CSE 115: PROGRAMMING LANGUAGE I

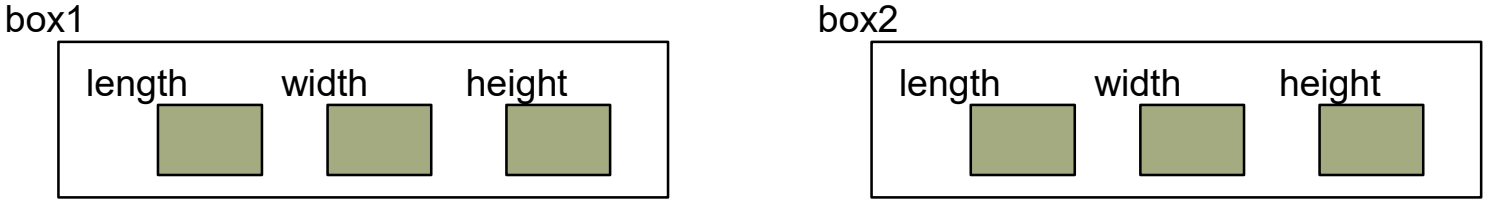
Structures

Organizing Data

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

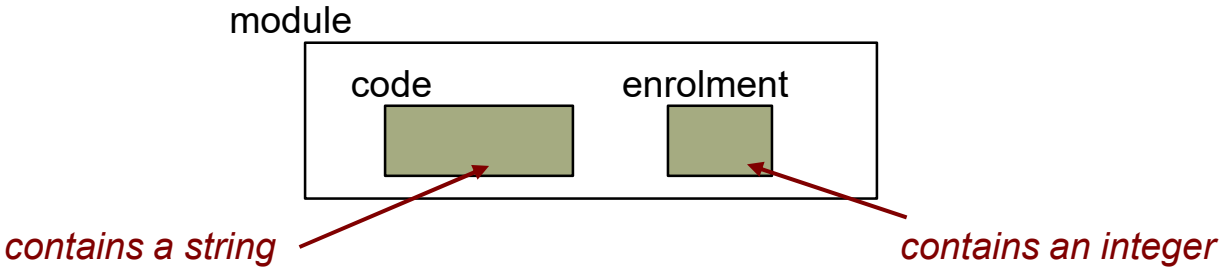
```
int length1, width1, height1; // for 1st box
int length2, width2, height2; // for 2nd box
```

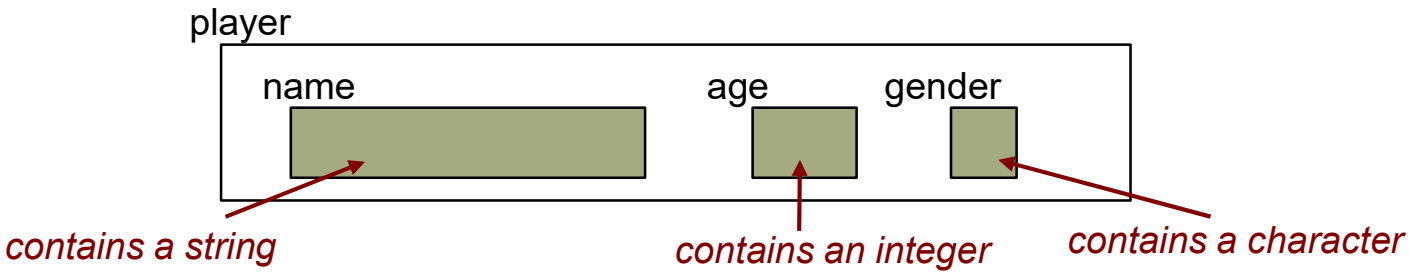
- More logical to organize related data as a "box" *group*, with length, width and height as its components (members). Then declare two variables box1 and box2 of such a *group*.



Organizing Data

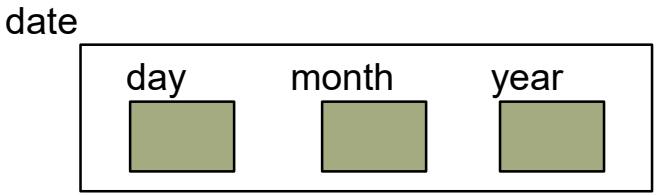
- The members of a *group* may be heterogeneous (of different types) (as opposed to an array whose elements must be homogeneous)
- Examples:

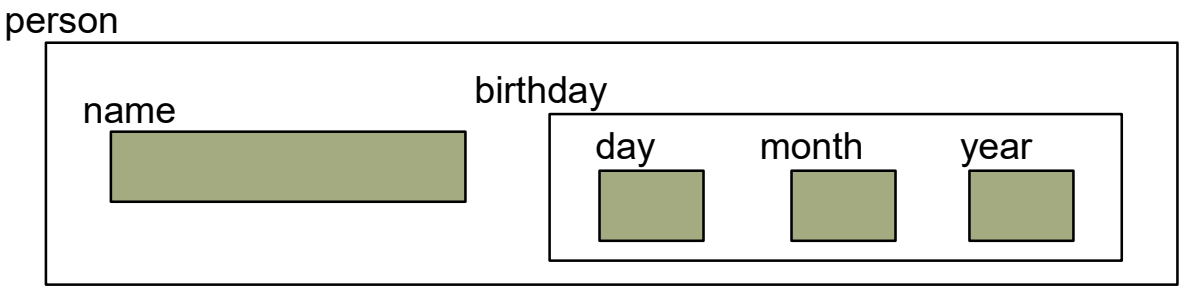




Organizing Data

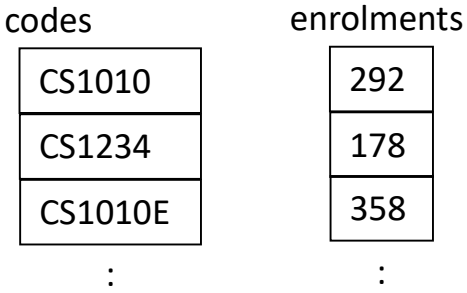
- A *group* can be a member of another *group*.
- Example: person's birthday is of "date" group



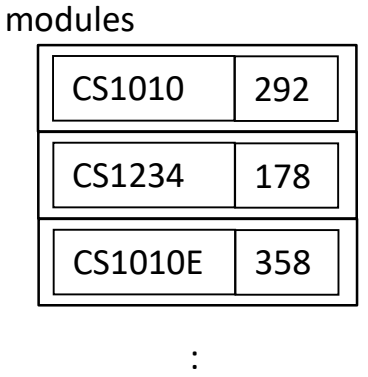


Organizing Data

- We can also create array of *groups*
  - 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?



Structures Types

- 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;
};
```

Structures Types

- 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;
};
```

Structures Variables

- 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;
```

Structures Variables

- 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

Structures Variables

- 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

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' };
```

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;
```

Demo #1: Initializing and Accessing Members

```
#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;
}
```

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

Type definition

Initialization

Accessing members

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?

Assigning Structures

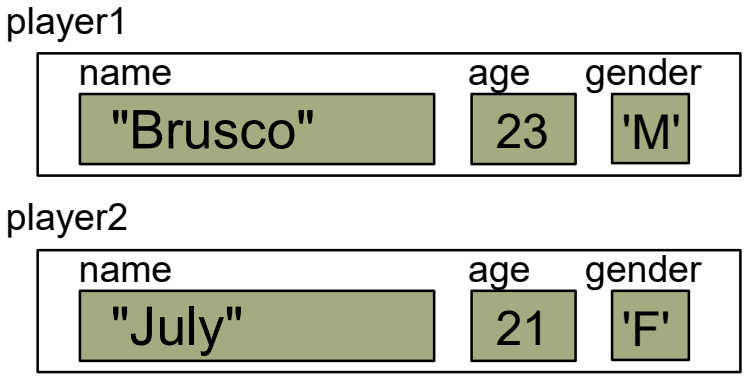
- We use the dot operator (.) to access individual member of a structure variable.
- If we use the structure variable's name, we are referring to the entire structure.
- Unlike arrays, we may do assignments with structures

```
player2 = player1;
```

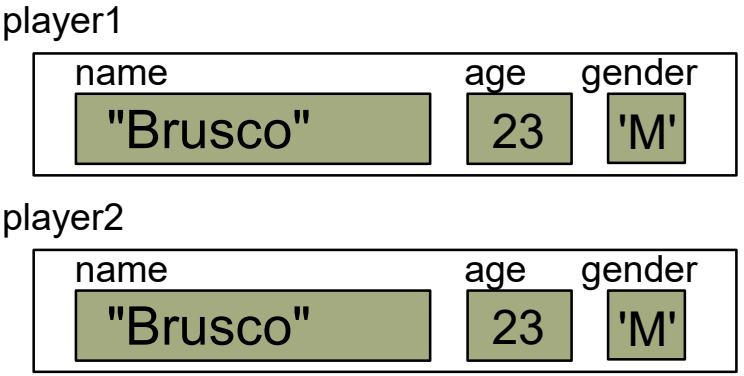
=

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

Before:



After:



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

Demo

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

```c
// #include statements and definition
// of player_t are omitted here for brevity
void print_player(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);
    print_player(player2);

    return 0;
}

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

Passing a structure to a function

Receiving a structure from the caller

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.
- 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.

Array of Structures

- 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
```

Demo: Array of Structures

```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;
}
```

Demo: Array of Structures

```
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);
}
```

Demo: Array of Structures

```
// 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;
   }
}
```