第３回Ｃ＋＋講座

1.継承

継承はオブジェクト指向に欠かせない重要なものです。継承は、ベースとなるクラスの内容を受け継いで新しいクラスを作成します。ベースとなるクラスを「基底クラス」「スーパークラス」などといい、継承されたクラスのことを「派生クラス」「サブクラス」などと呼んでいます。

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
C＋＋講座サンプル 1
継承
by y.t
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define GU		0
#define TYOKI		1
#define PA		2
class PLAYERBASE
{
	char *m_Name;
public:
	int m_Te;
	//コンストラクタ
	PLAYERBASE(char *name)
	{
		m_Name = name;
	}
	~PLAYERBASE(){};
	//メソッド
	void Print();
	//フレンド関数
	friend void Judgi(PLAYERBASE *p1, PLAYERBASE *p2);
};
void PLAYERBASE::Print()
{
	char *str;
	switch(m_Te)
	{
		case GU:
			str = "グー";
			break;
		case TYOKI:
			str = "チョキ";
			break;
		case PA:
			str = "パー";
			break;
	}
	printf("%s は%s を出した¥n", m_Name, str);
}
class HUMAN : public PLAYERBASE
```

```
{
public:
	//コンストラクタ
	HUMAN(char *name);
	~HUMAN(){};
	//メソッド
	void Action();
};
HUMAN::HUMAN(char *name) : PLAYERBASE(name)
{}
void HUMAN::Action()
{
	printf("何を出す？¥n");
	printf("1.グー　2.チョキ　3.パー¥n");
	scanf("%d", &m_Te);
	m_Te--;
}
class AI : public PLAYERBASE
{
public:
	//コンストラクタ
	AI(char *name);
	~AI(){};
	//メソッド
	void Action();
};
AI::AI(char *name) : PLAYERBASE(name)
{
	time_t a;
	srand((unsigned int)time(&a));
}
void AI::Action()
{
	m_Te = rand() % 3;
}
void Judgi(PLAYERBASE *p1, PLAYERBASE *p2)
{
	if (p1->m_Te == p2->m_Te)	printf("ドロー¥n");
	else
	{
		switch(p1->m_Te)
		{
			case GU:
				if (p2->m_Te == TYOKI)	printf("%s の勝ち¥n", p1->m_Name);
				else			printf("%s の勝ち¥n", p2->m_Name);
				break;
			case TYOKI:
				if (p2->m_Te == PA)	printf("%s の勝ち¥n", p1->m_Name);
				else			printf("%s の勝ち¥n", p2->m_Name);
				break;
			case PA:
				if (p2->m_Te == GU)	printf("%s の勝ち¥n", p1->m_Name);
				else			printf("%s の勝ち¥n", p2->m_Name);
				break;
		}
	}
}
//main 関数
int main()
{
```

```
	HUMAN Human("人間");
	AI Ai("機械");
	printf("じゃんけんゲーム¥n");
	Human.Action();
	Ai.Action();
	Human.Print();
	Ai.Print();
	Judgi(&Human, &Ai);
	return 0;
}
```

//

ちょっとばかり寄り道。PLAYERBASE クラスにメソッド以外の関数「フレンド関数」というものが宣言されています。フレンド関数とはメソッドでは無くてただの関数です。フレンド関数の役割は、その関数内でのみ、宣言したクラスのすべてのメンバを参照可能にするという効果があります。サンプル3では、本来アクセスできないはずの m_Name を参照しているのが分かるでしょう。

さて、ここでアクセスコントロールの説明をします。アクセスコントロールとは、クラスの各メンバについてどの程度外部から参照できるかを表しています。表 1.1 にアクセスコントロールの名称と機能をまとめました。

表 1.1 アクセスコントロール

名称	機能
public	どこからでもアクセスできる。
protected	.や->による外部からのアクセスはできない。
private	.や->による外部参照のほか、サブクラスからのアクセスもできない。 クラス宣言時の初期設定。

サンプル3では PLAYERBASE がスーパークラスで、HUMAN と AI がサブクラスです。HUMAN と AI では共通する部分は PLAYERBASE クラスを利用し、おのおの異なる部分をサブクラスとして作成することになります。

例によって、これも C 言語で書いてみました。

//

```
/*
C＋＋講座サンプル 2
サンプル 1 の C ヴァージョン
by y.t
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define GU		0
#define TYOKI		1
#define PA		2
typedef struct tag_PLAYER
{
	char *m_Name;
	int m_Te;
}PLAYER, *LPPLAYER;
void ActionHuman(LPPLAYER pPlayer)
{
	int te;
	printf("何を出す？¥n");
	printf("1.グー　2.チョキ　3.パー¥n");
	scanf("%d", &te);
	te--;
```

```
		pPlayer->m_Te = te;
}
void ActionAi(LPPLAYER pPlayer)
{
		pPlayer->m_Te = rand() % 3;
}
void Print(LPPLAYER pPlayer)
{
		char *str;
		switch(pPlayer->m_Te)
		{
			case GU:
				str = "グー";
				break;
			case TYOKI:
				str = "チョキ";
				break;
			case PA:
				str = "パー";
				break;
		}
		printf("%s は%s を出した¥n", pPlayer->m_Name, str);
}
void Judgi(LPPLAYER p1, LPPLAYER p2)
{
		if (p1->m_Te == p2->m_Te)	printf("ドロー¥n");
		else
		{
			switch(p1->m_Te)
			{
				case GU:
					if (p2->m_Te == TYOKI)		printf("%s の勝ち¥n", p1->m_Name);
					else				printf("%s の勝ち¥n", p2->m_Name);
					break;
				case TYOKI:
					if (p2->m_Te == PA)		printf("%s の勝ち¥n", p1->m_Name);
					else				printf("%s の勝ち¥n", p2->m_Name);
					break;
				case PA:
					if (p2->m_Te == GU)  printf("%s の勝ち¥n", p1->m_Name);
					else				printf("%s の勝ち¥n", p2->m_Name);
					break;
			}
		}
}
//main 関数
int main()
{
		PLAYER Human;
		PLAYER Ai;
		//人間の初期化
		Human.m_Name = "人間";
		//AI の初期化
		time_t a;
		srand((unsigned int)time(&a));
		Ai.m_Name = "機械";
		//メイン
		printf("じゃんけんゲーム¥n");
		ActionHuman(&Human);
		ActionAi(&Ai);
```

```
	Print(&Human);
	Print(&Ai);
	Judgi(&Human, &Ai);
	return 0;
}
```

//

2.ポリモーフィズム（多態性）

　ポリモーフィズムを用いるとメソッドの呼び出しを一本化することができます。どういうことなのか、それはサンプルを見ればよく分かります。

//

```
/*
C＋＋講座サンプル 3
ポリモーフィズム
by y.t
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define GU		0
#define TYOKI		1
#define PA		2
class PLAYERBASE
{
	char *m_Name;
public:
	int m_Te;
	//コンストラクタ
	PLAYERBASE(char *name)
	{
		m_Name = name;
	}
	~PLAYERBASE(){};
	//仮想関数
	virtual void Action(){};
	//メソッド
	void Print();
	//フレンド関数
	friend void Judgi(PLAYERBASE *p1, PLAYERBASE *p2);
};
void PLAYERBASE::Print()
{
	char *str;
	switch(m_Te)
	{
		case GU:
			str = "グー";
			break;
		case TYOKI:
			str = "チョキ";
			break;
		case PA:
			str = "パー";
			break;
	}
	printf("%s は%s を出した¥n", m_Name, str);
```

```
}
class HUMAN : public PLAYERBASE
{
public:
	//コンストラクタ
	HUMAN(char *name);
	~HUMAN(){};
	//メソッド
	void Action();
};
HUMAN::HUMAN(char *name) : PLAYERBASE(name)
{}
void HUMAN::Action()
{
	printf("何を出す？¥n");
	printf("1.グー　2.チョキ　3.パー¥n");
	scanf("%d", &m_Te);
	m_Te--;
}
class AI : public PLAYERBASE
{
public:
	//コンストラクタ
	AI(char *name);
	~AI(){};
	//メソッド
	void Action();
};
AI::AI(char *name) : PLAYERBASE(name)
{
	time_t a;
	srand((unsigned int)time(&a));
}
void AI::Action()
{
	m_Te = rand() % 3;
}
void Judgi(PLAYERBASE *p1, PLAYERBASE *p2)
{
	if (p1->m_Te == p2->m_Te)	printf("ドロー¥n");
	else
	{
		switch(p1->m_Te)
		{
			case GU:
				if (p2->m_Te == TYOKI)		printf("%s の勝ち¥n", p1->m_Name);
				else				printf("%s の勝ち¥n", p2->m_Name);
				break;
			case TYOKI:
				if (p2->m_Te == PA)		printf("%s の勝ち¥n", p1->m_Name);
				else				printf("%s の勝ち¥n", p2->m_Name);
				break;
			case PA:
				if (p2->m_Te == GU)  printf("%s の勝ち¥n", p1->m_Name);
				else				printf("%s の勝ち¥n", p2->m_Name);
				break;
		}
	}
}
//main 関数
```

```
int main()
{
	HUMAN Human("人間");
	AI Ai("機械");
	PLAYERBASE *PlayerManager[2];
	PlayerManager[0] = &Human;
	PlayerManager[1] = &Ai;
	printf("じゃんけんゲーム¥n");
	for(int i = 0; i < 2; i++)
	{
		PlayerManager[i]->Action();
	}
	for(int i = 0; i < 2; i++)
	{
		PlayerManager[i]->Print();
	}
	Judgi(&Human, &Ai);
	return 0;
}
```

//

ポリモーフィズムを実装するには仮想関数というものが必要です。仮想関数の宣言には virtual を使います。仮想関数は実装を再定義することが可能です（オーバーライト）。また、サブクラスのポインタは、スーパークラスのポインタ変数に代入することができます（アップキャスト）。

スーパークラスから仮想関数を呼ぶ場合、実行される処理はスーパークラスに代入されたサブクラスのポインタでオーバーライトされた処理です。これにより、共通の呼び出しで、異なる実装を処理させることができます。

3.this ポインタ

外部からインスタンスのポインタを取得しようとしたとき、変数のポインタを取得するとき同様&を使います。では、インスタンスの内部（メソッド）から、そのインスタンス自身のポインタを取得するにはどうすればいいのでしょうか。

それを実現するのが this ポインタです。

//

```
/*
C＋＋講座サンプル 4
this ポインタ
by y.t
*/
#include <stdio.h>
class A
{
public:
	void print();
};
void A::print()
{
	printf("this = %p¥n", this);
}
int main()
{
	A a;
	printf("&a = %p¥n", &a);
	a.print();
	return 0;
```

```
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

this と書いてやればそれで this ポインタが使えます。