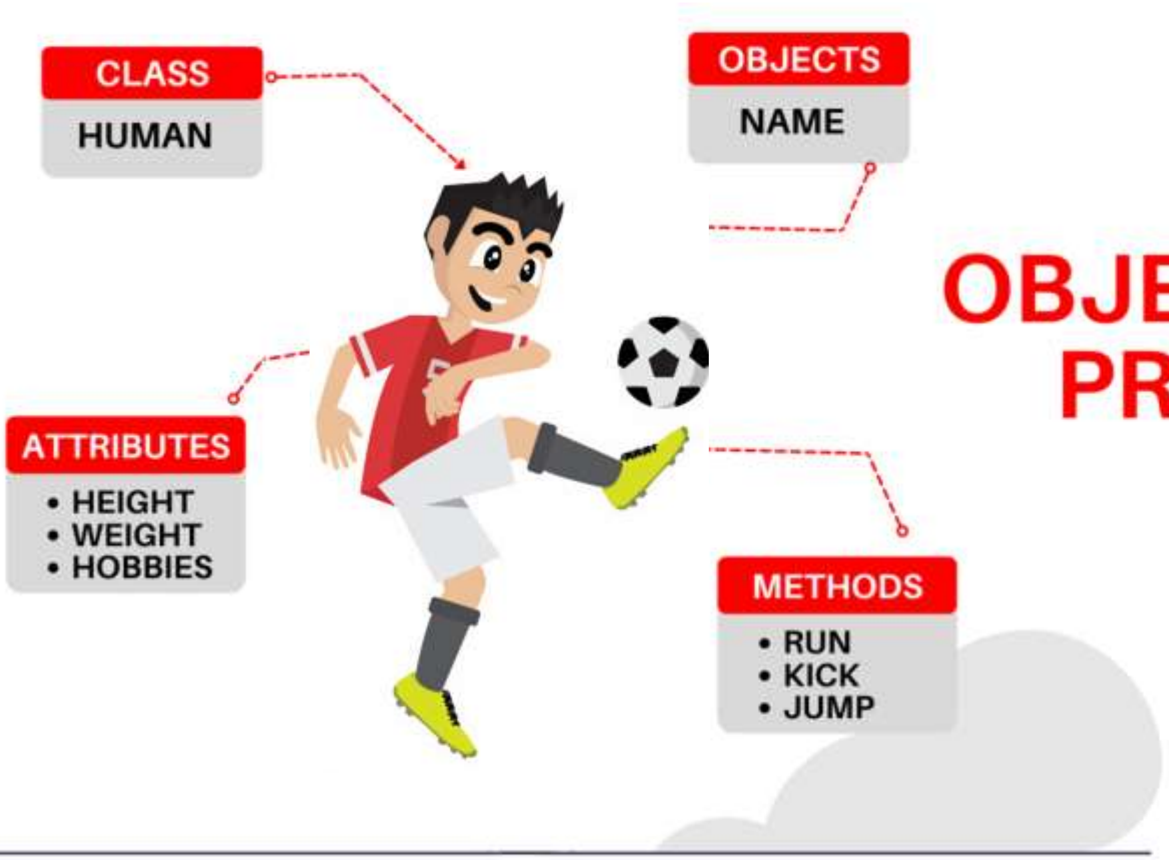
National University of Computer and Emerging Sciences



OBJECT-ORIENTED PRORAMMING

Spring 2022

Pir Sami Ullah Shah

Lecture # 5 Structures

Abstract Data Type

- You have seen many primitive data types like **`int`**, **`float`**, **`double`**, **`bool`** **`etc.`**

- An abstract data type (ADT) is a data type created by the programmer and is composed of one or more primitive data types.

Abstract Data Type

- So far you’ve written programs that keep data in individual variables.
- If you need to group items together, C++ allows you to create arrays.
- The limitation of arrays, however, is that all the elements must be of the same data type.
- Sometimes a relationship exists between different types of elements.

Abstract Data Type

Their definition statements do not make it clear that **they belong together**.

Variable Definition	Data Held
int empNumber;	Employee number
string name;	Employee’s name
double hours;	Hours worked
double payRate;	Hourly pay rate
double grossPay;	Gross pay

All these variables hold data about the same employee

Combining Data into Structures

- Structure: is like a container that allows multiple variables to be grouped together

- Variables can be of any type

```
struct structName
{
  dataType field1;
  dataType field2;
  . . .
};
```

Example struct Declaration

```
struct Student
{
   int studentID;
   string name;
   short yearInSchool;
   double gpa;
};
```

structure name

structure members

- Organize related data (variables) into a nice neat package (single unit)

`struct` Declaration Notes

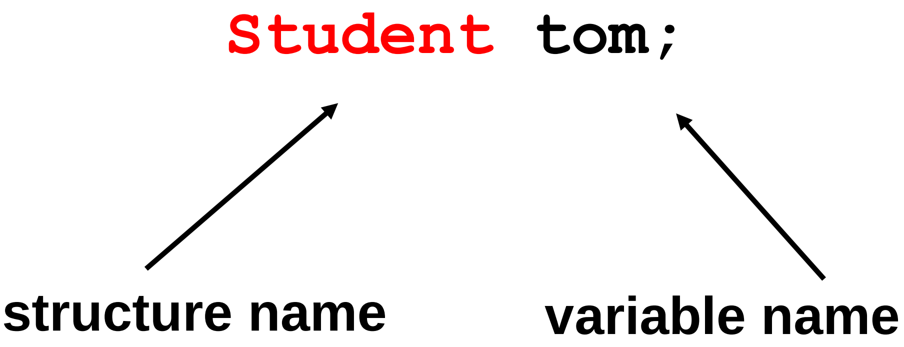
- Must have `;` after closing `}`
- **`struct`** names commonly begin with uppercase letter
- Multiple fields of same type can be in comma-separated list:

```
    string name, address;
```

Creating `struct` Variables

- `struct` declaration does not allocate memory or create variables

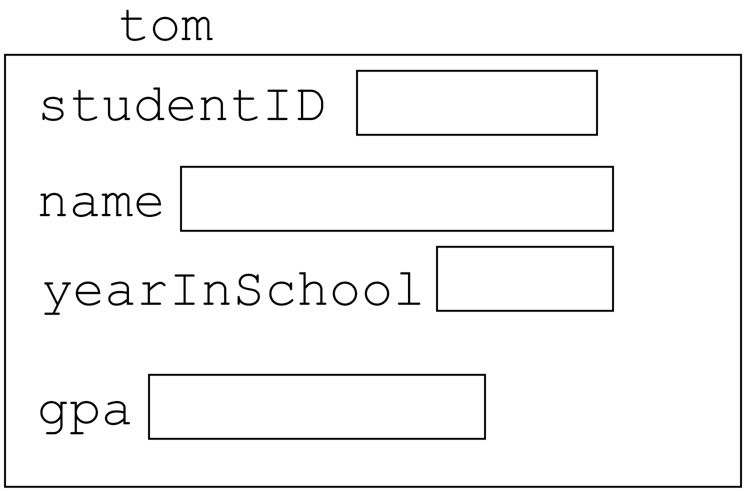
- Must create a struct variable
- To create variables, use structure name as type name



```
Student tom;
```

structure name

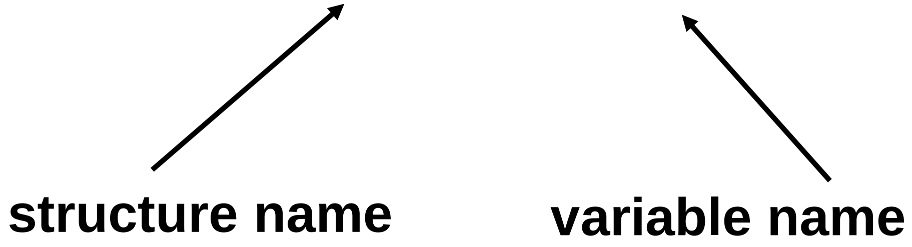
variable name

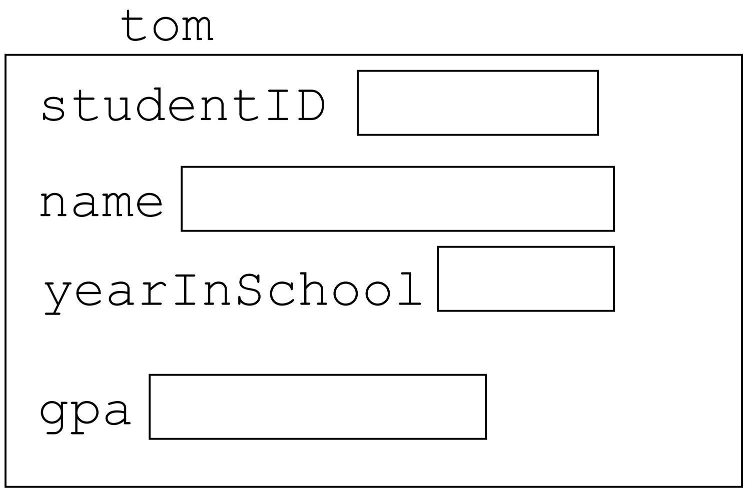


Creating `struct` Variables

- Must declare a structure before creating a structure variable

```
Student tom;
```





Creating `struct` Variables – Another way

- Can also create a structure variable with its declaration

```
struct Student
{  int studentID;
   string name;
   short yearInSchool;
   double gpa;
} student1;
```

Creating struct Variables Two Ways

```
struct Employee
{
    string    firstName;
    string    lastName;
    string    address;
    double    salary;
    int       deptID;
};

Employee  e1;
```

```
struct Student
{
    int studentID;
    string name;
    short yearInSchool;
    double gpa;
} s1, s2;
```

Accessing Structure Members

- Use the dot (.) operator to refer to members of **`struct`** variables:

```
cin >> s1.studentID;
s1.name = “Alex Stone”;
s1.gpa = 3.75;
```

- Member variables can be used in any manner appropriate for their data type

Displaying a struct Variable

- To display the contents of a **struct** variable, must display each field separately, using the dot operator:

```
cout << s1; // won't work
cout << s1.studentID << endl;
cout << s1.name << endl;
cout << s1.yearInSchool;
cout << " " << s1.gpa;
```

Initializing a Structure

Values should be in the **same sequence** as the structure

```
struct Student
{  int studentID;
   string name;
   short yearInSchool;
   double gpa;
};
```

- **struct** variable can be initialized when created

```
Student s1 = {11465, "Joan", 2, 3.75};
```

Initializing a Structure

Can also be initialized member-by-member after definition:

```
s1.name = "Joan";
s1.gpa = 3.75;
```

More on Initializing a Structure

- May initialize only some members:

```
Student s1 = {14579};
```

- Cannot skip over members:

```
// illegal
Student s1 = {1234, "John", , 2.83};
```

More on Initializing a Structure

- You can also give default values inside a struct definition

```
struct Student
{
   int studentID = 0;
   string name = “”;
   short yearInSchool = 1;
   double gpa = 1.0;
};
```

Accessing Structure Members

```
void main{

emp1.empNumber = 489;

emp1.name = “Jill Smith”;

emp1.hours = 23;

emp1.payRate = 20;

emp1.grossPay = emp1.hours * emp1.payRate;
}
```

```
struct PayRoll  {
 int empNumber;
 string name;
 double hours;
 double payRate;
 double grossPay;
} emp1;
```

Comparing `struct` Variables

- Cannot compare `struct` variables directly:

```
   if (s1 == s2) // won't work
```

- Instead, must compare on a field basis:

```
 if (s1.studentID == s2.studentID)
```

Assigning `struct` Variables

- A structure variable can be assigned to another structure variable only if both are of same type

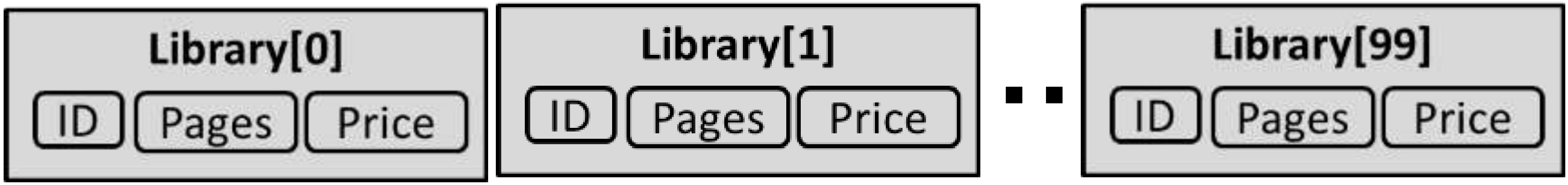
- A structure variable can be initialized by assigning another structure variable to it by using the assignment operator as follows:

```
Student s1 = { 1432, “Zoe”, 3, 2.99} ;
    Student s2 = s1;
```

Array of Structures

- An array of structures is a type of array in which each array element is a structure

```
struct Book
{       int ID;
        int Pages;
        float Price;
};
Book Library[100]; // declare array of
structures
```



Arrays of Structures

- Can be used in place of parallel arrays
  ```
  const int NUM_STUDENTS = 20;
  Student stuList[NUM_STUDENTS];
  ```

- Individual structures in an array accessible using subscript notation

- Fields within structures accessible using dot notation:
  ```
  cout << stuList[5].studentID;
  ```

Array of Structures

```
struct  Book
{    int      ID;
     int      Pages;
     float   Price;
};
Book b[3]; //declare of array of structures
```

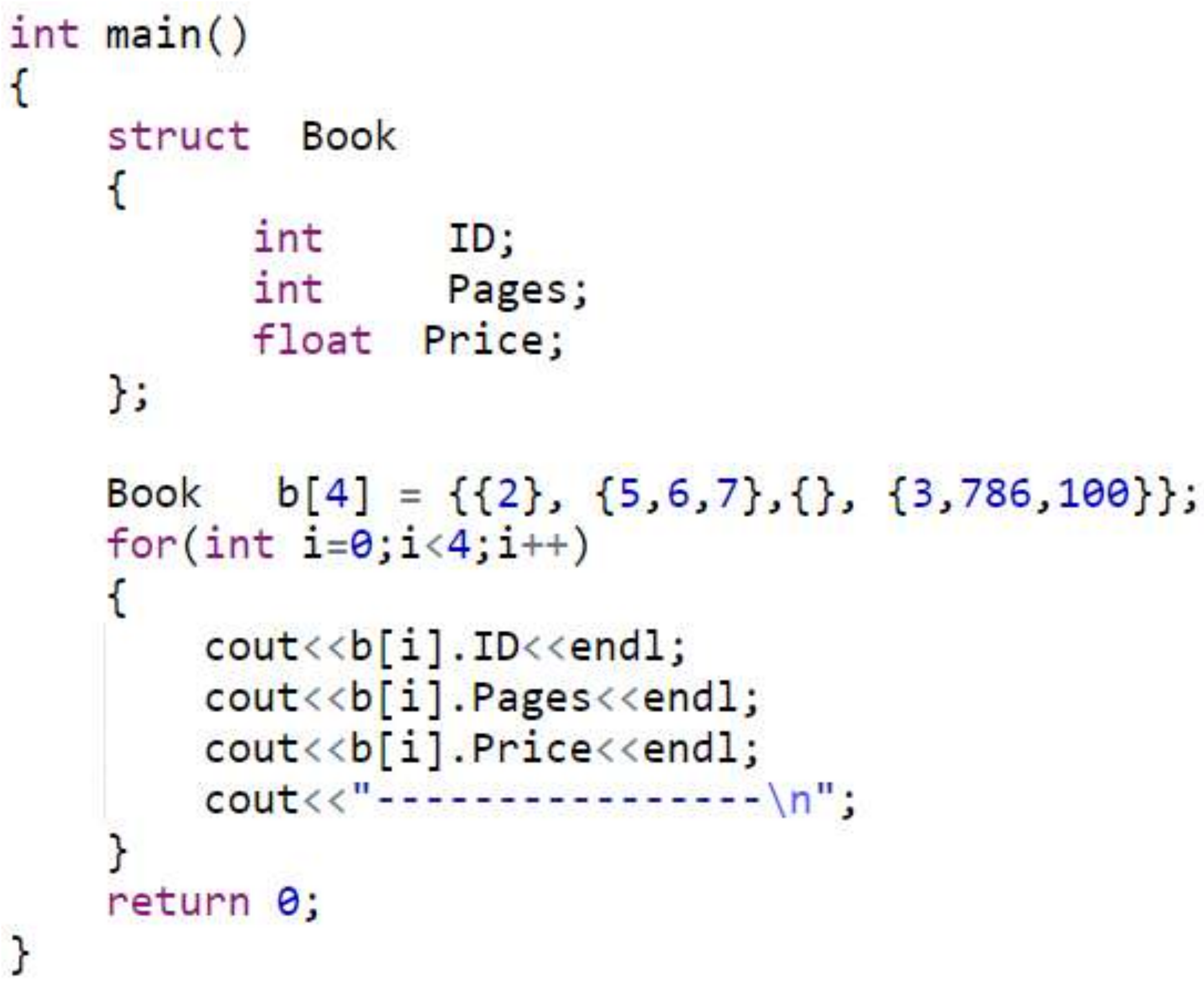
- Initializing can be at the time of declaration

  Book b[3] = { {1,275,70} , {2,600,90} , {3,786,100} };

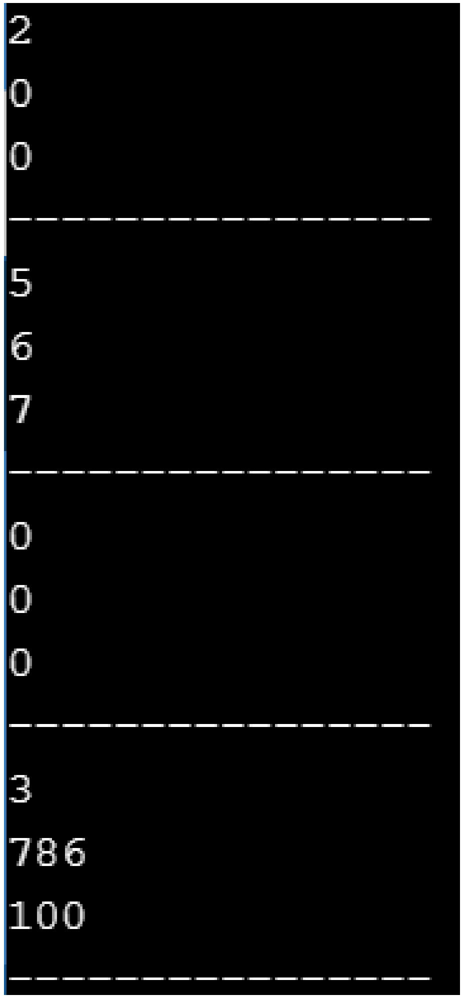
- Or can be assigned values using cin:

```
cin >> b[0].ID;
cin >> b[0].Pages;
```

Partial Initialization of Array of Structures



```
int main()
{
    struct  Book
    {
            int     ID;
            int     Pages;
            float  Price;
    };

    Book   b[4] = {{2}, {5,6,7},{}, {3,786,100}};
    for(int i=0;i<4;i++)
    {
        cout<<b[i].ID<<endl;
        cout<<b[i].Pages<<endl;
        cout<<b[i].Price<<endl;
        cout<<"-----------------\n";
    }
    return 0;
}
```



```
2
0
0
-----------------
5
6
7
-----------------
0
0
0
-----------------
3
786
100
-----------------
```

Array as Member of Structures

- A structure may also contain arrays as members.

```
struct  Student
{
    int    RollNo;
    float  Marks [3];
};
```

- Initialization can be done at time of declaration:

```
     Student s1 = {1, {70.0, 90.0, 97.0} };
```

Array as Member of Structures

- Can also assigned values later in the program:

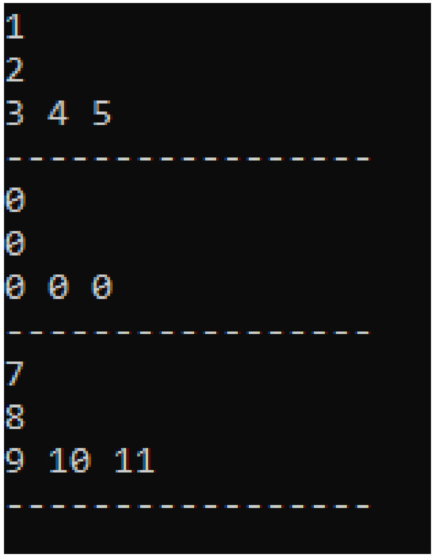
```
Student    s1;
s1.RollNo = 1;
s1.Marks[0] = 70.0;
s1.Marks[1] = 90.0;
s1.Marks[2] = 97.0;
```

- Or user can use cin to get input directly:

```
cin >> s1.RollNo;
cin >> s1.Marks[0];
cin >> s1.Marks[1];
cin >> s1.Marks[2];
```

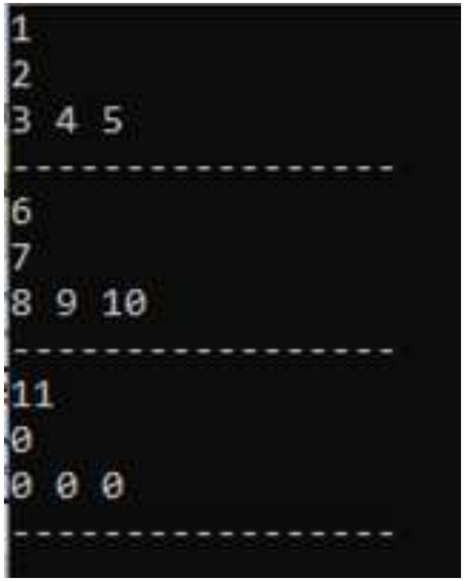
Array as Member of Structures

```
struct Student {
       int age;
       int marks;
       int arr[3];
};
void main() {
Student s[3] = { 1,2,3,4,5,{},7,8,9,10,11 };
for (int i = 0; i < 3; i++) {
       cout << s[i].age << endl;
       cout << s[i].marks << endl;
       cout << s[i].arr[0] << " " << s[i].arr[1] << "
       " << s[i].arr[2] << endl;;
       cout<<“------------”<<endl; }
}
```



```
1
2
3 4 5
-----------------
0
0
0 0 0
-----------------
7
8
9 10 11
-----------------
```

Array as Member of Structures



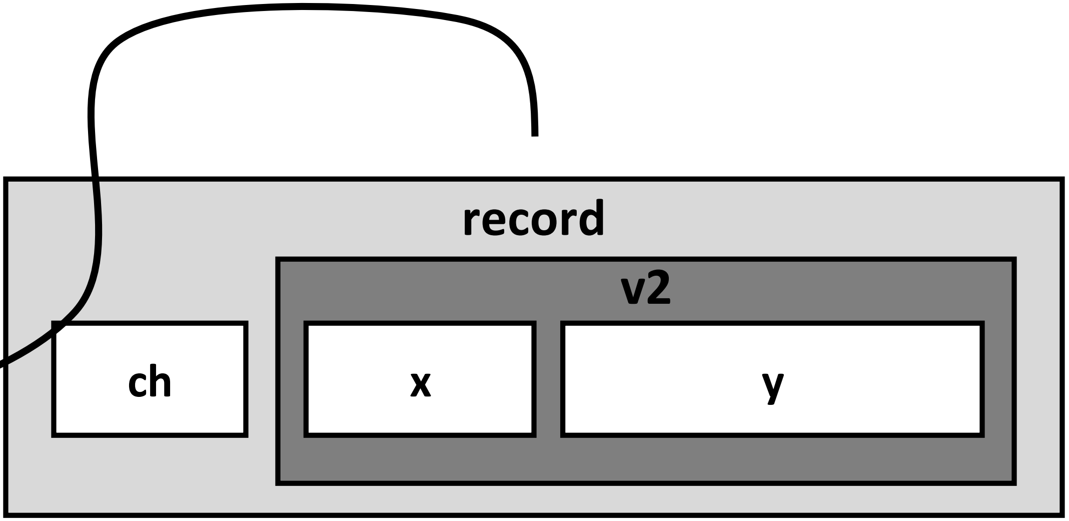
```
1
2
3 4 5
-----------------
6
7
8 9 10
-----------------
11
0
0 0 0
-----------------
```

```
struct Student {
      int age;
      int marks;
      int arr[3];
};
void main() {
Student s[3] = { 1,2,3,4,5,6,7,8,9,10,11 };
for (int i = 0; i < 3; i++) {
      cout << s[i].age << endl;
      cout << s[i].marks << endl;
      cout << s[i].arr[0] << " " << s[i].arr[1] << "
      " << s[i].arr[2] << endl;;
      cout<<“------------”<<endl;   }
}
```

Nested Structure

- A structure can be a member of another structure: called nested structure

```
struct A
{
   int    x;
   double y;
};

struct B
{
    char   ch;
    A      v2;
};

 B  record;
```



Initializing/Assigning to a Nested Structure

```
struct A{
   int x;
   float y;
};

struct B{
   char ch;
   A    v2;
};
```

```
void main()
//Initialization
{
B record ={‘S’,{100, 3.6}};
}
```

```
void main() // Input
{
 B record;
 cin >> record.ch;
 cin >> record.v2.x;
 cin >> record.v2.y;
}
```

```
void main()
//Assignment
{
 B record;
 record.ch = ‘S’;
 record.v2.x = 100;
 record.v2.y = 3.6;
}
```

Pointers to Structures

- A structure variable has an address
- Pointers can be used to point to structure variables.
- Pointers to structures are variables that can hold the address of a structure

```
 Student *stuPtr;
```

The **`stuPtr`** pointer can point at variables of the type **`Student`**

Accessing Structures with Pointers

- The pointer variable should be of the type:

```
                 Your Structure

struct Rectangle {
       int width;
       int height;
};

void main( )
{
        Rectangle rect1 = {22,33};
        Rectangle* rect1Ptr = &rect1;
}
```

Accessing Structures with Pointers

- How to access the structure members (using pointer)?
  - Use dereferencing operator (*) with dot operator (.)

```
struct Rectangle {
    int width;
    int height;
};

void main( )
{
     Rectangle rect1 = {22,33};
     Rectangle* rect1Ptr = &rect1;
     cout<<(*rectPtr1).width << endl;
     cout<<(*rectPtr1).height << endl;
}
```

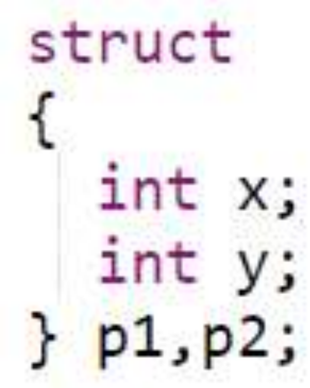
Accessing Structures with Pointers

- Is there some easier way to do this?
  - Use arrow operator (->) instead of * and .

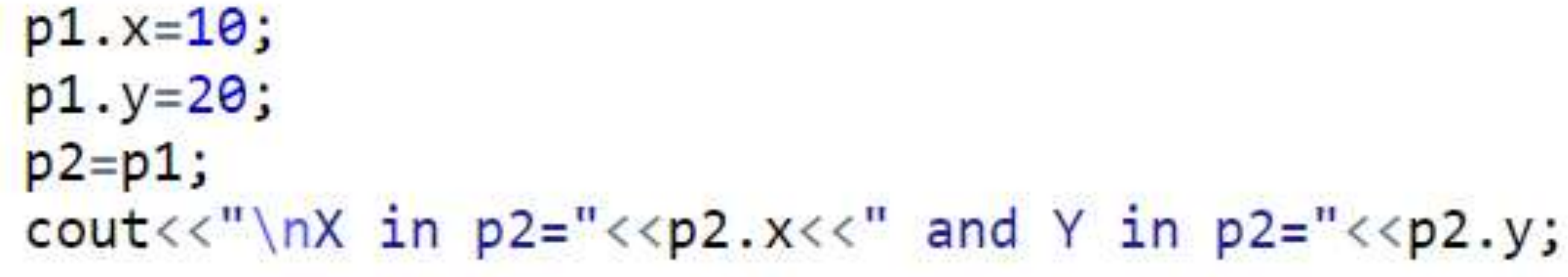
```
struct Rectangle {
    int width;
    int height;
};

void main( )
{
     Rectangle rect1 = {22,33};
     Rectangle* rect1Ptr = &rect1;
     cout<< rectPtr1->width << endl;
     cout<< rectPtr1->height << endl;
}
```

Anonymous Structure

- Structures can be anonymous
- Must create variable after declaration



```
struct
{
    int x;
    int y;
} p1,p2;
```



```
p1.x=10;
p1.y=20;
p2=p1;
cout<<"\nX in p2="<<p2.x<<" and Y in p2="<<p2.y;
```

Other Stuff You Can Do With a `struct`

- You can also associate functions with a structure (called member functions)

Quick Example

```
struct StudentRecord {
   string name;          // student name
   int marks[5];         // test grades
   double ave;           // final average

   void print_ave( ) {
      cout << "Name: " << name << endl;
      cout << "Average: " << ave << endl;
    }
};
```

Using a Member Function

- Use the dot operator to call member functions of a struct

```
StudentRecord stu;

stu.print_ave( );
```

Structures as Function Arguments

- May pass members of **struct** variables to functions:

```
//function definition
float computeGPA(float gpa){
……………
}

//function call
computeGPA(s1.gpa);
```

Structures as Function Arguments

- May pass entire `struct` variables to functions

1. Pass-by-value
2. Pass-by-reference
3. Pass-using pointers

Structures as Function Arguments – Pass by Value

```
struct Rectangle {
   double length;
   double width;
   double area;
};
void changeRect(Rectangle r) {
   r.length = 5;
   r.width = 6;
   r.area = 30;
}
void main(){
Rectangle box = {1, 2, 2};
changeRect(box); }
```

A **copy of the struct box** is created and saved in the function parameter **r**

Structures as Function Arguments – Pass by Value

```
void changeRect(Rectangle r) {
   r.length = 5;
   r.width = 6;
   r.area = 30;
}
void main(){
Rectangle box = {1, 2, 2};
changeRect(box);

cout << box.length << endl; prints 1
cout << box.width << endl; prints 2
cout << box.area << endl; prints 2
}
```

Structures as Function Arguments – Pass by Reference

```
struct Rectangle {
   double length;
   double width;
   double area;
};
void changeRect(Rectangle &r) {
   r.length = 5;
   r.width = 6;
   r.area = 30;
}

Rectangle box = {1, 2, 2};
changeRect(box);
```

The **actual struct variable box is passed by reference**
(parameter **r** is just another name for box)

Structures as Function Arguments – Pass by Value

```
void changeRect(Rectangle &r) {
   r.length = 5;
   r.width = 6;
   r.area = 30;
}
void main(){
Rectangle box = {1, 2, 2};
changeRect(box);

cout << box.length << endl; prints 5
cout << box.width << endl; prints 6
cout << box.area << endl; prints 30
}
```

Structures as Function Arguments - Notes

- Passing a structure to a function by value can slow down a program, waste space

- Passing a structure to a function by reference will speed up program, but the function may change data in structure

- Using a **`const`** reference parameter allows read-only access to reference parameter, it is fast and does not waste space

Structures as Function Arguments – Pass by const Reference

```
void changeRect(const Rectangle &r) {
   r.length = 5;//ERROR! Cannot modify const
   r.width = 6; //ERROR! Cannot modify const
   r.area = 30; //ERROR! Cannot modify const
}
void main(){
Rectangle box = {1, 2, 2};
changeRect(box);


}
```

Structures as Function Arguments – Pass by `const` Reference

```
void showRect(const Rectangle &r) {
   cout << r.length << endl;
   cout << r.width << endl;
   cout << r.area << endl;
}
void main(){
Rectangle box = {1, 2, 2};
showRect(box);
}
```

Output:

```
1
2
2
```

Returning a Structure from a Function

- A Function can return a **`struct`**:

```
Student getStudentData();  // prototype
stu1 = getStudentData();   // call
```

- Function must define a local structure variable
  - for internal use
  - for use with **`return`** statement

Returning a Structure from a Function - Example

```
Student getStudentData()
{
    Student tempStu;

    cin >> tempStu.studentID;
    cin >> tempStu.yearInSchool;
    cin >> tempStu.gpa;

    return tempStu;
}
```

Practice Question 1

- Define a structure called **“car”**. The member elements of the car structure are:
  - string Model;
  - int Year;
  - float Price

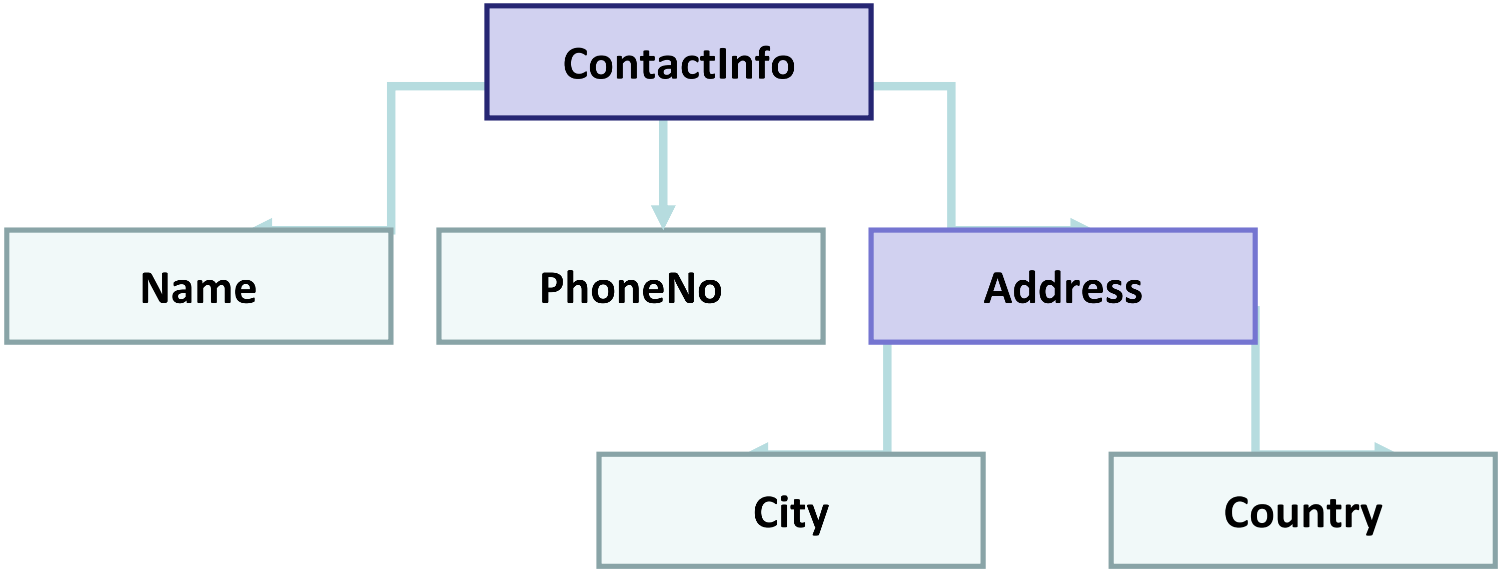
- Create an array of 30 cars. Get input for all 30 cars from the user. Then the program should display complete information (***Model***, ***Year***, ***Price***) of those cars only which are above 500,000 in price.

Practice Question 1

```
struct Car {
        string model;
        int year;
        float price;
};
void main() {
Car showroom[30]; //array of cars
for (int i = 0; i < 30; i++) {
        cin >> showroom[i].model;
        cin >> showroom[i].year;
        cin >> showroom[i].price;
}
for (int i = 0; i < 30; i++) {
        if (showroom[i].price > 500000) {
                cout << showroom[i].model<<" "<< showroom[i].year <<" "
                <<showroom[i].price;
                }
        }
}
```

Practice Question 2

- Write a program that implements the following using C++ struct. The program should finally displays contactInfo values for 10 people.



Practice Question 2

```
struct Address {
         string city;
         string country;   };
struct ContactInfo {
         string name;
         long int number;
         Address address;  };
void main() {
ContactInfo phonebook[10];
for (int i = 0; i < 10; i++) {
         cin >> phonebook[i].name;
         cin >> phonebook[i].number;
         cin >> phonebook[i].address.city;
         cin >> phonebook[i].address.country;
}
for (int i = 0; i < 30; i++) {
cout << phonebook[i].name << " " << phonebook[i].number << " "
<< phonebook[i].address.city << " " << phonebook[i].address.country
<< endl;;
}        }
```