ゲーム制作のための

C言語講座～第４回～

2007/08/29

みなさん、お久しぶりです。１０日間以上の休みの間にプログラミングは組みましたか？10日間の中であまり質問が無かったので、多分みんな大丈夫だと思っていてもいいのでしょうか・・・？今回のやる部分は初心者が一番挫折するところです。（正直、この部分を無視しても一応ゲームは作れるため、使わずにプログラムを組む人も結構いたりしますが・・。）前のところでつまずいたままでは、辛いと思います。復習も兼ねて自作関数についてのソースを書いてみました。（1日目、2日目の復習はしません）

source4-0.cpp　自作関数の復習＋変数領域

```
#include<stdio.h>
int change(int a , int b);
int main()
{
        int a = 1 , b = 2;        /*aは１、bは２*/
        printf("a = %d , b = %d¥n" , a , b);
        a = change(a , b);        /*aは？、bは？*/
        printf("a = %d , b = %d¥n" , a , b);
        return 0;
}
//変数の値を変える関数
int change(int a , int b)
{
        a++;              /*aは２*/
        b++;              /*bは３*/
        b = a + b;        /*bは５*/
        return b;
}
```

どうでしょう？理解できていますか？

メイン関数の前に「関数の宣言」をして（2行目）、メイン関数の後にその関数の処理内容を示す「関数の定義」を書きます（１２行目）。そして、どこかの関数の中で、「関数呼び出し」（7行目）をすることで、自作関数の処理が開始されます。復習してあれば、どういう順番で処理しているかはわかると思います。

ただ、１つ注意しておきたいところがあります。このプログラムを実行した時に何が出力されるでしょうか？まあ、１つ目のprintfは問題ないでしょう。では、２つ目はどうでしょうか？

「メイン関数の最初ではaが１で、bが２。次に・・・・。」と、いうように考えてみてください。

a が２で、b が５と出力されると思った人は・・・、
残念ながら間違っています。そもそも change 関数の return で b の値（この時は５）を返しています。a = change(a , b)なので、その５という値を a に代入していますね。そのため、a が５でなければおかしいですよ。
a も b も５だと思った人は・・・・、
それも違います。と、いうのもメイン関数の a、b と change 関数の a、b は名前が同じですが、別々の変数です。変数というのは変数宣言をした関数の中でしか使うことはできません。つまり、メイン関数で宣言した変数を別の関数内の計算に使おうとしても、エラーになってしまいます。こういうルールがあるため、別の関数内で同じ名前の変数を宣言しても、別の変数と勝手に認識してしまいます。（少しでも紛らわしいと思った人は違う名前の変数にしてしまうのがいいでしょう。）
と、いうことでメイン関数の b という変数の値は変化しないため、a は５、b は２になります。

このように、自作関数は処理を行うことで１つの結果を算出することはできますが、呼び出した関数（今回はメイン関数）内の変数の値を直接変えることはできません。では、別の関数によって変数の値を変えたい時はどうするかというと、今日教える「グローバル変数」と「ポインタ」を使えばいいのです。

４－１　グローバル変数

関数の中で宣言された変数はその関数内でしか使用できないのなら、関数の外で宣言しよう！というのが、「グローバル変数」です。自作関数の宣言や構造体の型の宣言と同じように、メイン関数の前に変数を宣言します。その変数はどの関数でも使用することができます。つまり、複数の関数で共有できる変数ということです。では、ソースを見てみましょう。

source4-1.cpp　グローバル変数

```
#include<stdio.h>
int global = 5;
void change(int a);
int main()
{
	int a = 1;
	printf("a = %d , c = %d¥n" , a , global);
	change(a);
	printf("a = %d , c = %d¥n" , a , global);
	return 0;
}
//値を変更する関数
void change(int a)
{
	a = 10;
	global = 20;
}
```

２行目の int global = 5 ; がグローバル変数です。ここで宣言した以上、どの関数内でも宣言することなく使うことができます。
７行目に global という変数を出力しました。この段階では何も変化が起きてないため、初期値の５が

出力されます。
　８行目に change 関数が呼び出されました。なので、一旦メイン関数から change 関数の処理に変わります。change 関数の中で、global という変数に２０という値を代入させました。その後、change 関数が終了し、メイン関数に戻ります。９行目にまた global を出力します。メイン関数の global と change 関数の global は同じものなので、値は２０です。よって、２０が出力されるようになります。

　普通の変数だと、宣言した関数が終了すると同時にメモリ上から必要性がなくなります。そのため、メモリの空きスペースが増えるので、メモリを効率よく使うことができます。しかし、グローバル変数はプログラムの最初から最後までずっと残り続けてしまいます。数個程度ならまだいいのですが、たくさん使用するとその分だけメモリを多く消費してしまいます。処理が遅くなる原因にもなるので気をつけてください。

４－２　ポインタ

　さて、初心者が一番挫折する部分がやってきました。「ポインタ」についてやっていきます。
　まず、メモリというのは知っていますよね？データを格納しているハードウェアです。そこには大量のデータを入れることができますが、そのデータを管理するために番地（アドレス）というものを使っています。どの位置かを表している数値です。
　今まで、変数を宣言して、計算をいろいろしてきましたが、変数を宣言した時にメモリ上にその変数のデータを格納するための場所が設けられます。そのアドレスはパソコン側が勝手に決めてしまいます。
　そのため、アドレスを気にすることなくプログラムを組むことができるのです。ですが、ここでアドレスを指定して、その中のデータを参照したり、計算に利用したりするのが「ポインタ」というものなのです。
　実際にポインタと自作関数を利用したソースを見て、どういうものか知っていきましょう。

source4-2.cpp　ポインタ

```
#include<stdio.h>
void change(int *pA , int *pB);
int main()
{
	int a = 1 , b = 2;
	printf("a = %d , b = %d¥n" , a , b);
	change(&a , &b);
	printf("a = %d , b = %d¥n" , a , b);
	return 0;
}
//変数の値を変える関数
void change(int *pA , int *pB)
{
	*pA += 2;			/*aは３*/
	*pB += 3;			/*bは５*/
	*pB = *pA + *pB;		/*bは８*/
}
```

　今日の最初のソースを、ポインタを使用した形に書き換えたものです。まあ、形だけ作れればいいという人は、「関数の宣言と定義のところで＊、関数呼び出しのところで&を前につける」というやり方

を知っておけばいいでしょう。呼び出した元の関数（メイン関数）にある a と b の値を直接変化させることができる自作関数になります。（a と pA、b と pB がそれぞれ対応しています。）

まあ、そんな話じゃ納得できない人や C 言語を勉強したい人のためにちゃんと説明します。

まず、関数呼び出しにある＆a と＆b について説明します。変数の前に＆をつけることで、その変数のデータが入っているアドレスを示すことになります。今回の場合だと、change 関数の引数に入る値をそれぞれ a と b のメモリの番地にしているということです。

int ＊pA と int ＊pB についてです。これは「ポインタ変数」と呼ばれるものです。変数名の前に＊をつけることでポインタ変数になります。通常の変数だと、その箱に合った何かしらの値が入りますが、＊をつけることでアドレスのみを扱う箱に変わるのです。

では次に、ポインタ変数の利用方法です。通常の変数では変数名を書くだけで、その中の値を計算に使用したり、出力したりできました。同じように、ポインタ変数も変数名だけで、その中に入っているアドレスを処理することができます（ポインタ演算）。

また、通常の変数では変数名の前に「＆」をつけることで、その変数のアドレスを示していました。ポインタ変数の場合、「＊」をつけることで、その中に入っているアドレスのところにある変数の値を利用することができるのです。たとえば・・・、

```
int a=１０；
int ＊pA；
pA = &a；
printf（“%d¥n”, ＊pA）；
```

と、書くと 3 行目に変数 a のアドレスを pA というポインタ変数の中に入れています。そうすると、4 行目の＊pA は「その中に入っているアドレス（＝a のアドレス）のところにある変数（＝a)」ということになります。つまり、変数 a に入っている１０という値が出力されるということです。

初めてポインタを習った人は結構わかりにくいかもしれません。ですが、今すぐ理解しなくてもかまいません。本やインターネットなどでいろいろ調べてみるといいでしょう。

４－３　まとめ

グローバル変数

通常の変数	関数の内部で宣言	宣言した関数内でしか使用できない。 また、関数が終了すると、メモリが開放される。
グローバル変数	メイン関数の前に宣言	どの関数でも使用可能。（別の関数内で値が変化すると、その値で処理される） プログラムが終了しない限り、メモリ上に残る。

ポインタ

通常の変数	型　変数名 （int a）	変数名…その変数内の値を指す。 ＆変数名…その変数のメモリアドレスを指す。
ポインタ変数	型　＊変数名 （int ＊a）	変数名…その変数内の値（何かのアドレス）を指す。 （pA = &a；だと変数 a のアドレスが pA に入っている） ＊変数名…その変数内に入っているアドレスが指している変数。 （上の続きであれば、＊pA は変数 a を指している）

ポインタ変数の中のアドレスが別の関数内で宣言された変数のアドレスであれば、その変数の値を直接変化させることが可能。

４－４　問題

次の問題の答えは何か考えてみよう。

int ＊Point；
int i ， a；

i = 800；
Point = &i；

Point = 100；
a = ＊Point ＊ 20；

i = i + 60；

（DX ライブラリ製作者のサイトより引用）

この時の、変数 a と i の値を求めよ。
答えは教えませんよ。解けた人も解けなかった人もプログラムを組んで、答えあわせをしてみましょう。

４－５　補足

ポインタ変数の使い方は大体わかったでしょうか？今のところ、ポインタ変数の型が int しか扱っていませんでした。ですが、型であれば、何でも OK なのです。double、配列、構造体・・・。ただし、ポインタ変数の型と参照している変数・構造体の型が同じであることが条件です。

あと、ポインタの中に入っているアドレスは参照する変数・構造体の最初のアドレスのみです。なので、配列や構造体のようなグループに対して使用すると、メモリの使用量が減るということになります。

まず、配列のポインタを見てみましょう。

source4-5a.cpp　配列のポインタ

```
#include<stdio.h>
int add(int *pT);
int main()
{
	int hairetu[6];
	int i , sum;
	for(i = 0 ; i < 6 ; i++)
	{
		hairetu[i] = 10 + i * 3;
	}
	sum = add(hairetu);
	printf("合計は%d¥n" , sum);
	return 0;
}
int add(int *pT)
{
	int i , sum = 0;
	for(i = 0 ; i < 6 ; i++)
	{
		sum += *(pT + i);
	}
```

```
        return sum;
}
```

１１行目と２０行目以外は大丈夫でしょう。普通の配列ですし、関数宣言もポインタ変数を引数にしているだけです。

１１行目に add(hairetu)というのがあります。この「hairetu」は「&hairetu[0]」と同じ意味です。配列の名前だけ書くと、配列の最初の部分のポインタになることを覚えておいてください。

次に、２０行目の pT＋i について話します。これは、pT に入っている要素から見て、i だけ先にある要素を参照するという意味です。配列を宣言した時、連続してメモリ上に格納されます。pT は配列 hairetu[0]のアドレスなので、i=2 の時は２つ先の要素、つまり hairetu[2]のアドレスになります。その計算をした後に、＊（ ）と括弧でくくっているため、hairetu[2]の中にある値を参照しているということになります。

このように、ポインタの中にあるアドレスを＋やーで計算することを「ポインタ演算」と呼びます。以下が、その一覧です。決して、アドレスに１を足すという意味ではないので注意してください。

ポインタ演算子

＋	p+2	p が指す要素の２つ先の要素のアドレス
－	p-2	p が指す要素の２つ前の要素のアドレス
	p1 – p2	p1 と p2 の間にある要素の数を調べる
＋＋	p++	p が指す要素の次の要素のアドレス
－－	p－－	p が指す要素の前の要素のアドレス

最後に、構造体のポインタの説明をします。構造体のポインタを使う時はちょっとやり方が変わる部分があります。3 日目の資料の source3-2.cpp を、ポインタを使って処理したソースを書きましたので、見てみましょう。

source4-5b.cpp　構造体のポインタ

```
#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);

int main()
{
```

```
        character player;
        character boss;
        player.life = 2 , player.speed = 2 , player.j_power = 10;
        player.atk = 7 , player.reach = 3;


        show(&player);

        return 0;
}

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);
}
```

まあ、関数宣言、呼び出しに関しては変数の時と変わりありません。ただ、定義の部分を見てみましょう。

ポインタを使わなかった時は、a. life というように「.」を使って、構造体の中の変数にアクセスしていました。しかし、今回は「－＞」という記号になっています。これは「アロー演算子」と呼ばれ、構造体のポインタの中の変数にアクセスする時に使用するものです。これは構造体のポインタを使う上でのルールなので、覚えておいて下さい。

－＞	アロー演算子	構造体のポインタを使用する時に必要。 中の変数にアクセスする時は「.」ではなく、「－＞」を使用する。

４－６　演習

「数学」「物理」「英語」の３教科の得点をユーザに入力させ、その「合計」と「平均」を算出するプログラムを作成しなさい。ただし、３つの値の「合計」と「平均」を算出する自作関数を作ること。

① 構造体でまとめれば、自作関数の引数を１つにできる。
② ポインタを使うと、なお良い。（というか、今日はポインタの講座なので・・・。）
③ 「平均」は小数で。