CS1010 http://www.comp.nus.edu.sg/~cs1010/

Programming Methodology

Unit 15

Structures

Unit 15: Structures

Objectives:

- Learn how to create and use structures
- Learn how to return 2 or more values from a function using structures

Reference:

- Chapter 10 Structure and Union Types

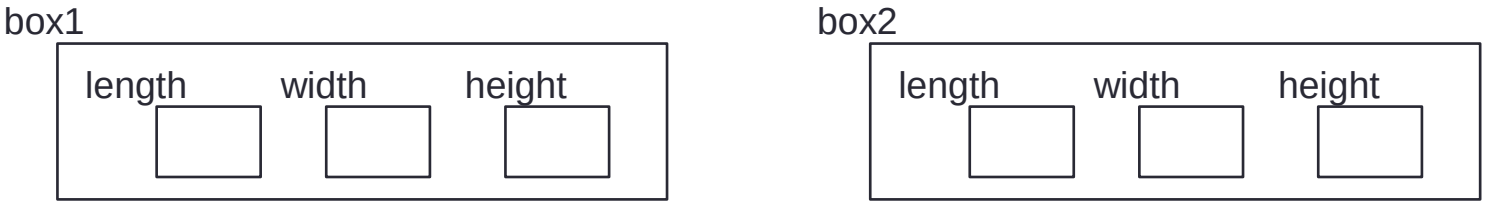
Unit 15: Structures

1. Organizing Data (1/4)

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

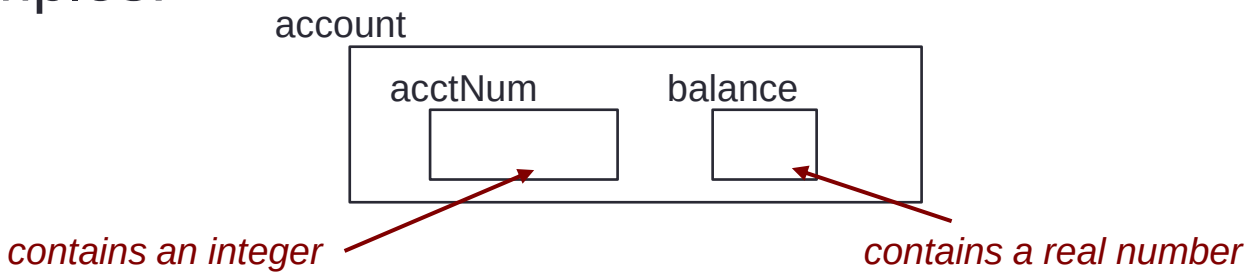
```
int length1, width1, height1; // for 1st box
int length2, width2, height2; // for 2nd box
```

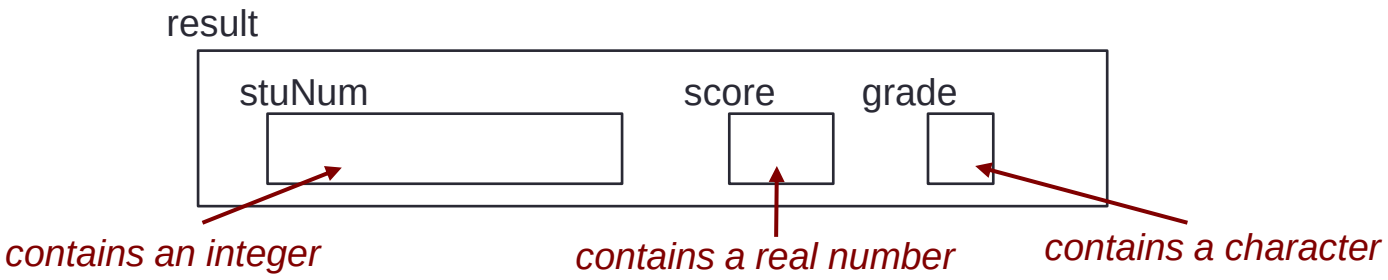
- More logical to organize related data as a "box" *group*, with length, width and height as its components (members). Then declare two variables box1 and box2 of such a *group*.



1. Organizing Data (2/4)

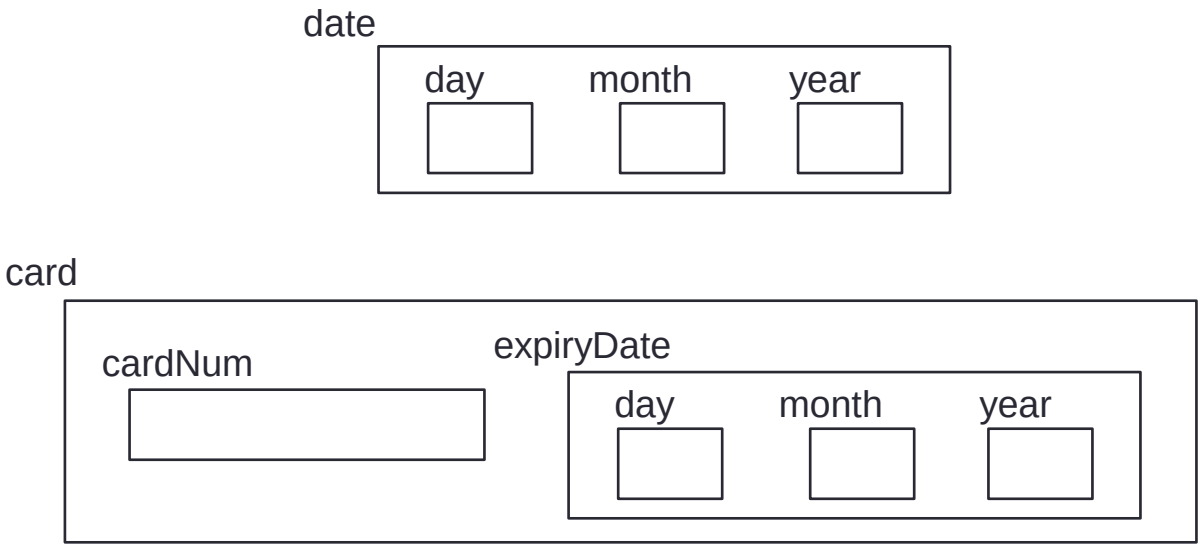
- The members of a *group* may be heterogeneous (of different types) (as opposed to an array whose elements must be homogeneous)
- Examples:





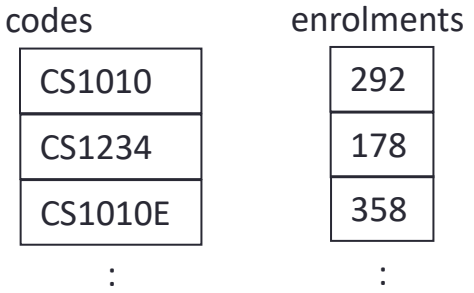
1. Organizing Data (3/4)

- A *group* can be a member of another *group*.
- Example: the expiry date of a membership card is of "date" group

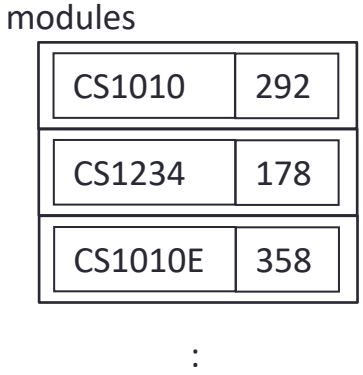


1. Organizing Data (4/4)

- We can also create array of *groups*
- Example: enrolment data for modules
  - 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?
- To be covered later in Unit 18

2. Structure Types (1/2)

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

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

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

Create a new type called box_t

```
typedef struct {
   int acctNum;
   float balance;
} account_t;
```

Create a new type called account_t

```
typedef struct {
   int stuNum;
   float score;
   char grade;
} result_t;
```

Create a new type called result_t

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

```
typedef struct {
   int acctNum;
   float balance;
} account_t;
```

3. Structure Variables

- Declaration
  - The syntax is similar to declaring ordinary variables.

```
typedef struct {
   int stuNum;
   float score;
   char grade;
} result_t;

result_t result1, result2;
```

Before function prototypes
(but after preprocessor directives)

Inside any function

3.1 Initializing Structure Variables

- The syntax is like array initialization
- Examples:

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

typedef struct {
   int cardNum;
   date_t birthday;
} card_t;

card_t card1 = {888888, {31, 12, 2020}};
```

```
typedef struct {
   int stuNum;
   float score;
   char grade;
} result_t;

result_t result1 = { 123321, 93.5, 'A' };
```

3.2 Accessing Members of a Structure Variable

- Use the dot (.) operator

```
result_t result2;

result2.stuNum = 456654;
result2.score = 62.0;
result2.grade = 'D';
```

```
card_t card2 = { 666666, {30, 6} };

card2.expiryDate.year = 2021;
```

3.3 Demo #1: Initializing and Accessing Members

Unit15_Demo1.c

```
result1: stuNum = 123321; score = 93.5; grade = A
result2: stuNum = 456654; score = 62.0; grade = D
```

```c
#include <stdio.h>

typedef struct  {
    int stuNum;
    float score;
    char grade;
} result_t;

int main(void) {
    result_t result1 = { 123321, 93.5, 'A' },
             result2;

    result2.stuNum = 456654;
    result2.score = 62.0;
    result2.grade = 'D';

    printf("result1: stuNum = %d; score = %.1f; grade = %c\n",
           result1.stuNum, result1.score, result1.grade);
    printf("result2: stuNum = %d; score = %.1f; grade = %c\n",
           result2.stuNum, result2.score, result2.grade);
    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:

```
result_t result1;

printf("Enter student number, score and grade: ");

scanf("%d %f %c", &result1.stuNum, &result1.score,
                  &result1.grade);
```

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

```
result2 = result1;
```

=

```
result2.stuNum = result1.stuNum;
result2.score = result1.score;
result2.grade = result1.grade;
```

Before:

result1

stuNum	score	grade
123321	93.5	'A'

result2

stuNum	score	grade
456654	62.0	'D'

After:

result1

stuNum	score	grade
123321	93.5	'A'

result2

stuNum	score	grade
123321	93.5	'A'

5. Exercise #1: Perimeter (1/2)

- Write a program Unit15_Perimeter.c to do the following:
  1. Define a structure type rectangle_t with 2 integer members: side1 and side2, which are the lengths of its 2 sides.
  2. Declare a variable of type rectangle_t and read values into its members.
  3. Compute the minimum perimeter if we fold the rectangle into halves once, either along the x-axis or the y-axis.
- Note
  - Do not use any additional variables besides the two given variables.
  - You may write the code in the **main()** function. You may modularise the program later.

5. Exercise #1: Perimeter (2/2)

Unit15_Perimeter.c

```
#include <stdio.h>

typedef struct {
   int side1, side2;
} rectangle_t;

int main(void) {
   rectangle_t rect;
   int perimeter;

   printf("Enter lengths: ");
   scanf("%d %d", &rect.side1, &rect.side2);

   if (rect.side1 > rect.side2)
      perimeter = rect.side1 + 2*rect.side2;
   else
      perimeter = rect.side2 + 2*rect.side1;

   printf("Perimeter = %d\n", perimeter);
   return 0;
}
```

6. Returning Structure from Functions (1/4)

- When combined with arrays and functions, structures give us a lot of flexibility in organizing and passing around data.

- One such example is that a function may return more than one outputs using structure.

- We will explore other examples later in Unit #18.

6. Returning Structure from Functions (2/4)

- Example:
  - Given this structure type result_t,

```
typedef struct {
    int max;
    float ave;
} result_t;
```

  - Define a function func() that returns a structure of this type:

```
result_t func( ... ) {
   ...
}
```

  - To call this function:

```
result_t result;

result = func( ... );
```

6. Returning Structure from Functions (3/4)

Unit15_Demo2.c

```
#include <stdio.h>

typedef struct {
    int max;
    float ave;
} result_t;

result_t max_and_average(int, int, int);

int main(void) {
    int num1, num2, num3; // inputs
    result_t result;

    printf("Enter 3 integers: ");
    scanf("%d %d %d", &num1, &num2, &num3);
    result = max_and_average(num1, num2, num3);

    printf("Maximum = %d\n", result.max);
    printf("Average = %.2f\n", result.ave);
    return 0;
}
...
```

returned structure is copied to *result*

max and average are printed

6. Returning Structure from Functions (4/4)

Unit15_Demo2.c

```
// Computes the maximum and average of 3 integers
result_t max_and_average(int n1, int n2, int n3) {
    result_t result;

    result.max = n1;
    if (n2 > result.max)
      result.max = n2;
    if (n3 > result.max)
      result.max = n3;

    result.ave = (n1+n2+n3)/3.0;

    return result;
}
```

the answers are stored in the structure variable *result*.

result is returned here

Summary

- In this unit, you have learned about
  - How to create and use structures
  - How to return 2 or more values from a function using structures

End of File