# 第11章 構造体とポインタ

## 対応C規格

* **主要対象:** C90
* **学習内容:** 構造体の基本、構造体ポインタ、構造体配列、->演算子、メモリ管理

## 学習目標

この章を完了すると、以下のことができるようになります：

* 構造体の基本概念を理解する
* 構造体ポインタを効果的に使える
* ->演算子と.演算子の使い分けができる
* 構造体配列とポインタ配列を活用できる
* 構造体を使ったデータ管理ができる

## 概要と詳細

### 構造体とは？

構造体は、複数の異なるデータ型をまとめて1つの新しいデータ型を作る機能です。現実世界の「モノ」をプログラムで表現するのに最適です！

#### 日常生活での構造体

身の回りにあるものを考えてみましょう：

1. **学生証の情報**
   * 学籍番号（整数）
   * 名前（文字列）
   * 学部（文字列）
   * 入学年（整数）
2. **商品の情報**
   * 商品コード（整数）
   * 商品名（文字列）
   * 価格（小数）
   * 在庫数（整数）

これらの「関連する情報の集まり」を1つにまとめるのが構造体です！

### なぜ構造体が必要なのか？

構造体を使わないと、関連するデータをバラバラに管理することになります：

/\* 構造体を使わない場合（管理が大変！） \*/  
int student\_id[100];  
char student\_name[100][50];  
int student\_age[100];  
  
/\* 構造体を使う場合（スッキリ！） \*/  
struct Student students[100];

### 構造体の基本概念

構造体は異なるデータ型を組み合わせて新しいデータ型を作る仕組みです。

#### 構造体の作り方の3ステップ

1. **定義する**：どんな情報を含むか決める
2. **宣言する**：構造体の変数を作る
3. **使う**：メンバーにアクセスする

#### 構造体の定義と宣言

#include <stdio.h>  
#include <string.h>  
  
/\* 学生情報を表す構造体 \*/  
struct Student {  
 int id;  
 char name[50];  
 int age;  
 double gpa;  
};  
  
int main(void)  
{  
 struct Student student;  
   
 /\* メンバーに値を代入 \*/  
 student.id = 1001;  
 strcpy(student.name, "田中太郎");  
 student.age = 20;  
 student.gpa = 3.5;  
   
 /\* メンバーの値を表示 \*/  
 printf("学生ID: %d\n", student.id);  
 printf("名前: %s\n", student.name);  
 printf("年齢: %d\n", student.age);  
 printf("GPA: %.1f\n", student.gpa);  
   
 return 0;  
}

#### 構造体の初期化

#include <stdio.h>  
  
struct Point {  
 int x;  
 int y;  
};  
  
struct Rectangle {  
 struct Point top\_left;  
 struct Point bottom\_right;  
};  
  
int main(void)  
{  
 /\* 初期化方法1: 順序指定 \*/  
 struct Point p1 = {10, 20};  
   
 /\* 初期化方法2: 部分初期化 \*/  
 struct Point p2 = {5}; /\* x=5, y=0 \*/  
   
 /\* 初期化方法3: ネストした構造体 \*/  
 struct Rectangle rect = {{0, 0}, {100, 50}};  
   
 /\* 初期化方法4: 個別代入 \*/  
 struct Point p3;  
 p3.x = 30;  
 p3.y = 40;  
   
 printf("p1: (%d, %d)\n", p1.x, p1.y);  
 printf("p2: (%d, %d)\n", p2.x, p2.y);  
 printf("p3: (%d, %d)\n", p3.x, p3.y);  
 printf("矩形: (%d,%d) から (%d,%d)\n",   
 rect.top\_left.x, rect.top\_left.y,  
 rect.bottom\_right.x, rect.bottom\_right.y);  
   
 return 0;  
}

### 構造体ポインタの基本

構造体もポインタで操作できます。大きな構造体を関数に渡す際に効率的です。

#### なぜ構造体ポインタを使うのか？

1. **メモリ効率**
   * 構造体全体をコピーするより、アドレスだけ渡す方が高速
   * 特に大きな構造体では効果的
2. **関数での値の変更**
   * ポインタなら関数内で元の構造体を変更できる
3. **動的メモリ管理**
   * 実行時に必要な数だけ構造体を作成可能

#### 構造体ポインタの宣言と使用

#include <stdio.h>  
#include <string.h>  
  
struct Book {  
 char title[100];  
 char author[50];  
 int pages;  
 double price;  
};  
  
int main(void)  
{  
 struct Book book;  
 struct Book \*book\_ptr;  
   
 /\* 構造体ポインタを構造体のアドレスに設定 \*/  
 book\_ptr = &book;  
   
 /\* ポインタを通じてメンバーにアクセス（方法1: (\*ptr).member） \*/  
 strcpy((\*book\_ptr).title, "C言語入門");  
 strcpy((\*book\_ptr).author, "山田花子");  
 (\*book\_ptr).pages = 350;  
 (\*book\_ptr).price = 2800.0;  
   
 /\* 直接アクセスとポインタアクセスの比較 \*/  
 printf("=== 直接アクセス ===\n");  
 printf("書名: %s\n", book.title);  
 printf("著者: %s\n", book.author);  
 printf("ページ数: %d\n", book.pages);  
 printf("価格: %.0f円\n", book.price);  
   
 printf("\n=== ポインタアクセス ===\n");  
 printf("書名: %s\n", (\*book\_ptr).title);  
 printf("著者: %s\n", (\*book\_ptr).author);  
 printf("ページ数: %d\n", (\*book\_ptr).pages);  
 printf("価格: %.0f円\n", (\*book\_ptr).price);  
   
 return 0;  
}

### アロー演算子（->）の使用

構造体ポインタには便利なアロー演算子（->）が用意されています。

#### アロー演算子の覚え方

構造体変数 → . (ドット)を使う  
構造体ポインタ → -> (矢印)を使う

**記憶のコツ**：ポインタは「矢印」で指すから「->」（矢印記号）を使う！

#### ->演算子と.演算子の比較

#include <stdio.h>  
#include <string.h>  
  
struct Employee {  
 int id;  
 char name[50];  
 char department[30];  
 double salary;  
};  
  
void print\_employee\_info(struct Employee \*emp)  
{  
 printf("=== 社員情報 ===\n");  
 printf("ID: %d\n", emp->id); /\* emp->id は (\*emp).id と同じ \*/  
 printf("名前: %s\n", emp->name); /\* emp->name は (\*emp).name と同じ \*/  
 printf("部署: %s\n", emp->department); /\* emp->department は (\*emp).department と同じ \*/  
 printf("給与: %.0f円\n", emp->salary); /\* emp->salary は (\*emp).salary と同じ \*/  
}  
  
int main(void)  
{  
 struct Employee emp;  
 struct Employee \*emp\_ptr = &emp;  
   
 /\* ->演算子を使用してメンバーに値を設定 \*/  
 emp\_ptr->id = 2001;  
 strcpy(emp\_ptr->name, "佐藤次郎");  
 strcpy(emp\_ptr->department, "開発部");  
 emp\_ptr->salary = 350000.0;  
   
 /\* 3つの方法でアクセス \*/  
 printf("=== 直接アクセス（.演算子） ===\n");  
 printf("ID: %d\n", emp.id);  
   
 printf("\n=== ポインタ経由（(\*ptr).member） ===\n");  
 printf("ID: %d\n", (\*emp\_ptr).id);  
   
 printf("\n=== ポインタ経由（->演算子） ===\n");  
 printf("ID: %d\n", emp\_ptr->id);  
   
 printf("\n");  
 print\_employee\_info(emp\_ptr);  
   
 return 0;  
}

#### ->演算子の実践例

#include <stdio.h>  
#include <string.h>  
  
struct Car {  
 char brand[30];  
 char model[30];  
 int year;  
 double mileage;  
 int is\_electric;  
};  
  
/\* 車の情報を更新する関数 \*/  
void update\_mileage(struct Car \*car, double new\_mileage)  
{  
 if (car != NULL && new\_mileage >= car->mileage) {  
 car->mileage = new\_mileage;  
 printf("%s %s の走行距離を %.1f km に更新しました\n",   
 car->brand, car->model, car->mileage);  
 }  
}  
  
/\* 車の詳細情報を表示する関数 \*/  
void display\_car\_details(struct Car \*car)  
{  
 if (car == NULL) {  
 printf("無効な車両データです\n");  
 return;  
 }  
   
 printf("=== 車両情報 ===\n");  
 printf("ブランド: %s\n", car->brand);  
 printf("モデル: %s\n", car->model);  
 printf("年式: %d年\n", car->year);  
 printf("走行距離: %.1f km\n", car->mileage);  
 printf("電気自動車: %s\n", car->is\_electric ? "はい" : "いいえ");  
 printf("\n");  
}  
  
int main(void)  
{  
 struct Car my\_car;  
 struct Car \*car\_ptr = &my\_car;  
   
 /\* ->演算子で車の初期設定 \*/  
 strcpy(car\_ptr->brand, "Toyota");  
 strcpy(car\_ptr->model, "Prius");  
 car\_ptr->year = 2020;  
 car\_ptr->mileage = 15000.0;  
 car\_ptr->is\_electric = 0; /\* ハイブリッド車 \*/  
   
 display\_car\_details(car\_ptr);  
   
 /\* 走行距離を更新 \*/  
 update\_mileage(car\_ptr, 18500.0);  
   
 display\_car\_details(car\_ptr);  
   
 return 0;  
}

### 構造体配列とポインタ

構造体の配列とそのポインタ操作について学習します。

#### 構造体配列の使いどころ

構造体配列は、同じ種類のデータを複数管理する場合に便利です：

* **学生名簿**：100人の学生情報
* **商品在庫**：1000個の商品データ
* **座標点**：図形を構成する頂点の集まり

/\* 1人分 \*/  
struct Student student;  
  
/\* 100人分 \*/  
struct Student students[100];

#### 構造体配列の基本操作

#include <stdio.h>  
#include <string.h>  
  
struct Product {  
 int code;  
 char name[50];  
 double price;  
 int stock;  
};  
  
void print\_product(struct Product \*prod)  
{  
 printf("商品コード: %d\n", prod->code);  
 printf("商品名: %s\n", prod->name);  
 printf("価格: %.0f円\n", prod->price);  
 printf("在庫: %d個\n", prod->stock);  
 printf("---\n");  
}  
  
void print\_all\_products(struct Product products[], int count)  
{  
 int i;  
 printf("=== 全商品リスト ===\n");  
 for (i = 0; i < count; i++) {  
 print\_product(&products[i]); /\* 配列要素のアドレスを渡す \*/  
 }  
}  
  
int main(void)  
{  
 struct Product inventory[3];  
 int i;  
   
 /\* 商品データの初期化 \*/  
 inventory[0] = (struct Product){101, "ノートPC", 98000, 5};  
   
 strcpy(inventory[1].name, "マウス");  
 inventory[1].code = 102;  
 inventory[1].price = 2980;  
 inventory[1].stock = 20;  
   
 inventory[2].code = 103;  
 strcpy(inventory[2].name, "キーボード");  
 inventory[2].price = 4500;  
 inventory[2].stock = 15;  
   
 print\_all\_products(inventory, 3);  
   
 /\* ポインタを使った配列操作 \*/  
 printf("\n=== ポインタを使った操作 ===\n");  
 struct Product \*ptr = inventory; /\* 配列の先頭要素を指す \*/  
   
 for (i = 0; i < 3; i++) {  
 printf("%d番目: %s (%.0f円)\n",   
 i + 1, (ptr + i)->name, (ptr + i)->price);  
 }  
   
 return 0;  
}

#### ポインタ演算による構造体配列操作

#include <stdio.h>  
#include <string.h>  
  
struct Score {  
 char subject[30];  
 int points;  
 char grade;  
};  
  
/\* 成績を計算してグレードを設定 \*/  
void calculate\_grade(struct Score \*score)  
{  
 if (score->points >= 90) {  
 score->grade = 'A';  
 } else if (score->points >= 80) {  
 score->grade = 'B';  
 } else if (score->points >= 70) {  
 score->grade = 'C';  
 } else if (score->points >= 60) {  
 score->grade = 'D';  
 } else {  
 score->grade = 'F';  
 }  
}  
  
/\* 配列内の最高得点を見つける \*/  
struct Score\* find\_best\_score(struct Score scores[], int count)  
{  
 struct Score \*best = &scores[0];  
 int i;  
   
 for (i = 1; i < count; i++) {  
 if (scores[i].points > best->points) {  
 best = &scores[i];  
 }  
 }  
   
 return best;  
}  
  
int main(void)  
{  
 struct Score my\_scores[4];  
 struct Score \*ptr;  
 struct Score \*best;  
 int i;  
   
 /\* 成績データの入力 \*/  
 strcpy(my\_scores[0].subject, "数学");  
 my\_scores[0].points = 85;  
   
 strcpy(my\_scores[1].subject, "英語");  
 my\_scores[1].points = 92;  
   
 strcpy(my\_scores[2].subject, "理科");  
 my\_scores[2].points = 78;  
   
 strcpy(my\_scores[3].subject, "社会");  
 my\_scores[3].points = 88;  
   
 /\* 各科目のグレードを計算 \*/  
 ptr = my\_scores; /\* 配列の先頭を指すポインタ \*/  
 for (i = 0; i < 4; i++) {  
 calculate\_grade(ptr + i); /\* ポインタ演算でi番目の要素 \*/  
 }  
   
 /\* 結果表示 \*/  
 printf("=== 成績表 ===\n");  
 for (i = 0; i < 4; i++) {  
 printf("%s: %d点 (評価: %c)\n",   
 my\_scores[i].subject,   
 my\_scores[i].points,   
 my\_scores[i].grade);  
 }  
   
 /\* 最高得点の科目を見つける \*/  
 best = find\_best\_score(my\_scores, 4);  
 printf("\n最高得点: %s (%d点)\n", best->subject, best->points);  
   
 return 0;  
}

### 構造体ポインタ配列

ポインタの配列で複数の構造体を効率的に管理します。

#### 構造体ポインタ配列の基本

#include <stdio.h>  
#include <string.h>  
  
struct Person {  
 char name[50];  
 int age;  
 char city[30];  
};  
  
void print\_person\_info(struct Person \*person)  
{  
 printf("名前: %s, 年齢: %d歳, 居住地: %s\n",   
 person->name, person->age, person->city);  
}  
  
int main(void)  
{  
 /\* 個別の構造体変数 \*/  
 struct Person person1 = {"山田太郎", 25, "東京"};  
 struct Person person2 = {"佐藤花子", 30, "大阪"};  
 struct Person person3 = {"田中次郎", 28, "名古屋"};  
 struct Person person4 = {"鈴木美咲", 22, "福岡"};  
   
 /\* 構造体ポインタの配列 \*/  
 struct Person \*people[4];  
 int i;  
   
 /\* ポインタ配列に各構造体のアドレスを設定 \*/  
 people[0] = &person1;  
 people[1] = &person2;  
 people[2] = &person3;  
 people[3] = &person4;  
   
 printf("=== 全員の情報 ===\n");  
 for (i = 0; i < 4; i++) {  
 printf("%d. ", i + 1);  
 print\_person\_info(people[i]);  
 }  
   
 /\* 特定の条件で検索 \*/  
 printf("\n=== 25歳未満の人 ===\n");  
 for (i = 0; i < 4; i++) {  
 if (people[i]->age < 25) {  
 print\_person\_info(people[i]);  
 }  
 }  
   
 return 0;  
}

#### 動的な構造体ポインタ配列

#include <stdio.h>  
#include <string.h>  
  
struct Student {  
 int id;  
 char name[50];  
 double gpa;  
};  
  
/\* 学生データを作成する関数 \*/  
struct Student create\_student(int id, const char\* name, double gpa)  
{  
 struct Student student;  
 student.id = id;  
 strcpy(student.name, name);  
 student.gpa = gpa;  
 return student;  
}  
  
/\* GPA順でソート（バブルソート） \*/  
void sort\_students\_by\_gpa(struct Student \*students[], int count)  
{  
 int i, j;  
 struct Student \*temp;  
   
 for (i = 0; i < count - 1; i++) {  
 for (j = 0; j < count - 1 - i; j++) {  
 if (students[j]->gpa < students[j + 1]->gpa) {  
 temp = students[j];  
 students[j] = students[j + 1];  
 students[j + 1] = temp;  
 }  
 }  
 }  
}  
  
/\* 学生一覧を表示 \*/  
void display\_students(struct Student \*students[], int count)  
{  
 int i;  
 printf("ID\t名前\t\tGPA\n");  
 printf("------------------------\n");  
 for (i = 0; i < count; i++) {  
 printf("%d\t%s\t\t%.1f\n",   
 students[i]->id,   
 students[i]->name,   
 students[i]->gpa);  
 }  
 printf("\n");  
}  
  
int main(void)  
{  
 /\* 学生データの作成 \*/  
 struct Student student1 = create\_student(1001, "田中太郎", 3.2);  
 struct Student student2 = create\_student(1002, "佐藤花子", 3.8);  
 struct Student student3 = create\_student(1003, "山田次郎", 2.9);  
 struct Student student4 = create\_student(1004, "鈴木美咲", 3.5);  
 struct Student student5 = create\_student(1005, "高橋健太", 3.1);  
   
 /\* 構造体ポインタ配列 \*/  
 struct Student \*class\_roster[] = {  
 &student1, &student2, &student3, &student4, &student5  
 };  
   
 int class\_size = sizeof(class\_roster) / sizeof(class\_roster[0]);  
   
 printf("=== 入学順（元の順序） ===\n");  
 display\_students(class\_roster, class\_size);  
   
 /\* GPA順でソート \*/  
 sort\_students\_by\_gpa(class\_roster, class\_size);  
   
 printf("=== GPA順（高い順） ===\n");  
 display\_students(class\_roster, class\_size);  
   
 /\* 優秀な学生を見つける \*/  
 printf("=== GPA 3.5以上の学生 ===\n");  
 int i;  
 for (i = 0; i < class\_size; i++) {  
 if (class\_roster[i]->gpa >= 3.5) {  
 printf("%s (GPA: %.1f)\n",   
 class\_roster[i]->name,   
 class\_roster[i]->gpa);  
 }  
 }  
   
 return 0;  
}

### ネストした構造体とポインタ

構造体の中に他の構造体を含む場合のポインタ操作について学習します。

#### ネストした構造体の基本

#include <stdio.h>  
#include <string.h>  
  
struct Address {  
 char street[100];  
 char city[50];  
 char postal\_code[20];  
};  
  
struct Contact {  
 char phone[20];  
 char email[50];  
};  
  
struct Employee {  
 int id;  
 char name[50];  
 struct Address address; /\* ネストした構造体 \*/  
 struct Contact contact; /\* ネストした構造体 \*/  
 double salary;  
};  
  
void print\_employee\_details(struct Employee \*emp)  
{  
 printf("=== 社員詳細情報 ===\n");  
 printf("ID: %d\n", emp->id);  
 printf("名前: %s\n", emp->name);  
 printf("住所: %s, %s %s\n",   
 emp->address.street,   
 emp->address.city,   
 emp->address.postal\_code);  
 printf("電話: %s\n", emp->contact.phone);  
 printf("メール: %s\n", emp->contact.email);  
 printf("給与: %.0f円\n", emp->salary);  
 printf("\n");  
}  
  
void update\_address(struct Employee \*emp, const char\* street,   
 const char\* city, const char\* postal\_code)  
{  
 strcpy(emp->address.street, street);  
 strcpy(emp->address.city, city);  
 strcpy(emp->address.postal\_code, postal\_code);  
 printf("%s の住所を更新しました\n", emp->name);  
}  
  
int main(void)  
{  
 struct Employee emp;  
 struct Employee \*emp\_ptr = &emp;  
   
 /\* 基本情報の設定 \*/  
 emp\_ptr->id = 5001;  
 strcpy(emp\_ptr->name, "田中太郎");  
 emp\_ptr->salary = 450000.0;  
   
 /\* ネストした構造体メンバーの設定 \*/  
 strcpy(emp\_ptr->address.street, "新宿区西新宿1-2-3");  
 strcpy(emp\_ptr->address.city, "東京都");  
 strcpy(emp\_ptr->address.postal\_code, "160-0023");  
   
 strcpy(emp\_ptr->contact.phone, "03-1234-5678");  
 strcpy(emp\_ptr->contact.email, "tanaka@company.co.jp");  
   
 print\_employee\_details(emp\_ptr);  
   
 /\* 住所更新 \*/  
 update\_address(emp\_ptr, "渋谷区渋谷2-3-4", "東京都", "150-0002");  
   
 print\_employee\_details(emp\_ptr);  
   
 return 0;  
}

#### 構造体ポインタのネスト

#include <stdio.h>  
#include <string.h>  
  
struct Engine {  
 char type[50];  
 double displacement;  
 int horsepower;  
};  
  
struct Car {  
 char model[50];  
 int year;  
 struct Engine \*engine; /\* エンジン情報へのポインタ \*/  
 double price;  
};  
  
struct Dealership {  
 char name[50];  
 struct Car \*inventory; /\* 車の在庫配列へのポインタ \*/  
 int car\_count;  
};  
  
void print\_car\_info(struct Car \*car)  
{  
 printf("=== 車両情報 ===\n");  
 printf("モデル: %s (%d年)\n", car->model, car->year);  
 if (car->engine != NULL) {  
 printf("エンジン: %s %.1fL %dHP\n",   
 car->engine->type,   
 car->engine->displacement,   
 car->engine->horsepower);  
 }  
 printf("価格: %.0f万円\n", car->price);  
 printf("\n");  
}  
  
void print\_dealership\_inventory(struct Dealership \*dealer)  
{  
 int i;  
 printf("=== %s の在庫 ===\n", dealer->name);  
 for (i = 0; i < dealer->car\_count; i++) {  
 printf("%d. %s (%.0f万円)\n",   
 i + 1,   
 (dealer->inventory + i)->model,   
 (dealer->inventory + i)->price);  
 }  
 printf("\n");  
}  
  
int main(void)  
{  
 /\* エンジン情報 \*/  
 struct Engine engine1 = {"V6ガソリン", 3.5, 280};  
 struct Engine engine2 = {"直4ハイブリッド", 1.8, 120};  
 struct Engine engine3 = {"V8ガソリン", 5.0, 450};  
   
 /\* 車両情報 \*/  
 struct Car cars[3] = {  
 {"セダンLX", 2023, &engine1, 450},  
 {"ハイブリッドEC", 2024, &engine2, 380},  
 {"スポーツST", 2023, &engine3, 850}  
 };  
   
 /\* ディーラー情報 \*/  
 struct Dealership dealer = {  
 "トーキョー自動車",  
 cars,  
 3  
 };  
   
 /\* ディーラーの在庫一覧 \*/  
 print\_dealership\_inventory(&dealer);  
   
 /\* 各車両の詳細情報 \*/  
 int i;  
 for (i = 0; i < dealer.car\_count; i++) {  
 print\_car\_info(dealer.inventory + i);  
 }  
   
 return 0;  
}

### 構造体とメモリ管理

構造体のメモリレイアウトとアライメントについて理解します。

#### 構造体のメモリサイズとアライメント

#include <stdio.h>  
  
struct Example1 {  
 char a; /\* 1バイト \*/  
 int b; /\* 4バイト \*/  
 char c; /\* 1バイト \*/  
};  
  
struct Example2 {  
 char a; /\* 1バイト \*/  
 char c; /\* 1バイト \*/  
 int b; /\* 4バイト \*/  
};  
  
struct Example3 {  
 double d; /\* 8バイト \*/  
 char a; /\* 1バイト \*/  
 int b; /\* 4バイト \*/  
};  
  
void analyze\_struct\_memory(void)  
{  
 struct Example1 ex1;  
 struct Example2 ex2;  
 struct Example3 ex3;  
   
 printf("=== 構造体メモリ分析 ===\n");  
   
 printf("Example1 (char, int, char):\n");  
 printf(" sizeof: %lu バイト\n", sizeof(struct Example1));  
 printf(" a のオフセット: %lu\n", (unsigned long)&ex1.a - (unsigned long)&ex1);  
 printf(" b のオフセット: %lu\n", (unsigned long)&ex1.b - (unsigned long)&ex1);  
 printf(" c のオフセット: %lu\n", (unsigned long)&ex1.c - (unsigned long)&ex1);  
 printf("\n");  
   
 printf("Example2 (char, char, int):\n");  
 printf(" sizeof: %lu バイト\n", sizeof(struct Example2));  
 printf(" a のオフセット: %lu\n", (unsigned long)&ex2.a - (unsigned long)&ex2);  
 printf(" c のオフセット: %lu\n", (unsigned long)&ex2.c - (unsigned long)&ex2);  
 printf(" b のオフセット: %lu\n", (unsigned long)&ex2.b - (unsigned long)&ex2);  
 printf("\n");  
   
 printf("Example3 (double, char, int):\n");  
 printf(" sizeof: %lu バイト\n", sizeof(struct Example3));  
 printf(" d のオフセット: %lu\n", (unsigned long)&ex3.d - (unsigned long)&ex3);  
 printf(" a のオフセット: %lu\n", (unsigned long)&ex3.a - (unsigned long)&ex3);  
 printf(" b のオフセット: %lu\n", (unsigned long)&ex3.b - (unsigned long)&ex3);  
 printf("\n");  
}  
  
int main(void)  
{  
 analyze\_struct\_memory();  
   
 printf("=== 基本データ型のサイズ ===\n");  
 printf("char: %lu バイト\n", sizeof(char));  
 printf("int: %lu バイト\n", sizeof(int));  
 printf("double: %lu バイト\n", sizeof(double));  
 printf("ポインタ: %lu バイト\n", sizeof(void\*));  
   
 return 0;  
}

#### 構造体の配列とポインタの効率性

#include <stdio.h>  
#include <string.h>  
#include <time.h>  
  
struct LargeData {  
 char description[256];  
 double values[100];  
 int status;  
};  
  
/\* 値渡しで構造体を処理（非効率） \*/  
double process\_by\_value(struct LargeData data)  
{  
 int i;  
 double sum = 0.0;  
   
 for (i = 0; i < 100; i++) {  
 sum += data.values[i];  
 }  
   
 return sum / 100.0; /\* 平均値 \*/  
}  
  
/\* ポインタ渡しで構造体を処理（効率的） \*/  
double process\_by\_pointer(struct LargeData \*data)  
{  
 int i;  
 double sum = 0.0;  
   
 for (i = 0; i < 100; i++) {  
 sum += data->values[i];  
 }  
   
 return sum / 100.0; /\* 平均値 \*/  
}  
  
/\* 構造体にテストデータを設定 \*/  
void setup\_test\_data(struct LargeData \*data)  
{  
 int i;  
 strcpy(data->description, "テストデータセット");  
   
 for (i = 0; i < 100; i++) {  
 data->values[i] = (double)(i + 1) \* 1.5;  
 }  
   
 data->status = 1;  
}  
  
int main(void)  
{  
 struct LargeData test\_data;  
 clock\_t start, end;  
 double result;  
 int iterations = 10000;  
 int i;  
   
 setup\_test\_data(&test\_data);  
   
 printf("=== 構造体サイズ情報 ===\n");  
 printf("LargeData のサイズ: %lu バイト\n", sizeof(struct LargeData));  
 printf("テスト回数: %d 回\n\n", iterations);  
   
 /\* 値渡しのテスト \*/  
 printf("値渡しでのテスト開始...\n");  
 start = clock();  
 for (i = 0; i < iterations; i++) {  
 result = process\_by\_value(test\_data);  
 }  
 end = clock();  
 printf("値渡し結果: %.1f\n", result);  
 printf("実行時間: %.3f 秒\n\n", (double)(end - start) / CLOCKS\_PER\_SEC);  
   
 /\* ポインタ渡しのテスト \*/  
 printf("ポインタ渡しでのテスト開始...\n");  
 start = clock();  
 for (i = 0; i < iterations; i++) {  
 result = process\_by\_pointer(&test\_data);  
 }  
 end = clock();  
 printf("ポインタ渡し結果: %.1f\n", result);  
 printf("実行時間: %.3f 秒\n", (double)(end - start) / CLOCKS\_PER\_SEC);  
   
 return 0;  
}

### 実践的な構造体活用例

#### データベース風の学生管理システム

#include <stdio.h>  
#include <string.h>  
  
#define MAX\_STUDENTS 100  
#define MAX\_NAME\_LEN 50  
  
struct Student {  
 int id;  
 char name[MAX\_NAME\_LEN];  
 int age;  
 double gpa;  
 char major[30];  
};  
  
struct StudentDatabase {  
 struct Student \*students[MAX\_STUDENTS];  
 int count;  
};  
  
/\* 学生データベースの初期化 \*/  
void init\_database(struct StudentDatabase \*db)  
{  
 int i;  
 db->count = 0;  
 for (i = 0; i < MAX\_STUDENTS; i++) {  
 db->students[i] = NULL;  
 }  
}  
  
/\* 学生を追加 \*/  
int add\_student(struct StudentDatabase \*db, struct Student \*student)  
{  
 if (db->count >= MAX\_STUDENTS) {  
 printf("エラー: データベースが満杯です\n");  
 return 0;  
 }  
   
 db->students[db->count] = student;  
 db->count++;  
 printf("学生 %s を追加しました (ID: %d)\n", student->name, student->id);  
 return 1;  
}  
  
/\* IDで学生を検索 \*/  
struct Student\* find\_student\_by\_id(struct StudentDatabase \*db, int id)  
{  
 int i;  
 for (i = 0; i < db->count; i++) {  
 if (db->students[i]->id == id) {  
 return db->students[i];  
 }  
 }  
 return NULL;  
}  
  
/\* 専攻で学生を検索 \*/  
void find\_students\_by\_major(struct StudentDatabase \*db, const char\* major)  
{  
 int i;  
 int found = 0;  
   
 printf("=== %s専攻の学生 ===\n", major);  
 for (i = 0; i < db->count; i++) {  
 if (strcmp(db->students[i]->major, major) == 0) {  
 printf("ID: %d, 名前: %s, GPA: %.1f\n",   
 db->students[i]->id,  
 db->students[i]->name,  
 db->students[i]->gpa);  
 found = 1;  
 }  
 }  
   
 if (!found) {  
 printf("該当する学生が見つかりませんでした\n");  
 }  
 printf("\n");  
}  
  
/\* 全学生の一覧表示 \*/  
void display\_all\_students(struct StudentDatabase \*db)  
{  
 int i;  
   
 printf("=== 全学生一覧 (%d名) ===\n", db->count);  
 printf("ID\t名前\t\t年齢\tGPA\t専攻\n");  
 printf("------------------------------------------------\n");  
   
 for (i = 0; i < db->count; i++) {  
 printf("%d\t%s\t\t%d\t%.1f\t%s\n",  
 db->students[i]->id,  
 db->students[i]->name,  
 db->students[i]->age,  
 db->students[i]->gpa,  
 db->students[i]->major);  
 }  
 printf("\n");  
}  
  
/\* 平均GPAを計算 \*/  
double calculate\_average\_gpa(struct StudentDatabase \*db)  
{  
 double total = 0.0;  
 int i;  
   
 if (db->count == 0) return 0.0;  
   
 for (i = 0; i < db->count; i++) {  
 total += db->students[i]->gpa;  
 }  
   
 return total / db->count;  
}  
  
int main(void)  
{  
 struct StudentDatabase db;  
 init\_database(&db);  
   
 /\* 学生データの作成 \*/  
 struct Student student1 = {2001, "田中太郎", 20, 3.2, "コンピュータ"};  
 struct Student student2 = {2002, "佐藤花子", 21, 3.8, "数学"};  
 struct Student student3 = {2003, "山田次郎", 19, 2.9, "コンピュータ"};  
 struct Student student4 = {2004, "鈴木美咲", 20, 3.5, "物理"};  
 struct Student student5 = {2005, "高橋健太", 22, 3.1, "数学"};  
   
 /\* データベースに学生を追加 \*/  
 add\_student(&db, &student1);  
 add\_student(&db, &student2);  
 add\_student(&db, &student3);  
 add\_student(&db, &student4);  
 add\_student(&db, &student5);  
   
 /\* 全学生表示 \*/  
 display\_all\_students(&db);  
   
 /\* 特定IDの学生を検索 \*/  
 struct Student \*found = find\_student\_by\_id(&db, 2003);  
 if (found != NULL) {  
 printf("ID 2003 の学生: %s (%s専攻)\n\n", found->name, found->major);  
 }  
   
 /\* 専攻別検索 \*/  
 find\_students\_by\_major(&db, "コンピュータ");  
 find\_students\_by\_major(&db, "数学");  
   
 /\* 統計情報 \*/  
 printf("=== 統計情報 ===\n");  
 printf("登録学生数: %d名\n", db.count);  
 printf("平均GPA: %.2f\n", calculate\_average\_gpa(&db));  
   
 return 0;  
}

## サンプルコード

### 構造体の基本操作

プログラムファイル: examples/struct\_basics.c

構造体の定義、初期化、メンバーアクセスの基本を学習します。

### 構造体ポインタと->演算子

プログラムファイル: examples/struct\_pointers.c

構造体ポインタと->演算子の使用方法を学習します。

### 構造体配列の操作

プログラムファイル: examples/struct\_arrays.c

構造体配列とポインタを使ったデータ管理を学習します。

### コンパイルと実行

# examples ディレクトリに移動  
cd examples  
  
# C90準拠でコンパイル  
gcc -std=c90 -Wall -Wextra -pedantic struct\_basics.c -o struct\_basics  
  
# 実行  
./struct\_basics

## 演習課題

### 基礎問題

1. **学生情報管理**
   * 学生の情報（ID、名前、年齢、成績）を格納する構造体を定義し、ポインタを使って情報を表示・更新するプログラムを作成してください
2. **座標計算**
   * 2D座標を表す構造体を定義し、2点間の距離を計算する関数をポインタを使って実装してください
3. **商品管理**
   * 商品情報（コード、名前、価格、在庫）の構造体を作成し、構造体配列で複数商品を管理するプログラムを作成してください

### 応用問題

1. **従業員データベース**
   * 従業員情報と部署情報をネストした構造体で管理し、部署別の給与統計を算出するプログラムを作成してください
2. **図書管理システム**
   * 本の情報（タイトル、著者、出版年、貸出状況）を管理し、検索・貸出・返却機能を実装してください
3. **成績管理システム**
   * 学生と科目の構造体を使って、学生別・科目別の成績統計を管理するプログラムを作成してください

### 発展問題

1. **動的配列システム**
   * 構造体ポインタ配列を動的に拡張できるシステムを実装してください
2. **データソート**
   * 構造体ポインタ配列を複数の条件（名前、年齢、成績など）でソートできるプログラムを作成してください
3. **階層データ構造**
   * 会社組織（部署→チーム→従業員）のような階層構造を構造体とポインタで表現してください

## コンパイル方法

この章では以下のMakefileを使用してコンパイルができます：

# 全ての例題をコンパイル  
make all  
  
# 特定のプログラムをコンパイル  
make struct\_basics  
  
# 実行  
make run  
  
# クリーンアップ  
make clean

## 規格による違い

### C90での制限事項

* 構造体の初期化は順序通りのみ
* 可変長配列メンバーは不可
* 匿名構造体・共用体は不可

### C99以降の拡張

* 指定イニシャライザが使用可能
* 可変長配列メンバー（C99）
* 匿名構造体・共用体（C11）

## よくある間違い

### 1. NULLポインタの参照

/\* NG: NULLポインタの参照 \*/  
struct Student \*student = NULL;  
printf("%s\n", student->name); /\* セグメンテーション違反 \*/  
  
/\* OK: NULLチェック \*/  
if (student != NULL) {  
 printf("%s\n", student->name);  
}

### 2. スタック上の構造体アドレスの返却

/\* NG: ローカル構造体のアドレス返し \*/  
struct Point\* create\_point(int x, int y) {  
 struct Point p = {x, y};  
 return &p; /\* 危険: ローカル変数のアドレス \*/  
}  
  
/\* OK: 静的変数または呼び出し元で用意 \*/  
void create\_point(struct Point \*p, int x, int y) {  
 p->x = x;  
 p->y = y;  
}

### 3. 構造体の比較

/\* NG: 構造体の直接比較 \*/  
struct Point p1 = {10, 20};  
struct Point p2 = {10, 20};  
if (p1 == p2) { /\* コンパイルエラー \*/  
}  
  
/\* OK: メンバーごとの比較 \*/  
if (p1.x == p2.x && p1.y == p2.y) {  
 printf("同じ座標です\n");  
}

## 次の章へ

構造体を理解したら、[関数ポインタ](../function-pointers/README.md) に進んでください。

## 参考資料

* [C言語構造体リファレンス](https://ja.cppreference.com/w/c/language/struct)
* [メモリアライメントの詳細](https://ja.cppreference.com/w/c/language/object)
* [構造体の初期化](https://ja.cppreference.com/w/c/language/struct_initialization)

# サンプルコード

## struct\_arrays.c

/\*  
 \* struct\_arrays.c - 構造体配列の操作  
 \*   
 \* このプログラムでは以下を学習します：  
 \* - 構造体配列の宣言と初期化  
 \* - 構造体配列とポインタ演算  
 \* - 構造体ポインタ配列  
 \* - 実践的なデータ管理  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
  
#define MAX\_STUDENTS 50  
#define MAX\_BOOKS 100  
#define MAX\_NAME\_LEN 50  
  
/\* 学生情報を表す構造体 \*/  
struct Student {  
 int id;  
 char name[MAX\_NAME\_LEN];  
 double scores[5]; /\* 5教科の点数 \*/  
 double average;  
 char grade;  
};  
  
/\* 書籍情報を表す構造体 \*/  
struct Book {  
 int isbn;  
 char title[100];  
 char author[50];  
 int year;  
 int is\_borrowed;  
 int borrower\_id; /\* 借りている学生のID \*/  
};  
  
/\* 図書館を表す構造体 \*/  
struct Library {  
 struct Book books[MAX\_BOOKS];  
 int book\_count;  
 struct Student \*students[MAX\_STUDENTS];  
 int student\_count;  
};  
  
/\* 学生の平均点を計算する関数 \*/  
void calculate\_average(struct Student \*student)  
{  
 int i;  
 double sum = 0.0;  
   
 for (i = 0; i < 5; i++) {  
 sum += student->scores[i];  
 }  
   
 student->average = sum / 5.0;  
   
 /\* 成績評価 \*/  
 if (student->average >= 90.0) {  
 student->grade = 'A';  
 } else if (student->average >= 80.0) {  
 student->grade = 'B';  
 } else if (student->average >= 70.0) {  
 student->grade = 'C';  
 } else if (student->average >= 60.0) {  
 student->grade = 'D';  
 } else {  
 student->grade = 'F';  
 }  
}  
  
/\* 学生情報を表示する関数 \*/  
void print\_student(const struct Student \*student)  
{  
 int i;  
   
 printf("ID: %d, 名前: %s\n", student->id, student->name);  
 printf("点数: ");  
 for (i = 0; i < 5; i++) {  
 printf("%.0f ", student->scores[i]);  
 }  
 printf("\n平均: %.1f, 評価: %c\n", student->average, student->grade);  
 printf("---\n");  
}  
  
/\* 成績順にソート（バブルソート） \*/  
void sort\_students\_by\_average(struct Student students[], int count)  
{  
 int i, j;  
 struct Student temp;  
   
 for (i = 0; i < count - 1; i++) {  
 for (j = 0; j < count - 1 - i; j++) {  
 if (students[j].average < students[j + 1].average) {  
 temp = students[j];  
 students[j] = students[j + 1];  
 students[j + 1] = temp;  
 }  
 }  
 }  
}  
  
/\* 書籍を追加する関数 \*/  
int add\_book(struct Library \*lib, int isbn, const char \*title,   
 const char \*author, int year)  
{  
 if (lib->book\_count >= MAX\_BOOKS) {  
 return 0; /\* 追加失敗 \*/  
 }  
   
 struct Book \*book = &lib->books[lib->book\_count];  
 book->isbn = isbn;  
 strcpy(book->title, title);  
 strcpy(book->author, author);  
 book->year = year;  
 book->is\_borrowed = 0;  
 book->borrower\_id = -1;  
   
 lib->book\_count++;  
 return 1; /\* 追加成功 \*/  
}  
  
/\* 書籍を検索する関数 \*/  
struct Book\* find\_book\_by\_isbn(struct Library \*lib, int isbn)  
{  
 int i;  
 for (i = 0; i < lib->book\_count; i++) {  
 if (lib->books[i].isbn == isbn) {  
 return &lib->books[i];  
 }  
 }  
 return NULL;  
}  
  
/\* 書籍を借りる関数 \*/  
int borrow\_book(struct Library \*lib, int isbn, int student\_id)  
{  
 struct Book \*book = find\_book\_by\_isbn(lib, isbn);  
   
 if (book == NULL) {  
 printf("ISBN %d の書籍が見つかりません\n", isbn);  
 return 0;  
 }  
   
 if (book->is\_borrowed) {  
 printf("「%s」は既に貸出中です\n", book->title);  
 return 0;  
 }  
   
 book->is\_borrowed = 1;  
 book->borrower\_id = student\_id;  
 printf("ID %d の学生が「%s」を借りました\n", student\_id, book->title);  
 return 1;  
}  
  
/\* 学生の借りている本を表示 \*/  
void show\_borrowed\_books(struct Library \*lib, int student\_id)  
{  
 int i;  
 int count = 0;  
   
 printf("\nID %d の学生が借りている本:\n", student\_id);  
 for (i = 0; i < lib->book\_count; i++) {  
 if (lib->books[i].is\_borrowed &&   
 lib->books[i].borrower\_id == student\_id) {  
 printf("- %s (%s著, %d年)\n",   
 lib->books[i].title,   
 lib->books[i].author,   
 lib->books[i].year);  
 count++;  
 }  
 }  
   
 if (count == 0) {  
 printf(" 借りている本はありません\n");  
 }  
}  
  
int main(void)  
{  
 printf("===== 構造体配列の操作 =====\n\n");  
   
 /\* 1. 構造体配列の基本 \*/  
 printf("1. 構造体配列の基本\n");  
   
 struct Student students[5] = {  
 {1001, "田中太郎", {85, 90, 78, 92, 88}, 0, ' '},  
 {1002, "佐藤花子", {92, 88, 95, 90, 93}, 0, ' '},  
 {1003, "山田次郎", {75, 80, 72, 78, 70}, 0, ' '},  
 {1004, "鈴木美咲", {88, 85, 90, 87, 91}, 0, ' '},  
 {1005, "高橋健太", {60, 65, 58, 70, 62}, 0, ' '}  
 };  
   
 int student\_count = 5;  
 int i;  
   
 /\* 平均点の計算 \*/  
 for (i = 0; i < student\_count; i++) {  
 calculate\_average(&students[i]);  
 }  
   
 printf("=== 全学生の成績 ===\n");  
 for (i = 0; i < student\_count; i++) {  
 print\_student(&students[i]);  
 }  
 printf("\n");  
   
 /\* 2. 構造体配列のソート \*/  
 printf("2. 構造体配列のソート\n");  
   
 sort\_students\_by\_average(students, student\_count);  
   
 printf("=== 成績順（高い順） ===\n");  
 for (i = 0; i < student\_count; i++) {  
 printf("%d位: %s (平均: %.1f)\n",   
 i + 1, students[i].name, students[i].average);  
 }  
 printf("\n");  
   
 /\* 3. 構造体ポインタ配列 \*/  
 printf("3. 構造体ポインタ配列\n");  
   
 struct Student \*top\_students[3];  
 int top\_count = 0;  
   
 /\* 平均80点以上の学生を選抜 \*/  
 for (i = 0; i < student\_count; i++) {  
 if (students[i].average >= 80.0 && top\_count < 3) {  
 top\_students[top\_count++] = &students[i];  
 }  
 }  
   
 printf("=== 優秀学生（平均80点以上） ===\n");  
 for (i = 0; i < top\_count; i++) {  
 printf("%s (平均: %.1f, 評価: %c)\n",   
 top\_students[i]->name,   
 top\_students[i]->average,   
 top\_students[i]->grade);  
 }  
 printf("\n");  
   
 /\* 4. より複雑な構造体の管理 \*/  
 printf("4. 図書館システム\n");  
   
 struct Library library = {.book\_count = 0, .student\_count = 0};  
   
 /\* 書籍の追加 \*/  
 add\_book(&library, 1001, "プログラミング入門", "山田太郎", 2020);  
 add\_book(&library, 1002, "データ構造とアルゴリズム", "佐藤花子", 2019);  
 add\_book(&library, 1003, "オペレーティングシステム", "田中次郎", 2021);  
 add\_book(&library, 1004, "ネットワーク基礎", "鈴木一郎", 2022);  
 add\_book(&library, 1005, "データベース設計", "高橋美咲", 2020);  
   
 printf("図書館の蔵書数: %d冊\n\n", library.book\_count);  
   
 /\* 学生を図書館に登録 \*/  
 for (i = 0; i < 3; i++) {  
 library.students[i] = &students[i];  
 library.student\_count++;  
 }  
   
 /\* 本の貸出 \*/  
 borrow\_book(&library, 1001, students[0].id);  
 borrow\_book(&library, 1003, students[0].id);  
 borrow\_book(&library, 1002, students[1].id);  
 borrow\_book(&library, 1001, students[2].id); /\* 既に貸出中 \*/  
   
 /\* 各学生の借りている本を表示 \*/  
 for (i = 0; i < 3; i++) {  
 show\_borrowed\_books(&library, students[i].id);  
 }  
 printf("\n");  
   
 /\* 5. ポインタ演算による配列走査 \*/  
 printf("5. ポインタ演算による配列走査\n");  
   
 struct Book \*book\_ptr = library.books;  
 struct Book \*book\_end = library.books + library.book\_count;  
   
 printf("=== 貸出中の書籍一覧 ===\n");  
 while (book\_ptr < book\_end) {  
 if (book\_ptr->is\_borrowed) {  
 printf("「%s」 → 学生ID: %d\n",   
 book\_ptr->title, book\_ptr->borrower\_id);  
 }  
 book\_ptr++;  
 }  
   
 return 0;  
}  
  
/\*  
実行結果例:  
===== 構造体配列の操作 =====  
  
1. 構造体配列の基本  
=== 全学生の成績 ===  
ID: 1001, 名前: 田中太郎  
点数: 85 90 78 92 88   
平均: 86.6, 評価: B  
---  
ID: 1002, 名前: 佐藤花子  
点数: 92 88 95 90 93   
平均: 91.6, 評価: A  
---  
ID: 1003, 名前: 山田次郎  
点数: 75 80 72 78 70   
平均: 75.0, 評価: C  
---  
ID: 1004, 名前: 鈴木美咲  
点数: 88 85 90 87 91   
平均: 88.2, 評価: B  
---  
ID: 1005, 名前: 高橋健太  
点数: 60 65 58 70 62   
平均: 63.0, 評価: D  
---  
  
2. 構造体配列のソート  
=== 成績順（高い順） ===  
1位: 佐藤花子 (平均: 91.6)  
2位: 鈴木美咲 (平均: 88.2)  
3位: 田中太郎 (平均: 86.6)  
4位: 山田次郎 (平均: 75.0)  
5位: 高橋健太 (平均: 63.0)  
  
3. 構造体ポインタ配列  
=== 優秀学生（平均80点以上） ===  
佐藤花子 (平均: 91.6, 評価: A)  
鈴木美咲 (平均: 88.2, 評価: B)  
田中太郎 (平均: 86.6, 評価: B)  
  
4. 図書館システム  
図書館の蔵書数: 5冊  
  
ID 1001 の学生が「プログラミング入門」を借りました  
ID 1001 の学生が「オペレーティングシステム」を借りました  
ID 1002 の学生が「データ構造とアルゴリズム」を借りました  
「プログラミング入門」は既に貸出中です  
  
ID 1001 の学生が借りている本:  
- プログラミング入門 (山田太郎著, 2020年)  
- オペレーティングシステム (田中次郎著, 2021年)  
  
ID 1002 の学生が借りている本:  
- データ構造とアルゴリズム (佐藤花子著, 2019年)  
  
ID 1003 の学生が借りている本:  
 借りている本はありません  
  
5. ポインタ演算による配列走査  
=== 貸出中の書籍一覧 ===  
「プログラミング入門」 → 学生ID: 1001  
「データ構造とアルゴリズム」 → 学生ID: 1002  
「オペレーティングシステム」 → 学生ID: 1001  
\*/

## struct\_basics.c

/\*  
 \* struct\_basics.c - 構造体の基本操作  
 \*   
 \* このプログラムでは以下を学習します：  
 \* - 構造体の定義方法  
 \* - 構造体変数の宣言と初期化  
 \* - 構造体メンバーへのアクセス  
 \* - 構造体の関数への渡し方  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
  
/\* 座標を表す構造体 \*/  
struct Point {  
 int x;  
 int y;  
};  
  
/\* 学生情報を表す構造体 \*/  
struct Student {  
 int id;  
 char name[50];  
 int age;  
 double gpa;  
};  
  
/\* 商品情報を表す構造体 \*/  
struct Product {  
 int code;  
 char name[100];  
 double price;  
 int stock;  
};  
  
/\* 構造体を値渡しで受け取る関数 \*/  
void print\_point\_by\_value(struct Point p)  
{  
 printf("座標: (%d, %d)\n", p.x, p.y);  
}  
  
/\* 構造体をポインタ渡しで受け取る関数 \*/  
void print\_point\_by\_pointer(struct Point \*p)  
{  
 if (p != NULL) {  
 printf("座標: (%d, %d)\n", p->x, p->y);  
 }  
}  
  
/\* 学生情報を表示する関数 \*/  
void display\_student(struct Student s)  
{  
 printf("=== 学生情報 ===\n");  
 printf("ID: %d\n", s.id);  
 printf("名前: %s\n", s.name);  
 printf("年齢: %d歳\n", s.age);  
 printf("GPA: %.2f\n", s.gpa);  
 printf("\n");  
}  
  
/\* 商品情報を更新する関数（ポインタ使用） \*/  
void update\_stock(struct Product \*p, int quantity)  
{  
 if (p != NULL) {  
 p->stock += quantity;  
 if (quantity > 0) {  
 printf("%s の在庫を %d 個追加しました。現在の在庫: %d\n",   
 p->name, quantity, p->stock);  
 } else {  
 printf("%s の在庫を %d 個減らしました。現在の在庫: %d\n",   
 p->name, -quantity, p->stock);  
 }  
 }  
}  
  
int main(void)  
{  
 printf("===== 構造体の基本操作 =====\n\n");  
   
 /\* 1. 構造体の宣言と初期化 \*/  
 printf("1. 構造体の宣言と初期化\n");  
   
 /\* 方法1: 宣言後に個別に代入 \*/  
 struct Point p1;  
 p1.x = 10;  
 p1.y = 20;  
   
 /\* 方法2: 初期化リストを使用 \*/  
 struct Point p2 = {30, 40};  
   
 /\* 方法3: 部分的な初期化（残りは0） \*/  
 struct Point p3 = {50}; /\* y は 0 になる \*/  
   
 printf("p1: (%d, %d)\n", p1.x, p1.y);  
 printf("p2: (%d, %d)\n", p2.x, p2.y);  
 printf("p3: (%d, %d)\n", p3.x, p3.y);  
 printf("\n");  
   
 /\* 2. 構造体のコピー \*/  
 printf("2. 構造体のコピー\n");  
 struct Point p4 = p1; /\* 構造体全体のコピー \*/  
 p4.x = 100; /\* p4を変更してもp1は影響を受けない \*/  
   
 printf("p1（元）: (%d, %d)\n", p1.x, p1.y);  
 printf("p4（コピー後変更）: (%d, %d)\n", p4.x, p4.y);  
 printf("\n");  
   
 /\* 3. 構造体と関数 \*/  
 printf("3. 構造体と関数\n");  
 print\_point\_by\_value(p1); /\* 値渡し \*/  
 print\_point\_by\_pointer(&p2); /\* ポインタ渡し \*/  
 printf("\n");  
   
 /\* 4. 文字列を含む構造体 \*/  
 printf("4. 文字列を含む構造体\n");  
 struct Student student1;  
 student1.id = 1001;  
 strcpy(student1.name, "田中太郎");  
 student1.age = 20;  
 student1.gpa = 3.5;  
   
 display\_student(student1);  
   
 /\* 5. 構造体の配列 \*/  
 printf("5. 構造体の配列\n");  
 struct Product products[3] = {  
 {101, "ノートPC", 98000.0, 5},  
 {102, "マウス", 2980.0, 20},  
 {103, "キーボード", 4500.0, 15}  
 };  
   
 int i;  
 printf("=== 商品リスト ===\n");  
 for (i = 0; i < 3; i++) {  
 printf("商品コード: %d, 名前: %s, 価格: %.0f円, 在庫: %d\n",  
 products[i].code, products[i].name,   
 products[i].price, products[i].stock);  
 }  
 printf("\n");  
   
 /\* 6. 構造体ポインタの使用 \*/  
 printf("6. 構造体ポインタの使用\n");  
 struct Product \*prod\_ptr = &products[0];  
   
 /\* アロー演算子でメンバーアクセス \*/  
 printf("ポインタ経由でアクセス: %s (%.0f円)\n",   
 prod\_ptr->name, prod\_ptr->price);  
   
 /\* 在庫の更新 \*/  
 update\_stock(prod\_ptr, 3);  
 update\_stock(&products[1], -5);  
 printf("\n");  
   
 /\* 7. 構造体のサイズ \*/  
 printf("7. 構造体のサイズ\n");  
 printf("sizeof(struct Point): %lu バイト\n", sizeof(struct Point));  
 printf("sizeof(struct Student): %lu バイト\n", sizeof(struct Student));  
 printf("sizeof(struct Product): %lu バイト\n", sizeof(struct Product));  
 printf("\n");  
   
 /\* 8. ネストした構造体 \*/  
 printf("8. ネストした構造体\n");  
 struct Rectangle {  
 struct Point top\_left;  
 struct Point bottom\_right;  
 };  
   
 struct Rectangle rect = {{0, 0}, {100, 50}};  
 printf("矩形: 左上(%d, %d) - 右下(%d, %d)\n",  
 rect.top\_left.x, rect.top\_left.y,  
 rect.bottom\_right.x, rect.bottom\_right.y);  
   
 /\* 矩形の幅と高さを計算 \*/  
 int width = rect.bottom\_right.x - rect.top\_left.x;  
 int height = rect.bottom\_right.y - rect.top\_left.y;  
 printf("幅: %d, 高さ: %d, 面積: %d\n", width, height, width \* height);  
   
 return 0;  
}  
  
/\*  
実行結果例:  
===== 構造体の基本操作 =====  
  
1. 構造体の宣言と初期化  
p1: (10, 20)  
p2: (30, 40)  
p3: (50, 0)  
  
2. 構造体のコピー  
p1（元）: (10, 20)  
p4（コピー後変更）: (100, 20)  
  
3. 構造体と関数  
座標: (10, 20)  
座標: (30, 40)  
  
4. 文字列を含む構造体  
=== 学生情報 ===  
ID: 1001  
名前: 田中太郎  
年齢: 20歳  
GPA: 3.50  
  
5. 構造体の配列  
=== 商品リスト ===  
商品コード: 101, 名前: ノートPC, 価格: 98000円, 在庫: 5  
商品コード: 102, 名前: マウス, 価格: 2980円, 在庫: 20  
商品コード: 103, 名前: キーボード, 価格: 4500円, 在庫: 15  
  
6. 構造体ポインタの使用  
ポインタ経由でアクセス: ノートPC (98000円)  
ノートPC の在庫を 3 個追加しました。現在の在庫: 8  
マウス の在庫を 5 個減らしました。現在の在庫: 15  
  
7. 構造体のサイズ  
sizeof(struct Point): 8 バイト  
sizeof(struct Student): 72 バイト  
sizeof(struct Product): 120 バイト  
  
8. ネストした構造体  
矩形: 左上(0, 0) - 右下(100, 50)  
幅: 100, 高さ: 50, 面積: 5000  
\*/

## struct\_pointers.c

/\*  
 \* struct\_pointers.c - 構造体ポインタと->演算子  
 \*   
 \* このプログラムでは以下を学習します：  
 \* - 構造体ポインタの宣言と使用  
 \* - アロー演算子（->）の使い方  
 \* - 構造体ポインタと関数  
 \* - 効率的なデータ処理  
 \*/  
  
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
  
/\* 従業員情報を表す構造体 \*/  
struct Employee {  
 int id;  
 char name[50];  
 char department[30];  
 double salary;  
 int years\_of\_service;  
};  
  
/\* 日付を表す構造体 \*/  
struct Date {  
 int year;  
 int month;  
 int day;  
};  
  
/\* プロジェクト情報を表す構造体 \*/  
struct Project {  
 int id;  
 char name[100];  
 struct Date start\_date;  
 struct Date end\_date;  
 struct Employee \*manager; /\* 管理者へのポインタ \*/  
 double budget;  
};  
  
/\* 従業員情報を表示する関数 \*/  
void print\_employee(struct Employee \*emp)  
{  
 if (emp == NULL) {  
 printf("エラー: 無効な従業員データ\n");  
 return;  
 }  
   
 printf("=== 従業員情報 ===\n");  
 printf("ID: %d\n", emp->id);  
 printf("名前: %s\n", emp->name);  
 printf("部署: %s\n", emp->department);  
 printf("給与: %.0f円\n", emp->salary);  
 printf("勤続年数: %d年\n", emp->years\_of\_service);  
 printf("\n");  
}  
  
/\* 給与を更新する関数 \*/  
void update\_salary(struct Employee \*emp, double percentage)  
{  
 if (emp == NULL || percentage < -100) {  
 printf("エラー: 無効なパラメータ\n");  
 return;  
 }  
   
 double old\_salary = emp->salary;  
 emp->salary = old\_salary \* (1 + percentage / 100.0);  
   
 printf("%s の給与を更新しました\n", emp->name);  
 printf("変更前: %.0f円 → 変更後: %.0f円 (%.1f%%)\n",   
 old\_salary, emp->salary, percentage);  
}  
  
/\* 最高給与の従業員を見つける関数 \*/  
struct Employee\* find\_highest\_paid(struct Employee employees[], int count)  
{  
 if (employees == NULL || count <= 0) {  
 return NULL;  
 }  
   
 struct Employee \*highest = &employees[0];  
 int i;  
   
 for (i = 1; i < count; i++) {  
 if (employees[i].salary > highest->salary) {  
 highest = &employees[i];  
 }  
 }  
   
 return highest;  
}  
  
/\* プロジェクト情報を表示する関数 \*/  
void print\_project(struct Project \*proj)  
{  
 if (proj == NULL) {  
 printf("エラー: 無効なプロジェクトデータ\n");  
 return;  
 }  
   
 printf("=== プロジェクト情報 ===\n");  
 printf("ID: %d\n", proj->id);  
 printf("プロジェクト名: %s\n", proj->name);  
 printf("開始日: %d年%d月%d日\n",   
 proj->start\_date.year, proj->start\_date.month, proj->start\_date.day);  
 printf("終了日: %d年%d月%d日\n",   
 proj->end\_date.year, proj->end\_date.month, proj->end\_date.day);  
 printf("予算: %.0f円\n", proj->budget);  
   
 if (proj->manager != NULL) {  
 printf("管理者: %s (%s)\n",   
 proj->manager->name, proj->manager->department);  
 } else {  
 printf("管理者: 未定\n");  
 }  
 printf("\n");  
}  
  
/\* 部署別の統計を計算する関数 \*/  
void calculate\_department\_stats(struct Employee employees[], int count,   
 const char \*department)  
{  
 int dept\_count = 0;  
 double total\_salary = 0.0;  
 int total\_years = 0;  
 int i;  
   
 for (i = 0; i < count; i++) {  
 if (strcmp(employees[i].department, department) == 0) {  
 dept\_count++;  
 total\_salary += employees[i].salary;  
 total\_years += employees[i].years\_of\_service;  
 }  
 }  
   
 if (dept\_count > 0) {  
 printf("=== %s の統計 ===\n", department);  
 printf("人数: %d名\n", dept\_count);  
 printf("平均給与: %.0f円\n", total\_salary / dept\_count);  
 printf("平均勤続年数: %.1f年\n", (double)total\_years / dept\_count);  
 printf("\n");  
 } else {  
 printf("%s に該当する従業員はいません\n\n", department);  
 }  
}  
  
/\* 構造体のスワップ（ポインタのみ交換） \*/  
void swap\_employees(struct Employee \*\*emp1, struct Employee \*\*emp2)  
{  
 struct Employee \*temp = \*emp1;  
 \*emp1 = \*emp2;  
 \*emp2 = temp;  
}  
  
int main(void)  
{  
 printf("===== 構造体ポインタと->演算子 =====\n\n");  
   
 /\* 1. 構造体ポインタの基本 \*/  
 printf("1. 構造体ポインタの基本\n");  
   
 struct Employee emp1 = {  
 1001, "田中太郎", "開発部", 350000.0, 5  
 };  
   
 struct Employee \*emp\_ptr = &emp1;  
   
 /\* ドット演算子とアロー演算子の比較 \*/  
 printf("ドット演算子: %s\n", emp1.name);  
 printf("アロー演算子: %s\n", emp\_ptr->name);  
 printf("間接参照とドット: %s\n", (\*emp\_ptr).name);  
 printf("\n");  
   
 /\* 2. ポインタを使った関数呼び出し \*/  
 printf("2. ポインタを使った関数呼び出し\n");  
 print\_employee(emp\_ptr);  
   
 /\* 3. ポインタを通じた構造体の更新 \*/  
 printf("3. ポインタを通じた構造体の更新\n");  
 update\_salary(emp\_ptr, 10.0); /\* 10%昇給 \*/  
 printf("\n");  
   
 /\* 4. 構造体配列とポインタ \*/  
 printf("4. 構造体配列とポインタ\n");  
   
 struct Employee employees[] = {  
 {2001, "佐藤花子", "営業部", 300000.0, 3},  
 {2002, "山田次郎", "開発部", 400000.0, 7},  
 {2003, "鈴木美咲", "人事部", 320000.0, 4},  
 {2004, "高橋健太", "営業部", 280000.0, 2},  
 {2005, "伊藤良子", "開発部", 450000.0, 10}  
 };  
   
 int emp\_count = sizeof(employees) / sizeof(employees[0]);  
   
 /\* 最高給与の従業員を検索 \*/  
 struct Employee \*highest\_paid = find\_highest\_paid(employees, emp\_count);  
 if (highest\_paid != NULL) {  
 printf("最高給与の従業員:\n");  
 printf("名前: %s, 給与: %.0f円\n\n",   
 highest\_paid->name, highest\_paid->salary);  
 }  
   
 /\* 5. ポインタ配列 \*/  
 printf("5. ポインタ配列\n");  
   
 struct Employee \*dept\_members[5];  
 int dept\_count = 0;  
 int i;  
   
 /\* 開発部のメンバーを抽出 \*/  
 for (i = 0; i < emp\_count; i++) {  
 if (strcmp(employees[i].department, "開発部") == 0) {  
 dept\_members[dept\_count++] = &employees[i];  
 }  
 }  
   
 printf("開発部のメンバー:\n");  
 for (i = 0; i < dept\_count; i++) {  
 printf("- %s (ID: %d)\n",   
 dept\_members[i]->name, dept\_members[i]->id);  
 }  
 printf("\n");  
   
 /\* 6. ネストした構造体とポインタ \*/  
 printf("6. ネストした構造体とポインタ\n");  
   
 struct Project project1 = {  
 101,  
 "新システム開発",  
 {2024, 1, 1},  
 {2024, 12, 31},  
 &employees[1], /\* 山田次郎を管理者に設定 \*/  
 5000000.0  
 };  
   
 print\_project(&project1);  
   
 /\* 7. 部署別統計 \*/  
 printf("7. 部署別統計\n");  
 calculate\_department\_stats(employees, emp\_count, "開発部");  
 calculate\_department\_stats(employees, emp\_count, "営業部");  
   
 /\* 8. ポインタのスワップ \*/  
 printf("8. ポインタのスワップ\n");  
   
 struct Employee \*ptr1 = &employees[0];  
 struct Employee \*ptr2 = &employees[1];  
   
 printf("スワップ前:\n");  
 printf("ptr1 → %s\n", ptr1->name);  
 printf("ptr2 → %s\n", ptr2->name);  
   
 swap\_employees(&ptr1, &ptr2);  
   
 printf("\nスワップ後:\n");  
 printf("ptr1 → %s\n", ptr1->name);  
 printf("ptr2 → %s\n", ptr2->name);  
   
 return 0;  
}  
  
/\*  
実行結果例:  
===== 構造体ポインタと->演算子 =====  
  
1. 構造体ポインタの基本  
ドット演算子: 田中太郎  
アロー演算子: 田中太郎  
間接参照とドット: 田中太郎  
  
2. ポインタを使った関数呼び出し  
=== 従業員情報 ===  
ID: 1001  
名前: 田中太郎  
部署: 開発部  
給与: 350000円  
勤続年数: 5年  
  
3. ポインタを通じた構造体の更新  
田中太郎 の給与を更新しました  
変更前: 350000円 → 変更後: 385000円 (10.0%)  
  
4. 構造体配列とポインタ  
最高給与の従業員:  
名前: 伊藤良子, 給与: 450000円  
  
5. ポインタ配列  
開発部のメンバー:  
- 山田次郎 (ID: 2002)  
- 伊藤良子 (ID: 2005)  
  
6. ネストした構造体とポインタ  
=== プロジェクト情報 ===  
ID: 101  
プロジェクト名: 新システム開発  
開始日: 2024年1月1日  
終了日: 2024年12月31日  
予算: 5000000円  
管理者: 山田次郎 (開発部)  
  
7. 部署別統計  
=== 開発部 の統計 ===  
人数: 2名  
平均給与: 425000円  
平均勤続年数: 8.5年  
  
=== 営業部 の統計 ===  
人数: 2名  
平均給与: 290000円  
平均勤続年数: 2.5年  
  
8. ポインタのスワップ  
スワップ前:  
ptr1 → 佐藤花子  
ptr2 → 山田次郎  
  
スワップ後:  
ptr1 → 山田次郎  
ptr2 → 佐藤花子  
\*/