Mahidol University

ITCS113
Fundamentals of Programming

Lecture 13 - Struct

Instructor: Asst. Prof. Dr. Akara Supratak

Contact: akara.sup@mahidol.edu

Agenda

- Structure
- Structure declaration
- Access members of structure
- Structure initialization
- Structure copy
- Why Structure?
- Structure + Functions

What is Structure?

Structure: a user-defined data type in C/C++.

A structure creates a data type that can be used to group items of possibly different types into a single type.

```
struct struct_name
{
  datatype var_name1;
  datatype *var_name2;
  datatype var_name3[size];
  ...
};
```

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

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

Structure Declaration

Below is how to create a variable of the structure:

```
struct struct_name
{
  datatype var_name1;
  datatype *var_name2;
  datatype var_name3[size];
  ...
};
```

```
struct struct_name var_name;
```

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

```
struct Point
{
  int x;
  int y;
};
struct Point p1, p2;
```

Access Members of Structure

To access any variable of a structure, we use the member access operator "." (i.e., period).

```
struct struct_name
{
  datatype var_name1;
  datatype *var_name2;
  datatype var_name3[size];
  ...
};
```

```
struct struct_name var_name;
```

```
var_name.var_name1 = ...
var_name.var_name2 = ...
var_name.var_name3 = ...
```

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

struct Date birth_day;

birth_day.month = 12;
birth_day.day = 29;
birth_day.year = 2018;

printf("%d", birth_day.month);
printf("%d", birth_day.day);
printf("%d", birth_day.year);
```

Access Members of Structure

To access any variable of a structure, we use the member access operator "." (i.e., period).

```
struct struct_name
{
  datatype var_name1;
  datatype *var_name2;
  datatype var_name3[size];
  ...
};
```

```
struct struct_name var_name;
```

```
var_name.var_name1 = ...
var_name.var_name2 = ...
var_name.var_name3 = ...
```

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

struct Date birth_day;

birth_day.month = 12;
birth_day.day = 29;
birth_day.year = 2018;

printf("%d", birth_day.month);
printf("%d", birth_day.day);
printf("%d", birth_day.year);
```

Structure Initialization

Structure members can be initialized using curly braces '{ }'.

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

struct Date birth_day;

birth_day.month = 12;
birth_day.day = 29;
birth_day.year = 2018;

printf("%d", birth_day.month);
printf("%d", birth_day.day);
printf("%d", birth_day.year);
```

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

struct Date birth_day = {12, 29, 2018};

printf("%d", birth_day.month);
printf("%d", birth_day.day);
printf("%d", birth_day.year);
```

Example

```
#include <stdio.h>

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

int main()
{
    struct Date d1 = {1, 1, 1999}; // initialization

    // direct member access
    struct Date d2, d3;
    d2.day = 25;
    d2.month = 12;
    d2.year = 2019;
    scanf("%d", &d3.day);
    scanf("%d", &d3.month);
    scanf("%d", &d3.year);

    printf("%d-%d-%d\n", d1.day, d1.month, d1.year);
    printf("%d-%d-%d\n", d2.day, d2.month, d2.year);
    printf("%d-%d-%d\n", d3.day, d3.month, d3.year);

    return 0;
}
```

Exercise: Design the following `struct`

Create a structure for Book including book id, title, author, and number of pages *note that suppose the title and author can have at most 50 characters

Struct Declaration

Struct Initialization

Example: `struct Book`

Structure Copy

Assignment operator '=' copies all member values to another structure variable.

```
#include <stdio.h>
#include <string.h>
struct Account{
   int id;
   char name[20];
   float amount;
};
int main() {
   struct Account acc1 = {1, "Akara", 178.7};
   struct Account acc2;
   acc2 = acc1; // copy values from 'acc1' to 'acc2'
   acc2.id = 2;
   acc2.amount -= 100;
   acc2.name[strlen(acc2.name)-1] = 'A';
   printf("%d %s %.2f\n", acc1.id, acc1.name, acc1.amount);
   printf("%d %s %.2f\n", acc2.id, acc2.name, acc2.amount);
   return 0;
}
```

Structure Copy

Assignment operator '=' copies all member values to another structure variable.

```
1 Akara 178.70
2 AkarA 78.70
```

```c
#include <stdio.h>
#include <string.h>
struct Account{
   int id;
   char name[20];
   float amount;
};
int main() {
   struct Account acc1 = {1, "Akara", 178.7};
   struct Account acc2;
   acc2 = acc1; // copy values from 'acc1' to 'acc2'
   acc2.id = 2;
   acc2.amount -= 100;
   acc2.name[strlen(acc2.name)-1] = 'A';
   printf("%d %s %.2f\n", acc1.id, acc1.name, acc1.amount);
   printf("%d %s %.2f\n", acc2.id, acc2.name, acc2.amount);
   return 0;
}
```

Exercise: What is the output?

```
#include <stdio.h>

struct Point{
    int x;
    int y;
};

int main() {
    struct Point p1;
    p1.x = 1;
    p1.y = 1;
    struct Point p2;
    p2 = p1;
    p2.x +=3;
    p2.y = p1.y-4;
    printf("P1(%d, %d) ", p1.x, p1.y);
    printf("P2(%d, %d) ", p2.x, p2.y);
}
```

Why Structure?

- Group various data in one place
- Easy to access (e.g., `acc1.amount`, `acc1.name`, etc.)
- Easy to manage (e.g., copy, pass to a function, etc.)
- Array of structures

`struct` Account

```
int id;
char name[20];
float amount;
```

Account: `acc1`

```
acc1.id = 1;
strcpy(acc1.name,"Peter");
acc1.amount = 10.50;
```

Account: `acc2`

```
acc2.id = 2;
strcpy(acc2.name,"Mary");
acc2.amount = 1112.99;
```

Why Structure?

For example, in a vending machine program, each **item** has a number of attributes associated with it (e.g., ID, name, quantity, price, etc.)

What if we want to increase from 10 to 100 items?

(Without Structure)

```
#include <stdio.h>

int main(){
    int item_id[10] = ...;
    char item_name[10][20] = ...;
    int item_qty[10] = ...;
    float item_price[10] = ...;

    int i;
    for (i=0 ; i<10 ; i++) {
        printf("ID: %d\n", item_id[i]);
        printf("Name: %s\n",
                    item_name[i]);
        printf("Qty: %d\n",
                    item_qty[i]);
        printf("Price: %.2f\n",
                    item_price[i]);
    }

    return 0;
}
```

Why Structure?

Rewrite with struct

Each item consists of id, name, quantity, and price: group in one unit

(With Structure)

```
#include <stdio.h>

struct Item {
    int id;
    char name[20];
    int qty;
    float price;
};
int main() {
    struct Item items[10] = ...;
    for (int i=0 ; i<10 ; i++) {
        printf("ID: %d\n",
                    items[i].id);
        printf("Name: %s\n",
                    items[i].name);
        printf("Qty: %d\n",
                    items[i].qty);
        printf("Price: %.2f\n",
                    items[i].price);
    }
    return 0;
}
```

Using a `struct` in a function

- `struct` can be used as a parameter in a function, similar to other variables
- `struct` can also be used as a output (return) from a function

Using a `struct` in a function

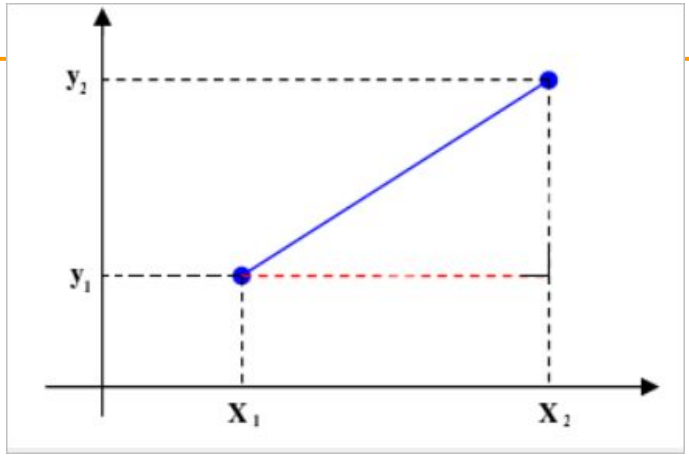
`struct` can be used as a parameter in a function, similar to other variables

Using a `struct` in a function

`struct` can also be used as a output (return) from a function

Example: `struct` as parameters

Write a function to calculate a **Euclidean distance** of two points. Use `struct` to store a point `(x,y)`




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

struct Point{
    float x;
    float y;
};


float calEuclidDist(struct Point p1,struct Point p2){
    float d = sqrt(pow((p2.x - p1.x),2) + pow((p2.y - p1.y),2));
    return d;
}
```

Example: `struct` as parameters

Write a function to calculate a **Euclidean distance** of two points. Use `struct` to store a point `(x,y)`

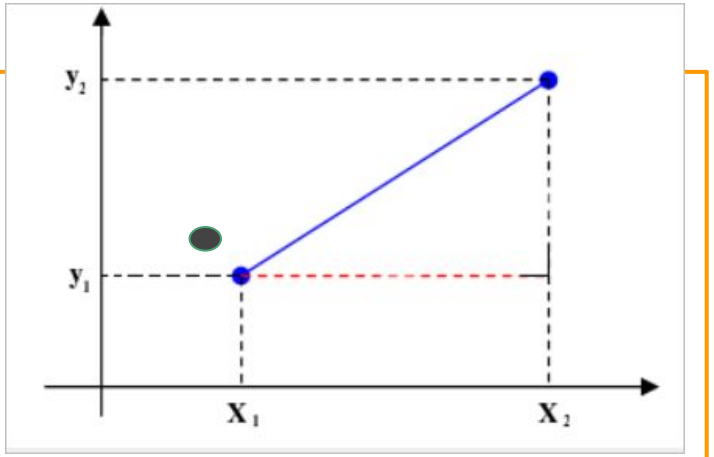
```
struct Point{
    float x;
    float y;
};
```

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

struct Point{...}; // Continue from a previous slide
float calEuclidDist(struct Point p1,struct Point p2){
 …
}
int main(){
    struct Point myP1, myP2;
    scanf("%f %f", &myP1.x, &myP1.y);
    scanf("%f %f", &myP2.x, &myP2.y);
    float dist = calEuclidDist(myP1, myP2);
    printf("Distance: %.2f\n", dist);
    return 0;
}
```

Example: `struct` as a return value

Write a function to calculate a **middle point** of two points. Use `struct` to store a point `(x,y)`




```
#include <stdio.h>


struct Point{
    float x;
    float y;
};


struct Point calMidPoint(struct Point p1,struct Point p2){
    struct Point midP;
    midP.x = (p1.x+p2.x)/2.0;
    midP.y = (p1.y+p2.y)/2.0;
    return midP;
}
```

Example: `struct` as a return value (cont.)

Write a function to calculate a **middle point** of two points. Use `struct` to store a point `(x,y)`

```
#include <stdio.h>
struct Point{...};
struct Point calMidPoint(struct Point p1,struct Point p2){
    ...
    return midP;
}
int main(){
    struct Point myP1, myP2;
    scanf("%f %f", &myP1.x, &myP1.y);
    scanf("%f %f", &myP2.x, &myP2.y);
    struct Point midPoint = calMidPoint(myP1, myP2);
    printf("Mid Point: %.2f %.2f\n", midPoint.x,
                                     midPoint.y);
    return 0;
}
```

Lab Exercises