第５回Ｃ言語講座

１．　変数の範囲（スコープ）

変数には有効範囲があります。この有効範囲をしっかりと把握しないと狙い通りのプログラムは作れません。

変数には特定の範囲でのみ使えるローカル変数と、プログラム全体で使えるグローバル変数があります。さらに、有効範囲を超えるとデータが無効になる自動変数とデータを保持し続ける静的変数があります。

それぞれの変数についてのサンプルを以下に示します。

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/*
C言語講座サンプル1
変数のスコープ
by y.t
*/
#include <stdio.h> /*入出力のヘッダーファイル*/

int g_d;  //グローバル変数

void func();

/*メイン関数*/
int main()
{
	int a;	//ローカル変数
	g_d = 0;
	for (a = 0; a < 10; a++)
	{
		func();
		g_d++;
	}
	return 0;
}

void func()
{
	int a = 0;		//ローカル変数
	static b = 0;		//静的変数
	a++;
	b++;
```

```
        printf("a = %d¥nb = %d¥n", a, b);
        printf("g_d = %d¥n", g_d);
}
```

//

func 関数の a は自動変数なので関数が終わるたびに a の値は破棄されてしまいます。ところが、b は静的変数なので関数が終わっても b の値を保持し続けます。

２．　ポインタ

ポインタはC 言語の核となる重要な知識です。ポインタとは、変数が確保されている場所を表します。変数を宣言すると、その変数はメモリ上のどこかに確保されます。C言語ではそれをメモリのアドレスという形で表現します。

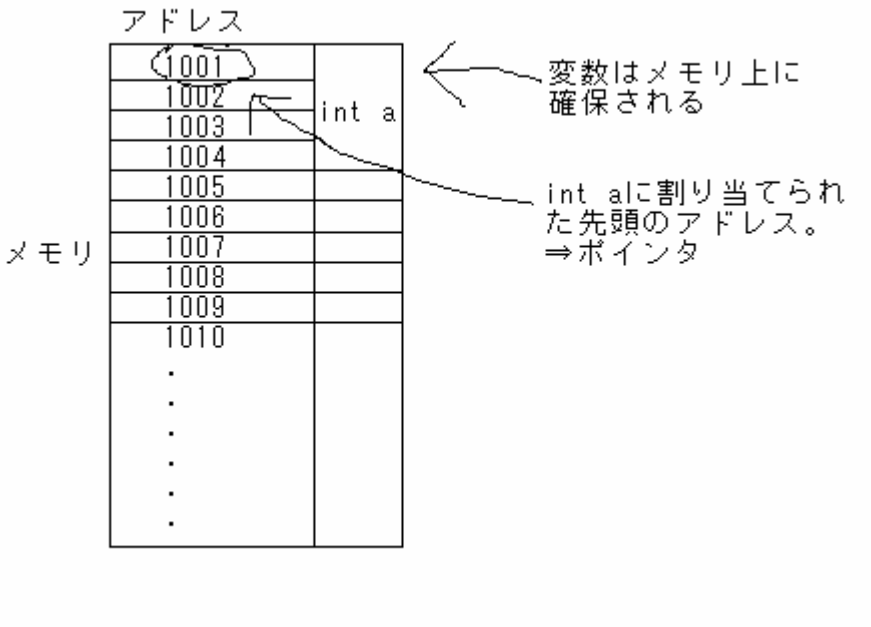

図1. 変数とメモリの関係

変数のポインタを知るには変数名の左に&をつけます。例えば、a という変数があったらそのポインタは&a となります。scanf 関数で、引数には&をつけていましたが、あれは変数のポインタを関数に渡していたのです。

ポインタを格納するポインタ変数というものがあります。ポインタ変数の宣言は変数宣言時にデータ型の右に*をつけます。ポインタのサンプルを以下に示します。

//

```
/*
C言語講座サンプル2
ポインタ
by y.t
*/
#include <stdio.h> /*入出力のヘッダーファイル*/
/*メイン関数*/
int main()
```

```
{
	int a;
	int *p;
	a = 10;
	printf("a = %d¥n", a);		//a のデータ
	printf("&a = %p¥n", &a); //a のポインタ
	p = &a;				//ポインタ変数 p に a のポインタを代入
	printf("p = %p¥n", p);		//ポインタ変数 p のデータ
	printf("*p = %d¥n", *p);	//p が参照しているデータ
	return 0;
}
```

//

ポインタ変数にはポインタを代入できます。ポインタ変数に*をつけるとポインタ変数が指しているアドレスの値を参照することが出来ます。

scanf 関数の引数の&はポインタである。というタネ明かしをしました。それでは、なぜポインタが必要なのでしょうか？実は、関数の引数で渡された値はコピーであり、関数でどんな処理をしようと元の変数は影響を受けないのです。そこで、書き換えたい変数のポインタを渡すことで、その変数がメモリ上のどこにあるのかを知らせることが出来ます。scanf の場合、変数を入力されたデータで書き換える必要があるため変数のポインタを渡しているのです。ポインタを引数で渡したサンプルを以下に示します。

//

```
/*
C言語講座サンプル3
ポインタを引数で渡す
by y.t
*/
#include <stdio.h> /*入出力のヘッダーファイル*/
int Add1(int a, int b);
void Add2(int *a, int b);
/*メイン関数*/
int main()
{
	int a, b;
	a = 10;
	b = 10;
	Add1(a, b);
	printf("a = %d¥n", a);
	Add2(&a, b);
	printf("a = %d¥n", a);
	return 0;
}
```

```
int Add1(int a, int b)
{
	a += b;
	return a;
}

void Add2(int *a, int b)
{
	*a += b;
	return;
}
```

//

ポインタを引数として渡すと、書き換えたい変数がメモリ上のどこにあるかが分かるため、そのアドレスに格納されている値を書き換えることが出来るのです。

３．　ポインタの扱いいろいろ

今までは単独の変数についてのポインタの扱い方をやりました。それでは、配列や構造体はどのようにポインタを扱うのでしょうか。その方法をやっていきます。

配列からやりましょう。配列のポインタを表したサンプルを用意しました。

//

```
/*
C言語講座サンプル4
配列とポインタ
by y.t
*/
#include <stdio.h> /*入出力のヘッダーファイル*/

int Sum(int *a, int size)
{
	int i;
	int sum = 0;
	for (i = 0; i < size; i++)
	{
		sum += a[i];
	}
	return sum;
}

/*メイン関数*/
```

```
int main()
{
	int a[5] = {3, 6, 4, 7, 1};
	int s;
	printf("&a[3] = %p¥n", &a[3]);	//a[3]のポインタ
	printf("&a[3] = (a + 3) = %p¥n", a + 3);	//a[3]のポインタの別表現
	printf("&a[0] = a = %p¥n", a);	//a は配列の先頭のポインタ
	s = Sum(a, sizeof(a) / sizeof(int));	//配列の先頭を引数に渡す
	printf("s = %d¥n", s);
	return 0;
}
```

//

重要なことは、配列のポインタはその配列の先頭のアドレスを示す。ということだけです。配列というのは先頭のアドレスを基準に相対的にメモリを参照しているだけなので、最低限&a[0]と a は同じものであると覚えておけばいいのです。

次に構造体のポインタです。

//

```
/*
C言語講座サンプル5
構造体のポインタ
by y.t
*/
#include <stdio.h> /*入出力のヘッダーファイル*/
#include <stdlib.h>
#include <string.h>
//構造体の宣言
typedef struct HERO
{
	int hp;
	int mp;
	int atk;
	int def;
}HERO; //struct HERO は HERO というデータ型として扱う。
/*メイン関数*/
int main()
{
	HERO h;
	HERO *hero = &h;	//HERO 型のポインタを宣言
	char name[64];
	int nameSize;
```

```
        printf("名前を入力してください¥n");
        scanf("%s", name);
        nameSize = strlen(name);//strlen：文字列の長さを取得する
        srand(nameSize);
        hero->hp = rand() % name[0];
        hero->mp = rand() % name[nameSize - 1];
        hero->atk = rand() % name[nameSize / 2];
        hero->def = rand() % name[nameSize / 3];
        printf("HP = %3d¥nMP = %3d¥nATK = %3d¥nDEF = %3d¥n", hero->hp, hero->mp,
hero->atk, hero->def);
        return 0;
}
```

//

ポインタで宣言された構造体のメンバの参照は”.”ではなく”->”で行います。“.”を使うことも可能ですが、その場合は”(*変数名).”と記述しなければならないので面倒です。

４．　本日の演習

1. 配列の合計と平均を求める関数を作りなさい。