情報研究会ＣＡＣＴＵＳ

第六回　Ｃ言語講習

今回の内容

- 構造体
- ポインタ

サンプルプログラム

実行結果

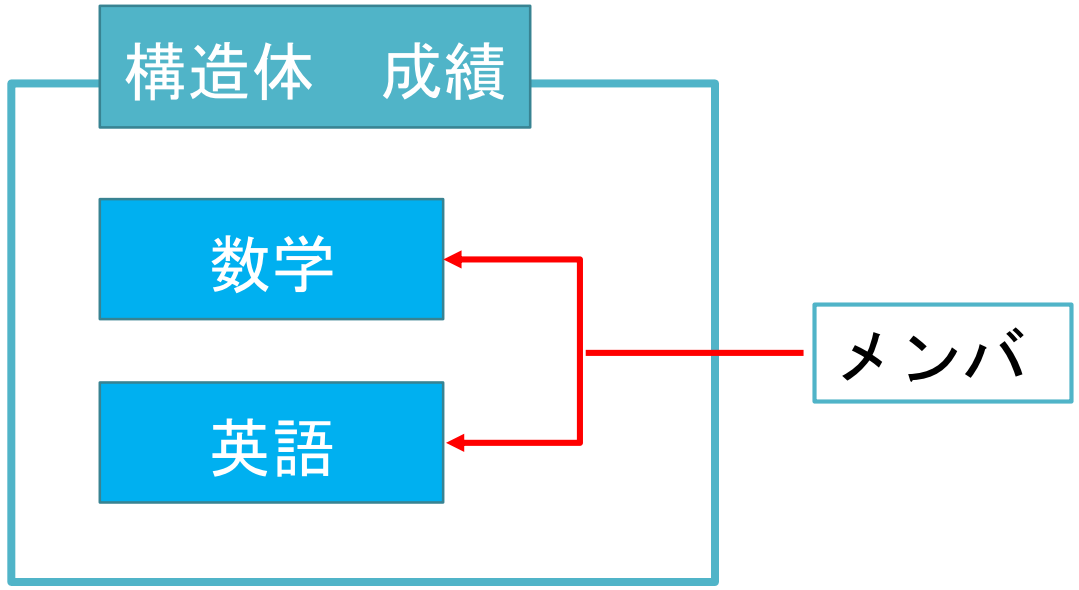
```
数学:90点 英語:60点
数学:80点 英語:70点
数学:80点 英語:70点
```

```c
#include <stdio.h>

struct grade
{
    int math;
    int english;
};

int main(void)
{
    struct grade student1 = {90, 60}; //初期化
    struct grade student2,someone;

    student2.math = 80;
    student2.english = 70;

    someone = student2; //メンバを一括代入

    printf("数学:%d点 英語:%d点\n", student1.math, student1.english);
    printf("数学:%d点 英語:%d点\n", student2.math, student2.english);
    printf("数学:%d点 英語:%d点\n", someone.math, someone.english);

    return 0;
}
```

構造体

・構造体とは複数の異なる型の変数が集まったデータ型である

・構造体に含まれる変数をメンバという



構造体の宣言

```
struct　タグ名
{
    データ型　メンバ名；
    データ型　メンバ名；
    …
}；
```

- struct タグ名が構造体のデータ型名となる
- プログラムの最初に宣言する
- { }の最後にセミコロンを忘れずに書く

構造体変数

- 構造体変数の宣言

```
struct grade student;
```

型名　　変数名

- メンバへのアクセス

```
構造体変数名.メンバ名
```

構造体変数名+ドット+メンバ名で構造体変数のメンバにアクセスすることができる

例）　構造体grade型変数studentのメンバ変数mathに値を代入

```
student.math = 100 ;
```

構造体変数の初期化

・配列の初期化と同様な方法で構造体変数も初期化できる

```
構造体型　変数名　＝　｛ 値１，値２，…｝
```

・構造体変数に構造体変数を代入するとメンバが一括代入される

確認問題１

構造体gradeに理科、平均点のメンバを追加し、以下の表の各生徒の平均点を求めてください

ただし平均点メンバを必ず使用し、平均点の有効数字は小数点第一位までとする

	数学	英語	理科	平均点
生徒1	98	52	75	
生徒2	66	74	81	
生徒3	57	68	83	

サンプルプログラム

実行結果

```
生徒1の平均点:95.0
生徒2の平均点:75.0
生徒3の平均点:55.0
生徒4の平均点:35.0
生徒5の平均点:15.0
```

```c
#include <stdio.h>
#define N 2

typedef struct
{
    int math;
    int english;
    double average;
}grade;

double average(grade); //プロトタイプ宣言

int main(void)
{
    int i;
    grade g[]={ {100,90},{80,70},{60,50},{40,30},{20,10} };

    for(i=0;i<5;i++){
        g[i].average = average(g[i]);
    }

    for(i=0;i<5;i++){
        printf("生徒%dの平均点:%.1f\n", i+1, g[i].average);
    }

    return 0;
}

double average(grade n)
{
    return (double)(n.math + n.english)/N;
}
```

typedef

- typedefを使用すると既存のデータ型名に新しい型名を加えることができる

- typedefを使用して構造体を宣言すると構造体変数の宣言でstructを省略することができる

```
typedef struct
{
    …

}タグ名;
```

構造体と配列

通常の変数と同様に構造体型の配列も宣言することができる
複数の異なるデータ型に対して同じ処理を繰り返したい場合に構造体型配列が役立つ

構造体と関数

関数の戻り値や引数に構造体型を用いることもできる

※先に構造体を宣言してからでないとその構造体を戻り値や引数とする関数を宣言することはできない

サンプルプログラム

実行結果

```
aの中身:1
aのアドレス:1638212
pの中身:1638212
aの中身:2
```

```c
#include <stdio.h>

int main(void)
{
    int *p; //ポインタ型変数の宣言
    int a = 1;

    printf("aの中身:%d\n", a);
    printf("aのアドレス:%u\n", &a);

    p = &a; //aのアドレスをpに代入
    printf("pの中身:%u\n", p);

    *p = 2; //アドレスが指す変数に2を代入

    printf("aの中身:%d\n", a);

    return 0;
}
```

変数のアドレス

メモリにはアドレス（番地）という住所のようなものが与えられている

変数を宣言するとデータ型のサイズに応じたメモリのアドレスが割り当てられる

番地	メモリ
1000	int a
1001	
1002	
1003	
1004	
1005	
1006	
1007	
1008	

アドレス演算子

変数に＆を付けるとその変数のアドレスを参照できる

＆をアドレス演算子という

```
＆a　→　1000
```

例）aのアドレスを出力

```
printf(“aのアドレスは%u”,&a);
```

scanfで入力された値を格納する変数に付けている
＆もアドレス演算子

番地	メモリ
1000	int a
1001	
1002	
1003	
1004	
1005	
1006	
1007	
1008	

ポインタ

ポインタとは変数のアドレスを記憶する変数である

ポインタの宣言は変数名の前に*をつける

```
型名　＊変数名　;
```

例）ポインタpに変数aのアドレスを代入

```
p = &a ;
```

間接演算子

ポインタに＊を付けるとそのポインタに保存されているアドレスが指す変数を参照することができる

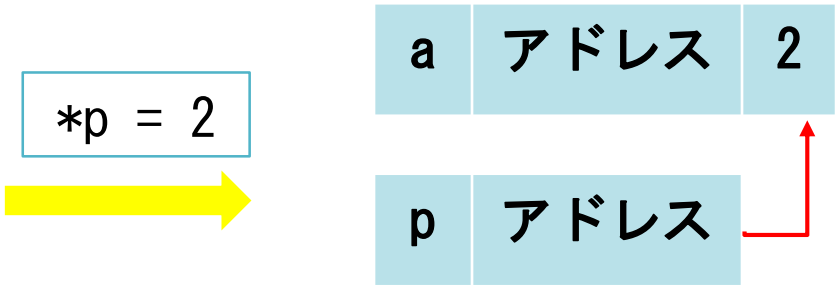
＊を間接演算子という

※ポインタを宣言するときの＊は間接演算子ではないことに注意

例）ポインタpに保存されているアドレスが指す
　　変数に２を代入

```
*p = 2 ;
```

サンプルプログラムの処理の流れ



1. ポインタpにaのアドレスを保存
2. pに保存されたアドレスが指す変数（すなわちaの中身）を参照
3. aの中身を書き換える

確認クイズ

問1　printf(“%d”,*p)が出力する数字は？

```
int a=0,b=1;
int *p;
p = &b;
b = a;
```

問2　printf(“%d%d”,a,b)が出力する数字は？

```
int a=0,b=1;
int *p;
p = &a;
b = *p;
a = b;
```

練習問題１

3次元空間における２点間の距離を求めるプログラム

ただし構造体と２点間の距離を求める関数を作成して使用すること

実行例

```
p(x,y,z): 1 2 3
q(x,y,z): 3 2 1
2点間の距離:2.828427
```

練習問題２

あなたは3種類のシリーズもの、各シリーズ1～5巻まで発売されている本を集めています
あなたが持っている本のリストと本屋に置いてある本のリストが与えられたとき、あなたが購入するべき本を出力してください
購入するべき本がない場合は「None」と出力してください

```
持っている本の数
シリーズ名（1～3）巻数（1～5）
…
売っている本の数
シリーズ名（1～3）巻数（1～5）
…
購入するべき本
…
```

```
持っている本の数:3
1 1
2 4
3 5
売っている本の数:4
2 4
1 5
3 5
3 5
購入するべき本
1 5
```

```
持っている本の数:2
2 5
1 4
売っている本の数:3
1 4
2 5
2 5
購入するべき本
None
```

実行例