C言語講座

第3回

第3回の内容♪♪♪

- ポインタ
- 構造体
- ファイル分割
- ライブラリ関数

今回の「ポインタ」について

「ポインタ」はC言語の中でとても重要な機能です。

しかし、C言語を始めたばかりの人の大半がこの「ポインタ」

が理解できずにつまずきます。（そしてそのままにします）

なので、今回は無理に理解することなく、わからないことがあったら、気軽に聞いてください。

アドレス

今までの講座で使ってきた「int～」や「char～」のような
変数は全てメモリ上に一時的に記憶（保存）されている。

```
int    a;
char  b;
```

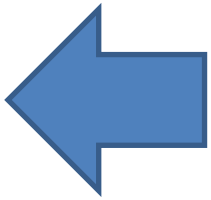
←いままでこんな感じで書いてるよね？

その保存されている場所（番地）をアドレスと言います。

アドレスを表示しよう！！

- アドレスを使うためには変数に「＆」をつけます。

```
#include<stdio.h>
void main(){
int a;
printf("aのアドレスは%p¥n",&a);
}
```



このように
つけます。

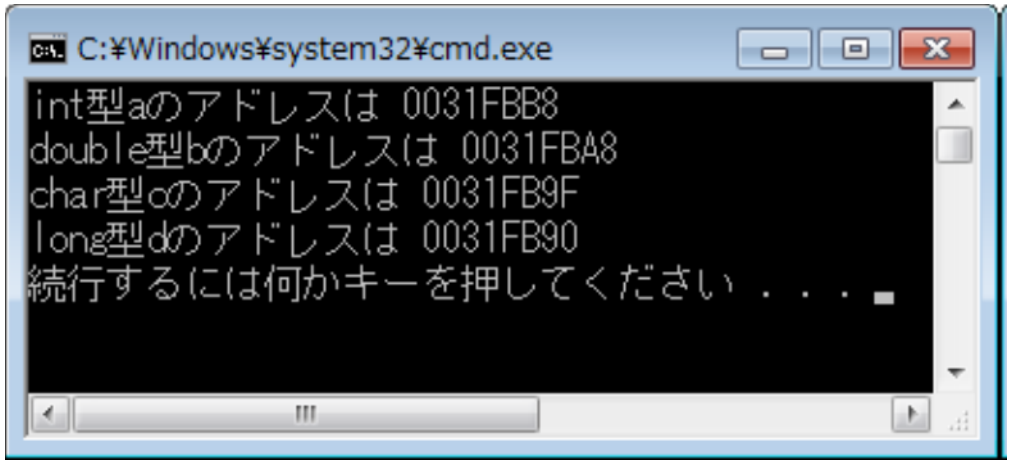
※%pはアドレスを表示するのに使います。

例文

```
#include<stdio.h>
int main(void){
int a;
double b;
char c;
long d;
printf("int型aのアドレスは %p¥n", &a);
printf("double型bのアドレスは %p¥n", &b);
printf("char型cのアドレスは %p¥n", &c);
printf("long型dのアドレスは %p¥n", &d);
return 0;
}
```

・上記の例文のようにいろんな型の変数をいくつか書いてみます。
・どんな型の変数も 変数名の前に「&」を付ければアドレスを示すことがわかります。

実行結果の解説



- 「～型～のアドレスは・・・」とコンソールに出力されます。
- この「・・・」が、その変数のメモリ上に保存されている場所、つまり、「アドレス」のことです。
- このアドレスは環境によって違うので、同じ必要はないです。

scanf関数

- 今までユーザーから値を入力する際に使ってきたscanf関数ですが、以下のように記述してきました。

scanf(“%d”,&a);

- この二つ目の引数「&a」、第一回でとりあえず変数に「&」をつけるように説明されてました。
- scanf関数は「アドレスが示すメモリを入力された値に書き換える」といったことをします。

ポインタ

ポインタとは？・・・変数のアドレスを入れる変数のことです。

ある変数が、メモリ上のどこに保存されているかを示すアドレスを保存しておけます。

ポインタの宣言

ポインタも変数なので、はじめに宣言します。

※以下のように宣言します。

int a; ⇒ int *p;

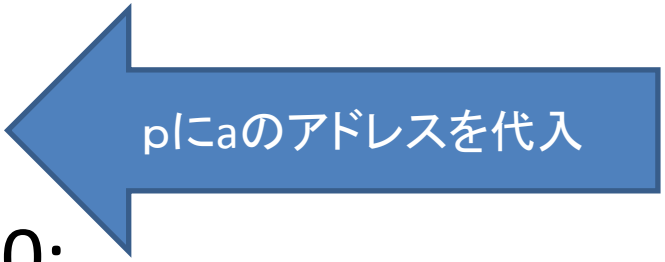
この様に、変数の前に「*」をつけます。

型は入れたいアドレスの変数の型です。

今回の例ではint型の変数aのアドレスを入れたいので、ポインタの型はint型にします。

ポインタの使い方

```
int main(void){
        int a;
        int *p;
          p=&a;
          return 0;
}
```



- ポインタを宣言したら、まずアドレスを入れます。
- アドレスを使う時は変数の前に「&」をつけます。

補足説明

- 最初にポインタを使うには「int *p」と書くと説明しましたが、別に「*p」という変数ではないです。
- .正確にいうと「int p」という変数に「*」をつけることによって他の変数のアドレスを変数「p」に入れて使用できるようになるといった考え方をしてください。
- .「p」という変数に「a」のアドレスを代入したいので「p=&a」となるわけです。（*p=&aという書き方はできない。）
- .ただし、宣言するときに初期化する場合のみ、以下のように書くことができます。

```
int *p=&a;
```

例文

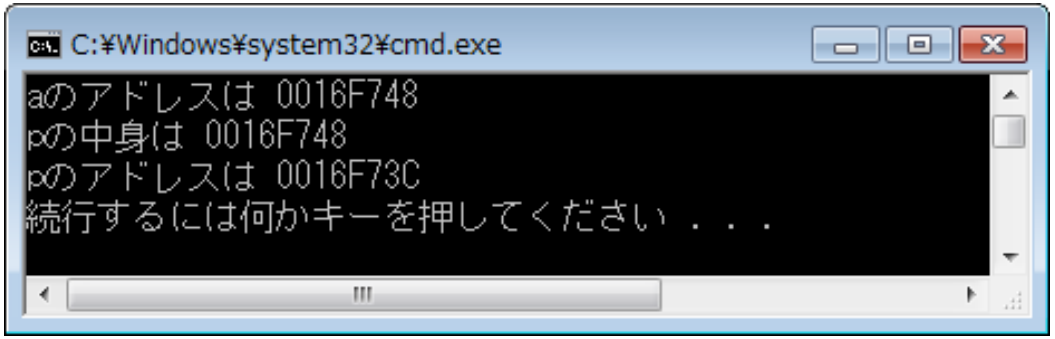
```
#include<stdio.h>

int main(void){
        int a;
        int *p;

        p=&a;

        printf("aのアドレスは %p¥n", &a);
        printf("pの中身は %p¥n", p);
        printf("pのアドレスは %p¥n", &p);
        return 0;
}
```

実行結果



- aのアドレスとpの中身が同じになります。

- pも変数ですので、メモリに保存されています。よって、しっかりとアドレスがあります。（aとpは違う変数なのでアドレスは違います。）

*の使い方

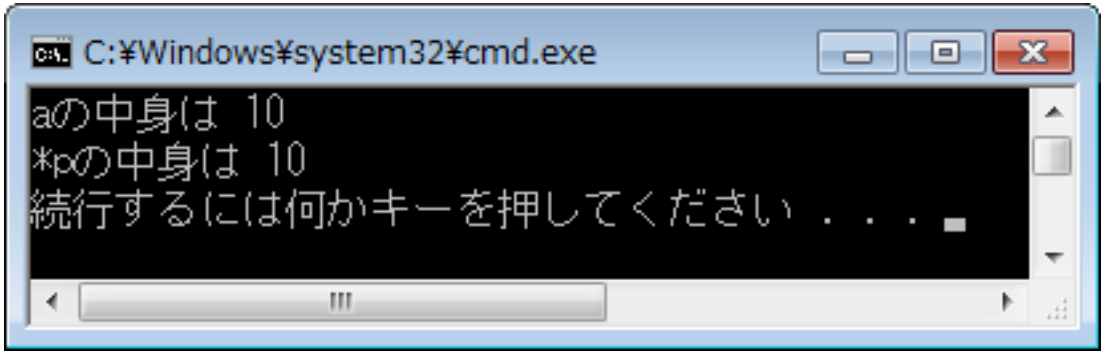
- ポインタに「*」のつけた時、*のついている変数に入っているアドレスの中身を参照します。

```
#include<stdio.h>

int main(void){
        int a=10;
        int *p=&a;
        printf("aの中身は %d¥n", a);
        printf("*pの中身は %d¥n", *p);
        return 0;
}
```

始めにaに10を代入して、そのあとにpにaのアドレスを入れています。

*の仕組み



- pにaのアドレスを入れると、*pの値がaと同じになりました。

- *のついたポインタにアクセスすると、*のついている変数（ここではp）に入っているアドレス（ここでは&a）の中身（ここではa）を参照します。

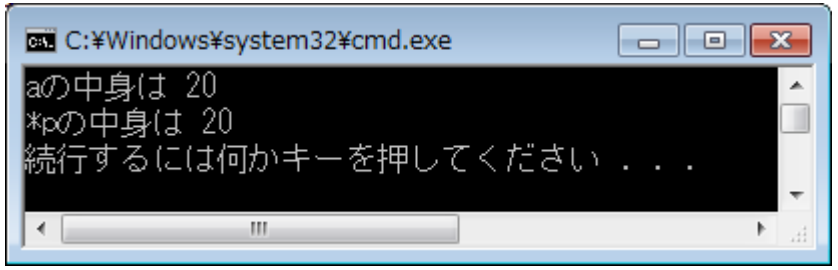
下記のようにpにaのアドレスを入れた
後に「a」の値を変えてみます。

```
#include<stdio.h>

int main(void){
        int a=10;
        int *p=&a;

        a=20;

        printf("aの中身は %d¥n", a);
        printf("*pの中身は %d¥n", *p);
         return 0;
}
```



*pの値は変更していませんが、
*pは、pに入っているアドレスの中身を参照するので、aの値を変えると、*pの値もaと同じになります。

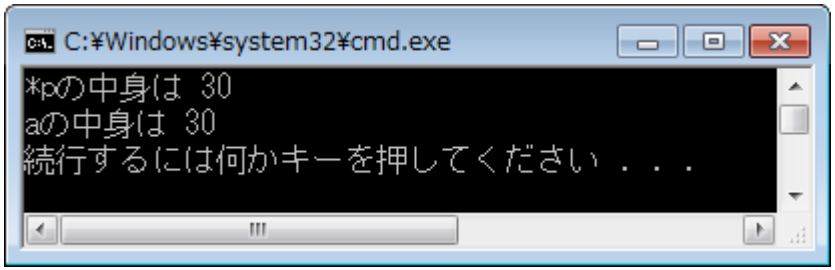
では今度は逆にpにaのアドレスを入れたあとに、「*p」の値を変えてみます。（「a」の値は変更しません。）

```
#include<stdio.h>

int main(void){
int a=10;
 int *p=&a;

*p=30;

printf("*pの中身は %d¥n", *p);
 printf("aの中身は %d¥n", a);
 return 0;
 }
```



*pは、pに入っているアドレスの中身を参照するので、*pの値を変えことで、aの値に変更することもできます。

- *のついたポインタにアクセスすると、その都度そのポインタに入っているアドレス先を参照します。

- つまり、「 *のついたポインタ」と「そのポインタに入っているアドレス先」は常に連動し、同じ値を示します。

- この機能こそが、ポインタがC言語の中で重要な機能であるということの理由です。

注意事項（ゼッタイ、ダメ）

- アドレスを渡さずにポインタを使用するのは 絶対にしないでください。
- ポインタはアドレスを得てその中身を間接的に書き換えたり参照する物です。
- ポインタにアドレスを渡さないと、そのポインタの中には宣言した際の適当な値が入っています。
- その状態で「*p」を使うと「p」に入っているどこのものかもわからないアドレスを参照してしまいます。
- もし、その参照したアドレスが他のシステムで使っている場所なら、そこを書き換えたりするとシステムがクラッシュする可能性があります。（大事な課題,OS…etc.）

ポインタを使用する？

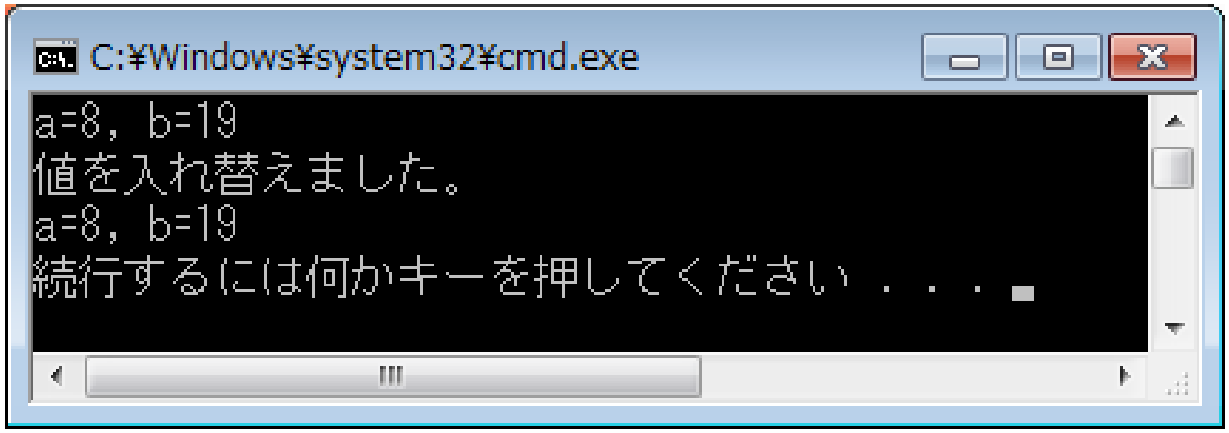
- 今までポインタの仕様について説明しましたが、正直、「aの値を変えたいなら、わざわざ*pなんて使わないで、直接aに値を代入すればいいのでは？」と思う人もいるかもしれません。
- 確かにこれまでの例のような使用方法は、実際にはほとんど行いません。（上記の通り、直接値をいじればいいのですから。）
- では、なぜポインタというものがC言語においてとても重要なのでしょうか。

例文

```
#include<stdio.h>
void swap(int x, int y){
        int tmp;
        tmp=x;
        x=y;
        y=tmp;
        printf("値を入れ替えました。¥n");
}
int main(void){
        int a=8, b=19;
        printf("a=%d, b=%d¥n", a, b);
        swap(a, b);
        printf("a=%d, b=%d¥n", a, b);
        return 0;
}
```

- このswap関数は、自作関数で、二つの引数を渡して、それらを入れ替えるという関数です。
- 一見ただしそうに見える。

実行結果



- 実行結果を見て、入れ替えました。なんて表示されていますが入れ替わってませんね。

どうしてこうなったwww

理由は・・・swap関数の引数「int x,int y」に「a」と「b」の数値を入れていますが

この「x」と「y」はそれぞれ「a」と「b」のコピーになっています。

コピーということになり「int x」と「int y」は別のアドレスを持っており

この2つの中身をいくら変えても最終的に表示するのは「a,b」なので

最終的にみても値が交換されていないというわけです。

使用例（改訂版）

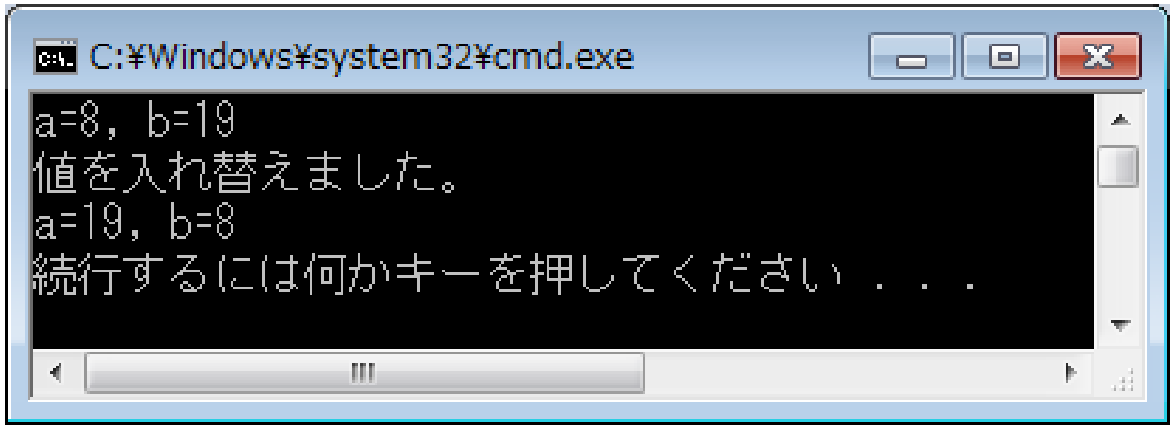
```
#include<stdio.h>
void swap(int *x, int *y){
int tmp;
tmp=*x;
*x=*y;
*y=tmp;
printf("値を入れ替えました。¥n");
}
int main(void){
int a=8, b=19;
printf("a=%d, b=%d¥n", a, b);
swap(&a, &b);
printf("a=%d, b=%d¥n", a, b);
return 0;
}
```

swap関数の引数を int*型に変えて、swap関数を呼び出す際に、それぞれの変数のアドレスを渡してあげるようにします。

訂正版の説明

- 今回は、main関数から swap関数が呼び出されたときに、swap関数にint *x, int *yというポインタが宣言され、渡されたアドレスを代入します。
- x,yにはa,bのアドレスが入っているので、*x,*yはメモリ上のa,bを参照します。よって、*x,*yを変更すればa,bも変更されます。

実行結果



- 実行結果もしっかりと入れ替わるようになりました。

他のポインタの機能

- 配列のポインタ
- ポインタの配列
- ポインタに対するポインタ
（ダブルポインタ）
- 関数ポインタ
（関数を参照するポインタ）

構造体

構造体とは・・・複数の変数を1つにまとめたものです。

配列とは違い、含まれる要素（構造体の場合はメンバ、フィールドなどと呼びます）の型は異なっても構いません。

```
typedef struct {
型 メンバ名;
int HP;
int MP;
char name[16];
:
} Status(構造体の型の名前);
```

宣言の仕方

```
構造体の型の名前 構造体の名前;
Ex)
typedef struct {
int HP;
int MP;
char name[16];
} Status;
void main(){
Status ch;
ch.HP=100;
ch.MP=50;
ch->HP=150;
ch.name[16]=“あああああ”;
}
```

さっきのスライドでは構造体をまだ変数として宣言していない点に注意！

typedefは新しい型を定義しただけなので、実際に使うときは←のように関数内でしっかりと構造体の名前を宣言しなくてはなりません

構造体メンバを使うにはドット演算子を使います

ポインタを使うときはアロー演算子を使います

構造体の初期化、メンバ全体の操作

```
Ex)
void main(){
Status ch1={100,50,”あああああ”};
Status ch2=ch1;
Status teki[50];
Status ch3[5]={
{130,50,”アルス”},
{150,20,”キーファ”},
{110,60,”マリベル”},
{120,30,”ガボ”}   };
}
```

構造体は、構造体変数を宣言した時に同時に初期値を与えて初期化することも可能です(ch1)

また構造体は、もっている変数を全部そのまま代入することができます(ch2)

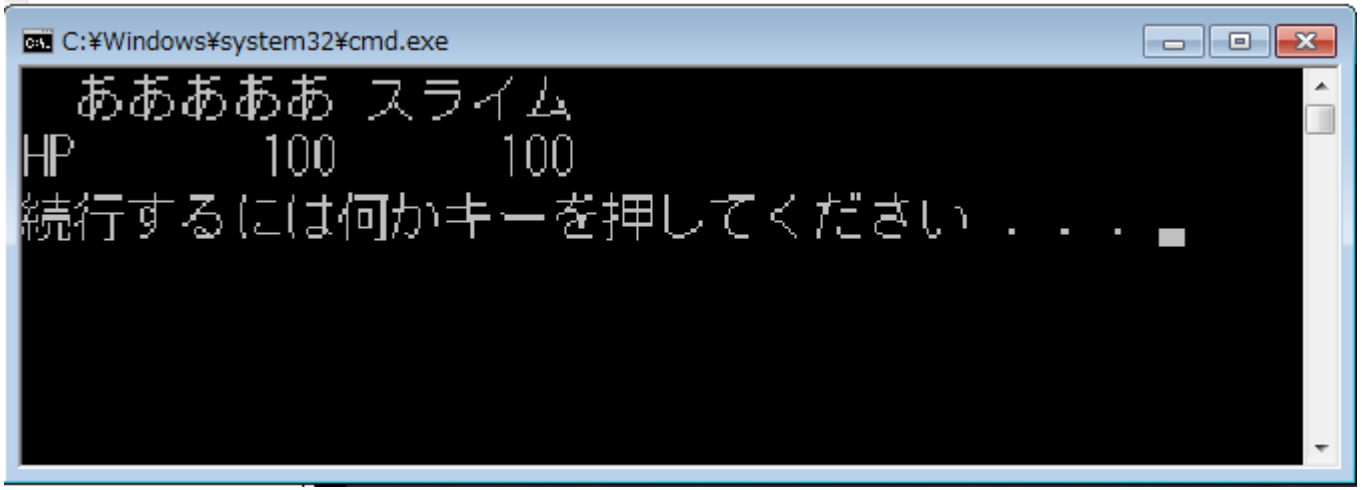
また構造体を配列で定義することもできます(teki)

配列で初期化は2次元配列の初期化のしかたと同じです(ch3)

演習問題(1)

構造体を使って↓の実行結果になるようソースコードを作成してください

ヒントは前のスライドを参考にしてね



解答

```
#include<stdio.h>
typedef struct {
int HP;
char name[16];
} Status;
void main(){
Status ch={100,“あああああ"};
Status teki={100,“スライム};
printf(" %s %s¥n",ch.name,teki.name);
printf("HP %d %d¥n",ch.HP,teki.HP);
}
```

ファイル分割

複数のファイルに分けてコンパイルする(1)

今は簡単なソースコードなので、1つのファイルで良いですが、
ゲームなど大規模なプロジェクトになると1つのファイルでつくる
わけには行けません。1つのファイルに全ての機能が含まれていると
どこになんの機能があるかさっぱりわからなくなります。
そこでなんらかの単位でソースコードをファイルごとに分割します。

まず、ファイルを分けるとグローバル変数であってもファイル間で変数
の参照はできません。関数の呼び出しもできません。これらをできる
ようにするには
「この変数や関数がどこで定義してあるからね」と一文
を書く必要があります

```
/*ch.c*/
#include<stdio.h>
void ch_test(){
printf(“分割できています。¥n");
}
/*ch.h*/
void ch_test();
/*main.c*/
#include<stdio.h>
#include”ch.h”
void main(){
ch_test();
}
```

ファイルを分割するのでとりあえずソースファイルにch.cを作りましょう。
ここから行うのがファイル分割で必要な重要な作業です。
ファイルは「.c」と「.h(ヘッダファイル)」を対でつくります(ファイル名を同じにするのが一般的)
.cには実際の処理、ヘッダファイルにはその関数のプロトタイプ宣言を書きます
次にmain.cを作りましょう

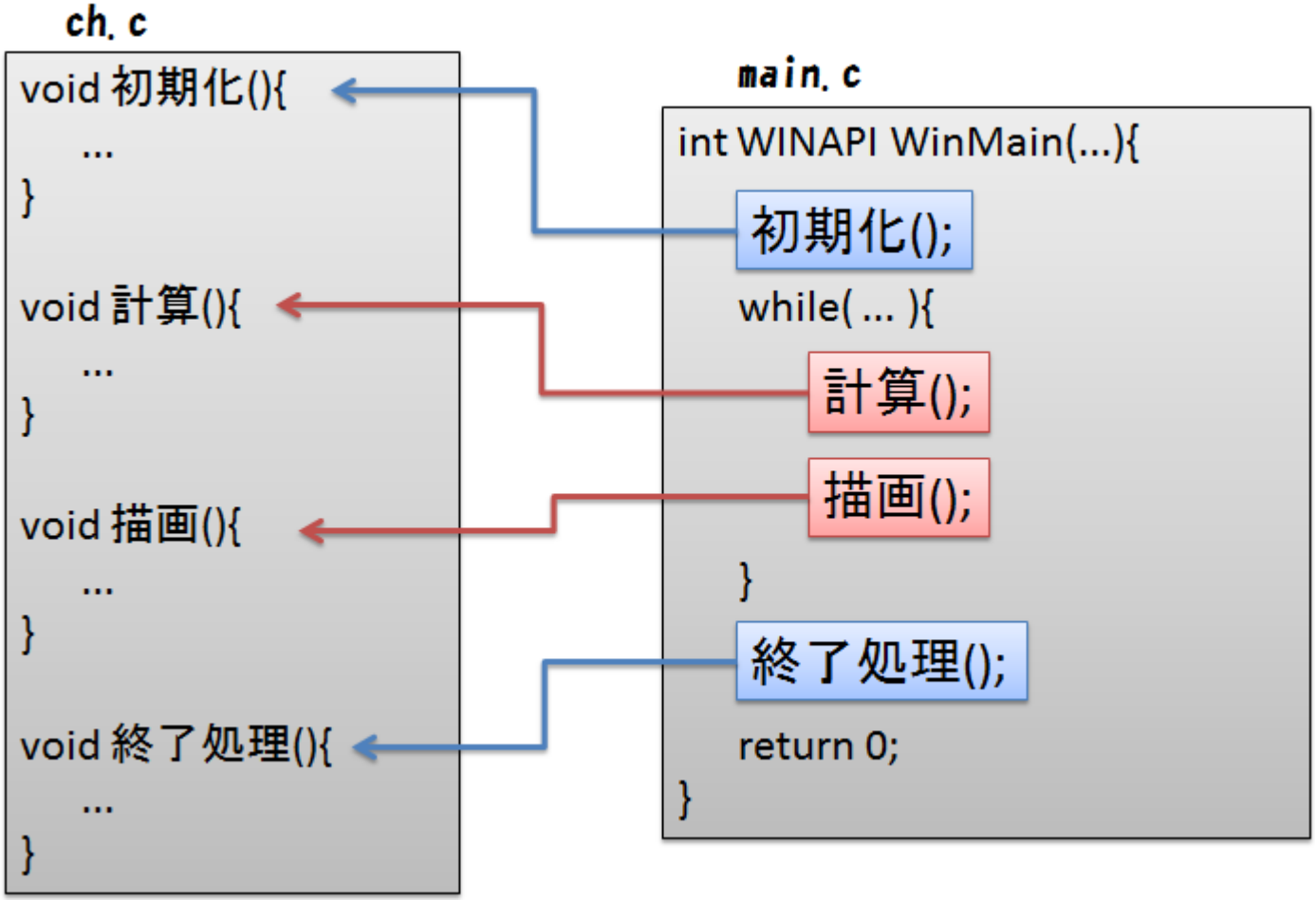
#include”ch.h”とかけばch.cに描いた内容をそこに書いたのと同じことになります。
自作のヘッダファイルは<>ではなく””で囲みます

※.hファイルに書かれている内容を#include<.h>と置き換えている

複数のファイルに分けてコンパイルする(2)

これで他のファイルに実体をもつ関数の呼び出しができるようになりました。しかし「メインファイルで宣言した変数を他のファイルで参照できるか？」と思うかもしれないけど・・・やめてください
グローバル変数でどのファイルからも参照可能にするとソースコードが巨大になったとき、いつどこで変更されているかさっぱりわからなくなるからです。
例えば、「敵.c」というファイルがあったとして、敵に関する変数はすべて「敵.c」にしか存在せず、このファイル内でしか変更できない仕組みだったら何か敵に関する変数にバグが生じたとき、原因を探すのは楽ですね。このように「見せる必要のないところからは変数を見せないようにする」ことをC++では「カプセル化」、「隠蔽化」といいます。C言語でもしっかりと隠蔽しましょう。
しかし変数参照できないのにどうやって処理をやり取りするのか？

まぁ、大丈夫です。関数さえ呼べればそれでいいのです。
必要な情報は関数の引数で与えてやり、取得したいときは関数の
返り値で得ればいいのです（ポインタを引数に持たせて入れて返す
仕組みでもok）



ここで一般的な処理の流れを見ておきましょう

- 初期化
- 計算
- 描画
- 終了処理

という4つの処理をもっています

複数のファイルに分けてコンパイルする(3)

```
/*ch.h*/
#ifndef DEF_CH_H //二重include防止
#define DEF_CH_H
void ch_Initialize(); // 初期化をする
void ch_Update(); // 動きを計算する
void ch_Draw(); // 描画する
void ch_Finalize(); // 終了処理をする
#endif
```

ヘッダファイルに書く一般的な内容は
・別ファイルに書かれている関数のプロトタイプ宣言
・グローバル関数
・構造体

例として、さっきの図のヘッダファイルを書いてみましょう
まず二重includeを防止するようにch.hを書きます
プログラムが大きくなると意図せず同じヘッダファイルをincludeしてしまうことがあります
二度目は通らないように#ifndefと#endifを使って条件わけしています。
DEF_CH_Hはインクルードガードといいヘッダファイルには必ずつけましょう

ライブラリ

標準でついているもの以外のものがある

1から全部作るのは面倒
既存のものを使うのが賢い
あるプラットフォーム上で動かすものはそれを使わないと動かない？

もっと高度なことをしたい人用

WindowsAPI

WindowsAPI

・Win32といわれるもの

・Windowsアプリケーションを CやC++

で作りたいならこれ

・ウィンドウの生成やDirectXでの描写、ネットワーク通信、スレッドなどたくさんの機能をもつ

・難易度は結構高め

DXライブラリ

前で説明したwindowsAPIのラッパー
ライブラリ

DirectXやWindows関連のプログラム
を使いやすくまとめた形で利用できる

難易度は低め

OpenCV

- 画像処理に特化したライブラリ
- 構造解析やモーション解析、パターン認識（物体検出）、機械学習（顔認識とか）

- 難易度は中ぐらい？