記述

Int nLife;

Int nAttack;

Int nSpeed;

}

・メンバ関数の実装

定義しただけではメンバはメンバ関数は使えない

実際の動作を記述して初めて実体を持つ

例）

Void CPlayer::inputData(void)

{

(関数動作の記述);

}

・クラスメンバへのアクセス

（変数の参照・書き換え・関数の呼び出し）

・クラス外から

構造体同様、[.][->]が必要

Player.nLife = 100;

Player.InputData();

・クラス内からのアクセスとは（privateメンバも可）

メンバ関数から自クラスのメンバ変数・関数を扱う

[.][->]は不要

nAttack = 80;

OutputData();

・アクセス指定子の使い分け

Publicなメンバ変数/メンバ関数

{

メンバ変数:参照:書き換えが自由

（不正な値、書き換えが許される）

メンバ関数:呼び出すだけで動作などの書き換え不可

}

関数: public化しても安全性が高い

変数:privateにし関数から必要な参照書き換えを行うのが良い