CS1010E Lecture #11

Structures

Your own data type

NUS National University of Singapore | Department of Computer Science School of Computing

String Review #1

PREVIOUS LECTURE

- A string is an array of characters, terminated by a null character '\0'.

```
char str[12] = "Chan Tan";
printf("%s\n", str);
```

```
Chan Tan
```

str[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
C	h	a	n		T	a	n	\0	\0	\0	\0

- Why string?
  - convenient string output (**`printf`**, **`puts`**)
  - convenient string input (**`scanf`**, **`fgets`**)
  - convenient string processing (various string functions)

Quiz

(CS1101C AY2005/06 Semester 1 Exam, Q2)

- **What is printed out by the following code fragment?**

```
char s[] = "abcdefg";
s[3] = '\0';
printf("%s\n", s);
```

```
abc
```

```
char s[] = "abcdefg";
s[3] = 0;
printf("%s\n", s);
```

```
abc
```

a	b	c	d	e	f	g	\0

```
char s[] = "abcdefg";
s[3] = '0';
printf("%s\n", s);
```

```
abc0efg
```

Quiz

(CS1010E AY2010/11 Semester 2 Exam, Q8)

- **What is printed out by the following code fragment?**

```
char c = 'A', d = 5;
printf("%c %c\n", c + d, 'c' + d);
```

F h

c A

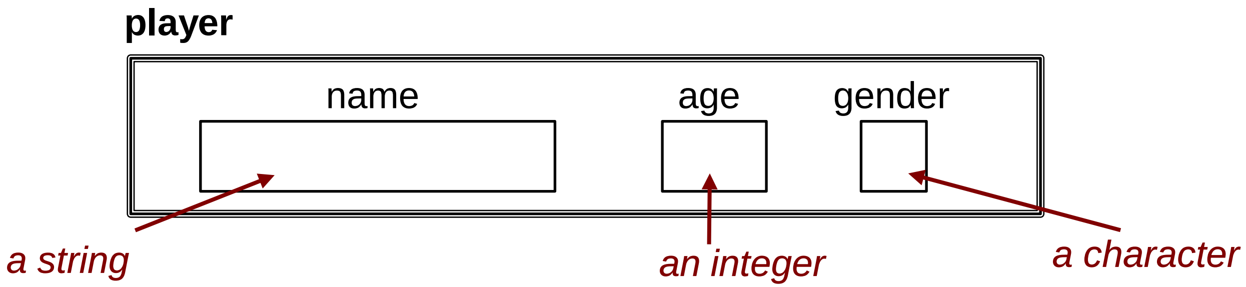
d 5

Learning Objectives

- At the end of this lecture, you should understand:
  - What is structure and why do we need structure.
  - How to create and use structure.
  - How to pass structure variable to and return structure variable from function call.

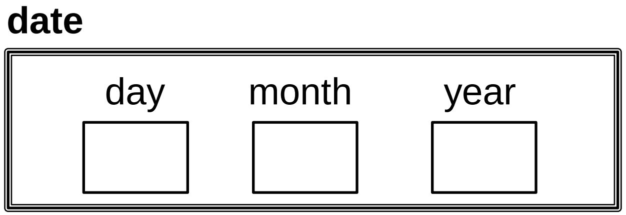
Motivation : Organizing Data (1/2)

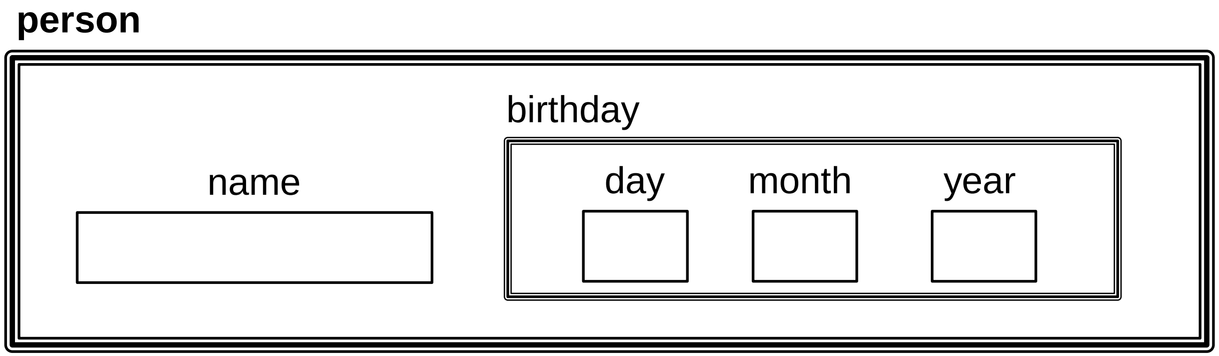
- In many cases, data we want to store and manipulate is too complex to be represented by a primitive data type.
- Example:



Motivation : Organizing Data (2/2)

- In many cases, data we want to store and manipulate is too complex to be represented by a primitive data type.
- More examples:





Defining Structure Data Types

- Structure is used to describe such complex data which may contain several members of heterogeneous types.

- Examples:

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

Do NOT miss this semi-colon ;

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

Defining Structure Variables

- Structure is a user-defined data type.

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


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

define structure before all the functions

player_t is the new data type you create

Initializing Structure Variables

- The syntax is like array initialization.

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


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

Accessing Members of a Structure Variable

- Use the dot (.) operator

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

int main(void) {

  player_t player2;

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

dot operator

Demo #1 : Defining & Using Structures

```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;
}
```

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

type definition

initialization

accessing members

print out members

Reading a Structure Member

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

- Example:

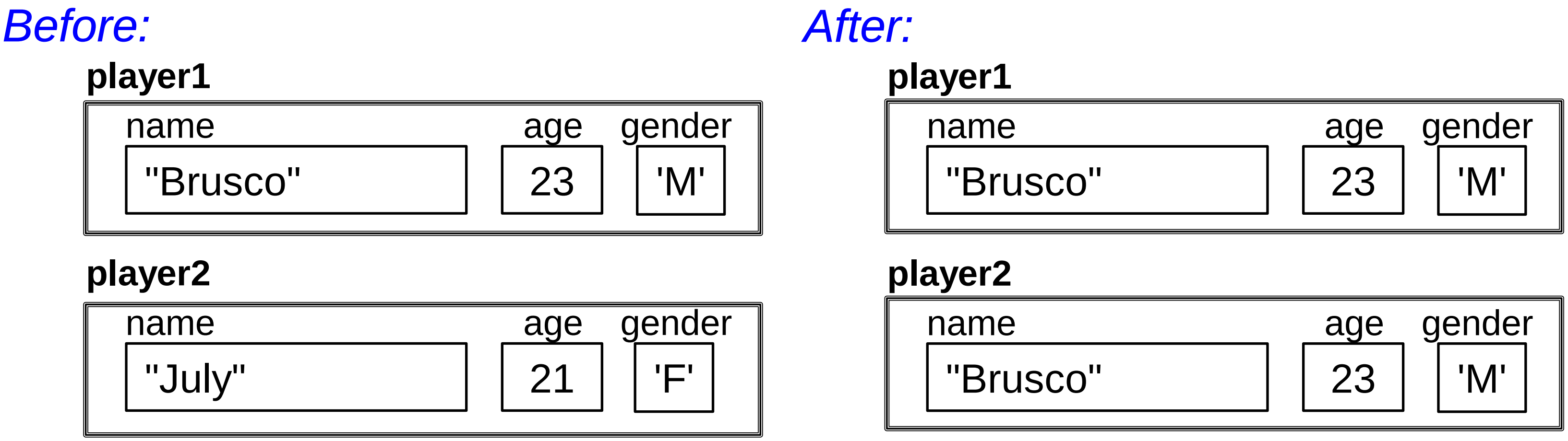
```
...

player_t player1;

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

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

Q: Why there is no & in front of player1.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;
```



Before:

player1

name "Brusco" age 23 gender 'M'

player2

name "July" age 21 gender 'F'

After:

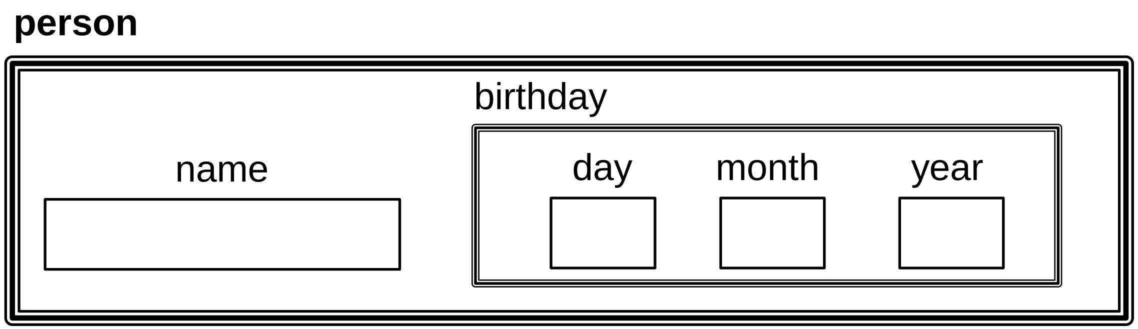
player1

name "Brusco" age 23 gender 'M'

player2

name "Brusco" age 23 gender 'M'

Nested Structures



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

typedef struct {
  char name[11];
  date_t birthday;
} person_t;

...
person_t person;
...
scanf("%s %d %d %d", person.name,  &person.birthday.day,
           &person.birthday.month, &person.birthday.year);
```

Demo #2 : Perimeter

This is Problem Set 4
Ex #11 on CodeCrunch

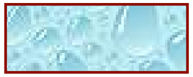
- Write a program perimeter.c to:
  - Define a structure type **rectangle_t** that contains 2 **double** members, *side1* and *side2*, which are the lengths of the 2 sides of a rectangle.
  - Declare a variable of **rectangle_t** type and read values into its members.
  - Compute the minimum perimeter if we fold the rectangle into halves once, either along the x-axis or the y-axis.

- Sample run:

```
Enter lengths of two sides: 3 4
Min perimeter after fold = 10.0
```

3×4

3×2



1.5×4

Demo #2 : Reference Solution

```c
#include <stdio.h>

typedef struct {
  double side1, side2;
} rectangle_t;

int main(void) {

  rectangle_t rect;
  double perimeter;

  printf("Enter lengths of two sides: ");
  scanf("%lf %lf", &rect.side1, &rect.side2);

  if (rect.side1 > rect.side2) {
    perimeter = rect.side1 + 2 * rect.side2;
  } else {
    perimeter = rect.side2 + 2 * rect.side1;
  }
  printf("Min perimeter after fold = %.1f\n", perimeter);
  return 0;
}
```

Passing Structure Variables to Functions

- The entire structure is copied, i.e. members of the actual parameter are copied into the corresponding members of the formal parameter.

- Let’s modify the Demo #1 program to illustrate this.

Demo #3 : Passing Structure Variables

```c
// #include statements, definition of player_t structure skipped
void print_player(char header[], player_t player);

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;
}

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

second parameter is of type player_t

pass a structure variable to a function

receive a structure variable

```
player1: Brusco; 23; M
player2: July; 21; F
```

Passing Address of Structure to Functions

- Like an ordinary variable (int, char, double…), when a structure variable is passed to a function, a local copy is made in the function been called.
  - Pass-by-value

- Hence, the original structure variable will not be modified by the function.

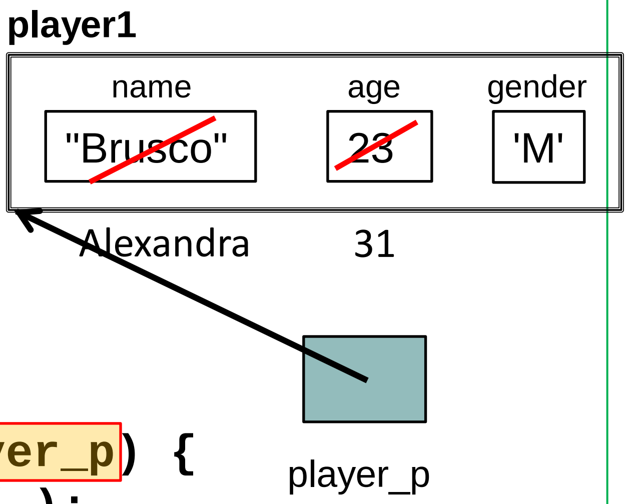
- To allow the function to modify the content of the original structure variable, you need to pass in the address (pointer) of the structure variable to the function.

Demo #4 : Passing Address of Structure

```
// #include statements, definition of player_t,
// and function prototype are omitted here for brevity
int main(void) {

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

  // to change player1's name and age
  change_name_and_age(&player1);
  ...

}


// to change a player’s name and age
void change_name_and_age(player_t *player_p) {
  strcpy( (*player_p).name, "Alexandra" );
  (*player_p).age = 31;
}
```

pass address to function

use pointer to change the original copy



The Arrow Operator (->)

- Expressions like (*player_p).name appear very often. Hence an alternative “shortcut” syntax is created for it.
- The arrow operator: ->

`(*player_p).name` *is equivalent to* `player_p->name`

`(*player_p).age` *is equivalent to* `player_p->age`

- Dot (.) has a higher precedence than *, that’s why you need the braces for (*player_p).age.

Demo #5 : The Arrow Operator (->)

```
void change_name_and_age(player_t *player_p) {

  strcpy( (*player_p).name, new_name );
  (*player_p).age = new_age;
}
```

```
void change_name_and_age(player_t *player_p) {

  strcpy( player_p->name, new_name );
  player_p->age = new_age;
}
```

Quiz

(CS1010 AY2013/14 Semester 1 Exam, Q1.2)

- What is the correct way to assign values to members of structure variable **`tray`**, given the following code fragment?

```
typedef struct {
  int length, width;
} tray_t;
...
tray_t tray;
```

A. **`tray->length = 12; tray->width = 12;`**

B. **`tray = {12, 12};`**

C. **`tray.length = tray.width = 12;`**

D. **`tray = 12;`**

Returning Structure from Functions

- A function can return a structure variable
  - Example: define a function func() that returns a structure of type player_t:

```
player_t func( ... ) {

   player_t player;
   ...
   return player;
}
```

  - To call **func()**:

```
player_t player3;

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

Demo #6 : Returning Structure Variable

```
// #include statements, definition of player_t,
// and function prototype are omitted here for brevity
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;
}

// Read particulars of a player and return it 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;
}
```

store return value in player2

return a structure variable

An Array of Structures

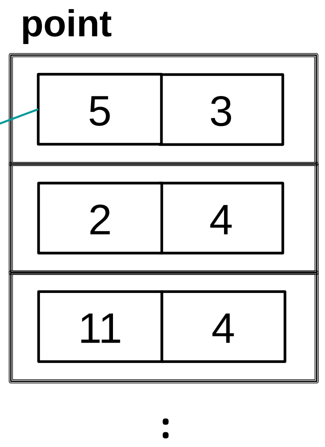
- Combining structures and arrays gives us a lot of flexibility in organizing data.
- For example, we may have a structure comprising 2 members: student’s name (string) and an array of 5 test scores he obtained.
- Or, we may have an array whose elements are structures.
- Or, even more complex combinations such as an array whose elements are structures which comprises array as one of the members.

Demo #7 : An Array of Points (1/4)

- You are given a list of points on a 2-dimensional plane, each point represented by its integer x- and y-coordinates. You are to sort the points in ascending order of their x-coordinates, and for those with the same x-coordinates, in ascending order of their y-coordinates.

- Thinking: we may create an array of points and then sort this array according to the given criteria.
  - Each point can be described by point_t structure

```
typedef struct {
  int  x, y;
} point_t;
```



point

5	3
2	4
11	4

:

Demo #7 : An Array of Points (2/4)

```
// preprocessor directives omitted for brevity
typedef struct {
  int  x, y; // x- and y-coordinates of a point
} point_t;

void scan_points(point_t points[], int *num_points);
void sort_points(point_t points[], int num_points);
int less_than(point_t points[], int p, int q);

int main(void) {

  point_t points[20];
  int num_points; // actual number of points

  scan_points (points, &num_points); // not shown on slides
  sort_points (points,  num_points);
  print_points(points,  num_points); // print in order

  return 0;
}
```

an array of point_t variables

Demo #7 : An Array of Points (3/4)

```c
// Sort the points in ascending order of x-coordinates and
// then y-coordinates, using selection sort.
void sort_points(point_t points[], int size) {

  int i, start_index, min_index;
  point_t temp;
  for (start_index=0; start_index<size-1; start_index++) {
    min_index = start_index;
    for (i=start_index+1; i<size; i++) {
      if ( less_than(points, i, min_index) ) {
        min_index = i;
      }
    }
    // swap point[start_index] with point[min_index]
    temp = points[start_index];
    points[start_index] = points[min_index];
    points[min_index] = temp;
  }
}
```

Demo #7 : An Array of Points (4/4)

```
// Return 1 if point[p] is "less than" point[q], 0 otherwise
// point[p] is "less than" point[q] if the former has a
// smaller x-coordinate, or if their x-coordinates are
// the same, but the former has a smaller y-coordinate.
int less_than(point_t points[], int p, int q) {
  if ( points[p].x < points[q].x ||
      (points[p].x==points[q].x && points[p].y<points[q].y) ) {
    return 1;
  } else {
    return 0;
  }
}
```

Today's Summary

Arrays I

Structure

- Define structure data type
- Store data into structure members
- Assign structure
- Create nested structure
- Pass structure to function
- Use pointer to structure variable
- Return structure from function
- Declare array of structures