第２回Ｃ＋＋講座

1.テンプレート関数

テンプレート関数とは、任意のデータ型を引数に渡したいときに使います。つまり、**どんな型にも対応する関数**を作ることが出来るのです。では、作り方を以下に記します。

```
template <class タイプ名>  関数のデータ型  関数名(タイプ名  引数)
{
        ・・・
}
```

といった感じです。実装例はサンプル 1 と共に解説します。

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
C＋＋講座サンプル 1
テンプレート関数
by y.t
*/
#include <stdio.h>
template <class A> A Max(A a, A b)        //テンプレートの定義
{
        X max = a;
        return max > b ? max : b;
}
/*メイン関数*/
int main()
{
        printf("%d¥n", Max(20, 5));
        printf("%c¥n", Max('d', 'x'));
        printf("%lf¥n", Max(2.5, 6.1));
        return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

<class A>となっている箇所があります。このとき、A は仮のデータ型と思ってください。プログラムを実行すると型 A の引数に代入された値のデータ型が A に当てはめられます。

2.new･delete 演算子

変数を使用するとき、今までは変数宣言をしていました。こうすることで変数に必要な分のメモリを確保している訳です。しかし、どれだけの配列を確保するのかわからないときや、必要なタイミングでメモリを確保したいときというのは変数宣言だけでは不可能なのです。そこで登場するのが**メモリの動的確保**です。これを実現するには、

第2回C＋＋講座（By y.t）

new **演算子**を使います。これを使うと**好きなタイミングで必要な数だけメモリを確保する**ことが出来ます。文法は、

ポインタ変数 = new データ型[配列の数];

です。new 演算子は確保したメモリのポインタを返します。なので、代入先はポインタとなっているのです。

もし、変数が配列で無いなら[]は外して下さい。

さて、メモリを確保したら**不要になったときに開放してやらないといけません**。メモリを開放しないと空きメモリが無くなってしまう為です。メモリを開放するには delete **演算子**というのを使います。文法は、

delete []変数;

です。

もし、変数が配列でなければ[]は外して下さい。

//

```
/*
C＋＋講座サンプル 2
new・delete 演算子
by y.t
*/
#include <stdio.h>
/*メイン関数*/
int main()
{
	int *a;
	int *b;
	a = new int;			//動的確保
	*a = 10;
	printf("*a = %d¥n", *a);
	delete a;			//メモリの開放
	int size = 0;
	printf("配列の個数を決めてください¥n");
	while(size <= 0)	//0以下はまずいので1以上が入るまで繰り返す。
	{
		scanf("%d", &size);
	}
	b = new int[size];	//要素が不定な配列を動的確保
	for (int i = 0; i < size; i++)
	{
		b[i] = i;
		printf("b[%d] = %d¥n", i, b[i]);
	}
	delete []b;			//メモリの開放
	return 0;
```

```
}
```

//

3.オブジェクト指向とは

C言語は、データ構造とアルゴリズムが別々になってます。いままでは、それらを組み合わせてプログラムを組んでいました。一方、C＋＋のようなオブジェクト指向言語では、データ構造とアルゴリズムが一体化しています。

オブジェクト指向プログラムの利点は、再利用性、拡張性、保守性に優れている点です。

C＋＋に限らず、オブジェクト指向言語では**クラス**という概念を用います。以降、クラスの使い方を中心に、オブジェクト指向でのプログラミングを目指します。

オブジェクト指向に欠かせないキーワードは、カプセル化・継承・ポリモーフィズム（多態性）の三つです。これらのキーワードも後々紹介していきます。

4.クラス

クラスを使ってみましょう。オブジェクト指向プログラミングではクラスを用いてプログラムを組んでいきます。C言語で構造体がちゃんと理解できている人はクラスの理解も容易だと思います。大雑把に言うと、クラスとはメンバに関数が持てる、構造体の拡張版です（本当はこう書くのはよろしくないですがイメージを掴みやすくする為あえてこう書きました）。ではクラスの作り方を書きます。

```
class クラス名
{
    ・・・メンバ・・・
}
```

構造体と似ていますね。使い方は、

```
クラス名 変数名;
```

です。クラスを実体化したオブジェクトを**インスタンス**と呼びます。

メンバの呼び出し方法は構造体のときと同じように.（メンバ参照演算子）を使い、ポインタのときは->（アロー演算子）を使います。

クラスでは、構造体と違いメンバに関数を含めることが出来ます。オブジェクト指向言語ではこのような関数を**メソッド**といいます。メソッドの定義の仕方を書きます。

```
データ型 クラス名::関数名（引数）
{
        処理;
}
```

::はスコープ解決演算子です。要するに関数がどのクラスに属するのかを明示しているのです。

自動販売機をクラスで表現するとサンプル3のようになります。

//

```
/*
C＋＋講座サンプル 3
クラス
by y.t
*/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct tag_SYOUHIN
{
	char *name;
	int nedan;
	int zaiko;
}SYOUHIN, *LPSYOUHIN;
class JIHANNKI
{
//初期のアクセスコントロールは private（非公開メンバ）
	int m_InputMoney;
	int m_ItemNum;
	char *m_pCompany;
	LPSYOUHIN m_pSyouhin;
	int Disp();
public:                                          //公開メンバ
	JIHANNKI(char *companyName);                 //コンストラクタ
	~JIHANNKI();                                 //デストラクタ
	//メソッドの宣言
	int InputMoney(int Money);
	int ReturnMoney();
	LPSYOUHIN SelectItem(int ItemNumber);
};
//コンストラクタの定義
JIHANNKI::JIHANNKI(char *companyName)
{
	m_pCompany = companyName;
	m_pSyouhin = new SYOUHIN[3];
	m_pSyouhin[0].name = "MAX コーヒー";
	m_pSyouhin[0].nedan = 120;
	m_pSyouhin[0].zaiko = 3;
	m_pSyouhin[1].name = "ポカリスエット";
	m_pSyouhin[1].nedan = 150;
	m_pSyouhin[1].zaiko = 3;
	m_pSyouhin[2].name = "水";
	m_pSyouhin[2].nedan = 100;
	m_pSyouhin[2].zaiko = 3;
	m_ItemNum = 3;
```

```
	m_InputMoney = 0;
	Disp();
}
//デストラクタの定義
JIHANNKI::~JIHANNKI()
{
	delete []m_pSyouhin;
}
int JIHANNKI::InputMoney(int Money)
{
	m_InputMoney += Money;
	Disp();
	return 0;
}
int JIHANNKI::ReturnMoney()
{
	int Money = m_InputMoney;
	m_InputMoney = 0;
	return Money;
}
LPSYOUHIN JIHANNKI::SelectItem(int ItemNumber)
{
	LPSYOUHIN pSyouhin = NULL;
	if (ItemNumber > 0 && ItemNumber <= m_ItemNum)
	{
		int i = ItemNumber - 1;
		if (m_pSyouhin[i].nedan <= m_InputMoney && m_pSyouhin[i].zaiko > 0)
		{
			m_InputMoney -= m_pSyouhin[i].nedan;
			m_pSyouhin[i].zaiko--;
			pSyouhin = &m_pSyouhin[i];
		}
	}
	Disp();
	return pSyouhin;
}
int JIHANNKI::Disp()
{
	char msg[64];
	char state[16];
```

```
		printf("*****************************¥n");
		printf("%s¥n", m_pCompany);
		printf("投入金額:%d¥n", m_InputMoney);
		for (int i = 0; i < m_ItemNum; i++)
		{
			sprintf(msg, "%2d.%-10s", i + 1, m_pSyouhin[i].name);
			state[0] = '¥0';
			if (m_pSyouhin[i].nedan <= m_InputMoney)
			{
				if (m_pSyouhin[i].zaiko > 0)
				{
					sprintf(state, "■");
				}
				else
				{
					sprintf(state, "売り切れ");
				}
			}
			printf("%s %s¥n", msg, state);
		}
		printf("*****************************¥n");
		return 0;
}
int main()
{
		JIHANNKI Jihannki("sofume");
		int flag;
		LPSYOUHIN pSyouhin = NULL;
		while(1)
		{
			printf("1.金額投入¥n");
			printf("2.商品選択¥n");
			printf("3.返金¥n");
			printf("0.終了¥n");
			scanf("%d", &flag);
			switch(flag)
			{
				case 1:
					int Money;
					scanf("%d", &Money);
```

```
                                        Jihannki.InputMoney(Money);
                                        break;
                                case 2:
                                        int SelNumber;
                                        scanf("%d", &SelNumber);
                                        pSyouhin = Jihannki.SelectItem(SelNumber);
                                        if (pSyouhin != NULL)           printf("%s を購入¥n", pSyouhin->name);
                                        break;
                                case 3:
                                        printf("%d 円の返金¥n", Jihannki.ReturnMoney());
                                        break;
                        }
                        if (flag == 0)          break;
                }

                return 0;
}
```

//

クラスにはコンストラクタとデストラクタというものが存在します。コンストラクタはインスタンスが作成されたときに実行されるメソッドで、インスタンスを初期化する役割を持っています。デストラクタはインスタンスが破棄されたときに実行されるメソッドで、インスタンスの開放処理を行う役割を持っています。コンストラクタ・デストラクタを使用すると初期化処理・開放処理を自動化できます。

コンストラクタは型を持たない関数です。しかも、void 型ですらないので、コンストラクタの宣言では関数のデータ型を記述しません。コンストラクタの名前はクラス名を同じでなければいけません。引数は自由に設定することができます。また、コンストラクタはオーバーロードすることが出来ます。

デストラクタもコンストラクタ同様、型を持たない関数です。デストラクタの名前はクラス名と同じであり、かつ~（チルダ）を先頭に付けなければなりません。デストラクタは引数を持つこともオーバーロードすることも出来ません。

クラスにはアクセスコントロールというものがあります。アクセスコントロールとは、メンバ変数がどの範囲まで参照可能かを制御するものです。現段階では、private はクラス内でしかアクセスできない。public はどこからでも参照できる。と覚えておいてください。アクセスコントロールについては「継承」の時に詳しくやります。アクセスコントロールを利用してプログラマが不必要にメンバにアクセスできないようにすることでクラスの内部のデータ構造を隠蔽することをカプセル化といいます。

さて、サンプル3をCで書き換えるとどうなるでしょうか？サンプル4に書き換えた場合のソースコードを乗せます。

//

```
/*
C＋＋講座サンプル 4
サンプル 3Cで書いてみた
by y.t
```

```
第2回C＋＋講座（By y.t）
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//構造体の宣言
typedef struct tag_SYOUHIN
{
	char *name;
	int nedan;
	int zaiko;
}SYOUHIN, *LPSYOUHIN;
//関数の定義
int Disp(int InputMoney, int ItemNum, LPSYOUHIN pSyouhin, char *pCompany)
{
	char msg[64];
	char state[16];
	int i;
	printf("*****************************¥n");
	printf("%s¥n", pCompany);
	printf("投入金額:%d¥n", InputMoney);
	for (i = 0; i < ItemNum; i++)
	{
		sprintf(msg, "%2d.%-10s", i + 1, pSyouhin[i].name);
		state[0] = '¥0';
		if (pSyouhin[i].nedan <= InputMoney)
		{
			if (pSyouhin[i].zaiko > 0)
			{
				sprintf(state, "■");
			}
			else
			{
				sprintf(state, "売り切れ");
			}
		}
		printf("%s %s¥n", msg, state);
	}
	printf("*****************************¥n");
	return 0;
}
```

```
LPSYOUHIN SelectItem(int ItemNumber, int *InputMoney, int ItemNum, LPSYOUHIN pSyouhin)
{
	LPSYOUHIN tmp = NULL;
	if (ItemNumber > 0 && ItemNumber <= ItemNum)
	{
		int i = ItemNumber - 1;
		if (pSyouhin[i].nedan <= *InputMoney && pSyouhin[i].zaiko > 0)
		{
			*InputMoney -= pSyouhin[i].nedan;
			pSyouhin[i].zaiko--;
			tmp = &pSyouhin[i];
		}
	}
	return tmp;
}
int main()
{
	int InputMoney;
	int ItemNum;
	char *pCompany;
	LPSYOUHIN pSyouhin = NULL, tmp = NULL;
	int flag;
	int Money;
	int SelNumber;
	//初期化
	pCompany = "sofume";
	pSyouhin = (LPSYOUHIN)malloc(sizeof(SYOUHIN) * 3);
	pSyouhin[0].name = "MAX コーヒー";
	pSyouhin[0].nedan = 120;
	pSyouhin[0].zaiko = 3;
	pSyouhin[1].name = "ポカリスエット";
	pSyouhin[1].nedan = 150;
	pSyouhin[1].zaiko = 3;
	pSyouhin[2].name = "水";
	pSyouhin[2].nedan = 100;
	pSyouhin[2].zaiko = 3;
	ItemNum = 3;
	InputMoney = 0;
	Disp(InputMoney, ItemNum, pSyouhin, pCompany);
	while(1)
```

```
	{
		printf("1.金額投入¥n");
		printf("2.商品選択¥n");
		printf("3.返金¥n");
		printf("0.終了¥n");
		scanf("%d", &flag);
		switch(flag)
		{
			case 1:
				scanf("%d", &Money);
				InputMoney += Money;
				Disp(InputMoney, ItemNum, pSyouhin, pCompany);
				break;
			case 2:
				scanf("%d", &SelNumber);
				tmp = SelectItem(SelNumber, &InputMoney, ItemNum, pSyouhin);
				Disp(InputMoney, ItemNum, pSyouhin, pCompany);
				if (tmp != NULL)	printf("%s を購入¥n", tmp->name);
				break;
			case 3:
				printf("%d 円の返金¥n", InputMoney);
				InputMoney = 0;
				break;
		}
		if (flag == 0)		break;
	}
	free(pSyouhin);

	return 0;
}
```

//

サンプル 3 ではコード的に自販機とユーザの境界が明確になっていました（オブジェクト指向の利点）。ところが、サンプル 4 では自販機とユーザの境界がはっきりしません。入力に対して対応する変数が変化し、出力されているだけのプログラムに見えます。表向きは同じ「自販機プログラム」ですが、ソースの印象はまったく異なるものになります。

5.本日の関数

表 5.1 本日の関数

関数	引数	戻り値	処理内容
int sprintf(char *buffer, const char *format [, argument]…);	buffer 出力の格納場所 format 書式付き文字列 argument 表示したい変数	書き込まれた文字数	buffer に文字列を書き込む 必須ヘッダー stdio.h