ソフトゼミ∀　第6回

ポインタ

■ 補足

■ 配列はポインタ

4回で配列を学習しましたが、実は配列というのはポインタと等しいのです。
例えば char a[5];を宣言した時、それは a というポインタから始まる変数を連続で５つ宣言したことと等しいことになります。すると&a[0]と a は等しくなるため&a[1]=a+1 , &a[2]=a+2 のようになります。

わかりやすく表をまとめると以下のようになります。

ポインタ		アドレス
a	&a[0]	1000
a+1	&a[1]	1001
a+2	&a[2]	1002
a+3	&a[3]	1003
a+4	&a[4]	1004

a07_4.c

```
#include<stdio.h>
int main(void){
	int a[10];
	int i;
	for(i=0;i<10;i++){
		a[i]=i;
	}
	for(i=0;i<10;i++){
		printf("%d = %d¥n",&a[i],a+i);
	}
	printf("¥n");
	for(i=0;i<10;i++){
		printf("a[%d]:%d= *(a+%d):%d¥n",i,a[i],i,*(a+i));
	}
}
```

上の出力はアドレスを&a[i]とa+iで出力したものです。
下の出力は a[i]の値を配列 a[i]とポインタ*(a+i)から出力したものです。

■ **構造体へのポインタ**

通常の変数だけでなく、構造体に対してもポインタを使うことが出来ます。

a07_5.c

```
#include<stdio.h>
struct Test{
        int a;
        int b;
        int c;
};

int main(void){
        struct Test test;
        struct Test *p;
        test.a=10;
        test.b=20;
        test.c=50;
        p=&test;
        printf("test.a=%d,test.b=%d,test.c=%d¥n",(*p).a,(*p).b,(*p).c);
        printf("test.a=%d,test.b=%d,test.c=%d¥n",p->a,p->b,p->c);
        return (0);
}
```

この際、構造体のポインタ*pから構造体のメンバにアクセスするには
(*p).a という形をとらなければなりませんが、p->a のようにアロー演算子を使うことによりいちいち括弧で囲まなくても簡単に書くことが出来ます。

■ **線形リスト（ちょっとだけ）**

構造体の中で構造体を宣言したらどうなるのだろうか・・・？と考えたことはありませんか、それを実現したのが線形リストになります。
線形リストは連結リストの一種であり、今回は線形リストの中でも一番簡単な片方向リストを例として出してみました。

a07_6.c

```
#include <stdio.h>
struct A{
	int x;
	struct A *next;
};
int main(void){
	struct A a,b,c;
	struct A *p;
	p=&a;
	a.x=10;
	a.next=&b;
	b.x=20;
	b.next=&c;
	c.x=30;
	c.next=NULL;
	while(1){
		printf("%d¥n",p->x);
			if(p->next==NULL){
				break;
			}
			else{
				p=p->next;
			}
		}
	return 0;
}
```

初見だと大分わかりにくいとは思いますが、これが線形リストの基本になります。
これだけだとよくわからないので、上のプログラムを図にしてみるとこのようになります。

struct a		struct b		struct c	
int x	struct *next	int x	struct *next	int x	struct *next
10	&b	20	&c	30	NULL

上のプログラムではstruct Aのポインタ*pを宣言し、まずはstruct aを参照します。するとa.x=10なので10が出力されます。次にa.(*next)は&bですのでp=p->nextで次のstructであるbに移ることが出来ます。このようにあたかも配列のように使うことが出来ますが、決定的に違うのは動的に確保することが出来るということです。
今回のサンプルはあらかじめ線形リストを作った後に参照していますが、実はmallocという関数を使うことにより動的に次のポインタを確保することが出来ます。
これは何が便利かというと例えば配列a[100]と宣言した時に配列を10個しか使わなかった場合は残り90個分のメモリが無駄になってしまい、逆に110個使うことがあれば配列の予想外参照によりエラーになってしまいます。一方で連結リストは10個だとしても110個だとしても無駄なく対応することができます。

◆ 追加問題

以下の関数を作成せよ。main文を用意して動作確認をすること。

1. 2つの変数を受け取って、2つの変数の和と差を返す関数
   void wts(int *a , int *b);を作成せよ

2. 5の要素を持つ配列をそれぞれint型、char型、double型で作成し、それぞれのアドレスを求めよ。例えばint a[5];だったらa[0]～a[4]のアドレスをそれぞれ出力すればいい、またこの時、何故このような出力結果になるのか考察せよ。

◆ 応用問題 y=ー(゜д゜)・∵. ﾀｰﾝ A

y=ー(゜д゜)・∵. ﾀｰﾝAは練習問題、追加問題共に終わってしまい、余力のある人だけで構いません。

1. **int型配列aのa[0]～a[n]を降順に並び替える関数sort(int a[], int n); を作れ。**
   例えば元の配列がa={5,3,4,1,2};だったらsort(a,5)を読み出すと、a={1,2,3,4,5};になるように並び替えろ。
   また、並び替える際、配列をポインタの形に置き換えた時に、置き換える前と同じように動くことを確認せよ。
   できたら、sort(int *a, int n);**も作ってみよう！**