ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

PROGRAMMING METHODOLOGY
(PHƯƠNG PHÁP LẬP TRÌNH)

UNIT 19: Structures

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

Recording of modifications

- Currently, there are no modification on these contents.

Unit 19: Structures

Objectives:

- Learn how to create and use structures
- Learn how to pass structures to functions
- Return structures as results of functions
- Learn how to use an array of structures

Reference:

- Chapter 8, Lessons 8.1 – 8.5

Unit 19: Structures (1/2)

1. Organizing Data
2. Structure Types
3. Structure Variables
   - 3.1 Initializing Structure Variables
   - 3.2 Accessing Members of a Structure Variable
   - 3.3 Demo #1: Initializing and Accessing Structure Members
   - 3.4 Reading a Structure Member
4. Assigning Structures

Unit 19: Structures (2/2)

5. Passing Structures to Functions (with Demo #2)
6. Array of Structures (with Demo #3)
7. Passing Address of Structure to Functions (with Demos #4 and #5)
8. The Arrow Operator (->) with Demo #6
9. Returning Structure from Functions with Demo #7

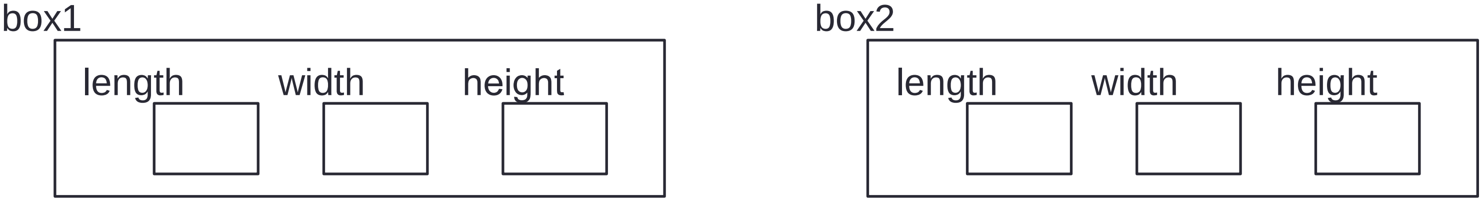
1. Organizing Data (1/4)

- Write a program to compute the volume of 2 boxes.

```
int length1, width1, height1; // for 1st box
int length2, width2, height2; // for 2nd box
```

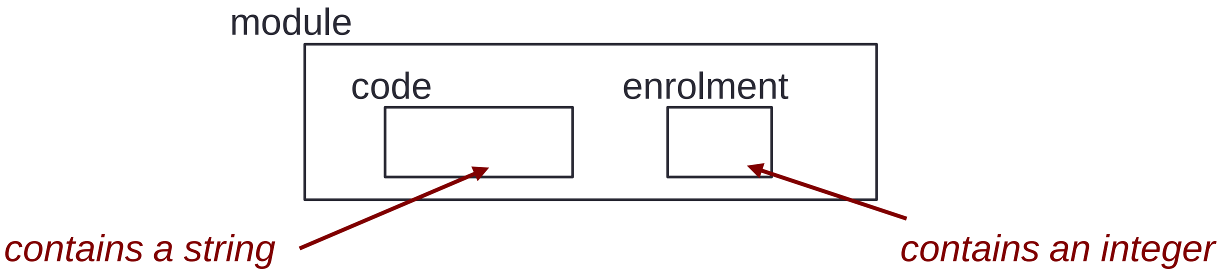
length1
width1
height1
length2
width2
height2

- More logical to organize related data as a "box" *group*, with length, width and height as its components (members). Then declare two variables box1 and box2 of such a *group*.



1. Organizing Data (2/4)

- The members of a *group* may be heterogeneous (of different types) (as opposed to an array whose elements must be homogeneous)
- Examples:



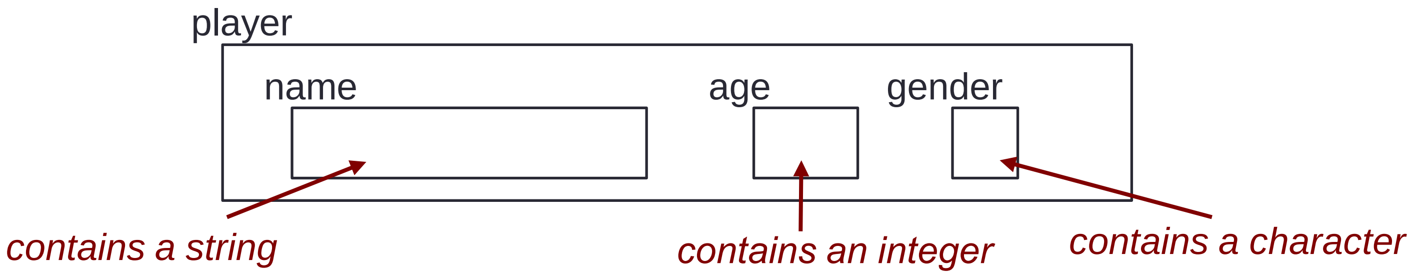
module

code

enrolment

contains a string

contains an integer



player

name

age

gender

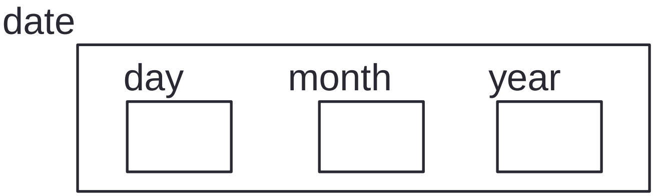
contains a string

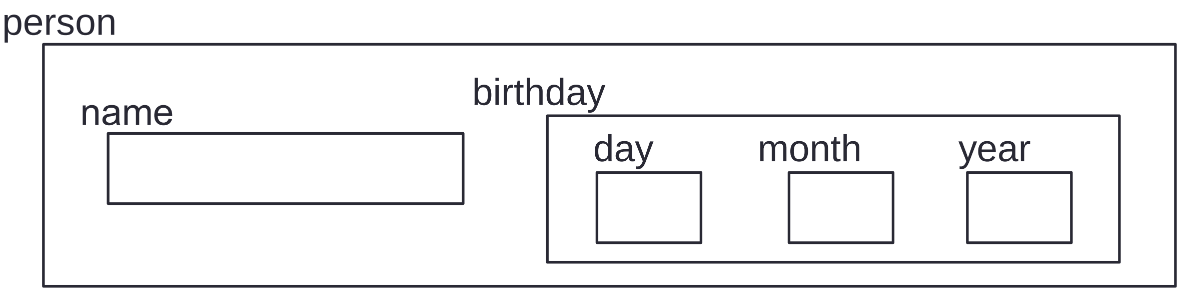
contains an integer

contains a character

1. Organizing Data (3/4)

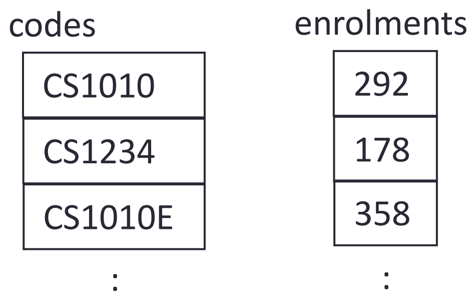
- A *group* can be a member of another *group*.
- Example: person’s birthday is of “date” group





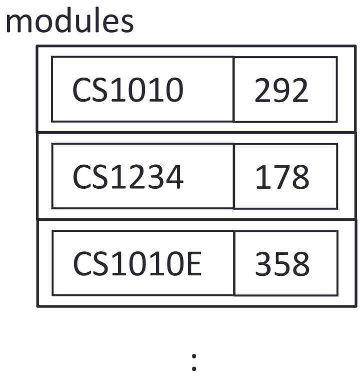
1. Organizing Data (4/4)

- We can also create array of *groups*
- Recall Week 10 Exercise #3: Module Sorting
  - Using two parallel arrays
    - codes[*i*] and enrolments[*i*] are related to the same module *i*



codes	enrolments
CS1010	292
CS1234	178
CS1010E	358
:	:

  - Using an array of "module" *group*



modules

CS1010	292
CS1234	178
CS1010E	358

:

- Which is more logical?

2. Structure Types (1/2)

- Such a group is called structure type
- Examples of structure types:

```
struct{
   int length, width, height;
};
```

This semi-colon ; is very important and is often forgotten!

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

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

2. Structure Types (2/2)

- A type is NOT a variable!
  - what are the differences between a type and a variable?
- The following is a definition of a type, NOT a declaration of a variable
  - A type needs to be defined before we can declare variable of that type
  - No memory is allocated to a type

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

3. Structure Variables (1/3)

- Three methods to declare structure variables
- Examples: To declare 2 variables player1 and player2

- Method 1 (anonymous structure type)
  - seldom used

```
struct {
   char name[12];
   int  age;
   char gender;
} player1, player2;
```

3. Structure Variables (2/3)

- Method 2
  - Name the structure using a tag, then use the tag name to declare variables of that type
  - Some authors prefer to suffix a tag name with “_t” to distinguish it from the variables

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

struct player_t player1, player2;
```

Usually before all functions

3. Structure Variables (3/3)

- Method 3
  - Use typedef to define and name the structure type

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

player_t player1, player2;
```

Usually before all functions

Create a new type called player_t

We will use this syntax in our module.

3.1 Initializing Structure Variables

- The syntax is like array initialization
- Examples:

```
typedef struct {
   int day, month, year;
} date_t;

typedef struct {
   char matric[10];
   date_t birthday;
} student_t;

student_t john = {"A0123456Y", {15, 9, 1990}};
```

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

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

3.2 Accessing Members of a Structure Variable

- Use the dot (.) operator

```
player_t player2;

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

```
student_t john = { "A0123456Y", {15, 9} };

john.birthday.year = 1990;
```

3.3 Demo #1: Initializing and Accessing Members

Unit19_Demo1.c

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

```c
#include <stdio.h>
#include <string.h>
typedef struct  {
    char name[12];
    int  age;
    char gender;
} player_t;

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

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

    printf("player1: name = %s; age = %d; gender = %c\n",
           player1.name, player1.age, player1.gender);
    printf("player2: name = %s; age = %d; gender = %c\n",
           player2.name, player2.age, player2.gender);
    return 0;
}
```

Type definition

Initialization

Accessing members

3.4 Reading a Structure Member

- The structure members are read in individually the same way as we do for ordinary variables
- Example:

```
player_t player1;

printf("Enter name, age and gender: ");

scanf("%s %d %c", player1.name,
      &player1.age, &player1.gender);
```

- Why is there no need for & to read in player1’s name?

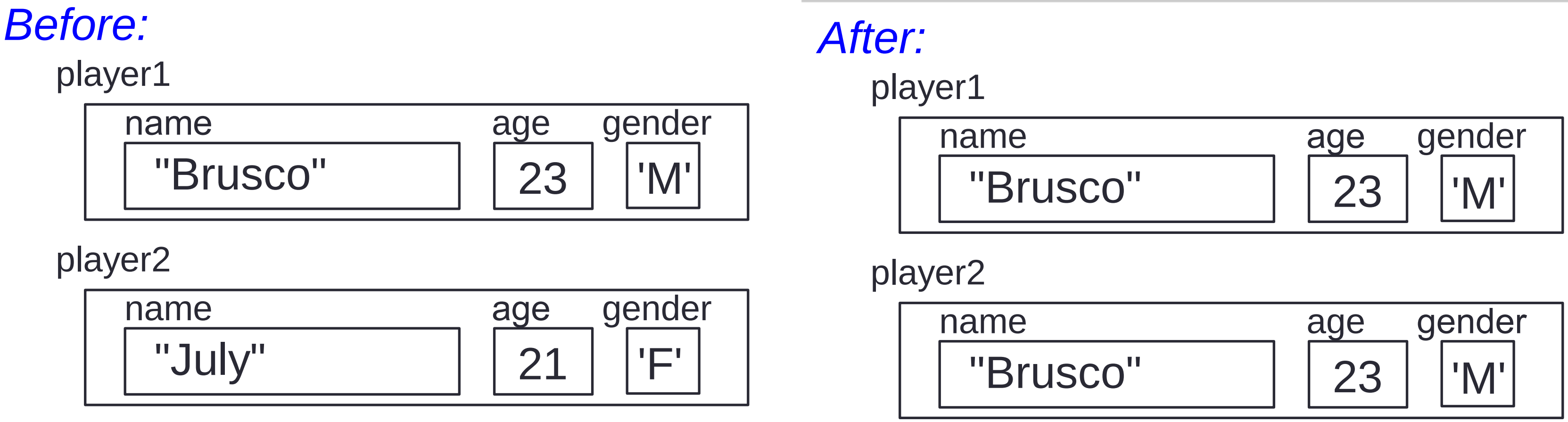
4. Assigning Structures

- We use the dot operator (.) to access individual member of a structure variable.
- If we use the structure variable's name, we are referring to the entire structure.
- Unlike arrays, we may do assignments with structures

```
player2 = player1;
```

≡

```
strcpy(player2.name, player1.name);
player2.age = player1.age;
player2.gender = player1.gender;
```



Before:

player1

name "Brusco" age 23 gender 'M'

player2

name "July" age 21 gender 'F'

After:

player1

name "Brusco" age 23 gender 'M'

player2

name "Brusco" age 23 gender 'M'

5. Passing Structures to Functions

- Passing a structure to a parameter in a function is akin to assigning the structure to the parameter.
- As seen earlier, the entire structure is copied, i.e., members of the actual parameter are copied into the corresponding members of the formal parameter.
- We modify Unit19_Demo1.c into Unit19_Demo2.c to illustrate this.

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

5. Demo #2: Passing Structures to Functions

Unit19_Demo2.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

6. Array of Structures (1/2)

- 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.
- Recall Week 11 Exercise #3: Module Sorting (see next slide)
- Instead of using two parallel arrays modules[] and students[], we shall create a structure comprising module code and module enrolment, and use an array of this structure.
- We will show the new implementation in Unit19_SortModules.c for comparison with Week11_SortModules.c (both programs are given)

6. Array of Structures (2/2)

- Given an array with 10 elements, each a structure containing the code of a module and the number of students enrolled in that module. Sort the array by the number of students enrolled, using Selection Sort.

- Sample run:

```
Enter number of modules: 10
Enter module codes and students enrolled:
CS1010 292
CS1234 178
CS1010E 358
CS2102 260
IS1103 215
IS2104 93
IS1112 100
GEK1511 83
IT2002 51
MA1101S 123
```

```
Sorted by student enrolment:
IT2002    51
GEK1511   83
IS2104    93
IS1112   100
MA1101S  123
CS1234   178
IS1103   215
CS2102   260
CS1010   292
CS1010E  358
```

6. Demo #3: Array of Structures (1/3)

Unit19_SortModules.c

```c
#include <stdio.h>
#define MAX_MODULES 10   // maximum number of modules
#define CODE_LENGTH 7    // length of module code

typedef struct {
   char code[CODE_LENGTH+1];
   int  enrolment;
} module_t;

// Function prototypes omitted here for brevity

int main(void) {
   module_t modules[MAX_MODULES];
   int num_modules;

   num_modules = scanModules(modules);
   sortByEnrolment(modules, num_modules);
   printModules(modules, num_modules);
   return 0;
}
```

6. Demo #3: Array of Structures (2/3)

Unit19_SortModules.c

```
int scanModules(module_t mod[]) {
   int size, i;

   printf("Enter number of modules: ");
   scanf("%d", &size);
   printf("Enter module codes and student enrolment:\n");
   for (i=0; i<size; i++)
      scanf("%s %d", mod[i].code, &mod[i].enrolment);

   return size;
}

void printModules(module_t mod[], int size) {
   int i;

   printf("Sorted by student enrolment:\n");
   for (i=0; i<size; i++)
      printf("%s\t%3d\n", mod[i].code, mod[i].enrolment);
}
```

6. Demo #3: Array of Structures (3/3)

Unit19_SortModules.c

```
// Sort by number of students
void sortByEnrolment(module_t mod[], int size) {
   int i, start, min_index;
   module_t temp;

   for (start = 0; start < size-1; start++) {
      // find index of minimum element
      min_index = start;
      for (i = start+1; i < size; i++)
         if (mod[i].enrolment < mod[min_index].enrolment)
            min_index = i;

      // swap minimum element with element at start index
      temp = mod[start];
      mod[start] = mod[min_index];
      mod[min_index] = temp;
   }
}
```

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

- Given this code, what is the output?

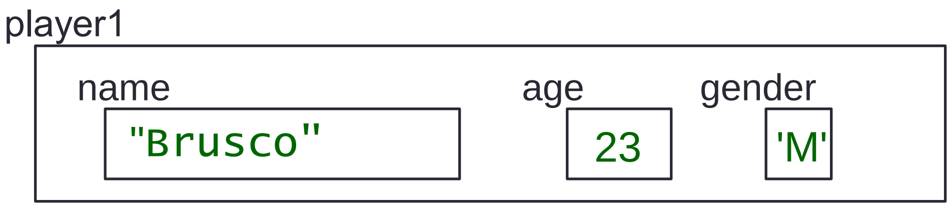
Unit19_Demo4.c

```
// #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);
}
```

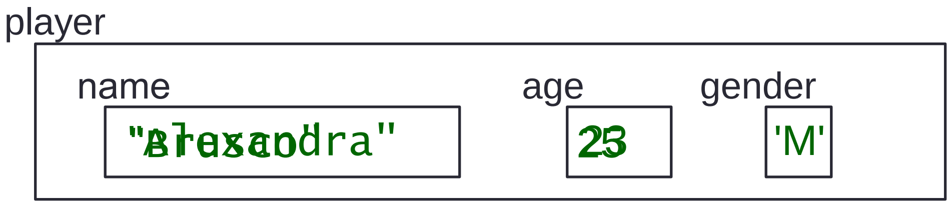
```
player1: name = Brusco; age = 23; gender = M
```

7. 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;
```

7. 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.)

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

- Need to pass address of the structure variable

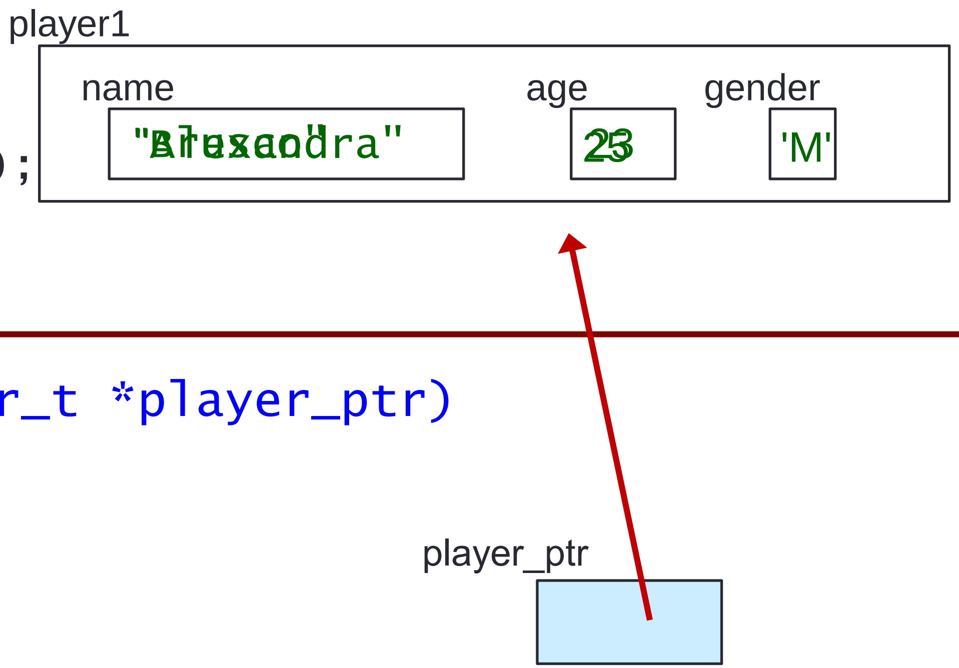
Unit19_Demo5.c

```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
```

7. 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;
```

8. 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)!

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

- Function change_name_and_age() in Unit19_Demo5.c modified to use the -> operator.

Unit19_Demo6.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;
}
```

9. Returning Structure from Functions

- 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( ... );
```

9. Demo #9: Returning Structure

Unit19_Demo7.c

```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

Summary

- In this unit, you have learned about
  - How to aggregate data in structures
  - How to pass structures to functions
  - How to return structures in functions
  - How to declare arrays of structures

End of File