第４・５回Ｃ言語講座

2011/5/30 & 6/6

メインテーマ：ポインタ、配列＆文字列とその扱い

①ポインタ

〇メモリのアドレス（イメージ）

変数	メモリ		アドレス※
	・		・
	・		・
X	3	:	2000
Y	5	:	2004
num	3	:	2008
sum		:	2012
	・		・
	・		・
西千葉	千葉大学		東経 140 北緯 35

※実際のアドレスは実行時にコンピュータが空いているところを探して勝手に割り当てる。

コンピュータがプログラムを実行する際、変数などのデータをメモリ上に置いて処理を行う。
前回まで変数の宣言、定義などを行ってきたが、それらはメモリ上のどこかに記録されていた。どこかはわからないし、とくに意識する必要はない。
現実のもので表すと、左下のようなイメージになる。

〇ポインタ

・ポインタ

変数はメモリ上にその値が保存されている“場所”がある。
その場所（アドレス・住所）の情報を持てる変数がポインタともいえる。
主に後述する変数の値を書き換える関数や配列、文字列などに利用される。

```c
#include <stdio.h>
int main(void)
{
	int num; //いつもの変数
	int* pnum; //ポインタ（場所を示す変数）
	pnum = &num; //ポインタに変数numの場所を教える
	num = 100;

	printf("numの値は%d¥n", num);
	printf("pnumが指す住所にある変数の値は%dです¥n¥n", *pnum);

	printf("変数numの住所は%d¥n", &num);
	printf("pnumが指す住所は%dです¥n¥n", pnum);


	return 0;
}
```

・ 関数②（引数の値を書き換える）

ｘとｙの値を交換する関数 swap を自作したいとする。

実は普通に swap（ int x , int y）と宣言して中身を記述しても x の値とｙの値は入れ替わらない。

なぜかというと上記のように書くとｘとｙの本体ではなく、その値のコピーが関数に渡されるから。

そこで変数本体を渡すにはその変数のアドレスを渡してあげる必要がある。

scanf 関数で scanf（ "%d" , &num ）としていたのもこのため。（ &num の部分）

```c
#include <stdio.h>
//ｘとｙを入れ替える関数
int swap( int* px , int* py )
{
        int temp;
        temp = *px; //ｘとｙの値の交換
        *px = *py;
        *py = temp;
        return 0;
}

int main(void)
{
        int num1 , num2;
        num1 = 10; //値は適当に
        num2 = 7;
        printf("num1 = %d , num2 = %d¥n", num1, num2);
        swap( &num1 , &num2);
        printf("swap関数を実行しました¥n");
        printf("num1 = %d , num2 = %d¥n", num1, num2);
}
```

メモリ	アドレス
int *px	10004
int *py	10008
int temp	10012
int num1	10016
int num2	10020
num1 = 10	20000
num2 =7	20004
	20008
	20012
	20016
	20020
	20024

パソコンのメモリ上では、宣言と値は別々の場所に記録されている。

上 swap 関数にて、swap(int x, int y)と宣言するとメモリ上の宣言の部分をパソコンは取得しようとする。しかし実際に必要なのは値であるため、思わぬ挙動をする。

1Pにて、"変数 num のメモリ上の住所"と"ポインタ変数 pnum が指す値のメモリ上の住所" が同じ数値だった。この住所が実際に値が格納されている場所であるため、入れ替える値のポインタを宣言し、アドレスを渡す。

②配列・文字列

・ 配列

同じ型の多量の変数を一度に扱う方法。

int a[要素数] といった形で宣言する。

実際に変数を使うには

a[0]（１つ目の要素にアクセス）

といった形で使用する。

例：変数 5 個の配列を作った場合

int a[5] = { 0,1,2,3,4 };

変数名 a[0] a[1] a[2] a[3] a[4]

値 0 1 2 3 4

ただしこの場合 a[0] ～ a[4] の 5 個しか扱ってはいけない。

a[5]や a[6]以降は「住所として」存在はするが、これらの領域はＰＣの他の部分が使用している可能性があり、間違ってここに値を書き込んでしまうと重大なエラーにつながる可能性がある。

```c
#include <stdio.h>
int main(void)
{
	int arr[5]; //配列宣言
	int i;
	printf("値を５つ入力してください¥n");
	//配列はfor文で全要素にアクセスできる
	for(i = 0;i < 5;i++)
	{
		scanf("%d", &arr[i]);
	}
	for(i = 0;i < 5;i++)
	{
		printf("arr[%d]の値は%dです¥n", i, arr[i]);
	}
	return 0;
}
```

③文字列

Ｃ言語で言う文字列は char 型の配列を指す。

例： char a[6] = "Hello";

a[0] a[1] a[2] a[3] a[4] a[5]

H e l l o ¥0

このように配列の各 char 型変数に 1 文字ずつが保存されている。

最後の‘￥0’は文字列の終わりを表す文字（終端文字）で文字列には必ずこれがなくてはいけない。

（文字列の終わりがどこかわからなくなるため）

とはいえ、基本的には勝手に付加されるので気にしなくても問題はない。

```c
#include <stdio.h>
#include <string.h> //文字列操作系の関数がつかえるようになる
int main(void)
{
	char string[30] = "Hello World!"; //文字列≒char型の配列

	printf("%s¥n", string); //&string[0]とstringは同じ意味

	printf("文字列を入力してください¥n");
	scanf("%s", string); //scanfを使った文字列入力
	printf("文字列は¥n%s¥nです¥n", string);
	 //strlen(文字列)→文字列の長さを得る
	printf("文字列の長さは¥n%d¥nです¥n", strlen(string));
	return 0;
}
```

④関数と配列

配列の値を関数に渡す場合、
ポインタを使って配列の先頭の場所だけを関数側に教えます。
なぜかというと、配列は必ず連続に確保されるので、
（イメージとしては、配列を宣言した場合、その住所は最初の要素が101 号室なら次が 102 号室,103 号室・・・と連番になる）
最初の要素と要素の個数がわかれば、後は隣の住所を調べるだけですべての要素にアクセスできるため。

ここで重要なのは住所として存在してはいるが、配列の場所以外にはアクセスしてはいけないということ。

```
#include <stdio.h>

int sum_array(int* a , int num)
{
        int i;   //ループ用
        int value = 0;   //戻り値用
        for(i = 0; i < num ; i++ )
        {
                value += a[i];
        }
        return value;
}

int main(void)
{
        int i_array[5] = {1,10,100,1000,10000};
        int sum;

        sum = sum_array( i_array , 5 );
        printf( "配列の合計は%dです¥n" , sum );
        return 0;
}
```

住所	1000	1004	1008	1012	1016	1020	1024	1028	1032
値	a[0]	a[1]	a[2]	a[3]	a[4]	この辺はアドレスとして存在はするが、ほかの処理で使っているかもしれない			
	配列が使っている領域								

演習問題　（基本的に問題に書かれた動作をすれば正解とします。）

①配列にｎ個（数字は各自に任せる）数字を格納し、要素中の２の倍数をすべて表示せよ。

②入力された 5 つの数字を配列に格納し、それぞれについて大小比較し
昇順に並べ替えなさい。
できたら昇順に並べ替える関数を作ってやってみること。

例（10, 3, 7, 21,15） → （3, 7, 10, 15, 21）