ICT 5101

Lecture 10

Dr. Hossen A Mustafa

Structures

- A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling
- Example:
  - int studentID
  - char *studentName
  - double cgpa
  - char gender
  - All these can be grouped together using a structure

Structure Declaration

- Combined variable and type declaration

```
struct tag {member-list} variable-list;
```

- Any one of the three portions after the **struct** keyword can be omitted
- Example:

```
struct students {
        int studentID;
        char *studentName;
        double cgpa;
        char gender;
} std1, std2;
```

Structure Declaration

- struct {int a, b; char *p;} x, y; /* omit tag */
  - variables x, y declared with members as described:
  - int members a, b and char pointer p.
  - x and y have same type, but differ from all others even if there is another declaration:
- struct {int a, b; char *p;} z;
  - z has different type from x, y

Structure Declaration

- struct S {int a, b; char *p;}; /* omit variables */
  - No variables are declared, but there is now a type struct S that can be referred to later
- struct S z; /* omit members */
  - Given an earlier declaration of struct S, this declares a variable of that type

Recursively defined structures

- Within a structure, it can refer to structures of the same type, via pointers

```
struct TREENODE {
  char *label;
  struct TREENODE *leftchild, *rightchild;
}
```

Member access

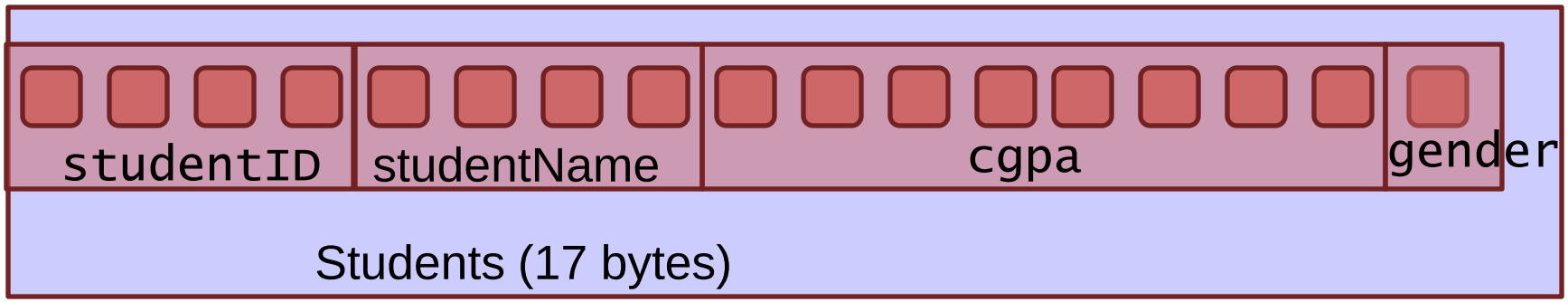
```
struct students {
    int studentID;
    char *studentName;
    double cgpa;
    char gender;
}
struct students s1;
struct students *s2;
struct students allstudents[100];
```

Member access

- Direct access
  - s1. studentID
  - s1.cgpa
  - allstudents[0].studentID
- Indirect access
  - Dereference a pointer to a structure, then return a member of that structure
  - s2-> studentID or
  - s2->cgpa

Memory layout

```
struct students {
    int studentID;  // 4byte
    char *studentName; // 4byte
    double cgpa; // 8 byte
    char gender; // 1 byte
}
```



Structures as function arguments

- Structures can be returned and passed as arguments – just like int, char, etc.
- struct students updateInfo(struct students s);
  - Call by value: temporary copy of structure is created
  - Caution: passing large structures is inefficient
- void updateInfo (struct students *s);
  - Call by reference: address of structure is passed

Unions

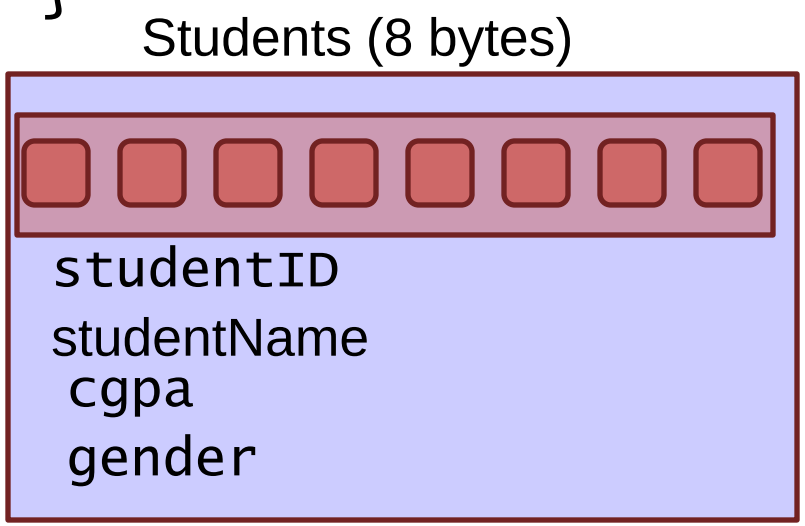
- Like structures, but every member occupies the same region of memory!
- Size of a union is equal to the size of the max sized member
- Example

```
union VALUE {
  float f;
  int i;
  char *s;};
```

- Up to programmer to determine how to interpret a union (i.e. which member to access)

Memory layout

```
union students {
    int studentID;  // 4byte
    char *studentName; // 4byte
    double cgpa; // 8 byte
    char gender; // 1 byte
}
```



Class Assignment

- Write a program named classassignment10.c
- The program should define a structure named **students** with ID, name, and CGPA and declare a global variable array of students
- Implement the following functions
  - void addStudentInfo(int ID, char *name, double cgpa)
  - void showStudentInfo()
  - students getBestCGPA()