

C言語講座第6回目

構造体などなど

みっしえる

遊び人

せいべつ：おんな

レベル： 0

HP：133

MP：132

ちから：248

すばやさ： 43

たいりょく：208

かしこさ：217

うんのよさ：234

さいたいHP：231

さいたいMP：196

こうげき力：378

しゅび力：456

Ex：72156862

E トンカチ

E いかしたスーツ

E ダンボール

E むぎわら帽子

構造体

Ex)

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

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。

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)

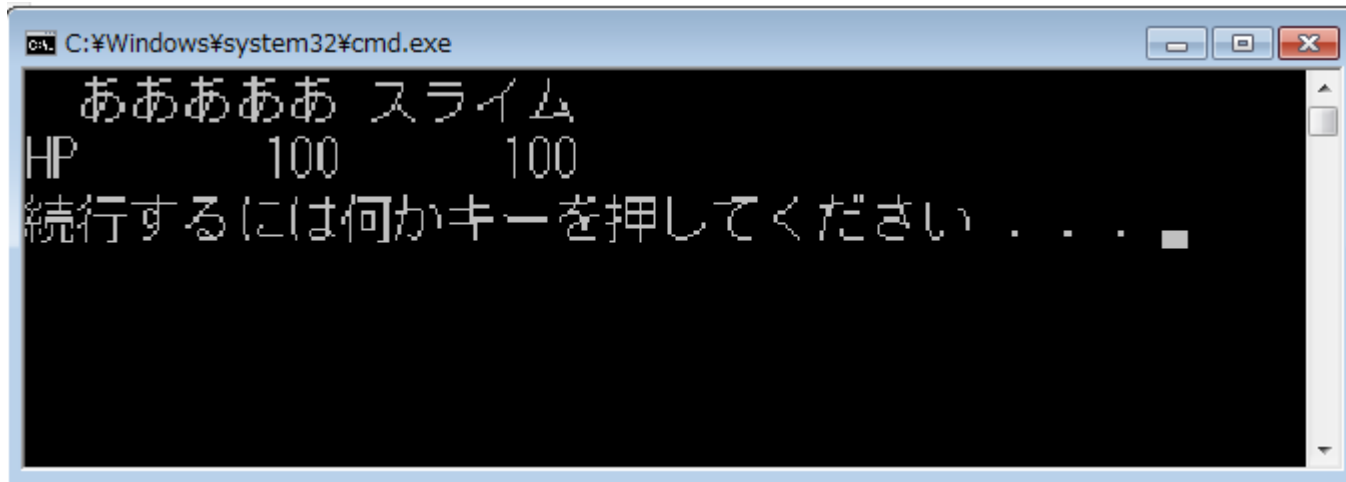
また構造体は、もっている変数を全部そのまま代入することができます(ch2)

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とね



```
C:\Windows\system32\cmd.exe
ああああ スライム
HP      100      100
続行するには何かキーを押してください . . . .
```

解答例

```
#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);
}
```

ここからは、C言語の授業でおしえてくれなさそうのでプログラムを組むときに何かと必要になるかもしれないことを教えようと思います。

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、ファイルに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。これらが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にかく
ソースファイルにch.cを作らしまし
ょう。

ここから行お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を作らしまし

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)

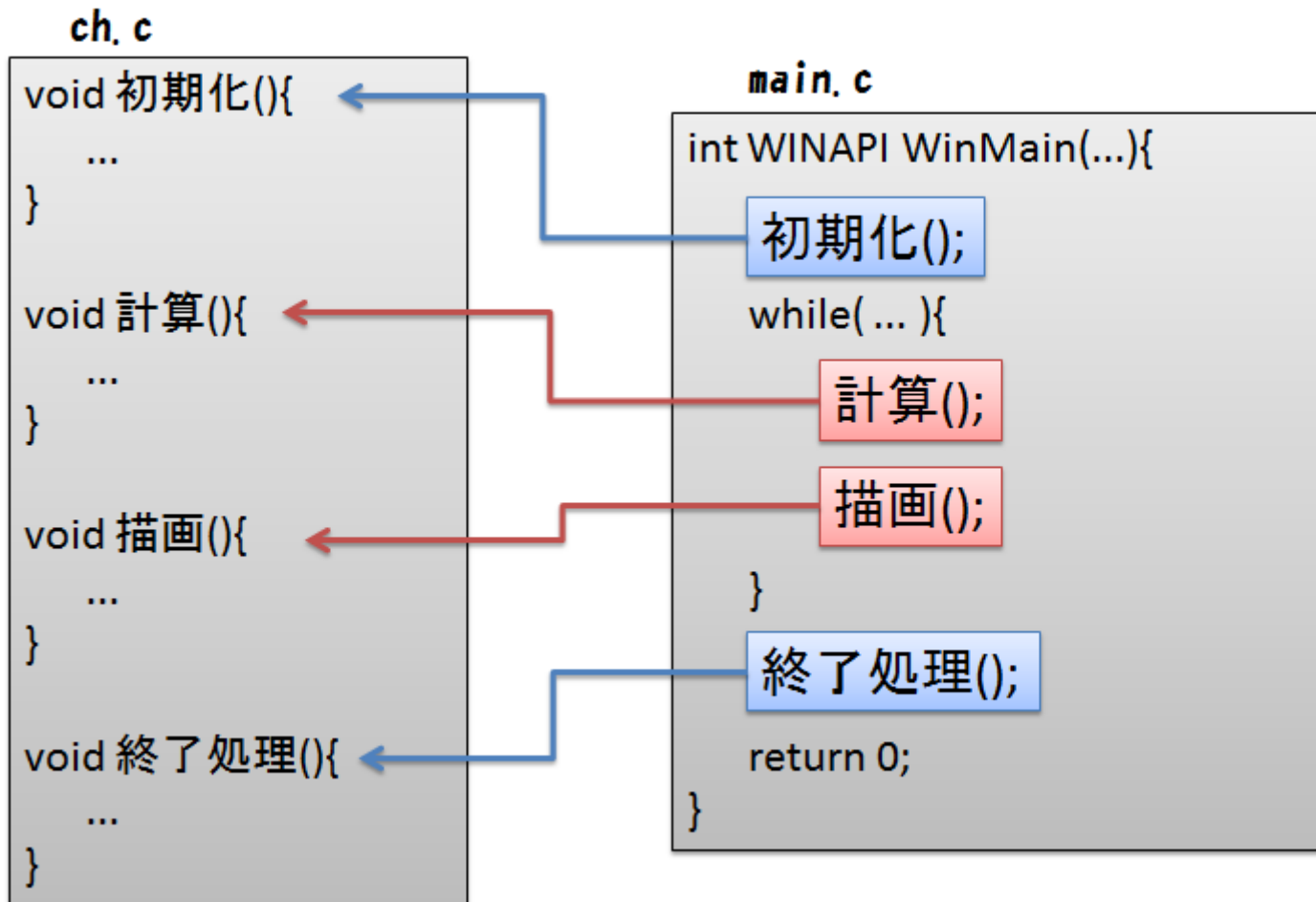
これで他のファイルに実体をもつ関数の呼び出しができるようになり
ました。しかし「メインファイルで宣言した変数を他のファイルで参照
できるか？」と思うかもしれないけど…**やめてください**

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？

まあ、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
処理をもち
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

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**を使って条件付けしています。
DEF_CH_Hはインクルードガードといふヘッダファイルには必ずつけましょう

標準ライブラリ関数

- コンパイラメーカーがよく使う機能をオブジェクトライブラリとして提供してくれるもの
- 使うときは`#include<>`
- 今までよく使っていた`stdio.h`は入出力用の関数

Ex

- 入出力 < `stdio.h` >
- 汎用 < `stdlib.h` >
- 文字処理 < `ctype.h` >
- 文字列処理 < `string.h` >
- 数学関数 < `math.h` >
- 時間 < `time.h` >

ライブ러리

標準でついているもの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

- 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

**言語をCやC++にこだわる必要はまった
くない**

他にも言語はたくさんある

C C++ C# objective-c

Visual Basic.NET JAVA

PHP Perl Python Ruby

JavaScript ...

上達するには？

- **本で基礎知識を身につける。**
- **自力でプログラムを組む力を身につける。**

- **あくまで学ぶことなので自分との闘いでもわからないところは聞ける**
- **まずは簡単な物を作ろう。そこから順に難しいものをつくっていく**
- **アルゴリズム的問題を解いてみる**
- **毎日、プログラムに触れよう**