CS1010 Programming Methodology

http://www.comp.nus.edu.sg/~cs1010/

Unit 18

More about Structures

NUS National University of Singapore | School of Computing

Unit 18: More about Structures

Objectives:

- Learn how to create and use structures with strings
- Learn how to pass structures to functions
- Learn how to use an array of structures

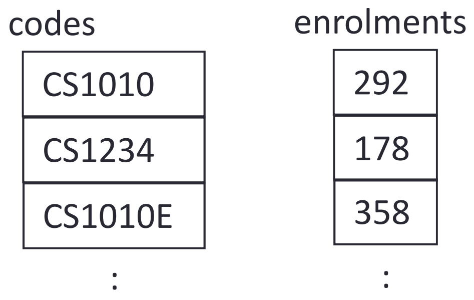
Reference:

- Chapter 10 Structure and Union Types

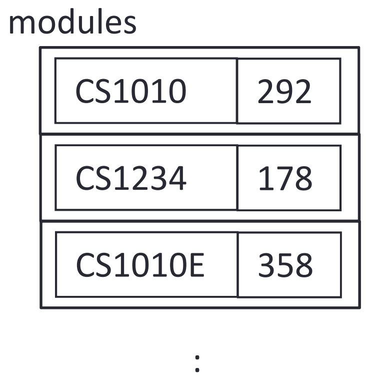
Unit 18: More about Structures (1/2)

1. Organizing Data (4/4)

- We can also create array of *groups*
- Example: codes and enrolments for modules can be stored
  - Using two parallel arrays
    - codes[*i*] and enrolments[*i*] are related to the same module *i*



  - Using an array of “module” *group*



- Which is more logical?

1. Structures with Strings

- Besides the primitive data types, structures may include Strings (Unit #16) as well.
- Examples of structure types:

```
typedef struct {
   char code[8];
   int  enrolment;
} module_t;
```

```
typedef struct {
   char name[12];
   int  age;
   char gender;
} player_t;
```

1.1 Initializing Structures with Strings (1/2)

```
typedef struct {
   char name[12];
   int  age;
   char gender;
} player_t;
```

- Using initializers:

```
player_t player1 = { "Brusco", 23, 'M' };
```

- Using string functions:

```
player_t player2;

strcpy(player2.name, "July");
player2.age = 21;
player2.gender = 'F';
```

1.1 Initializing Structures with Strings (2/2)

```
typedef struct {
   char name[12];
   int  age;
   char gender;
} player_t;
```

- Using scanf():

```
player_t player3;

printf("Enter name, age and gender: ");
scanf("%s %d %c", player3.name,
      &player3.age, &player3.gender);
```

Why is there no need for & to read in name?

- Using assignment:

```
player_t player4 = player3;
```

=

```
strcpy(player4.name, player3.name);
player4.age = player3.age;
player4.gender = player3.gender;
```

2. Passing Structures to Functions

- Passing a structure to a parameter in a function is akin to assigning the structure to the parameter.
- The entire structure is copied, i.e., members of the actual parameter are copied into the corresponding members of the formal parameter.
- We use Unit18_Demo1.c to illustrate this.

```
player1: name = Brusco; age = 23; gender = M
player2: name = July; age = 21; gender = F
```

2. Demo #1: Passing Structures to Functions

Unit18_Demo1.c

```
// #include statements and definition
// of player_t are omitted here for brevity
void print_player(char [], player_t);

int main(void) {
    player_t player1 = { "Brusco", 23, 'M' }, player2;

    strcpy(player2.name, "July");
    player2.age = 21;
    player2.gender = 'F';

    print_player("player1", player1);
    print_player("player2", player2);

    return 0;
}

// Print player's information
void print_player(char header[], player_t player) {
    printf("%s: name = %s; age = %d; gender = %c\n", header,
           player.name, player.age, player.gender);
}
```

Passing a structure to a function

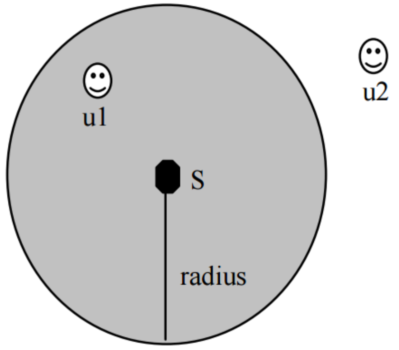
Receiving a structure from the caller

3. Array of Structures

- Combining structures and arrays gives us a lot of flexibility in organizing data.
  - For example, we may have a structure comprising 2 members: student's name and an array of 5 test scores he obtained.
  - Or, we may have an array whose elements are structures.
  - Or, even more complex combinations such as an array whose elements are structures which comprises array as one of the members.
- Case study:
  - A startup company decides to provide location-based services. Its customers are a list of stores.
  - Each store has a name, a location given by (x, y) coordinates, a radius that defines a circle of influence.
  - We can define a structure type store_t for the stores, and have a store_t array store_t variables. We call this array storeList and it represents the list of stores.

3. Case Study: Nearby Stores (1/4)

- Given a user's current location (x, y) and a list of stores, write a program that prints the names of stores where the user's current location is within their circles of influence.
- The diagram on the left shows the circle of influence of a store S. User u1 is within S's circle of influence while u2 is not.



- Sample run:

```
Enter number of stores: 5
Enter store information:
ABC_Store 3 4 5.0
Cheerful 1 1 3.0
Old_Papa 5 6 10.0
Seven_11 2 2 2.0
Lowson 4 1 2.5
Enter user location: 5 2
```

```
The nearby stores are:
ABC_Store
Old_Papa
Lowson
```

3. Case Study: Nearby Stores (2/4)

Unit18_NearbyStores.c

```
// Preprocessor directives and
// function prototypes omitted for brevity

typedef struct {
   char  sname[13];
   int   x, y;
   float radius;
} store_t;

int main(void){
   store_t storeList[MAX_STORES];
   int numStore, userX, userY;

   numStore = readStores(storeList);

   printf("Enter user location: ");
   scanf("%d %d", &userX, &userY);

   printNearbyStores(userX, userY, storeList, numStore);

   return 0;
}
```

3. Case Study: Nearby Stores (3/4)

Unit18_NearbyStores.c

```
int readStores(store_t storeList[]) {
   int i, numStore;

   printf("Enter number of stores: ");
   scanf("%d", &numStore);

   printf("Enter store information:\n");

   for (i=0; i<numStore; i++)
      scanf("%s %d %d %f", storeList[i].sname,
                            &storeList[i].x, &storeList[i].y,
                            &storeList[i].radius);

   return numStore;
}
```

3. Case Study: Nearby Stores (4/4)

Unit18_NearbyStores.c

```
int withinRadius(int x, int y, store_t store) {
   float distance = sqrt((store.x - x)*(store.x - x) +
                         (store.y - y)*(store.y - y));

   return (distance < store.radius);
}

void printNearbyStores(int x, int y,
                       store_t storeList[], int numStore) {
   int i;

   printf("The nearby stores are:\n");

   for (i=0; i<numStore; i++)
      if (withinRadius(x, y, storeList[i]))
         printf("%s\n", storeList[i].sname);
}
```

4. Passing Address of Structure to Functions (1/5)

- Given this code, what is the output?

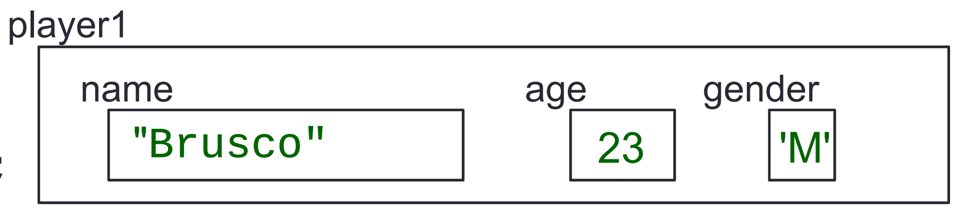
Unit18_Demo2.c

```
// #include statements, definition of player_t,
// and function prototypes are omitted here for brevity
int main(void) {
   player_t player1 = { "Brusco", 23, 'M' };

   change_name_and_age(player1);
   print_player("player1", player1);
   return 0;
}

// To change a player’s name and age
void change_name_and_age(player_t player) {
   strcpy(player.name, "Alexandra");
   player.age = 25;
}

// Print player’s information
void print_player(char header[], player_t player) {
   printf("%s: name = %s; age = %d; gender = %c\n", header,
          player.name, player.age, player.gender);
}
```

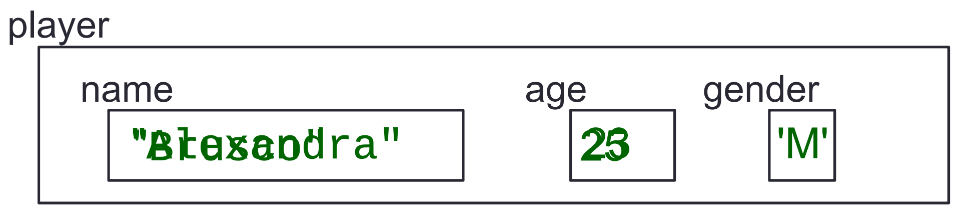
```
player1: name = Brusco; age = 23; gender = M
```

4. Passing Address of Structure to Functions (2/5)

```
main()

change_name_and_age(player1);
```



```
change_name_and_age(player_t player)
```



```
strcpy(player.name, "Alexandra");
player.age = 25;
```

4. Passing Address of Structure to Functions (3/5)

- Like an ordinary variable (eg: of type int, char), when a structure variable is passed to a function, a separate copy of it is made in the called function.
- Hence, the original structure variable will not be modified by the function.
- To allow the function to modify the content of the original structure variable, you need to pass in the address (pointer) of the structure variable to the function.
- (Note that passing an array of structures to a function is a different matter. As the array name is a pointer, the function is able to modify the array elements.)

4. Passing Address of Structure to Functions (4/5)

- Need to pass address of the structure variable

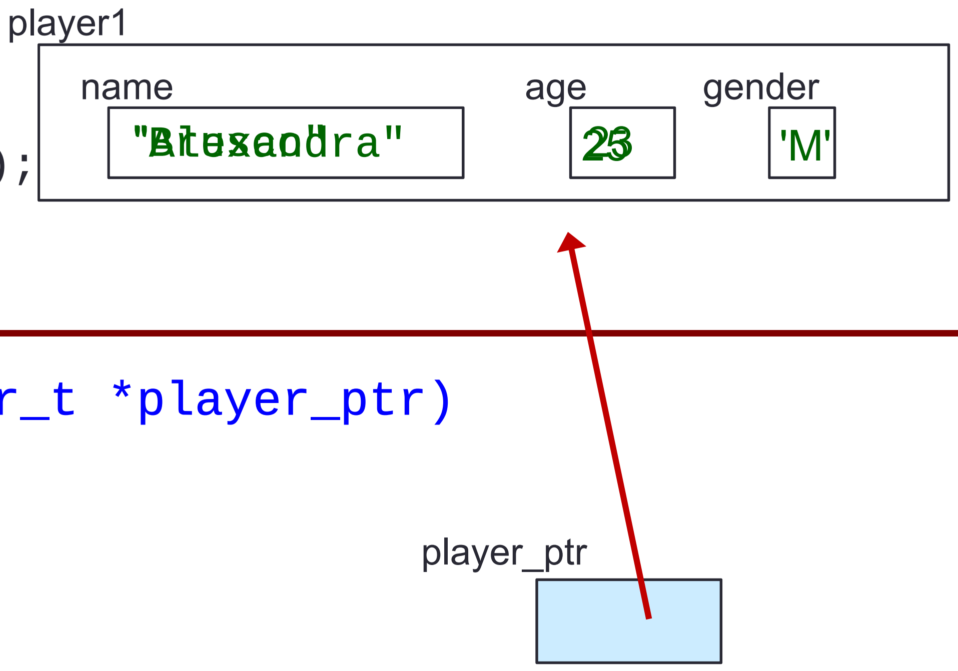
Unit18_Demo3.c

```
// #include statements, definition of player_t,
// and function prototypes are omitted here for brevity
int main(void) {
   player_t player1 = { "Brusco", 23, 'M' };

   change_name_and_age(&player1);
   print_player("player1", player1);
   return 0;
}

// To change a player's name and age
void change_name_and_age(player_t *player_ptr) {
   strcpy((*player_ptr).name, "Alexandra");
   (*player_ptr).age = 25;
}

// Print player's information
void print_player(char header[], player_t player) {
   printf("%s: name = %s; age = %d; gender = %c\n", header,
          player.name, player.age, player.gender);
}
```

```
player1: name = Alexandra; age = 25; gender = M
```

4. Passing Address of Structure to Functions (5/5)

```
main()

change_name_and_age(&player1);
```



```
change_name_and_age(player_t *player_ptr)

strcpy((*player_ptr).name, "Alexandra");
(*player_ptr).age = 25;
```

5. The Arrow Operator (->) (1/2)

- Expressions like `(*player_ptr).name` appear very often. Hence an alternative "shortcut" syntax is created for it.
- The arrow operator (->)

`(*player_ptr).name`	*is equivalent to*	`player_ptr->name`
`(*player_ptr).age`	*is equivalent to*	`player_ptr->age`

- Can we write *player_ptr.name instead of (*player_ptr).name?
- *No*, because . (dot) has higher precedence than *, so *player_ptr.name means *(player_ptr.name)!

5. The Arrow Operator (->) (2/2)

- Function change_name_and_age() in Unit18_Demo4.c modified to use the -> operator.

Unit18_Demo4.c

```c
// To change a player's name and age
void change_name_and_age(player_t *player_ptr) {
   strcpy(player_ptr->name, "Alexandra");
   player_ptr->age = 25;
}
```

6. Returning Structure from Functions

- As mentioned in Unit 15, a function can return a structure
  - Example: Define a function func() that returns a structure of type player_t:

```
player_t func( ... ) {
   ...

}
```

  - To call func():

```
player_t player3;

player3 = func( ... );
```

6. Returning Structure from Functions

Unit18_Demo5.c

```
int main(void){
   player_t player1, player2;

   printf("Enter player 1's particulars:\n");
   player1 = scan_player();
   printf("Enter player 2's particulars:\n");
   player2 = scan_player();
   . . .
   return 0;
}

// To read in particulars of a player and return structure to caller
player_t scan_player() {
   player_t player;

   printf("Enter name, age and gender: ");
   scanf("%s %d %c", player.name, &player.age, &player.gender);

   return player;
}
```

returned structure is copied to player1

variable player temporarily stores the user's inputs

player is returned here

7. Exercise #1: Points (1/5)

- Write a program Unit_Points.c that includes
  1. a structure type point_t whose members are the *x*- and *y*-coordinates of a point. The coordinates are integers.
  2. a function read_points() to read the number of points and points' data into an array of points, and return the number of points read. Each point is represented by its *x*- and *y*-coordinates.

- You may assume that the input data contain at least 1 point and at most 10 points.

- An example of input data of 5 points is as shown here

  (points.in)

```
5
3 4
-1 4
5 -2
-6 -2
0 3
```

7. Exercise #1: Points (2/5)

Unit18_Points.c

```
#include <stdio.h>
#define MAX_POINTS 10
typedef struct {
  int x, y; // x- and y-coordinates of a point
} point_t;
// Function prototypes omitted for brevity

int main(void) {
   point_t points[MAX_POINTS];
   int size; // number of points

   size = read_points(points);

   …

   return 0;
}
```

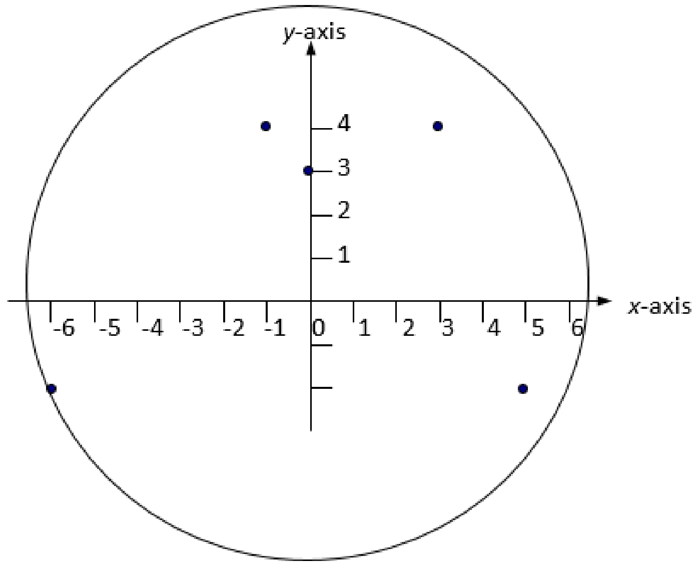
7. Exercise #1: Points (3/5)

```c
// Read input data
// Return the number of points read
int read_points(point_t points[]) {
   int size, i;

   printf("Enter number of points: ");
   scanf("%d", &size);

   printf("Enter data for %d points:\n", size);
   for (i=0; i<size; i++)
       scanf("%d %d", &points[i].x, &points[i].y);

   return size;




}
```

Unit18_Points.c

7. Exercise #1: Points (4/5)

- After reading the points, imagine that you draw the smallest circle with centre at the origin (0, 0) that encloses all the given points. Complete the function float circle_area() to return the area (of type float) of this smallest circle.
- You may assume that π is 3.14159.
- For the example input data, the area is 125.66.
- Hint: It may be useful to add a function for computing the square of distance of a point from the origin.



```
5
3 4
-1 4
5 -2
-6 -2
0 3
```

7. Exercise #1: Points (5/5)

Unit18_Points.c

```
// Compute the area of the smallest circle that
// encloses all the points.
float circle_area(point_t points[], int size) {
   int i, max_dist, dist;

   max_dist = dist_sq(points[0]);

   for (i=1; i<size; i++) {
      dist = dist_sq(points[i]);
         if (dist > max_dist)
            max_dist = dist;
   }
   return PI * max_dist;
}

// Square of distance of a point from the origin
int dist_sq(point_t pt) {
   return (pt.x * pt.x) + (pt.y * pt.y);
}
```

7. Exercise #2: Health Screening (1/2)

- Write a program Unit18_Health_Screen.c to read in a list of health screen readings
  - Each input line represents a reading consisting of 2 numbers: a float value indicating the health score, and an int value indicating the number of people with that score.
  - You may assume that there are at most 50 readings.
  - The input should end with the reading 0 0, or when 50 readings have been read. (see health.in)
- As the readings are gathered from various clinics, there might be duplicate scores in the input. You are to determine how many unique scores there are.
- A skeleton program Unit18_Health_Screen.c is given.
- This exercise is mounted on CodeCrunch.

7. Exercise #2: Health Screening (2/2)

- A sample run is shown below

```
Enter score and frequency (end with 0 0):
5.2135 3
3.123 4
2.9 3
0.87 2
2.9 2
8.123 6
3.123 2
7.6 3
2.9 4
0.111 5
0 0
Number of unique readings = 7
```

- Possible extension: Which is the score that has the highest combined frequency? (Do this on your own.)

Summary

- In this unit, you have learned about
  - How to create and use structures with strings
  - How to pass structures to functions
  - How to use an array of structures

End of File