ゲーム制作のための

C言語講座～第５回～

2007/08/30

さて、今日でこの講座最終回です。５回でうまく教えられるか不安でしたが、何とか最低限の知識は教えられそうです。ただ、ハイスピードで薄くやってきたため、あまり身についていない人は少ないかもしれません。この講座を踏み台にして、これから先、独学でやっていってもらえると嬉しいです。（まあ、後期にやってほしい講座があれば、考えなくも・・。むろん、私の実力で教えられる範囲で。）

では、今日はファイル入出力関係です。これも重要なものですね。ゲームのセーブ・ロードだけでなく、アプリケーションではユーザが入力した個人設定の保存など、再度プログラムを実行した時に前の状態から続けられるようにするには必須の技術です。

では、今日の講座を始めます。

５－１　ファイルへの書き込み

まずは、ファイル読み書きをする時の手順を説明しておきます。

① ファイルオープン（指定したファイルにいつでも読み書きができるようにするための準備）
② 実際にファイルにアクセスして読み書きを行う
③ ファイルクローズ（後片付けのようなもの）

という手順で、ファイルに読み書きをします。では、どういう命令でそのような操作をするか、ソースを見て確認してみましょう。

source5-1.cpp　ファイル書き込み

```
#include<stdio.h>
typedef struct PLAYER
{
        int HP;
        int MP;
        int atk;
        int def;
}Player;
int main()
{
        FILE *fp;                                       //ファイル構造体のポインタfp
        Player player;
        player.HP = 100 , player.MP = 50;
        player.atk = 30 , player.def = 20;
        fp = fopen("data.txt" , "w");                   //ファイルを開く
        fprintf(fp , "%d , %d¥n" , player.HP , player.MP);
        fprintf(fp , "%d , %d¥n" , player.atk , player.def);
        fclose(fp);                                     //ファイルを閉じる
        return 0;
```

}

１１行目 FILE ＊fp とあります。ファイル入出力にはファイルに関する情報を格納するために、「FILE」という構造体の型を持つ構造体のポインタが使うファイル分だけ必要です。そのため、fp というポインタを宣言しています。

①ファイルオープン

ファイルオープンは１行で済みます。１５行目の fopen（ ）関数がそうです。１つ目の引数には、対象となるファイルの名前を入れます（ファイル名も文字列なので“”は忘れずに）。２つ目の引数には、そのファイルを開く目的（オープンモード）を書きます。その関数で得た結果をファイルポインタの中に代入してください。

オープンモード

w	ファイル書き込みをするためのもの。（ファイルは新規作成） 指定したファイル名と同じ名前のファイルがある状態でやると、すべて上書きされるので注意。
r	ファイル読み込みをするためのもの。（ファイルは既存のもの） むろん、読み込むので、指定したファイルが無いと駄目です。
a	追加書き込みをするためのもの。 指定したファイルが無いと、新規作成。あるとデータの終わりの部分から追加書き込みをします。
w+ または r+	読み込みと書き込みの両方ができるもの。 w+は新規作成。r+は既存のファイル。
a+	読み込みと追加書き込みの両方ができるもの。
(w、r、a の後に)b	バイナリ（２進数）の状態で読み込みとか書き込みをするときのもの。

②ファイル書き込み

このソースでは、ファイル書き込みをしています。ファイル書き込みをするためには fprintf 関数を使います。使い方は printf とほぼ一緒で、文字列の前にファイルポインタを指定するということです。これにより、ファイルオープンした時に指定したファイルへ出力する形になります。

③ファイルクローズ

fclose 関数というのがあります。ソースの中では、fclose(ファイルポインタ)；というのがそれです。書き込んだ内容を保存する役割も持つため、ファイル書き込みが終わったら、ちゃんと書いてください。

どうでしょうか？割と簡単にファイルに書き込みができました。このソースを実行したら、ソースが入っているフォルダを見てみましょう。ちゃんと data.txt が存在していて、中に fprintf で出力したものが書いてあれば OK です。

５－２　ファイル読み込み

ファイル読み込みも①～③の流れは変わりません。変わっているところは、①でオープンモードをｗ→ｒにするのと、②では入力なので別の関数を使うことです。では、ソースを見てみましょう。

source5-2.cpp　ファイル読み込み

```
#include<stdio.h>
```

```
typedef struct PLAYER
{
	int HP;
	int MP;
	int atk;
	int def;
}Player;
void show(Player *p);
int main()
{
	FILE *fp;                                          //ファイル構造体のポインタfp
	Player player;
	fp = fopen("data.txt" , "r");                      //ファイルを開く
	if(fp == NULL)
	{
		printf("セーブファイルが存在しません¥n");
		return -1;
	}
	fscanf(fp , "%d , %d" , &player.HP , &player.MP);
	fscanf(fp , "%d , %d" , &player.atk , &player.def);
	show(&player);
	fclose(fp);                                        //ファイルを閉じる
	return 0;
}
void show(Player *p)
{
	printf("playerのHP = %d¥n" , p->HP);
	printf("playerのMP = %d¥n" , p->MP);
	printf("playerのatk = %d¥n" , p->atk);
	printf("playerのdef = %d¥n" , p->def);
}
```

今回のソースはさっきのファイル書き込みで出た data.txt からデータを取得しています。

①ファイルオープン

w→r になりました。読み込みようとしてファイルと開くということです。ただ、読み込む時に指定したファイルが無いとデータを取得することができません。その時のための対策が必要になってきます。

読み込みモードでファイルオープンした際に、指定ファイルが無いと fopen 関数の結果は NULL というものになります（つまり、fp の値が NULL になる）。なので、if(fp＝＝NULL)と書けば、指定ファイルが無い時の処理を書くことができるのです。このソースでは１５～１９行目がその部分です。「セーブファイルが存在しません」と表示された後、return を使ってメイン関数を終了させています。

②ファイル読み込み

書き込みでは fprintf ですが、読み込みでは fscanf 関数を使います。これも引数の最初に fp が追加されただけで、scanf と書き方はほとんど変化ありません。

③ファイルクローズ

書き込みと変わりありません。忘れずに行ってください。

ちゃんと思い通りの結果が出ましたか？data.txt に書かれていることと比較してくださいね。

Q、１つのファイルを複数の関数上で読み書きできますか？（ファイルオープンした関数以外の関数でもファイル読み書きは可能ですか？）

A、できます。一回ファイルオープンすれば、読み書きの準備は完了しているからです。ただ、fprintf 関数や fscanf 関数はどこに読み書きすればいいか、場所を指定していました。（文字列の前に毎回書いていた fp がそうです。）なので、ファイルポインタを別の関数の引数に渡す必要はあります。
参考として以下がそのソースです。

source5-2q.cpp　複数の関数でファイル入出力

```
#include<stdio.h>
typedef struct Number
{
        int a;
        int b;
        int sum;
}Number;
void read(Number *n , FILE *fp);
void add_write(Number *n , FILE *fp);
int main()
{
        FILE *fp;                                                    //ファイル構造体fp
        Number number;
        fp = fopen("data.txt" , "r+");          //ファイルを開く
        if(fp == NULL)
        {
                printf("セーブファイルが存在しません¥n");
                return -1;
        }
        read(&number , fp);
        add_write(&number , fp);
        fclose(fp);                                                  //ファイルを閉じる
        return 0;
}
void read(Number *n , FILE *fp)
{
        fscanf(fp , "%d" , &n->a);
        fscanf(fp , "%d" , &n->b);
}

void add_write(Number *n , FILE *fp)
{
        n->sum = n->a + n->b;
        fprintf(fp , "¥n%d" , n->sum);
```

```
}
```

５－３　その他の知識色々

まあ、ファイル入出力に関してはこれくらいで十分でしょう。まだ、時間がありそうなので、使えると便利な知識をちょっと紹介して、C 言語講座終了にします。

キャスト

これは変数の型を一時的に別の型にするものです。とりあえず、ソースを見ましょう。

source5-3a.cpp　キャスト

```
#include<stdio.h>
int main()
{
        int kazu = 100;
        int kazu2 = 3;
        float answer;
        answer = (double)kazu / kazu2;
        printf("%f¥n" , answer);
        return 0;
}
```

７行目の（double）というのがキャストです。元々、kazu は int の変数ですが、kazu / kazu2 の計算をする時だけ、一時的に double の型にして計算しました。

理由は（double）をはずすとわかります。はずしてみると、結果の小数部分が 0 になっています。これは、int / int で出る値は int なので、その値を answer に代入しているためです。最低でもどちらかが float や double でないと、割った時の答えの小数部分がなくなってしまいます。

マクロ（#define）

プログラムを書く時、結構定数を使っていませんか？for 文の繰り返しの回数とか、配列の個数とか・・・。そして、プログラムを組んでいる最中にそれを見て、その数字が何を示しているかちゃんと思い出せるでしょうか？

多分、大きなプログラムを書く場合、答えは NO です。定数がありすぎて、なぜその数値だったかを忘れてしまうでしょう。それを防ぐため、定数に名前をつけることができます。それが#define というものです。

source5-3b.cpp　マクロ

```
#include<stdio.h>
#define LOOP_NUMBER 5     /*ループする回数*/
int main()
{
        int i , sum = 0;
        for(i = 1 ; i <= LOOP_NUMBER ; i++)
        {
```

```
            sum += 10;
        }
        printf("%d¥n" , sum);
        return 0;
}
```

２行目に#define を使って、定数に名前をつけています。この場合、「LOOP_NUMBER」という名前が出てきたら、それは「５」であると定義しています。なので、６行目の for 文の中の LOOP_NUMBER というところが５として処理をされます。

ちなみに、名前とか作る以上メモリを消費しているのではないかと思った人もいるかと思いますが、これはコンパイル前の段階で LOOP_NUMBER を５に変換してしまいます。なので、このプログラムを実行しても#define のせいで多くメモリを確保するということはありません。

５－４　演習

今までの知識を生かして、コンソール上で動くゲームを作製すること。

第３回のあとに何かゲームを作った人は今回のファイル入出力を利用して、さらに拡張してみて下さい。

これで、C 言語講座は終了です。まあ、本当に最低限のことしか教えていない上に、わかりにくい部分もあったと思います。ですので、ちゃんと勉強してくださいね。プログラムは勉強するまでが大変という人も多いようです。勉強が終わると、自分が納得するまで「プログラムを書く」⇔「実行してみてイメージ通りか確認」を繰り返していくだけです。自分が作りたいと思っていたものが完成したときは、本当に感動すると思いますよ。では、頑張ってくださいね～。