目 次

第1章　第8回

1.1　第8回放送内容

1. 引数としてのポインタ
2. 構造体

1.2　引数としてのポインタ

前回の放送ではポインタの基本的な使い方を勉強しました。しかし、何のためにポインタが存在しているのかが良くわからなかったと思います。今回は、ポインタはなぜ存在するのか、またどのようにして使ったらよいかを勉強していきたいと思います。

では、次のような関数を作ったとします。

```
void swap( int x, int y )
{
int tmp = x;
        x = y;
        y = tmp;
}
```

このコードでは、二つの変数を引数として受け取り、その中身を交換する関数を作った「つもり」です。次のようなコードを書き、実行させてみましょう。

```
#include <stdio.h>
int main(void)
{
int hoge = 10; piyo = 5;
        swap( hoge , piyo );
        printf("hoge : %d, piyo : %d\n",hoge,piyo);
```

```
        return 0;
}
```

実行結果

```
hoge : 10, piyo : 5
```

このswap関数ではうまく値を交換できていません。なぜでしょうか？

ここで、関数の引数について思い出しましょう。第5回放送で、仮引数には実引数のコピーが渡されると説明しました。仮引数とは、この場合swap関数の括弧の中のことです。実引数とは、呼び出した側の括弧の中のことです。つまり、void swap(ここが仮引数) であり、main関数の中のswap(ここが実引数) となっています。忘れている人は復習しておきましょう。これらのコードの実行結果を図で表すと、図??となります。

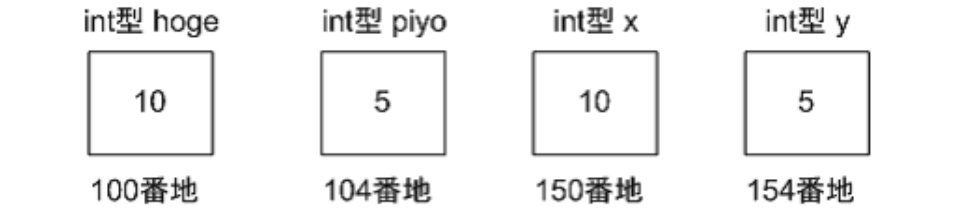

図 1.1: メモリ内部の様子

よって、仮引数であるxとyをいくら書き換えても元の変数hogeとpiyoにはなんら影響を及ぼしません。しかしそれでは困ります。

そこで登場するのがポインタです。ポインタを使ってもとの変数にアクセスすることで中身を書き換えることができます。次のようなコードを書いてみます。

```
#include <stdio.h>

void swap(int* x, int* y); //プロトタイプ宣言

int main(void)
{
int hoge = 10, piyo = 5;
```

```
        swap( &hoge, &piyo );
        printf("hoge : %d, piyo : %d\n",hoge,piyo);
        return 0;
}

void swap(int* x, int* y)
{
int tmp = *x;
        *x = *y;
        *y = tmp;
}
```

このコードの様子を図で表したものが、図??です。

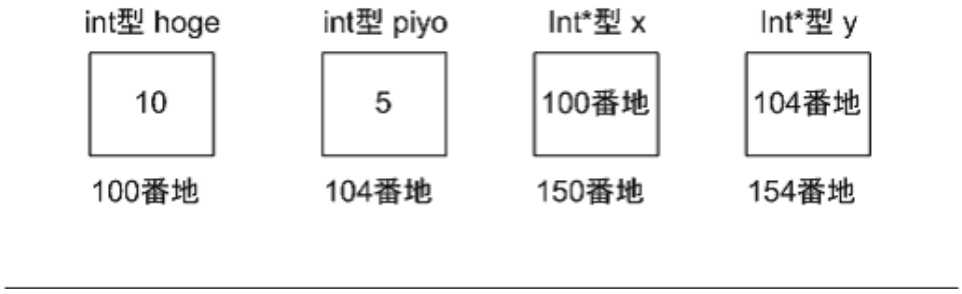

図 1.2: メモリ内部の様子

ここで、*x は hoge そのものであり、*y も piyo そのものです。よってこれらの値を入れ替えることで値を交換できることになります。よって実行結果は次のようになります。

実行結果

hoge : 5, piyo : 10

1.3 構造体

1.3.1 構造体

構造体とは、複数の変数をまとめて使いたい時に使います。使い方は次のようになります。

書式

```
struct 構造体の名前 {
構造体の中身
};
```

では、構造体を使ったプログラムの例を見てみましょう。

```
#define NUMBER_OF_STUDENT 3

struct Student{
char student_name[100];
int student_age;
};

int main(void)
{
struct Student st[NUMBER_OF_STUDENT];
int i;
int num;
printf("%d 人の生徒の情報を入力してください\n",NUMBER_OF_STUDENT);

for( i=0; i < NUMBER_OF_STUDENT ; i++ ) {
    printf("%d 人目の名前 :",i+1); scanf("%s",st[i].student_name);
        printf("%d人目の年齢 :",i+1); scanf("%d",&st[i].student_age);
}
printf("何人目の生徒の情報が見たいですか :");
scanf("%d",&num);
printf("名前 : %s, 年齢 : %d",st[num-1].student_name, st[num-1].student_age);
return 0;
}
```

まず、 `struct Student st[NUMBER_OF_STUDENT];` という部分で、Student という構造体の変数を NUMBER_OF_STUDENT 個だけ作っています。そして、構造体の中にある変数にアクセスするときには、 `st[0].student_name` といったようにします。

ここで、構造体とはあまり関係がないですが、scanf 関数のところに注目してみましょう。はじめに出てきた scanf の部分はまだちょっと難しいので、説明ができません。ですが、次に出てくる `scanf("%d",&st[i].student_age);` という部分の意味がポインタを勉強することによって、ようやく理解が出来るようになります。ここでは、scanf 関数の引数に構造体の中の `student_age` という int 型の変数のアドレスを渡しています。なぜアドレスを渡すかというと、前節で変数の中の値を書き換えたい場合は、アドレスを渡さないとできないということを説明しました。

1.3.2 構造体へのポインタ

int 型のポインタなどと同様に、構造体のポインタも作成することができます。int 型のポインタを作成したい場合、 `int* hoge;` などと宣言しました。ここで、仮に

```
struct Student{
char student_name[100];
int student_age;
};
```

という構造体を作ったとすると、Student 構造体のポインタを作成するには、

```
struct Student* hoge;
```

といったように宣言します。

ここで注意しなければならないのは、hoge の中にある要素にアクセスしたい場合です。そういった場合、 `hoge->student_age` といった用にアクセスします。`->` はアロー演算子と呼ばれ、 - (マイナス）と > （大なり）を合わせて作ったものです。なぜこのようなものが使われるのでしょうか。

例えば次のようにしてみましょう。

```
(*hoge).student_age
```

これは、問題ありません。hoge が指している先のデータ（構造体）の中の `student_age` というデータを取り出せ、という意味です。ちょっとややこしいです。では、次のようにしたらどうなるでしょうか。 `*hoge.student_age`

多くの人はこれを、
`(*hoge).student_age`
と勘違いしてしまいますが、実はこれは、
`*(hoge.student_age)`
となってしまい、意味がぜんぜん違うものになってしまいます。この混乱を解消するために、アロー演算子というものが開発されました。

参考文献

[1] Bjarne Stroustrup 著 長尾高弘 訳: プログラミング言語 C++第三版.

[2] Brian W. Kernighan, Dennis M. Ritchie: The C Programming Language Second Edition.

[3] Peter van der Linden, Expert C Programming Deep C Secrets.

[4] ハーバートシルト著 トップスタジオ訳: 独習 C 第三版.

[5] 前橋 和弥: ポインタ完全制覇.

[6] 浅井 淳: ポインタが理解できない理由.