構造体とポインタ

ここまでに構造体の基本的な使い方と typedef というユーザが定義する型について学んだ。今回以降は構造体へのポインタと構造体を引数や返却値に用いた関数の利用方法について学んでいく。

これは関連データをひとまとめにした上で関数という機能単位にまとめる方法である。多くのデータを引数として関数に渡す場合には構造体にまとめた上で行うと引数が一つで済む。これはプログラムの仕様が変更され、引数を増やしたい場合にも、構造体メンバに変数を追加するだけで関数のプロトタイプを変更しなくて良いというメリットもある。このように、今回学ぶ項目はプログラムを部品化することによって、プログラムを把握しやすくしたり、再利用性を高めたりすることができるため、とても役に立つ。

今回のテキストについては typedef を習った後なので、全てのプログラム例で使用する。

構造体へのポインタ

ポインタについては以前に学んでいる。構造体へのポインタも基本的には同じであるが、構造体の使用方法を含めると以下のようになる。

①構造体のテンプレート（枠型）の宣言
②構造体の宣言とポインタ変数の宣言
③値（アドレス）の設定
④構造体メンバの参照

それでは具体的な使用方法について、簡単なプログラムの例を以下に示す。

プログラム例１

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

// ①構造体のテンプレート（枠型）の宣言
typedef struct record                    // 成績
{
        int number;                      // 番号
        char name[20];                   // 名前
        double average;                  // 平均点
}RECORD;

int main(void)
{
        // ②構造体とポインタの宣言
        RECORD student1, *p;

        // ③アドレスの設定
        p = &student1;

        // ④構造体メンバの参照
        p->number = 1;
        strcpy(p->name, "Satou");
        p->average = 90.2;
```

```
        printf("%d %s %.1f¥n", p->number, p->name, p->average);

        return 0;
}
```

このプログラムは①の処理時に構造体 struct record の枠型について typedef を用いて RECORD 型として新しい型の定義を行っている。次に以下のようにして、構造体の本体（実際にメモリ上に領域が取られたもの）student1 とポインタ変数 p を宣言し、ポインタ変数 p に構造体 student1 のアドレスを設定する。この処理は基本的な使い方として前述した②、③の処理である。

```
RECORD student1, *p;
p = &student1;
```

特徴的なのは④の構造体メンバの参照であり、以下の書式に従う。

書式
構造体変数名->メンバ名
使用例
p->number = 1;

書式に使った「->」はアロー演算子（矢印演算子）といい、マイナス「-」と大なり「>」の二つの記号で空白を入れずに連続して記述する。構造体変数名からメンバ名に対して矢印が出ている形に見えるので直感的に分かりやすい。

また、使用例で書いた p->number = 1;は実は(*p).number = 1;という書き方と等しい。ただし、この場合は*p.number という書き方はできない。「.」の方が「*」よりも優先順位が高いため別の意味になってしまうため、括弧は必要である（注：gcc と VisualStudio2003 ではコンパイルエラーが発生)。(*p).number という書き方は手間がかかり、構造体へのポインタを利用したメンバの参照はプログラム中で良く利用されるため、アロー演算子が用意されている。

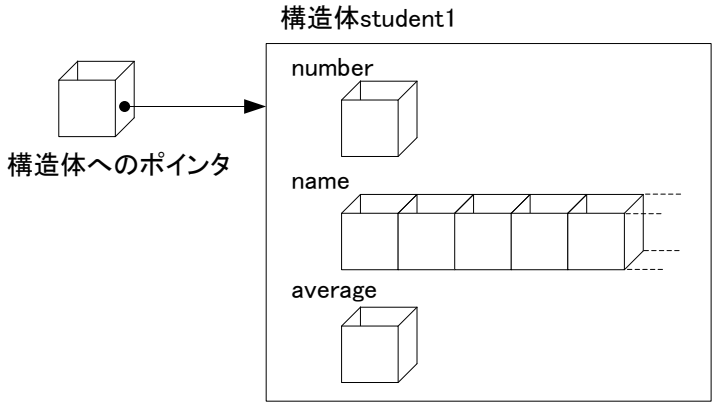

図１　構造体へのポインタ

構造体のネストとポインタ

構造体のネストとポインタを用いたプログラム例を以下に示す。

プログラム例２

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

// 構造体のテンプレート（枠型）の宣言
typedef struct subject            // 科目
{
        int mathematics;          // 数学
        int english;              // 英語
}SUBJECT;

typedef struct record             // 成績
{
        int number;               // 番号
        char name[20];            // 名前
        SUBJECT general;          // 一般科目
}RECORD;

int main(void)
{
        // 構造体とポインタの宣言
        RECORD student1, *p;

        // アドレスの設定
        p = &student1;

        // 構造体メンバの参照
        p->number = 1;
        strcpy(p->name, "Satou");
        p->general.mathematics = 90;
        p->general.english = 80;

        printf("%d %s %d %d¥n",
                p->number, p->name, p->general.mathematics, p->general.english);

        return 0;
}
```

注意点は構造体メンバの参照部分である。構造体のネストを用いた場合、構造体のメンバに構造体が含まれる。このメンバに含まれた構造体は変数とポインタの違いで、ドット演算子とアロー演算子を使い分けることになる。前回のテキストの「構造体のネスト」の項目では両方が変数だったため、**「構造体変数名.構造体変数名.メンバ名」(例:student1.general.mathematics)** で指定した。このプログラム例２では外側はポインタで内側は変数であるため、メンバの参照は**「構造体変数名->構造体変数名.メンバ名」(例：p->general.mathematics)** となる。

仮に両方がポインタだった場合にはどうなるだろうか？その場合は以下のようなプログラムとなり、**「構造体変数名->構造体変数名->メンバ名」(例：p->ps->mathematics)** となる。この場合のプログラム例を次に示す。

プログラム例３

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

// 構造体のテンプレート（枠型）の宣言
typedef struct subject            // 科目
{
        int mathematics;          // 数学
        int english;              // 英語
}SUBJECT;

typedef struct record             // 成績
{
        int number;               // 番号
        char name[20];            // 名前
        SUBJECT *ps;              // 一般科目(ポインタに変更)
}RECORD;

int main(void)
{
        // 構造体とポインタの宣言
        RECORD student1, *p;
        SUBJECT general;          // RECORD型のメンバであるSUBJECTの本体を宣言する

        // アドレスの設定
        p = &student1;
        p->ps = &general;         // ネストされた構造体変数にアドレスを設定する

        // 構造体メンバの参照
        p->number = 1;
        strcpy(p->name, "Satou");
        p->ps->mathematics = 90;
        p->ps->english = 80;

        printf("%d %s %d %d¥n", p->number, p->name, p->ps->mathematics, p->ps->english);

        return 0;
}
```

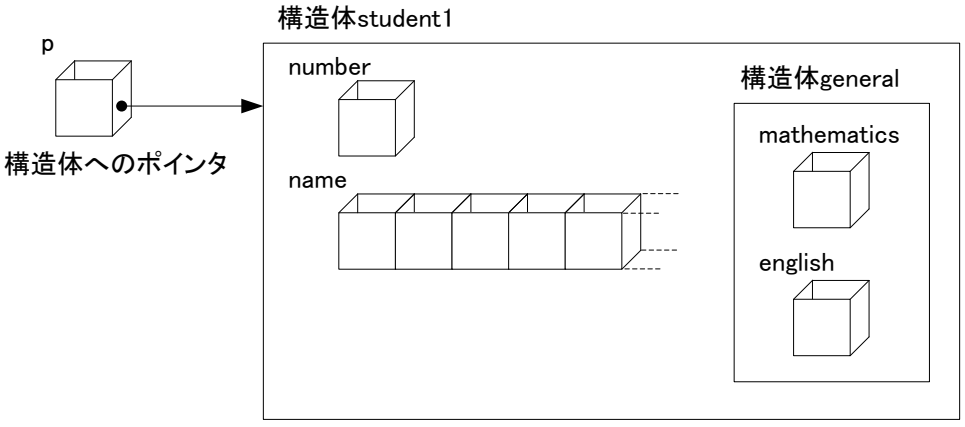

図２　構造体のネストとポインタ（プログラム例２）

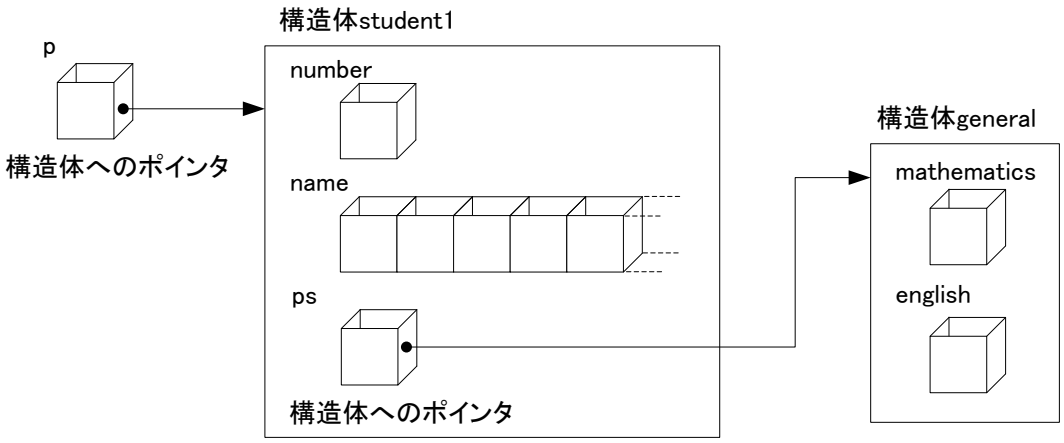

図３　構造体のネストとポインタ（プログラム例３）

構造体配列へのポインタ

それでは次に構造体配列へのポインタの使用例について示す。

プログラム例４

```
#include <stdio.h>

// 構造体のテンプレート（枠型）の宣言
typedef struct record             // 成績
{
        int number;               // 番号
        char name[20];            // 名前
        double average;           // 平均点
}RECORD;

int main(void)
{
        int i;

        // 構造体とポインタの宣言
        RECORD student1, student2[3], *pa, *pb;

        // アドレスの設定
        pa = &student1;
        pb = student2;

        // 構造体メンバの参照
        // 構造体student1のメンバへの入力
        printf("学生1¥n");
        printf("学籍番号:");
        scanf("%d", &pa->number);
        printf("名前:");
        scanf("%s", pa->name);
        printf("平均値:");
        scanf("%lf", &pa->average);

        // 構造体配列student2のメンバへの入力
```

```
        // 相対的なポインタ参照
        for(i = 0; i < 3; i++)
        {
                printf("学生%d¥n", i+2);
                printf("学籍番号:");
                scanf("%d", &(pb+i)->number);
                printf("名前:");
                scanf("%s", (pb+i)->name);
                printf("平均値:");
                scanf("%lf", &(pb+i)->average);
        }

        printf("¥n入力結果¥n");
        printf("%d %s %.1f¥n", pa->number, pa->name, pa->average);

        // 構造体配列student2のメンバの表示
        // 相対的なポインタ参照
        for(i = 0; i < 3; i++)
        {
                printf("%d %s %.1f¥n", (pb+i)->number, (pb+i)->name, (pb+i)->average);
                // 以下のどちらの書き方でも正しく動作する
                // printf("%d %s %.1f¥n", pb[i].number, pb[i].name, pb[i].average);
                // printf("%d %s %.1f¥n",
                        student2[i].number, student2[i].name, student2[i].average);
        }

        return 0;
}
```

構造体配列の宣言とポインタ変数へのアドレスの設定は以下のようにして行う。配列の先頭アドレスが欲しい場合には配列名だけを書けばよいことは以前に習った（例えば int a[10];と宣言したならば、a とだけ書くと配列の先頭アドレスとなる)。構造体配列の場合についても同様である。

```
RECORD student2[4];
pb = student2;
```

次にメンバの参照である。ポインタの授業で「相対的なポインタ参照と絶対的なポインタ参照」を習ったが、このプログラム例では相対的なポインタ参照で記述した。具体的な書式は以下ようになる。

```
(pb+i)->number              ・・・(1)
```

このメンバの参照では以下の書き方でも問題無い。アロー演算子はドット演算子に変わっていることに注意すること。

```
pb[i].number                ・・・(2)
```

これは通常の構造体配列のメンバの参照方法と同じように見えるが、実際、元々の構造体配列名

は student2 であるので、pb の部分を置き換えた以下の形でも、正しく動作する。

```
student2[i].number        ･･･(3)
```

構造体配列とポインタの関係を示すために、(1)～(3)の利用方法を示した。状況に応じて使い分けたり、既存のプログラムを改造したりする場合には役に立つと思う。プログラムを自分で作成する場合には(3)の使い方を勧めておく。理由として、構造体配列は宣言をすれば、その後は配列名を利用するとアドレスを扱うことができるため、改めてポインタ変数を宣言する必要が無くなり、プログラムが簡潔になるからである。

C 言語で扱うデータは大きく分けると「値」か「アドレス」になる（どちらも CPU では 2 進数のデータとして扱われるが、例えば、前者は int a;と宣言した変数に格納するデータであり、後者は int a;と宣言した際に&a で取り出せるデータである)。変数、配列、ポインタ変数、&や*の演算子などを組み合わせることで、いろいろな記述が可能となる。扱っているデータがどちらを意味しているかを理解することが大切である。

補足として、構造体配列 student2 のメンバの表示では相対的なポインタ参照で記述しているが、以下のようにして、絶対的なポインタ参照で表示することも可能である。

```
// 構造体配列student2のメンバの表示
// 絶対的なポインタ参照
printf("%d %s %.1f¥n", pb->number, pb->name, pb->average);
pb++;
printf("%d %s %.1f¥n", pb->number, pb->name, pb->average);
pb++;
printf("%d %s %.1f¥n", pb->number, pb->name, pb->average);
```

演習　プログラム例１～４を作成し、実行結果を確認しなさい。プログラム例４についてはテキストで解説した構造体メンバの参照方法(1)～(3)、及び絶対的なポインタ参照についても確認すること。

課題４

Kadai4-2　Kadai4-1（オリジナルのプログラム）で作成したプログラムについて構造体へのポインタを利用して書き直しなさい。Kadai4-1 で既にポインタを利用したプログラムを作成したものはレポートに「Kadai4-1 と同じ」と書きなさい。

課題の続きは後で示す。