CS 137

Programming Principles

Chapter 8

Structures

Victoria Sakhnini

Table of Contents

Structures

Previously we learned about arrays in C, which group elements of the **same type** into one logical entity. C provides another tool — the **structure** — for grouping elements of **different kinds**. A structure consists of named member variables stored together in memory.

For example, suppose we wanted to model a clock with hours and minutes. We call this `tod` for "time of day":

```
struct tod {
    int hours;
    int minutes;
};
```

We can then create instances of the type:

```
struct tod now = {16, 50};
// now.hours == 16, now.minutes == 50
struct tod later = {.hours = 18};
// What is the value of later.minutes?
```

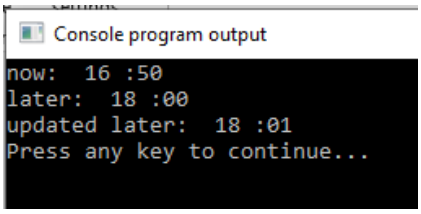
? What is the value of later.minutes when only .hours is initialized?

In C there is no such thing as "partial initialization". If a struct is initialized by specifying a value for a single member, all other members are automatically initialized to 0 of the proper kind. So later.minutes == 0.

Example

```
#include <stdio.h>
struct tod {
    int hours;
    int minutes;
};
void todPrint(struct tod when) {
    printf(" %0.2d:%0.2d\n", when.hours, when.minutes);
}

int main(void) {
    struct tod now   = { 16, 50 };
    struct tod later = { .hours = 18 };
    printf("now: ");
    todPrint(now);
    printf("later: ");
    todPrint(later);
    later.minutes = 1;
    printf("updated later: ");
    todPrint(later);
    return 0;
}
```



Function Returns a Structure

Functions can both accept and return structures. The function `todAddTime` below returns a value of type `struct tod`.

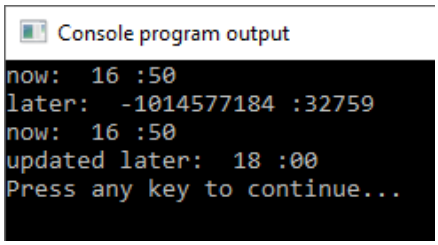
```
#include <stdio.h>

struct tod {
    int hours;
    int minutes;
};

void todPrint(struct tod when)
{
    printf(" %0.2d:%0.2d\n", when.hours, when.minutes);
}

struct tod todAddTime(struct tod when, int hours, int minutes)
{
    when.minutes += minutes;
    when.hours  += hours + when.minutes / 60;
    when.minutes %= 60;
    when.hours   %= 24;
    return when;
}

int main(void)
{
    struct tod now = { 16, 50 };
    struct tod later;
    printf("now: ");
    todPrint(now);
    printf("later: ");
    todPrint(later);  // later was NOT initialized — arbitrary values in
memory
    later = todAddTime(now, 1, 10);
    printf("now: ");
    todPrint(now);    // Structs are passed by value — a copy is made,
                      // so now is unchanged
    printf("updated later: ");
    todPrint(later);
    return 0;
}
```



📌 When passing structs to functions, values are passed by value — a copy is made. If you want to modify the original struct, you must pass a pointer to it and modify the contents through the pointer (more on this in the next chapter).

Using typedef to Simplify Struct Usage

To avoid writing `struct` every time, we can use `typedef`:

```c
#include <stdio.h>

typedef struct {
    int hours;
    int minutes;
} tod;

int main(void) {
    tod now = {14, 40};  // instead of: struct tod now = {14, 40};
    return 0;
}
```

Array of Structures

There is nothing new here syntactically — an array where each element is a structure.

```
#include <stdio.h>

struct student {
    int id;
    int assgn;
    int mid;
    int fexam;
    int fgrade;
};

void calc_fgrade(struct student a[], int n)
{
    for (int i = 0; i < n; i++)
        a[i].fgrade = 0.2 * a[i].assgn + 0.3 * a[i].mid + 0.5 * a[i].fexam;
}

int main(void)
{
    struct student cs137[5] = {
        {1111, 80, 88, 78},
        {2222, 77, 90, 81},
        {3333, 67, 66, 78},
        {4444, 90, 100, 98},
        {5555, 88, 77, 84}
    };
    calc_fgrade(cs137, 5);
    for (int i = 0; i < 5; i++)
        printf("student %d, final-grade=%d\n", cs137[i].id, cs137[i].fgrade);
    return 0;
}
```

Console program output

```
student 1111, final-grade=81
student 2222, final-grade=82
student 3333, final-grade=72
student 4444, final-grade=97
student 5555, final-grade=82
Press any key to continue...
```

cs137 is an array of length 5 where each element is of type struct student. So cs137[i] is a structure, and cs137[i].id accesses the id member of the i-th student. The array was initialized with data directly for simplicity.

Structures Containing Structures

Structures can have other structures as members. Again, there is nothing new here syntactically.

```
#include <stdio.h>

typedef struct {
    double x;
    double y;
} point;

typedef struct {
    point p1;
    point p2;
} line;

int main(void) {
    line ln1 = { {2.5, -6}, {-10.25, 17} };
    double a, b;
    a = (ln1.p1.y - ln1.p2.y) / (ln1.p1.x - ln1.p2.x);
    b = ln1.p1.y - a * ln1.p1.x;
    printf("y = %fx + %f\n", a, b);
    return 0;
}
```

ln1 represents a line defined by two points. ln1.p1 is of type point (as is ln1.p2). The output is the same line presented as y = ax + b.

Structures Containing Arrays

Structures can also contain arrays as members.

```
#include <stdio.h>

typedef struct {
    int id;
    int assgn[10];
    int mid;
    int fexam;
} student;

int main(void)
{
    student stud = { 11111,
                     {80, 90, 88, 99, 77, 78, 76, 89, 90, 100},
                     88, 78 };
    int fgrade, sum = 0;
    for (int i = 0; i < 10; i++)
        sum += stud.assgn[i];
    fgrade = 0.2 * sum / 10 + 0.3 * stud.mid + 0.5 * stud.fexam;
    printf("student %d, final-grade=%d\n", stud.id, fgrade);
    return 0;
}
```

stud.id is 11111. stud.assgn is an array of length 10. stud.mid is 88. stud.fexam is 78.

Additional Examples

Complex Number Arithmetic

The following program defines a `complex_t` structure and provides operations for addition, subtraction, multiplication, division, and absolute value. The `multiply_complex` and `divide_complex` functions are left for you to complete.

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

typedef struct {
    double real, imag;
} complex_t;

int       scan_complex    (complex_t *c);
void      print_complex   (complex_t c);
complex_t add_complex     (complex_t c1, complex_t c2);
complex_t subtract_complex(complex_t c1, complex_t c2);
complex_t multiply_complex(complex_t c1, complex_t c2);
complex_t divide_complex  (complex_t c1, complex_t c2);
complex_t abs_complex     (complex_t c);

int main(void)
{
    complex_t com1, com2;

    printf("Enter the real and imaginary parts of a complex number\n");
    printf("separated by a space> ");
    scan_complex(&com1);
    printf("Enter a second complex number> ");
    scan_complex(&com2);

    printf("\n");
    print_complex(com1);  printf(" + ");
    print_complex(com2);  printf(" = ");
    print_complex(add_complex(com1, com2));

    printf("\n\n");
    print_complex(com1);  printf(" - ");
    print_complex(com2);  printf(" = ");
    print_complex(subtract_complex(com1, com2));

    printf("\n\n|");
    print_complex(com1);  printf("| = ");
    print_complex(abs_complex(com1));
    printf("\n");

    return (0);
}

/* Returns 1 on valid scan, 0 on error, EOF on end of file */
int scan_complex(complex_t *c)
{
    int status;
```

```
    status = scanf("%lf%lf", &c->real, &c->imag);
    if (status == 2)      status = 1;
    else if (status != EOF) status = 0;
    return (status);
}

/* Displays value as (a + bi) or (a - bi); drops a or b if they round to 0 */
void print_complex(complex_t c)
{
    double a = c.real, b = c.imag;
    char sign;
    printf("(");
    if (fabs(a) < .005 && fabs(b) < .005)
        printf("%.2f", 0.0);
    else if (fabs(b) < .005)
        printf("%.2f", a);
    else if (fabs(a) < .005)
        printf("%.2fi", b);
    else {
        sign = (b < 0) ? '-' : '+';
        printf("%.2f %c %.2fi", a, sign, fabs(b));
    }
    printf(")");
}

complex_t add_complex(complex_t c1, complex_t c2)
{
    complex_t csum;
    csum.real = c1.real + c2.real;
    csum.imag = c1.imag + c2.imag;
    return (csum);
}

complex_t subtract_complex(complex_t c1, complex_t c2)
{
    complex_t cdiff;
    cdiff.real = c1.real - c2.real;
    cdiff.imag = c1.imag - c2.imag;
    return (cdiff);
}

complex_t multiply_complex(complex_t c1, complex_t c2)
{
    // try to complete this function and use it in main
}

complex_t divide_complex(complex_t c1, complex_t c2)
{
    // try to complete this function and use it in main
}

complex_t abs_complex(complex_t c)
{
    complex_t cabs;
    cabs.real = sqrt(c.real * c.real + c.imag * c.imag);
    cabs.imag = 0;
    return (cabs);
}
```

Extra Practice Problems

Consider the following definition for questions 1–3:

```
struct point {
    int x;
    int y;
};
```

1.

Write the C function scale_point that takes a struct point parameter p and an integer f, and returns a new struct point value equal to p with both coordinates multiplied by f.

2.

Write the C function distance that takes two struct point parameters p1 and p2, and returns the squared distance between p1 and p2.

3.

Write the C function find that reads pairs of integers (each pair representing a point) and returns the point with the largest distance from (0, 0). If there is more than one, return the one read last. Do not use float or double.

Test main for questions 1–3:

```
int main(void) {
    struct point p1 = {5, 10};
    struct point p2 = {0, 0};
    struct point p3 = {-2, 0};
    struct point pt;

    pt = scale_point(p1, 2);
    assert(pt.x == 10);
    assert(pt.y == 20);

    pt = scale_point(p3, -1);
    assert(pt.x == 2);
    assert(pt.y == 0);

    assert(distance(p1, p1) == 0);
    assert(distance(p1, p2) == 125);

    pt = find();  // as input enter: -2 0 5 10
    assert(pt.x == p1.x);
    assert(pt.y == p1.y);
    return 0;
}
```

4.

Write a function that takes an array of student structures and its length, and prints the id of every student who failed the midterm along with how much they need to score on the final to pass the course.

Marking scheme: assignments 30%, midterm 20%, final 50%.

```
struct student {
    int id;
    int assgn;
    int mid;
    int fexam;
    int fgrade;
};
```

5.

Complete the following inventory management program for a retail company with multiple store locations.

Your program must provide:
a) Add a new product (item ID, name, description, supplier, purchase price, selling price, quantity).
b) Update an existing product's purchase or selling price by item ID and store location.
c) Print all products at a specific store location.
d) Find and print the product with the highest profit margin (selling price - purchase price) across all locations.
e) Find and print the product with the lowest quantity in stock across all locations.
f) Switch between store locations.

```
#include <stdio.h>
#include <string.h>
#define MAX_PRODUCTS  500
#define MAX_LOCATIONS  10

struct Product {
    int   itemID;
    char  name[100];
    char  description[200];
    char  supplierName[100];
    float purchasePrice;
    float sellingPrice;
    int   quantity;
};

struct StoreLocation {
    char           locationName[50];
    struct Product products[MAX_PRODUCTS];
    int            productCount;
};

struct InventorySystem {
    struct StoreLocation locations[MAX_LOCATIONS];
    int locationCount;
    int currentLocation;
};
```

```
void addProduct(struct InventorySystem *inventory, int itemID, const char
*name,
                const char *description, const char *supplierName,
                float purchasePrice, float sellingPrice, int quantity) { }

void updateProductPrice(struct InventorySystem *inventory, int itemID,
                        float newPrice, int locationIndex, int
isPurchasePrice) { }

void listProductsInLocation(struct InventorySystem *inventory, int
locationIndex) { }

void findProductWithHighestProfitMargin(struct InventorySystem *inventory) { }

void findProductWithLowestQuantity(struct InventorySystem *inventory) { }

int main() {
    struct InventorySystem inventory;
    inventory.locationCount  = 0;
    inventory.currentLocation = 0;
    int choice;

    while (1) {
        printf("Current Location: %s\n",
               inventory.locations[inventory.currentLocation].locationName);
        printf("1. Add a new product\n");
        printf("2. Update purchase price of an existing product\n");
        printf("3. Update selling price of an existing product\n");
        printf("4. Print list of products in the current location\n");
        printf("5. Switch to a different location\n");
        printf("6. Find product with the highest profit margin\n");
        printf("7. Find product with the lowest quantity in stock\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1: {
            int itemID, quantity;
            float purchasePrice, sellingPrice;
            char name[100], description[200], supplierName[100];
            printf("Enter Item ID: ");        scanf("%d", &itemID);
            printf("Enter Name: ");           scanf("%s", name);
            printf("Enter Description: ");    scanf("%s", description);
            printf("Enter Supplier Name: ");  scanf("%s", supplierName);
            printf("Enter Purchase Price: "); scanf("%f", &purchasePrice);
            printf("Enter Selling Price: ");  scanf("%f", &sellingPrice);
            printf("Enter Quantity: ");       scanf("%d", &quantity);
            addProduct(&inventory, itemID, name, description,
                       supplierName, purchasePrice, sellingPrice, quantity);
            break;
        }
        case 2: {
            int updateID; float newPrice;
            printf("Enter Item ID to update: ");      scanf("%d", &updateID);
            printf("Enter new Purchase Price: ");     scanf("%f", &newPrice);
            updateProductPrice(&inventory, updateID, newPrice,
                               inventory.currentLocation, 1);
            break;
        }
```

```
        case 3: {
            int updateSellingID; float newSellingPrice;
            printf("Enter Item ID to update: ");    scanf("%d",
&updateSellingID);
            printf("Enter new Selling Price: ");    scanf("%f",
&newSellingPrice);
            updateProductPrice(&inventory, updateSellingID, newSellingPrice,
                               inventory.currentLocation, 0);
            break;
        }
        case 4:
            listProductsInLocation(&inventory, inventory.currentLocation);
            break;
        case 5: {
            int newLocation;
            printf("Enter the location index to switch to: ");
            scanf("%d", &newLocation);
            if (newLocation >= 0 && newLocation < inventory.locationCount)
                inventory.currentLocation = newLocation;
            else
                printf("Invalid location index.\n");
            break;
        }
        case 6: findProductWithHighestProfitMargin(&inventory); break;
        case 7: findProductWithLowestQuantity(&inventory);      break;
        case 8: return 0;
        default: printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

6.

Given the following Date structure, complete the three functions below.

```
struct Date {
    int day;
    int month;
    int year;
};
```

struct Date date_create(int d, int m, int y)
 Returns a Date struct with day d, month m, and year y.

bool date_eq(struct Date d, struct Date other)
 Returns true if and only if d and other represent exactly the same date.

struct Date add_days(struct Date d, int days)
 Returns the date that is exactly 'days' days after d.

Reminder: September, April, June, and November have 30 days. All the rest have 31, except February which has 28 (or 29 in a leap year). A helper is_leap_year function is provided.

```
int main() {
    struct Date d1 = date_create(1, 1, 2000);
    struct Date d2 = date_create(2, 1, 2000);
```

```
    assert(!date_eq(d1, d2));
    d1 = add_days(d1, 130);
    d2 = add_days(d2, 129);
    assert(date_eq(d1, d2));
}
```

7.

Create a C program to manage records for up to 100 students. Each student has an ID (int), exam score (float), and enrollment status (string: "enrolled" or "dropped").

Provide a menu-driven program with the following options:

1. Input student data
2. Display all student information
3. Calculate and display the average exam score of enrolled students
4. Find and display the student with the highest exam score
5. Count and display the number of enrolled students
6. Update the enrollment status of a specific student by ID
7. Exit

Use separate functions for each operation. Display appropriate error messages for invalid input or missing students.

8.

Given the struct point definition below, write a function:
bool colinear(struct point *points, size_t len)
that takes an array of points and its length and returns true if and only if all points lie on the same line; otherwise false.

```
struct point {
    int x;
    int y;
};
```