構造体

ここまでに char、int、doulbe などの基本的なデータ型に加えて、同じデータ型が連続している配列についてのデータ構造について習った。これ以外にも、もっと複雑なデータ型をユーザが定義することが可能である。それが構造体と呼ばれるもので、異なる型のデータをひとかたまりのデータとして扱うことができる。

異なるデータをまとめて扱いたい時とはどんな場合だろうか。例えば、住民データを管理したい場合を考える。個人を識別するデータとして、識別番号、氏名、年齢、住所、電話番号、…などなど、多くのデータが挙げられる。これらは文字列であったり、数値であったりする。このデータを元にして、管理台帳プログラムを作成する場合には、これらのデータはひとまとまりとして扱った方がすっきりするし、便利である。実際に異なるデータをまとめて扱いたい機会というのは、非常に多い。

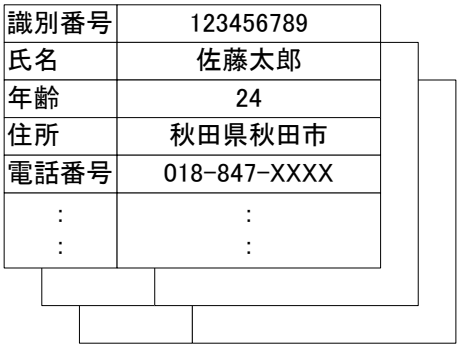

図１　データの管理例

構造体の使用手順

サンプルプログラムを見ながら構造体の使用手順を説明していく。以下は学生の平均点を管理し、表示するプログラムである。構造体に含まれるデータは学籍番号、名前、平均点としている。基本的な使用は以下に示す通り。

①構造体のテンプレート（枠型）の宣言
②構造体の宣言・初期化
③構造体メンバの参照

プログラム例１

```
#include <stdio.h>

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

int main(void)
{
        int i;
```

```
        // 構造体の宣言と初期化
        struct record student1 = { 1, "Satou", 90.2 };
        struct record student2[3] = {
                { 2, "Suzuki", 77.3 },
                { 3, "Takahashi", 59.1 },
                { 4, "Tanaka", 64.9 }
        };

        // 構造体メンバの参照
        printf("%d %s %.1f¥n", student1.number, student1.name, student1.average);
        for(i = 0; i < 3; i++)
        {
                printf("%d %s %.1f¥n",
                        student2[i].number, student2[i].name, student2[i].average);
        }

        return 0;
}
```

実行結果

```
1 Satou 90.2
2 Suzuki 77.3
3 Takahashi 59.1
4 Tanaka 64.9
```

それでは個別に説明をしていく。

①構造体のテンプレート（枠型）の宣言

最初に構造体のテンプレートを宣言する必要がある。この処理ではひとまとめにするデータを記述していく。書式と使用例を以下に示す。struct は構造体という意味である。またタグ名というのは構造体の枠型に付ける名前である。データ型は今までに習った int や char などを使うことができ、配列やポインタも利用できる。

構造体を構成するデータとして定めた変数（使用例での変数 number、name、average に当たる）のことを構造体のメンバと呼ぶ。

①の宣言自体は単に構造体の枠型を作ったにすぎず、メモリ上に実際のデータを格納する領域が割り当てられてはいない状態である。

書式

```
struct  タグ名 {
        データ型  変数名;
        データ型  変数名;
                :
};
```

使用例

```
struct record           // 成績
{
        int number;     // 学籍番号
        char name[20];  // 名前
```

```
        double average; // 平均点
};
```

②構造体の宣言と初期化

①で作ったテンプレートを使って実際に構造体を宣言し、メモリ上に領域を確保する。これを構造体の宣言という。構造体は使用例に示すように、変数（struct record student1;）や配列（struct record student2[4];）で扱うことができる。また、ポインタで扱うこともできるが、これについては後で述べる。

書式
struct タグ名 変数名;
使用例
struct record student1; struct record student2[3];

また、構造体は通常の変数や配列同様にして、以下に示すようにして初期化を行うことができる。

```
struct record student1 = { 1, "Satoh", 90.2 };
struct record student2[3] = {
        { 2, "Suzuki", 77.3 },
        { 3, "Takahashi", 59.1 },
        { 4, "Tanaka", 64.9 },
};
```

③構造体メンバの参照

構造体の各メンバを利用する場合には以下に示すようにして利用する。ここで使用しているピリオド（.）はドット演算子と呼ぶ。

書式
構造体変数名.メンバ名
使用例
student1.number = 1; scanf("%s", student2[0].name);

使用例に示したように記述することによって、各メンバは変数や配列の場合と同様に、値の代入や計算に利用することができる。

構造体の使用例

ここまでに構造体の使用手順について説明した。ここでは構造体の使用例として、scanf()でのデータ入力、構造体の一括代入、構造体のネスト（入れ子）について説明する。

（1）scanf( )を用いてのデータ入力

構造体に scanf( ) でデータを入力する場合も通常の変数への入力の場合と変わらない。各メン

バのデータ型を考えて引数を指定して行う。変数のアドレスをとりだすには、'&'を変数の前に付ける。配列のアドレスをとりだすには、'&'は不要である。

プログラム例２

```
#include <stdio.h>

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

int main(void)
{
        int i;

        // 構造体の宣言
        struct record student1, student2[3];

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

        // 構造体配列student2のメンバへの入力
        for(i = 0; i < 3; i++)
        {
                printf("学生%d¥n", i+2);
                printf("学籍番号:");
                scanf("%d", &student2[i].number);
                printf("名前:");
                scanf("%s", student2[i].name);
                printf("平均値:");
                scanf("%lf", &student2[i].average);
        }

        // 構造体メンバの参照
        printf("¥n入力結果¥n");
        printf("%d %s %.1f¥n", student1.number, student1.name, student1.average);
        for(i = 0; i < 3; i++)
        {
                printf("%d %s %.1f¥n",
                        student2[i].number, student2[i].name, student2[i].average);
        }

        return 0;
}
```

実行結果例

```
学生１
学籍番号:1
名前:Satou
平均値:90.2
学生２
学籍番号:2
名前:Suzuki
平均値:77.3
     :

入力結果
1 Satou 90.2
2 Suzuki 77.3
     :
```

（２）構造体の代入

同じ型を持った構造体は一括して代入することが可能である。以下の例では student1 の内容が student2 と student3[0]に代入されている。

構造体のメンバへの代入で用いている関数 strcpy は文字列のコピーを行う。この関数は string.h をインクルードすることで利用する。

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
struct record            // 成績
{
        int number;      // 番号
        char name[20];   // 名前
        double average; // 平均点
};

int main(void)
{
        // 構造体の宣言
        struct record student1;
        struct record student2, student3[3];

        // 構造体のメンバへの代入
        student1.number = 1;
        strcpy(student1.name, "Satou");
        student1.average = 90.2;

        // 構造体の一括代入
        student2 = student1;
        student3[0] = student1;

        // 構造体メンバの参照
        printf("%d %s %.1f¥n", student1.number, student1.name, student1.average);
```

```
        printf("%d %s %.1f¥n", student2.number, student2.name, student2.average);
        printf("%d %s %.1f¥n", student3[0].number, student3[0].name, student3[0].average);

        return 0;
}
```

実行結果

```
1 Satou 90.2
1 Satou 90.2
1 Satou 90.2
```

（3）構造体のネスト

構造体はそのメンバに構造体を持つことが出来る。これを構造体のネストという。以下のプログラム例では学生の平均点を扱う部分を数学と英語の点数に置き換えたものである。数学と英語は一般科目という枠組みでまとめている。

プログラム例４

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

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

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

int main(void)
{
        // 構造体の宣言と初期化
        //struct record student1 = { 1, "Satou", 90, 80};
        struct record student1 = { 1, "Satou", {90, 80} }; // 中括弧を付けたほうがより明示的
        struct record student2;

        // 構造体のメンバへの代入
        student2.number = 2;
        strcpy(student2.name, "Suzuki");
        student2.general.mathematics = 59;
        student2.general.english = 97;

        // 構造体メンバの参照
        printf("%d %s %d %d¥n", student1.number, student1.name,
                student1.general.mathematics, student1.general.english);
        printf("%d %s %d %d¥n", student2.number, student2.name,
                student2.general.mathematics, student2.general.english);
```

```
        return 0;
}
```

実行結果

```
1 Satou 90 80
2 Suzuki 59 97
```

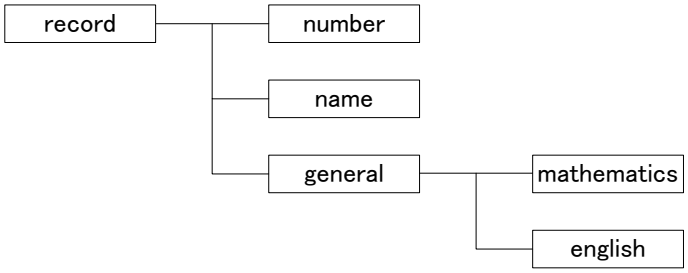

図２　構造体のネストのイメージ

ここでの大事な点は構造体テンプレートの宣言の順番である。2つ目のテンプレートのメンバでは struct subject general;を宣言している。この宣言の前に struct subject のテンプレートの宣言が必要となる。つまり、構造体テンプレートの宣言の順番を逆にするとコンパイルエラーが起きる。

構造体のネストは以下のようにしてまとめて記述することも可能である。

使用例

```
struct record                           // 成績
{
        int number;                     // 番号
        char name[20];                  // 名前
        struct subject                  // 一般科目
        {
                int mathematics;        // 数学
                int english;            // 英語
        }general;
};
```

「typedef」について

typedef は型の定義という意味があり、既にあるデータ型に対して、新しい名前を付けて利用するもので、以下のように記述する。

書式

```
typedef 既にある型 新しい名前
```

使用例

```
#include <stdio.h>

typedef unsigned int UINT;

int main(void)
{
        UINT data;
```

```
        data = 10;

        return 0;
}
```

typedef で宣言される新しい名前は、プログラム中で区別しやすいように一般的には大文字を使用する。typedef は#define とよく似ており、新しい型を作り出すのではなく、文字列の置き換えを行っている。ただし、型だけに限定しているために目的がはっきりしている。

使用例の場合には unsigned int 型を新しい名前 UINT として typedef 宣言した。この宣言以降、UINT という型を利用することができる。unsigned は符号無し変数を宣言する場合に利用し、通常の符号有りの場合に比べて、利用できる正の数の最大値を大きくしたい場合に用いる。このデータ型は変数の宣言の記述が長いので、より短い名前に置き換えて利用することが好まれる場合がある。

構造体の宣言では struct が付くため、型名が長くなる。構造体の宣言を簡潔にするための typedef は非常に良く使われる。プログラム例１で用いたプログラムを typedef を用いて書き直すと次のようになる。

また、構造体のテンプレートの宣言に用いたタグ名 record は、typedef を用いた場合に省略することができる。その際にはコンパイラによって適当なタグ名が割り振られる。プログラムする上では、typedef 以降、データ型を RECORD として構造体の宣言をするので、コンパイラによって付けられた「適当なタグ名」を意識することはない。

プログラム例５

```
#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;

        // 構造体の宣言と初期化
        RECORD student1 = { 1, "Satou", 90.2 };
        RECORD student2[3] = {
                { 2, "Suzuki", 77.3 },
                { 3, "Takahashi", 59.1 },
                { 4, "Tanaka", 64.9 }
        };

        // 構造体メンバの参照
        printf("%d %s %.1f¥n", student1.number, student1.name, student1.average);
        for(i = 0; i < 3; i++)
        {
                printf("%d %s %.1f¥n",
```

```
                        student2[i].number, student2[i].name, student2[i].average);
        }

        return 0;
}
```

実行結果

プログラム例１と同じ

補足：

以下は構造体のネストとプログラム例５の typedef で紹介した構造体テンプレートの宣言例である。ここで一つ確認したいことがある。構造体ネスト使用例の general と typedef 使用例の RECORD は同じものだろうか？

構造体ネスト使用例

```
struct record           // 成績
{
        int number;     // 番号
        char name[20];  // 名前
        struct subject  // 一般科目
        {
                int mathematics; // 数学
                int english;     // 英語
        }general;
};
```

typedef 使用例

```
typedef struct record           // 成績
{
        int number;             // 番号
        char name[20];          // 名前
        double average;         // 平均点
}RECORD;
```

確かに両方とも{}直後の位置に書いてあるが、これは別物である。構造体ネスト使用例の general は struct subject という構造体の枠型に対して宣言した「変数名」であり、typedef 使用例の RECORD は typedef によって、struct record に対応付けられた「型」である。この違いを理解していないとプログラム作成時の混乱の原因になるので気を付けたい。

最後に構造体ネスト使用例のプログラムの subject 構造体を typedef で型の宣言をして書き直したものを以下に示す。

typedef と構造体ネスト使用例

```
typedef struct subject  // 一般科目
{
        int mathematics; // 数学
        int english;     // 英語
}SUBJECT;

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

演習　プログラム例１～５を作成し、実行結果を確認しなさい。

課題４

Kadai4-1

　自分の身の回りのことについて構造体を利用したプログラムを考え、作成しなさい。このプログラムの頭の部分にはコメント文で簡単なプログラムの説明を記述すること。

課題の続きは後で示す。