UNIT 7: STRUCTURES

Nguyen Thi Thu Huong
Department of Computer Science
School of Information and Communication Technology
Hanoi University of Technology

DATA STRUCTURES (STRUCT)

- Arrays require that all elements be of the same data type.
- It is necessary to group information of different data types: list of products (name , part_number, dimensions, weight, cost)
- Structures can store combinations of character, integer floating point and enumerated type data. Name of the data type : *struct*.

STRUCTURES (STRUCT)

- A *struct* is a derived data type composed of members that are each fundamental or derived data types.
- A single *struct* would store the data for one object. An array of *structs* would store the data for several objects.
- A *struct* can be defined in several ways as illustrated in the following examples:

Declaring Structures (struct)

Does Not Reserve Space

```
struct my_example
{
    int label;
    char letter;
    char name[20];
} ;
/* The name "my_example" is
    called a structure tag    */
```

Reserves Space

```
struct my_example
{
    int label;
    char letter;
    char name[20];
} mystruct ;
```

USER DEFINED DATA TYPES (TYPEDEF)

- *typedef* : for creating synonyms for previously defined data type names.

  Example:

  ```
  typedef  int  Length;
  ```

  Length is a synonym (alias) for the data type *int*.

- The data "type" name *Length* can now be used in declarations in exactly the same way that the data type *int* can be used:

  ```
  Length   a, b, len ;
  Length   numbers[10] ;
  ```

TYPEDEF & STRUCT

- Often, *typedef* is used in combination with *struct* to declare a synonym (or an alias) for a structure:

```
typedef struct                    /* Define a structure */
{
   int label ;
   char letter;
   char name[20] ;
} Some_name ;                     /* The "alias" is Some_name */

Some_name  mystruct ;    /* Create a struct variable */
```

ACCESSING STRUCT MEMBERS

- Individual members of a *struct* variable may be accessed using the structure member operator (the dot, "."):

  mystruct.letter ;

- Or , if a pointer to the *struct* has been declared and initialized

  Some_name *myptr = &mystruct ;

  by using the structure pointer operator (the "->"):

  myptr -> letter ;

  which could also be written as:

  (*myptr).letter ;

SAMPLE PROGRAM WITH STRUCTS

```
/* This program illustrates creating structs and
   then declaring and using struct variables.  Note
   that struct personal is an included data type in
   struct "identity".
   */
#include <stdio.h>
struct personal  /*Create a struct but don't reserve
   space*/
  {  long id;
     float gpa;
  } ;
struct identity    /*Create a second struct that
   includes the first one. */
 {  char name[30];
     struct personal person;
 } ;
```

Sample Program With Structs (cont.)

```
int main ( )
{
   struct identity js = {"Joe Smith"}, *ptr = &js ;

   js.person.id = 123456789 ;
   js.person.gpa = 3.4 ;
   printf ("%s %ld %f\n", js.name, js.person.id,
            js.person.gpa) ;
   printf ("%s %ld %f\n", ptr->name, ptr->person.id,
            ptr->person.gpa) ;
}
```