CSE 220 – C Programming
Fall 2015

Structures, Unions
Basic Data Types

- C provides basic data types assign storage for the variable and define the type of data that will be held in the location.

  int
  float
  char
  double

- A single location of memory with varied size

<table>
<thead>
<tr>
<th></th>
<th>2.343</th>
<th>A</th>
<th>2.343</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Arrays make it possible to make a sequence whose elements are basic types.

```c
char name[20];
int    data[5][4];
```

Pitfall: Arrays require that all elements be of the same data type. Many times it is necessary to group information of different data types.
A Collection of Items of Different Types

• A program to store and process the information of all students in cse220!

Float GPA;

GPA : 4.00

char name [50];

name : xxxxx

int phone_number;

phone_number : 3538669

int PID;

PID : A305138669
A Collection of Items of Different Types

• A program to store and process the information of all books in EB library!

char name [50];
name : “Programming in C”

int ISBN;
ISBN : 49859392

char author [50];
author: “Dr. X Y”

int year;
Year: 2010
C supports data structures that can **group/store combinations of different data types** (character, integer, floating point, arrays, etc.) into a single data type:

- **Structures**
- **Union**
- **Enumeration**

Data structures of multiple values of different kinds
Structures
Structure

• **Structure**: A *collection* of one or more variables (members), typically of different *types*, grouped together under a single name for convenient handling
Structure

• **Structure:** A *collection* of one or more variables (members), typically of *different types*, *grouped* together under a single name for convenient handling.

• **Member:** each piece of information (i.e., each value)
  
  – Members may have different types
  – Members have names
    • To access a member, specify name not position
• There are two ways to define structures in C:

```c
struct {
    type_1 member_1;
    type_2 member_2;
    type_3 member_3;
    ...
} structure_name;
```

Don’t forget

Only one instance of structure at any time


**Declaration (I)**

```
struct {
    float x;
    float y;
    float r;
} circle;
```

You only have one instance of struct!

---

```
circle center (x, y) radius;
```

---

You only have one instance of struct!
If you need multiple circles, you need to define a separate struct for each circle.

```c
struct {  
  float x;  
  float y;  
  float r;  
} circle1;
```

```c
struct {  
  float x;  
  float y;  
  float r;  
} circle2;
```

If you need multiple circles, you need to define a separate struct for each circle.
If you need multiple circles, you need to define a separate struct for each circle.

How about 1000 circles?
struct {
    float x;
    float y;
    float r;
}
circle1;

struct {
    float x;
    float y;
    float r;
}
circle2;

struct {
    float x;
    float y;
    float r;
}
circle100;

struct {
    float x;
    float y;
    float r;
}
circle3;
Declaration (II)

- Alternatively (more easier and flexible), you can declare a structure **tag**:

```c
struct structure_tag_name
{
    type_1 member_1;
    type_2 member_2;
    type_3 member_3;
    ...
};
```

**Idea:** define a struct tag and create as many instances as we need
Examples

```c
struct Student
{
    char    name[100];
    int     PID;
    int     phone_num;
    double  grades[20];
};
```

Now you have a new datatype called **Student** and you can use this datatype define your instances of student type
Examples

<table>
<thead>
<tr>
<th>Book</th>
<th>struct Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>name (string)</td>
<td>char name[100];</td>
</tr>
<tr>
<td>subject (string)</td>
<td>char subject[100];</td>
</tr>
<tr>
<td>author (string)</td>
<td>char author[100];</td>
</tr>
<tr>
<td>ISBN (integer)</td>
<td>int ISBN;</td>
</tr>
<tr>
<td>year (integer)</td>
<td>int year;</td>
</tr>
</tbody>
</table>

Now you have a new datatype called Book and you can use this datatype define your instances of Book type
Instantiation of Structure Tags

• Defining structure tags does NOT reserve memory

• It is just considered as a definition of a new data type

• You need to make instances of defined structure to be able to use them
Instantiation of Structure (I)

```
struct structure_tag_name instance_name;
```

```
struct Student {
    char   name[100];
    int    PID;
    int    phone_num;
    double grades[20];
};
```
Instantiation of Structure (II)

Alternatively you can create instances when you define the structure

```c
struct Student
{
    char name[100];
    int PID;
    int phone_num;
    double grades[20];
};
```

```
struct Student
{
    char name[100];
    int PID;
    int phone_num;
    double grades[20];
};
```

```c
struct Student Alice;
struct Student Bob;
```
struct Circle
{
    float x;
    float y;
    float r;
}

How about 1000 circles?

Now you have a new datatype called Circle and you can use this datatype to define your instances of Circle type
Array of Structs

• Array whose elements are structures

• Example: Write a program that keeps the list of 1000 circles
Example: Array of Structs

```c
struct Circle
{
    float x;
    float y;
    float r;
} ;
```

- Defining structure tags does **NOT** reserve memory

```c
struct Circle circles[1000];
```