**Lecture 9 Structures**

**Concept of structures**

* A structure is a user-defined data type, consisting of a list of data elements, possibly of different types, stored in contiguous memory locations (memory block)
* The size of a structure is the sum of sizes of all its elements
* In algorithm/program, a structure is a composite data structure, combining different types of data values. A structure type of data object is also called a record.

Syntax:

Struct struct\_name{

Data\_type1 v1;

Data\_type v2;

…

};

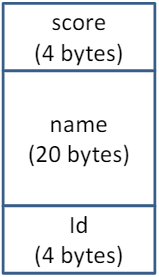
The data\_type can be any data type name which has been defined

Declare structure type variable:

Struct struct\_name struct\_var;

Use dot operator to represent each element as a variable, and be used to set/get values of get address

* Struct\_var.v1
* The structure definition does not allocate any memory. It defines a template that tells the compiler the types and ordering of structure elements. The structure type variable declaration tells compiler to allocate memory block.
* Structure variable initialization:
  + Initializing a structure means assigning some constrains to the elements of the structure
  + Syntax to initialize a structure variable
    - Struct struct\_name struct\_var = {constant1, constant2,…};
  + Initializers should match their corresponding element types in the structure definition. The rest will be set to 0

Example

Struct student{

Int id;

Char name[20];

Float score;

};

//this defines a structure type names struct student

Struct student s1;

//this declares a variable s1 of the struct student type

Sizeof(struct student) **or** sizeof(s1) is defined as

Sizeof(int) + sizeof(name) + sizeof(float) = 28 bytes

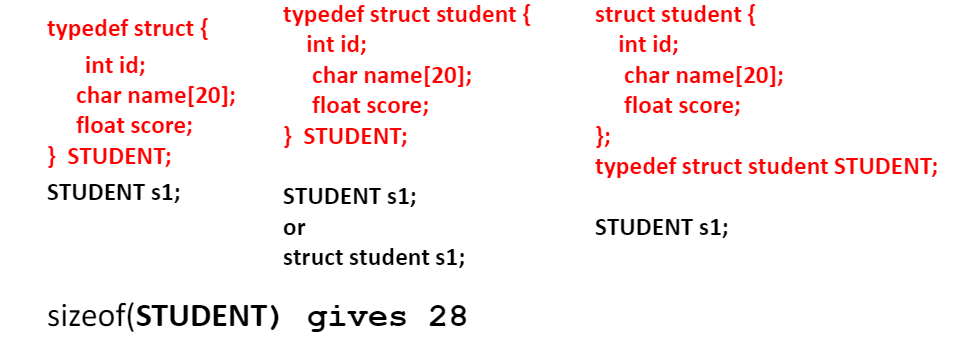
S1 is allocated a memory block of 28 bytes. Element id has the lowest address, followed by name and score in the memory block.

**Typedef for user-defined type name**

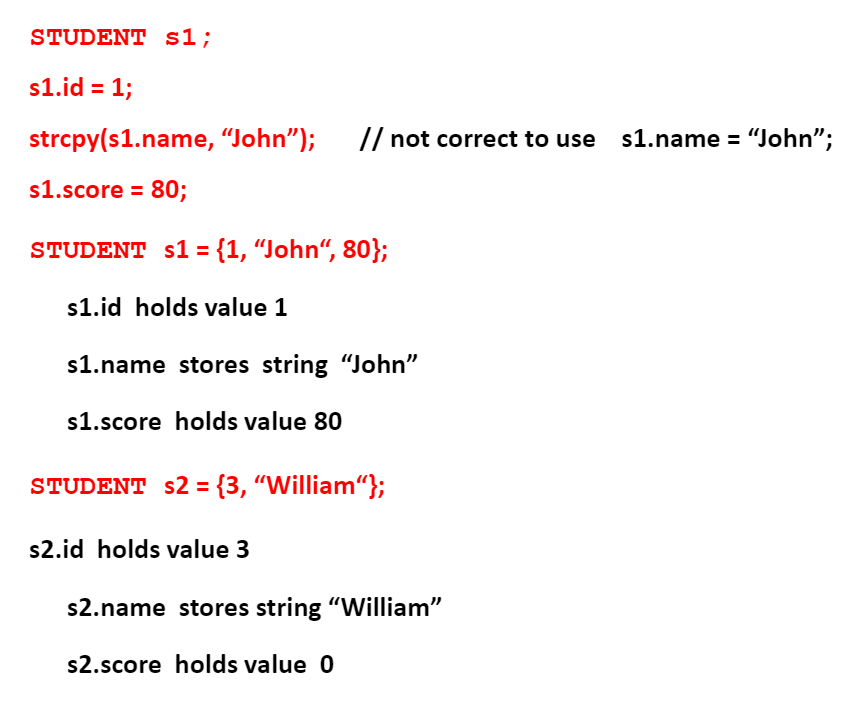
* Keyword typedef is used to define a type name
* Rename an existing type

Typedef int INT;

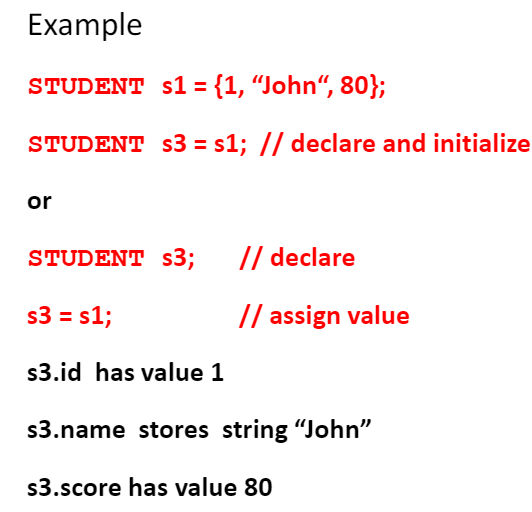
INT a = 10; // the same as int a=10;



* Instead of typing struct student, by using typedef you can just type STUDENT



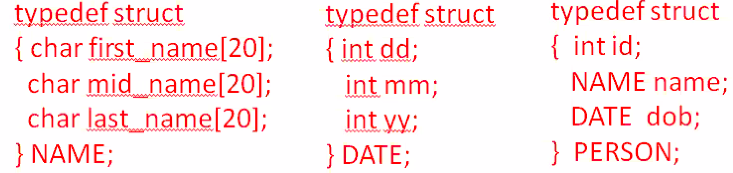
* Remember to assign strings in the special way above



* S1 and s3 refer to **different** locations in memory

**Nested structures**

* A structure can be placed inside another structure to create a nested structure



**Pointers on structures**

* A structure type pointer can be declared to hold the address of the structure type variable
* Then use point-to operator “->” to access the element of the structure

Syntax:

Struct struct\_name{

Data\_type v1;

Data\_type v2;

};

Struct struct\_name struct\_var, \*ptr;

Ptr = &struct\_var;

Ptr->v1 **or** struct\_var.v1

**Self-referential structures**

* Structures that contain a pointer variable of its own type

Example:

Typedef Struct node{

Int val;

Struct node \*next;

}NODE

* Struct node has two element variables *int val* and *struct node \*next* next is a pointer pointing to a struct node type variable
* This is how you make linked structures

Example:

NODE n1, n2, \*p1 = &n1;

N1.val = 1;

N1.next = &n2;

N2.val=2;

N2.next = NULL;

Printf(“%d”, p1->val); // prints 1

Printf(“%d”, p1->next->val); //prints 2

**Function with structures**

* Passing a structure variable to a function is the same as passing a variable to a function. That is, the value of the structure variable will be copied to the function.
* The general syntax for passing a structure variable to a function:
* Data\_type func\_name(struct struct\_name struct\_var);
* You can also pass by reference

**Structure type as a function return type**

* When a structure type is used as a return type of a function, a copy of each variable of the structure is copied out from the function local scope

**Arrays of structures**

* An array of structures (or structure array) is an array, in which all elements are of the structure type. The representations of structure arrays are same as ordinary arrays.

Syntax:

Struct struct\_name array\_name[length];

* Arrays of structure are commonly used to represent/store/process data records in application programs

**Structures vs Arrays**

1. Both structures and arrays store data elements in contiguous memory locations
2. All array elements have the same data type, elements are represented by subscript index. Structure’s elements may have different data types, elements are represented by dot operator
3. Structures support assignment, arrays do not
4. Structure support return type, arrays do not
5. Passing structure variable to a function copies the structure data to the function local, passing array name to a function copies the address of the array to function local.

* If you want to copy array data to function local, you can put the array in a structure and then, pass the structure name to the function

**Bit fields**

* A bit field is a structure used to store multiple logical values as a short series of bits where each of the single bits can be addressed separately
* Application of bit fields
  + A bit field is used to represent the sign of int type
  + A usage of bit fields is used to represent single bit flags with each flag stored in a separate bit
  + Packet header

