

# C言語講座 2回目

関数、再帰関数、構造体

# 関数とは？

- 何するの？
- 数学関係あるの？
- $y=f(x)$  ？

- 実は今まで使ってきたmain も関数でprintfもscanfも関数である！
- また計算だけでなくいろいろな処理をするものである。
- printfもscanfも最初から定義されているものですが今回は自作関数という文字どうり自分で関数を作っていきます！

# どのように関数を作るのか？

```
型名 名前(引数)
{
    内容...
}
```

- 型名には関数で返す値(戻り値)の型を宣言する。  
(値を返さない場合はvoid)
- 引数にはmain関数から代入されてくる変数、型を宣言する。
- 内容では処理の内容と処理したあと値を返す場合 **return 戻り値** を書く。(値を返さない場合不要)

# 関数の例1

○二乗の値を返す関数

```
#include<stdio.h>
```

```
int nijo(int x){ ←
```

```
    int y = x*x;
```

```
    return y; ←
```

```
}
```

```
int main(){
```

```
    int a = 4;
```

```
    int b;
```

```
    b = nijo(a); ←
```

```
    printf(“%d*%d = %d¥n”,a,a,b);
```

```
    return 0;
```

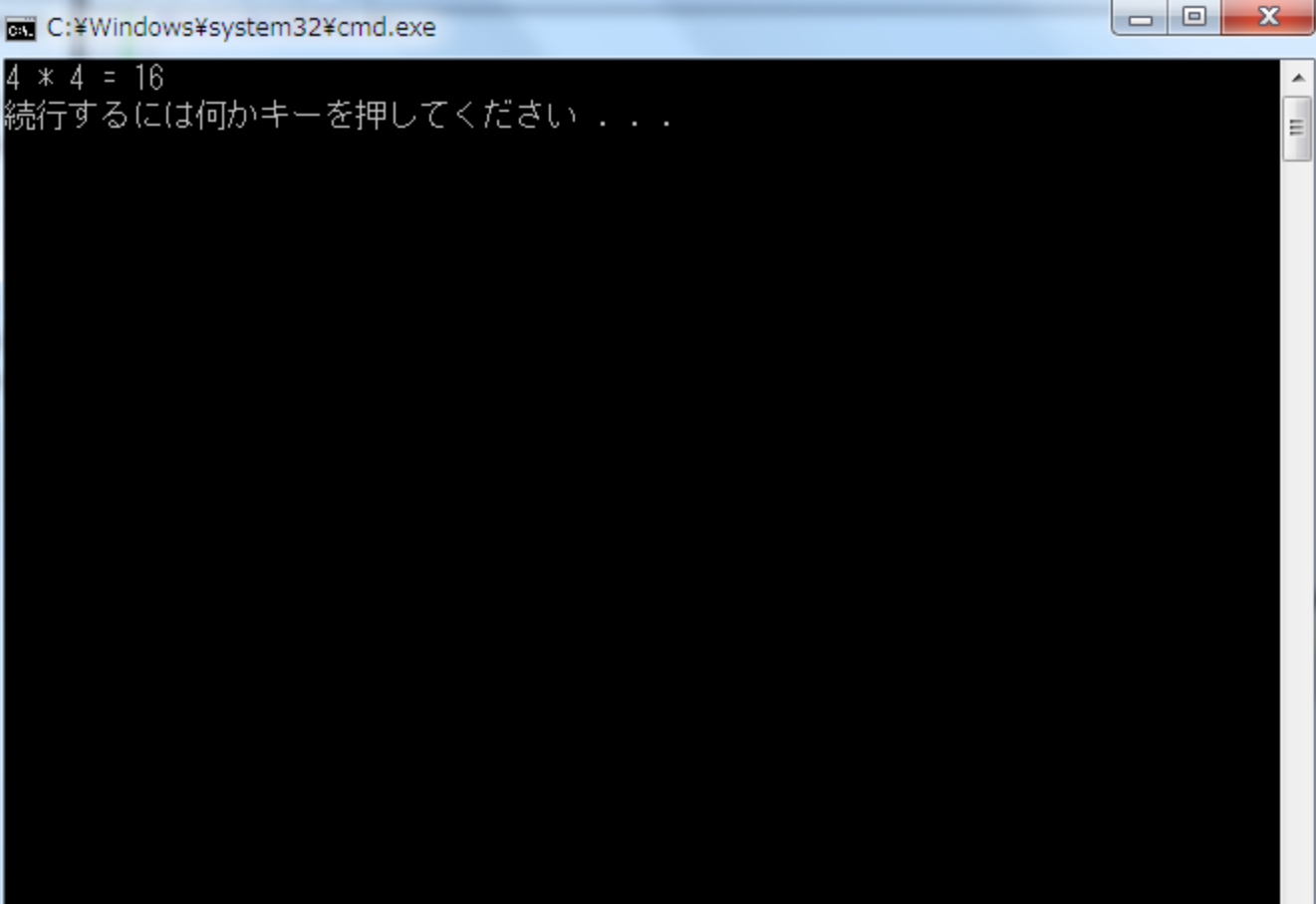
```
}
```

○関数はmain関数の前に定義する。

○int型のyを返すので関数の型名はint

○int型の関数なのでbに関数nijoの戻り値が代入される。

# 実行結果



```
C:\Windows\system32\cmd.exe
4 * 4 = 16
続行するには何かキーを押してください . . .
```

# 関数の例2

○二乗の値を出力する関数

```
#include<stdio.h>
```

```
void nijop(int x){  
    int y = x*x;  
    printf(“%d*%d = %d“,x,x,y)  
}
```

○値を返さず出力するだけなのでvoid型の関数

```
int main(){
```

```
    int a = 4;  
    nijop(a);  
    return 0;
```

○4が代入されて出力されるだけ

結果はさっきと同じになります。

# 問題

- 2つの値の積を返すint型の関数を作ってみよう！  
(代入する値は $a = 5$ ,  $b = 7$ とする。)

※複数の引数を扱う場合は引数を  
`Kansu(int x, int y, ... ,int z)`  
と書く。



- 関数の宣言にはもう1つ書き方がある。
- **それがプロトタイプ宣言！**
- 新しく出てきた言葉だけどいたって簡単で今までmain関数の前に関数を全部書いてきましたが、main関数の前に関数名だけを宣言して、最後に関数の内容を書く。

# 関数の例（プロトタイプ宣言）

```
#include<stdio.h>
```

```
int nijo(int x){
```

```
    int y = x*x;  
    return y;  
}
```

```
int main(){
```

```
    int a = 4;  
    int b;  
    b = nijo(a);  
    printf("b = %d\n",b);  
    return 0;
```

```
}
```

```
#include<stdio.h>
```

```
int nijo(int x); プロトタイプ宣言  
                セミコロンをつける
```

```
int main(){
```

```
    int a = 4;  
    int b;  
    b = nijo(a);  
    printf("b = %d\n",b);  
    return 0;
```

```
}
```

```
int nijo(int x){
```

```
    int y = x*x;  
    return y;
```

```
}
```



# 再帰関数

- 関数内で自分自身を呼び出す関数を再帰関数という。

※無限ループしないように終了する条件を忘れずに設定する。

- 具体的にどう作るのか？

# 再帰関数の例

代入された自然数から-1して  
0になるまで出力する関数

```
#include <stdio.h>
```

```
void countd(int);
```

```
int main(){
```

```
    int x;  
    printf("自然数：");  
    scanf("%d",&x);
```

```
    countd(x);  
    return 0;
```

```
}
```

```
void countd(int n){
```

```
    if(n >= 0){
```

```
        printf("%d¥n",n);
```

```
        countd(n-1);
```

```
    }
```

```
}
```

← ○関数の終了条件

← ○ここで再び関数が呼び出され  
代入したnの値が1引かれてい  
く

# 問題

- 再帰関数を使って入力した値で（例として5のとき）

\* \* \* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

が表示される関数を作ってみよう！

ヒント：再帰関数の例の関数に書き加える。

話は変わって

# 構造体

- 構造体とは複数の変数を1つのまとまりにしたもの。
- 配列と違って同じ型でデータをまとめるのではなく違った型のデータをまとめられる。
- 例えば人のデータとして「名前、年齢、身長、体重」を作れる。
  
- 作り方は次のページで

# どのように構造体を作るのか？

```
struct 構造体の名前{
```

```
    メンバー変数(作りたいデータの中身)
```

```
};
```

セミコロンを忘れずに

- mainの外で定義する。
- 構造体の名前はなんでもよい。(ただし最初は大文字)
- 構造体の名前は型名として扱う。



# 構造体の例

人のデータ{

- 名前
- 年齢
- 身長
- 体重

}

を作りたいとき



```
struct Person{  
    char name[20];  
    int age;  
    int height;  
    int weight;  
};
```

**Person**がこの構造体の名前となる。

name[20]やageはメンバと呼ばれる要素である。（mainで宣言する変数とは違う）

構造体はどのように使うのか？

```
struct Person{
    char name[20];
    int age;
    int height;
    int weight;
};
void main{
    Person yamada;
    yamada.name[20] = 0;
    yamada.age = 20;
    yamada.height = 165;
    yamada.weight = 56;
}
```

Person型の構造体yamadaを作る。

構造体のそれぞれの要素にアクセスするにはドット演算子を使って

**構造体名.メンバ**

と書く。

これはint aなどの変数と同じように扱うことができる。

文字列の直接初期化はポインタを用いるので省略。

このように構造体を作ることによって値の違ったデータをたくさん作ることができる。

# 構造体の初期化,配列,代入

```
void main{
```

```
    Person suzuki ={"鈴木",18,150,49};
```

○さきほどはメンバごとに設定していたが配列とおなじように初期化できる。

```
    Person famiry[3]={  
        {"父",50,170,66},  
        {"母",47,140,47},  
        {"弟",15,160,50};}
```

○構造体も変数とおなじように配列を作ることができる（初期化も）

```
    Person miyazaki = famiry[0];
```

○同じ構造体でなら中身をそのまま代入できる。

```
}
```

# 問題

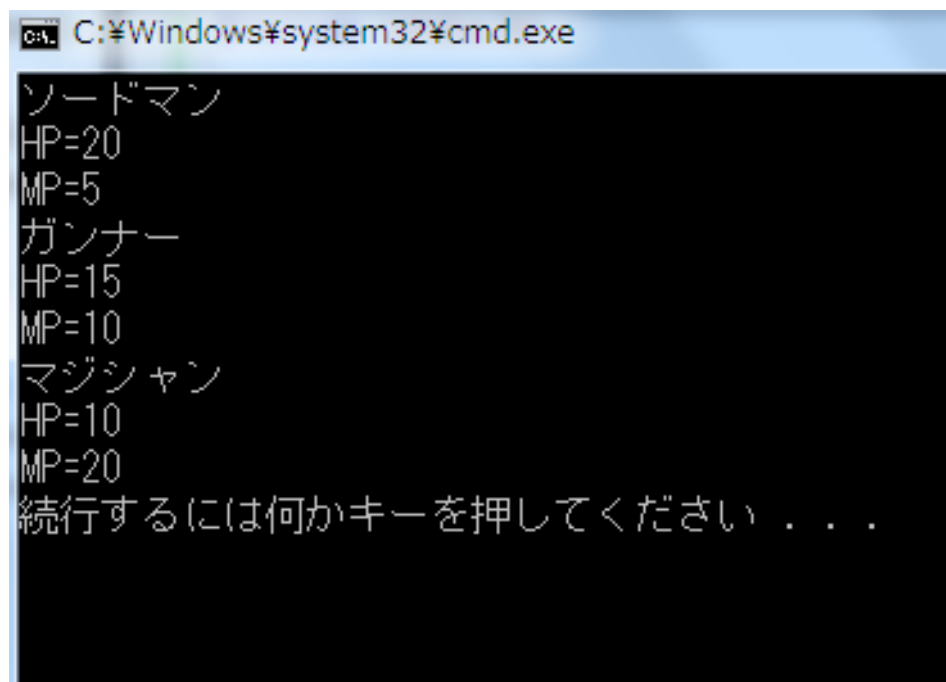
char name[20]

int HP

int MP

をメンバに持つ構造体  
Jobを作って右の図のよ  
うに印字するプログラ  
ムを書いてみよう。

※文字列を出力すると  
きは%sを使う



```
C:\Windows\system32\cmd.exe
ソードマン
HP=20
MP=5
ガンナー
HP=15
MP=10
マジシャン
HP=10
MP=20
続行するには何かキーを押してください...
```