ゲーム制作のための

C言語講座～第３回～

2007/08/16

２日間連続でやってきましたが、とうとう今日の講座を終えれば、一旦休みに入ります（１０日以上も）。連続で難しい話を続けてきたため、疲れている方も多いでしょう。でも、基礎的な部分を２日間でまとめたおかげで、残りの３日間はスローペースでもよさそうな感じかもしれません。（でも、内容はさらに難しくなっていくかと・・・）とりあえず、頑張って下さい。

あと、前回までの内容が理解できてない！と、思った方は気軽に質問してください。メール・メッセならいつでもいいですし､合宿とかに持ち込んできても OK です｡積極的にプログラミングを打った人、積極的に先輩たちに質問していった人とかが、最終的にプログラミングが使えるようになりますよ。

今日の講座は自作関数と構造体について話をしていきます。前回までの内容でとりあえず、何かを作ることはできるようになりました。ですが、これではまだ大きな作品を作る時に膨大なソースになってしまいます。そこで、この２つを使うことで、近い命令を１つにまとめたり、データの管理が楽になったりします。是非ともこの２つを吸収して、効率のよいソースを書いてみましょう。

では、今日の講座を始めます。

３－１　自作関数

今まで、関数といえば rand とか、printf とかやってきました。これは、stdlib.h や stdio.h といった別ファイルにどういうものかをあらかじめ書かれていました。メイン関数の中で、関数を呼び出す（srand とか printf とかを使う）時に括弧の中にデータを入れていたと思います。このデータを別ファイルにある関数の中身に持っていって、処理されます。その結果、データを出力できたり、入力できたり、乱数を出したりしていたのです。(2 次関数 $y=ax^2$ に x に数字を入れると計算されて y（別の値）が決まるのと同じです｡）今回、その関数を名前や出力する型、処理内容を自分で作ることができます。似たような処理の部分を関数にしてしまえば、それを呼び出すだけで、処理ができるようになります。つまり、何度も似たようなものを書く必要性がなくなるということですね。では、早速、書き方・動作を説明します。

source3-1a.cpp　自作関数１

```
#include<stdio.h>
int add(int data1 , int data2);
int main()
{
        int kazu1 = 10 , kazu2 = 20 , ans;
        ans = add(kazu1 , kazu2);
        printf("答えは%dです¥n" , ans);
        return 0;
}
int add(int data1 , int data2)
{
        data1 = data1 + data2;
        return data1;
```

}

まず、変数や配列の時と同様に宣言を行わなければなりません。2 行目がその宣言の部分です。これを行うことで、それ以降の文に add という名前が出てきても関数のことだと気がついてくれます。それで、宣言の方法ですが、　型 関数名（引数）；　という形になっています。

「型」というのは、変数と同じやつです。この型は「関数の計算結果の値の型」です。この関数の結果は整数を出すものなので、int 型で宣言しています。

「関数名」は自由に名前をつけて構いません。（ただし、main のような元々ある名前は不可）

「引数」というのは、今まで出てきた関数の括弧内のもののことです。関数に入れる処理前のデータのことを差しています。今回は、int 型の data1 と data2 の 2 つのデータを add 関数の引数にしています。

次に、メイン関数の中で呼び出しましょう。6 行目の右辺がそうです。関数名（引数にいれるデータ）という形で書いてください。この場合、add 関数で使われる data1 にメイン関数の kaze1 の値を代入しています。同じように、data2 に kaze2 の値を代入しています。そして、関数が呼び出された時点で、一度メイン関数での処理はストップして、add 関数の処理が始まります。

では、add 関数の処理を見てみましょう。10 行目に add 関数の中身が書かれています。自作関数の中身を書く時は　型 関数名（引数）｛内容｝という形で書きます。宣言の；が｛内容｝に変わっただけです。それ以外の部分（例：関数名）を変えてしまうとエラーが起きるので注意してください。

add 関数の中身を見てみると、最初に data1 = data1 + data2；と書かれています。これは、普通に演算しましょう。呼び出した時に引数として、値が代入されているはずなので、その値で計算すればよいです。

次の行に return data1；と書かれています。ここで、return の本当の意味を説明すると、この関数の計算結果として、値を呼び出された関数に返す働きがあります。今回では、data1 の値（＝３０）が結果として、メイン関数に返されます。つまり、ans = add(add1 , add2)；が add 関数の処理結果から add(add1 , add2) = 30 だとわかり、ans = 30；という風になったということです。

次に、関数を使う、使わないでどれくらい違うか比べてみましょう。

source3-1b.cpp　自作関数２before

```
#include<stdio.h>
int main()
{
	int i;				/*ループに使う変数*/
	printf("九九の5の段を出力します¥n");
	for(i = 1 ; i <= 9 ; i++)
	{
		printf("%d¥n" , i * 5);
	}
	printf("九九の7の段を出力します¥n");
	for(i = 1 ; i <= 9 ; i++)
	{
		printf("%d¥n" , i * 7);
	}
	printf("九九の8の段を出力します¥n");
	for(i = 1 ; i <= 9 ; i++)
```

```
        {
                printf("%d¥n" , i * 8);
        }
        return 0;
}
```

最初に関数を使わないプログラムです。全部で 21 行になりました。このソースは九九の 5 の段、7 の段、8 の段の答えを一通り出力するプログラムです。5～９行目、１０～１４行目、１５～１９行目の３つがワンパターンですね。ここの部分を、関数を作ってまとめてみましょう。

source3-1c.cpp　自作関数２after

```
#include<stdio.h>
void answer(int number);                    /*answer関数の宣言*/
int main()
{
        answer(5);                          /*５の段*/
        answer(7);                          /*７の段*/
        answer(8);                          /*８の段*/
        return 0;
}
/*answer関数の内容*/
void answer(int number)
{
        int i;          /*iはループ用の変数なので、answer関数の最初に宣言*/
        printf("九九の%dの段を出力します¥n" , number);
        for(i = 1 ; i <= 9 ; i++)
        {
                printf("%d¥n" , i * number);
        }
}
```

---

void 型の answer という関数を作りました。void 型というのは、返す値がない型です。つまり、return が不必要ということです。関数の引数に何の段かを表す int 型の number というものを宣言しています。メイン関数から呼び出す時に、number の値がいくつになるかを指定するだけで、「九九の～」という文章を出力してくれるし、number の値から計算をして、答えを出力してくれます。

このソースは全部合わせて１９行です。たった 5 行で 3 回出てくるところを関数にしてまとめただけで、行数を減らすことができました。行数が減るということは、処理が速くなるし、自作関数の書き方に慣れれば、少ない時間でソースを書くことができるということです。もっとたくさんの行の部分や、何回も近い形で出てくるところを関数にすれば、かなりの削減効果が期待できるでしょう。

３－２　構造体

そして、もう１つ構造体についてです。簡単に言うと、いくつかのデータを１つにまとめたグループです。まあ、ソースを見てみましょう。

source3-2.cpp　構造体

```
#include<stdio.h>
/*構造体の内容*/
typedef struct CHARACTER
{
        int life;               /*残りライフ*/
        int speed;              /*移動速度*/
        int j_power;            /*ジャンプ力*/
        int atk;                /*攻撃力*/
        int reach;              /*攻撃範囲*/
        int x , y;              /*座標*/
}character;
void show(character a);         /*show関数の宣言*/
int main()
{
        character player;
        character boss;
        player.life = 2 , player.speed = 2 , player.j_power = 10;
        player.atk = 7 , player.reach = 3;

                                （中略）

        show(player);

        return 0;
}
/*show関数の内容*/
void show(character a)
{
        printf("HP = %d¥n移動力= %d¥nジャンプ力= %d¥n" , a.life , a.speed , a.j_power);
        printf("攻撃力= %d¥nリーチ= %d¥n" , a.atk , a.reach);
}
```

まず、構造体のデータ内容を決めなければなりません。int が整数を扱うと事前に決まっているのと同じように、構造体を使う前に構造体にどんなデータを扱っているかを書かなければなりません。それははメイン関数の前に書きます。自作関数の宣言の前にしたほうがいいでしょう。それで、書き方は typedef struct 名前｛構造体の中の変数の宣言｝名前；となっています。

struct の後の名前は、その構造体の型の名前です。そして中括弧の後の名前は「struct 名前」の省略名称（愛称）になっています。つまり、character と書けば、struct CHARACTER だと判断してくれるということです。ちなみに、typedef を書いているから愛称を設定できます。typedef を使わない構造体ももちろん作れますが、こっちのほうが楽なので、typedef 使用のソースしか載せません。知りたければ、調べましょう。

これで構造体の型を作ることができました。あとは、その型を使用する関数の始めに宣言します。１５、１６行目がそうです。character の型を持つ player と boss の２つの構造体を宣言しました。変数の宣言とまったく変わりありませんね。ただ、複数のデータを持っているので、初期化は行うことはで

きませんので注意してください。１７、１８行目に player 構造体の中身である life や speed に値を代入しています。player. life で player 構造体の life を差しています。(構造体). (中の変数) という書き方をしてください。まあ、このソースではここに値を直接書いていますが、ゲームを作る時は、セーブデータから読み込んだり、初期化をする自作関数を呼び出したりして、値を決めると思います。

構造体の利点は 1 回型を作ると複数のものに汎用できるところです。今回、player と boss だけですが、friend とか enemy とか色々キャラクター用のデータが必要になっても、character を利用すれば関数の最初に 1 行宣言するだけで使用できるようになります。いちいち、全部の変数を宣言する必要性がなくなるのです。

また、作っている最中に変数が足らなくなることはよくあります。そのとき、構造体の型の内容を書いているところに追加するだけで、それを使用している全部の構造体に変数が追加できます。

さらにいうと、このソースの最後に show という自作変数があります。もし、構造体を作ってなければ、引数が 5 つ必要になるため、かなり面倒だと思います。構造体の型を事前に決めてあったため、構造体を引数にすれば 1 つだけ書くだけで十分なのです。

ただ、1 つだけ言っておきます。関係性の低いもの同士を 1 つの構造体内に入れないほうがいいです。構造体の利点を引き出せませんので。今回はキャラクターのデータというグループでまとめてありました。

３－３　まとめ

内容	関数・書き方
自作関数	(メイン関数前) 関数宣言－型　名前 (引数) ; (メイン関数後) 関数定義－型　名前 (引数) {内容} (使用する関数の中) 関数呼び出し－名前 (引数に入れる値)
構造体	(#include の後) 構造体の型の宣言－typedef　struct 名前{中身}愛称 ; (使用する関数の中) 構造体宣言－愛称　名前 ; (使用する関数の中) メンバアクセス－ (構造体名). (中身の変数)

３－４　演習

自作関数を利用して、昨日作った「じゃんけんゲーム」を簡単化できるかやってみる。(ソースの構造上できなければ、仕方ないです。でも、少しでも改善できそうならば挑戦してください。)

次の講座 (8/29) までに、簡単なゲームとかアプリケーションを作ってみる。

(例) 丁半・High and Low のようなギャンブル、簡単なすごろく、簡単な RPG の戦闘、マインスイーパー (去年、堤さん作製の C 言語テストで出てきた)、ソート (並べ替えのこと)・・etc