C言語講座
第参回

関数、再帰関数

関数について（1）

皆さん関数というと数学の y=f(x) が思い浮かぶのではないだろうか？

数学ではさらに　F(x)=2x+1　のようになっていたはず・・・

この　2x+1　というのが関数の中身だということ。

関数について（２）

さて、Ｃ言語でいう関数というと皆さんもう既に使っていて、main関数、printfなども関数と呼ばれるものである。

今までのプログラムはすべてmain関数の中に書かれていた。

今回の講座では「自分で作成する関数」についてメインに取り扱っていく。

自作関数のつくりかた（１）

大体書き方は決まっていて下のような形になる。

といっても、いきなり返却値型だか引き数だといわれてもなにがナンダカ・・・

```
返却値型　関数名（引き数宣言）
{
        関数の中身（プログラムをかく）
        return 戻り値;
}
```

自作関数のつくりかた(2)

```
返却値型　関数名(引き数宣言)
{
        関数の中身(プログラムをかく)
}
```

返却値型(戻り値型):関数内で行った処理をmain関数に返す時の値の型をここで指定する。

引き数～:逆にmain関数から呼び出すときに持ってくる値を入れる変数を宣言するところ。

元のプログラム

```
#include<stdio.h>

int main(){
        int x,y;
        printf("f(x) = 2x+1¥n");
        printf("x = ");
        scanf("%d",&x);

        y = 2*x+1;

        printf("f(x) = %d¥n",y);
}
```

関数を使ったプログラム

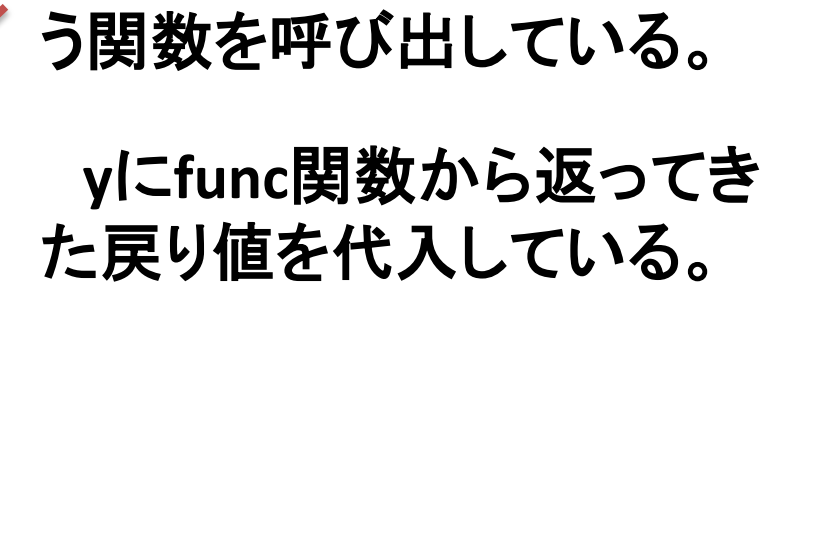
```
#include<stdio.h>
int func(int);
int main(){
        int x,y;
        printf("f(x) = 2x+1¥n");
        printf("x = ");
        scanf("%d",&x);

        y = func(x);
        printf("f(x) = %d¥n",y);
        return 0;
}
int func(int a){
        int b;
        b=2*a+1;
        return b;
}
```

○解説

```
#include<stdio.h>
int func(int);
int main(){
   int x,y;
   printf("f(x) = 2x+1¥n");
   printf("x = ");
   scanf("%d",&x);

   y = func(x);
   printf("f(x) = %d",y);
   return 0;
}
int func(int a){
   int b;
   b=2*a+1;
   return b;
}
```



ここで、下にあるfunc(x)という関数を呼び出している。

yにfunc関数から返ってきた戻り値を代入している。

○解説

```
#include<stdio.h>
int func(int);
int main(){
   int x,y;
   printf("f(x) = 2x+1¥n");
   printf("x = ");
   scanf("%d",&x);

   y = func(x);
   printf("f(x) = %d",y);
   return 0;
}
int func(int a){
   int b;
   b=2*a+1;
   return b;
}
```

ここで、下にあるfunc(x)という関数を呼び出している。

yにfunc関数から返ってきた戻り値を代入している。

自作関数はこの部分です。

この部分は自分で好きな処理を行うようにしよい。

今回は引き数で持ってきた値に2掛けて1足す作業をしてその答えを戻り値として返している。

○解説

```
#include<stdio.h>
int func(int);
int main(){
   int x,y;
   printf("f(x) = 2x+1¥n");
   printf("x = ");
   scanf("%d",&x);

   y = func(x);
   printf("f(x) = %d",y);
   return 0;
}
int func(int a){
   int b;
   b=2*a+1;
   return b;
}
```

自作関数の書き方

戻り値の変数型　関数名（引数型と名）

・戻り値の変数型

関数内の処理が終わったら、その結果を返す必要がある。これを戻り値と呼ぶ。

戻り値は様々な型で宣言できる。

今回は整数の計算結果のbのこと。

結果はint型なので、宣言ではintと書く。

もし小数を返すならdouble

もし何も返さないならvoid　　とかく。

・関数名・・・これは自由何でもよい（例えばitoko()とか…）

・引数型と引数名

main関数から持ってきた値（ここではx）を引数という。

ここでの型は引数と同じにし、それにも他に名前を付ける。（ここではa）

○解説

```
#include<stdio.h>
int func(int);
int main(){
   int x,y;
   printf("f(x) = 2x+1¥n");
   printf("x = ");
   scanf("%d",&x);

   y = func(x);
   printf("f(x) = %d",y);
   return 0;
}
int func(int a){
   int b;
   b=2*a+1;
   return b;
}
```

これをプロトタイプ宣言という。

プロトタイプ宣言はコンパイラにこういう関数を使うという宣言。

書き方は

戻り値の型　関数名（引数の型）;

意味はそんなに考えないでこういうもんだと知っておくといい。

ー ー ー ー ー ー ー ー ー ー ー ー ー ー

returnの説明

return は戻り値を返す指示。

ここでは、bの値をmainのfunction(x)に返すという指示。

void型の関数なら、returnは必要ない。

再帰関数

関数内で自分自身の関数を呼び出すことを再帰関数をいう。

注意事項

再帰関数は終了する条件を設定しておかないと永遠に自分を呼び続け、大変なことになりますよ。

再帰関数の例

```
#include <stdio.h>
void raretu(int);

void main(void){
        int a;
        printf(“整数を入力してください: ");
        scanf("%d",&a);

        raretu(a);
}
void raretu(int x){
        if(x < 50){
                printf("%d¥n",x);
                raretu(x+1);
        }

}
```

再帰状態を抜ける条件
　ここでは、持ってきた引数が50未満のとき、引数に1を加えて再び自分自身を呼び込む。

50以上であれば関数を抜ける

・演習問題

整数nの入力を促し、入力されたnの階乗を計算し表示する関数を作成せよ。

階乗とは、

n！ = n(n-1)(n-2)×・・・×2×1

のように1からnまでの自然数の総乗のこと。

終わったら再帰関数を使って作ってみる。

全く違った感じになる。

・参考

【階乗の定義】

n= 0 ならば 0! = 1

n≠0 ならば n! = n * (n – 1)!

例えば、n=3ならば

3! = 3 * (3 – 1)! = 3 * 2!

2! = 2 * (2 – 1)! = 2 * 1!

1! = 1 * (1 – 1)! = 1 * 0!

0! = 1

解答

```
#include <stdio.h>
int factorial(int n);
int main(){
	int n,result;

	printf(“整数を入力してください:");
	scanf("%d",&n);
	result =factorial(n);
	printf(“%dの階乗は%dです。¥n“n,result);
}
int factorial(int n){
	int i,result=1;
	for(i=n; i>0;i--){
	result *= i;
	}
	return result;
}
```

再帰関数を使ったnの階乗を求めるプログラム

```
#include <stdio.h>
int factorial(int n);

int main() {
        int n, result;
        printf(“整数nを入力してください:”);
        scanf(“%d”,&n);
        result=factorial(n);
        printf(“%dの階乗は%dです¥n”,n,result);
        return 0;
}
int factorial(int n) {
        if (n==0) {
           return 1;
        }
        else {
           return n*factorial(n-1);
        }
}
```