Object Oriented Programming

STRUCTURES - REVISION

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 items of different types of elements.

Abstract Data Type

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

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

All these variables hold data about the same employee

Combining Data into Structures

- Structure: is a **user-defined data type**. It is like a container that allows multiple variables to be grouped together. Structures are used to organize **related data** (variables) into a nice neat package.

- Variables can be of any type

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

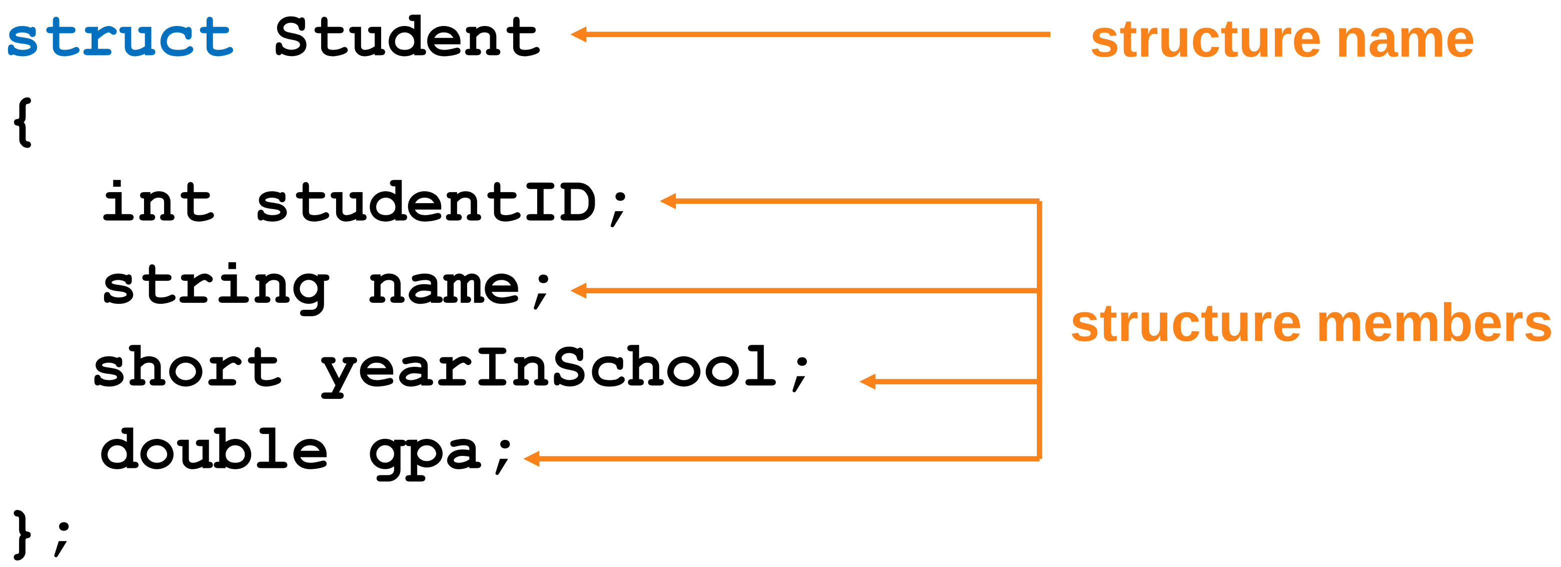
Introducing Structures

A **structure** is a collection and is referenced with **single name**.

The data items in structure are called structure **members**, **elements**, or **fields**.

The difference between **array** and **structure**: is that ***array must consists*** *of a* ***set of values*** *of same data type* but on the other hand, **structure may consist of different data types**

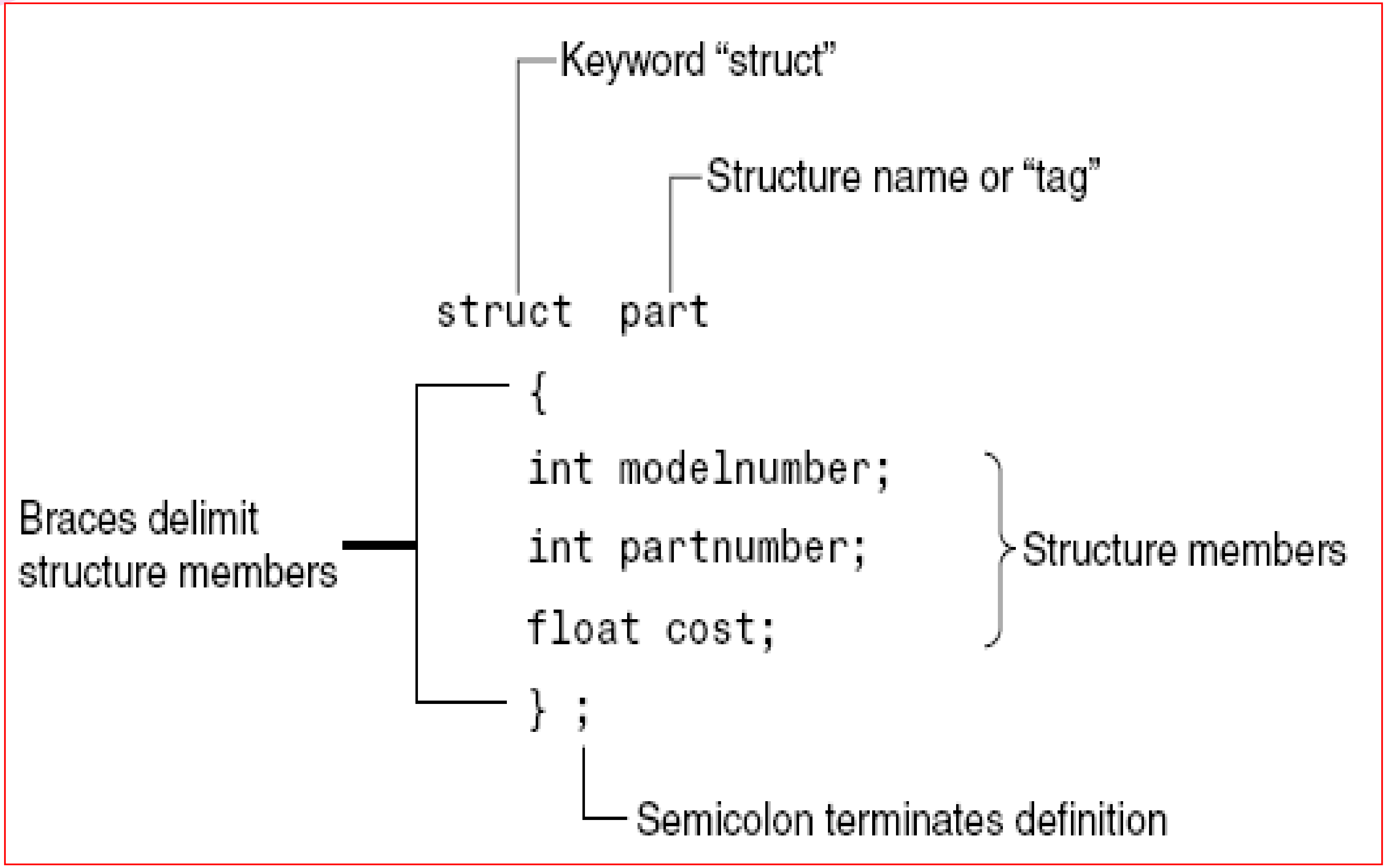
Example struct Declaration



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

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

Structure Definition Syntax



```
struct  part
{
int modelnumber;
int partnumber;
float cost;
} ;
```

Keyword “struct”
Structure name or “tag”
Braces delimit structure members
Structure members
Semicolon terminates definition

struct Declaration Notes

- Made in global scope usually, so can access the user-defined datatype in all functions

- Must have `;` after closing `}`

- **`struct`** names commonly begin with uppercase letter

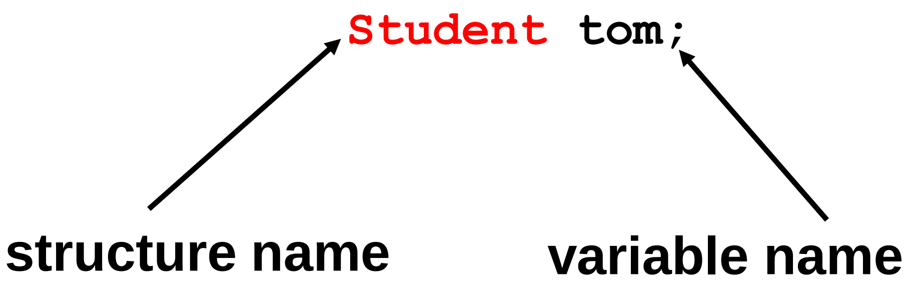
- Multiple fields of same type can be in comma-separated list:

```
    string name, address;
```

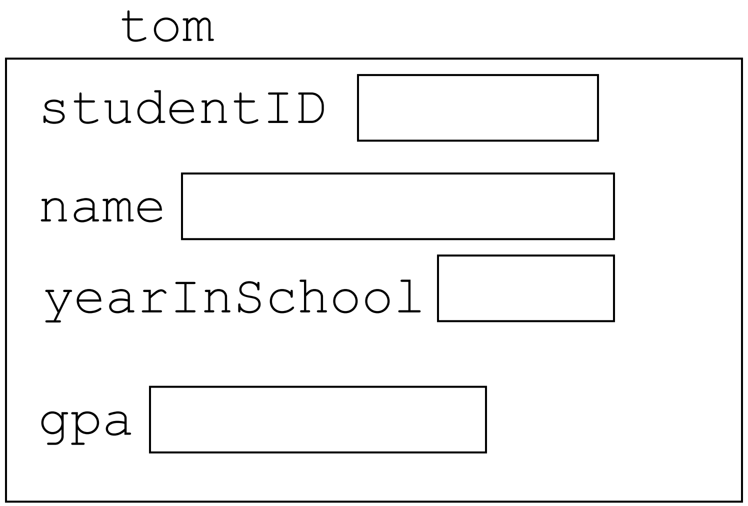
Creating struct Variables

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

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

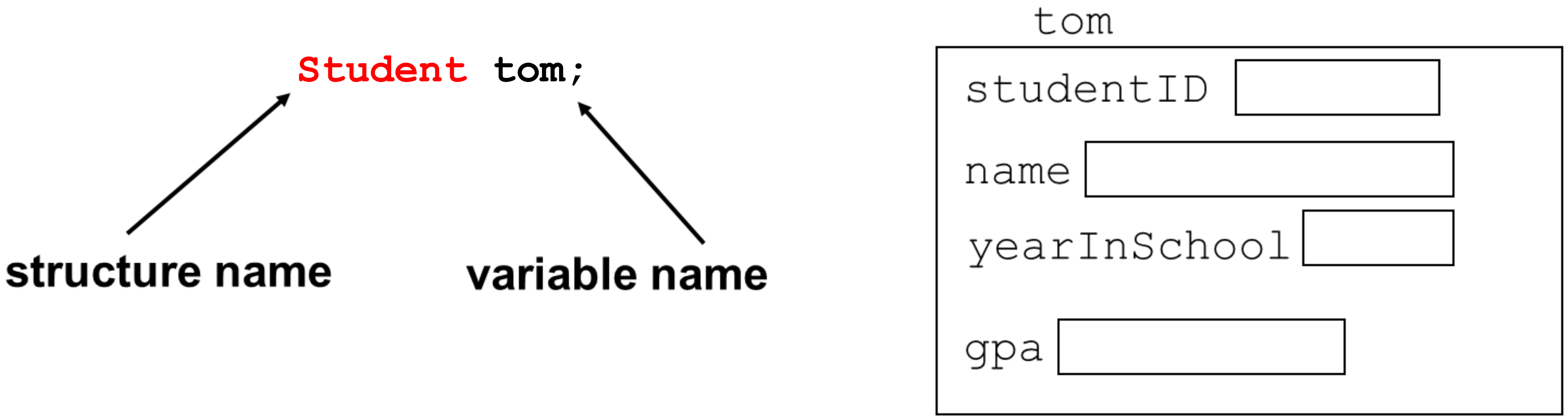


```
Student tom;
```



Creating struct Variables

- Must declare a structure before creating a structure variable

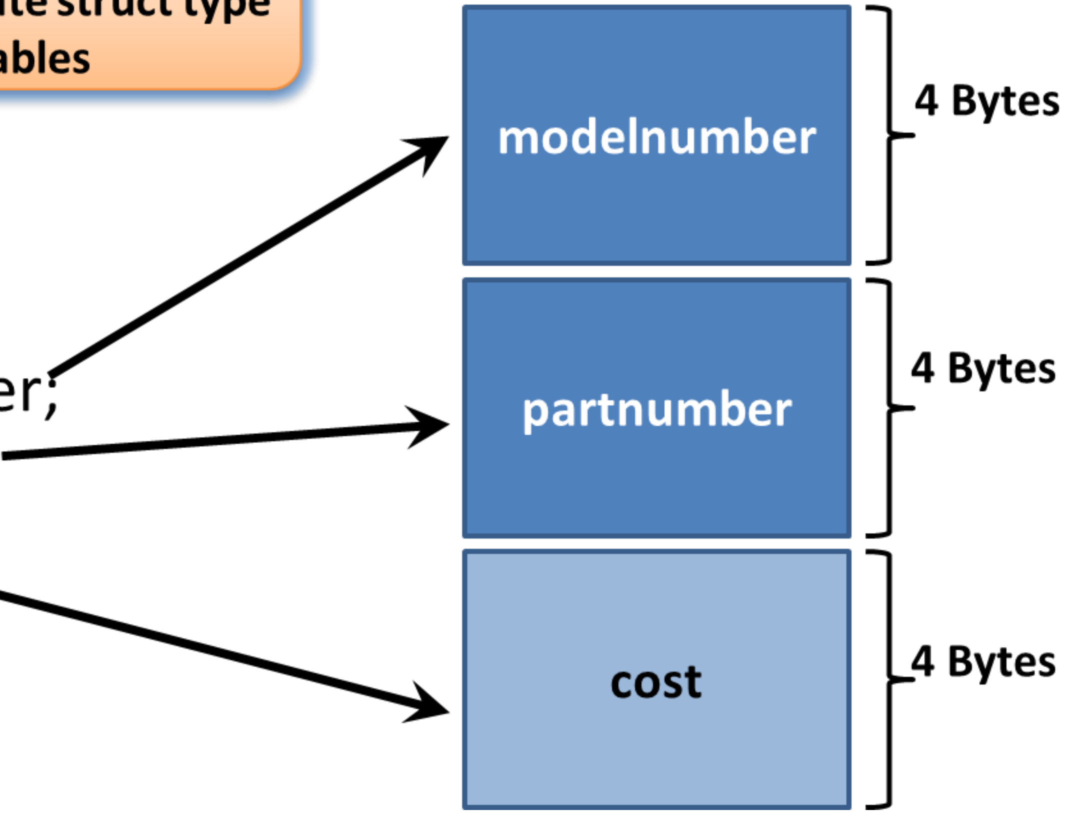


```
Student tom;
```

Structure Members In Memory

Memory is only allocated when we create struct type variables

```
struct   part
{
         int       modelnumber;
         int       partnumber;
         float     cost;
};
```

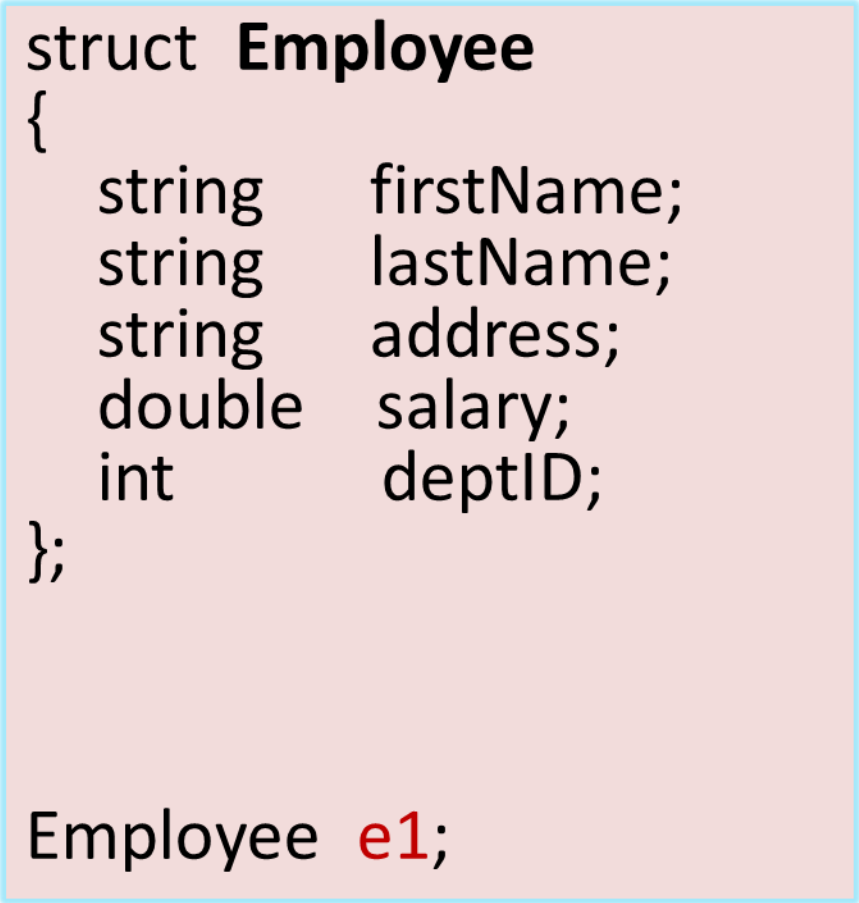


Creating struct Variables – Another way

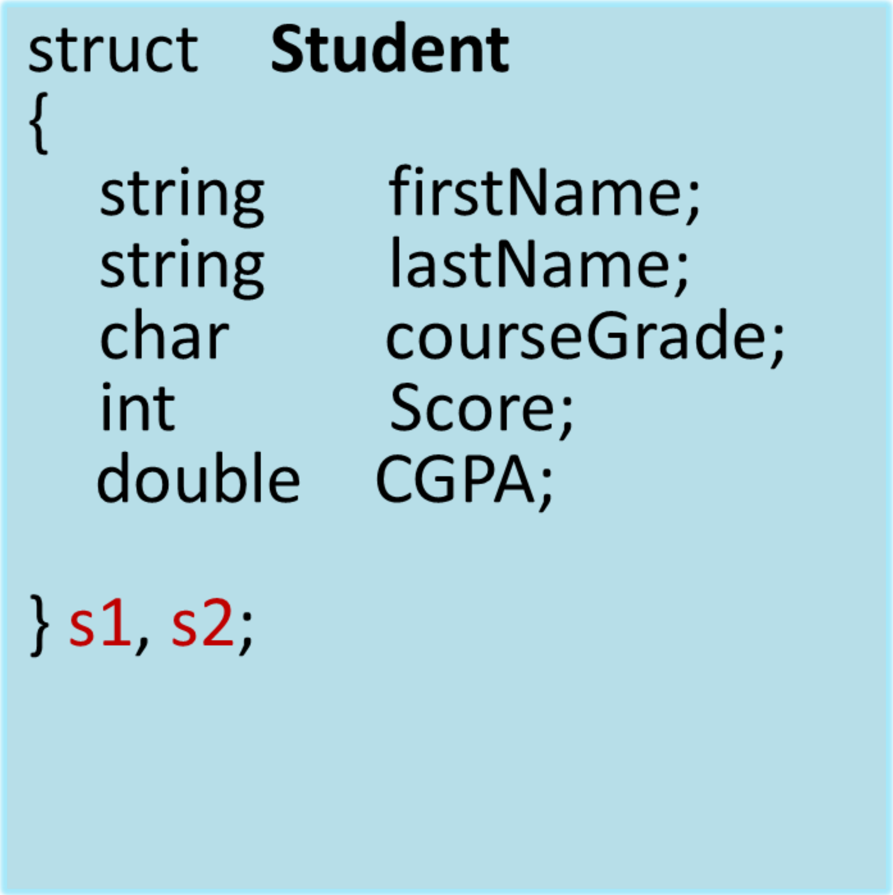
- Can also create a structure variable with its declaration

```
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;
```



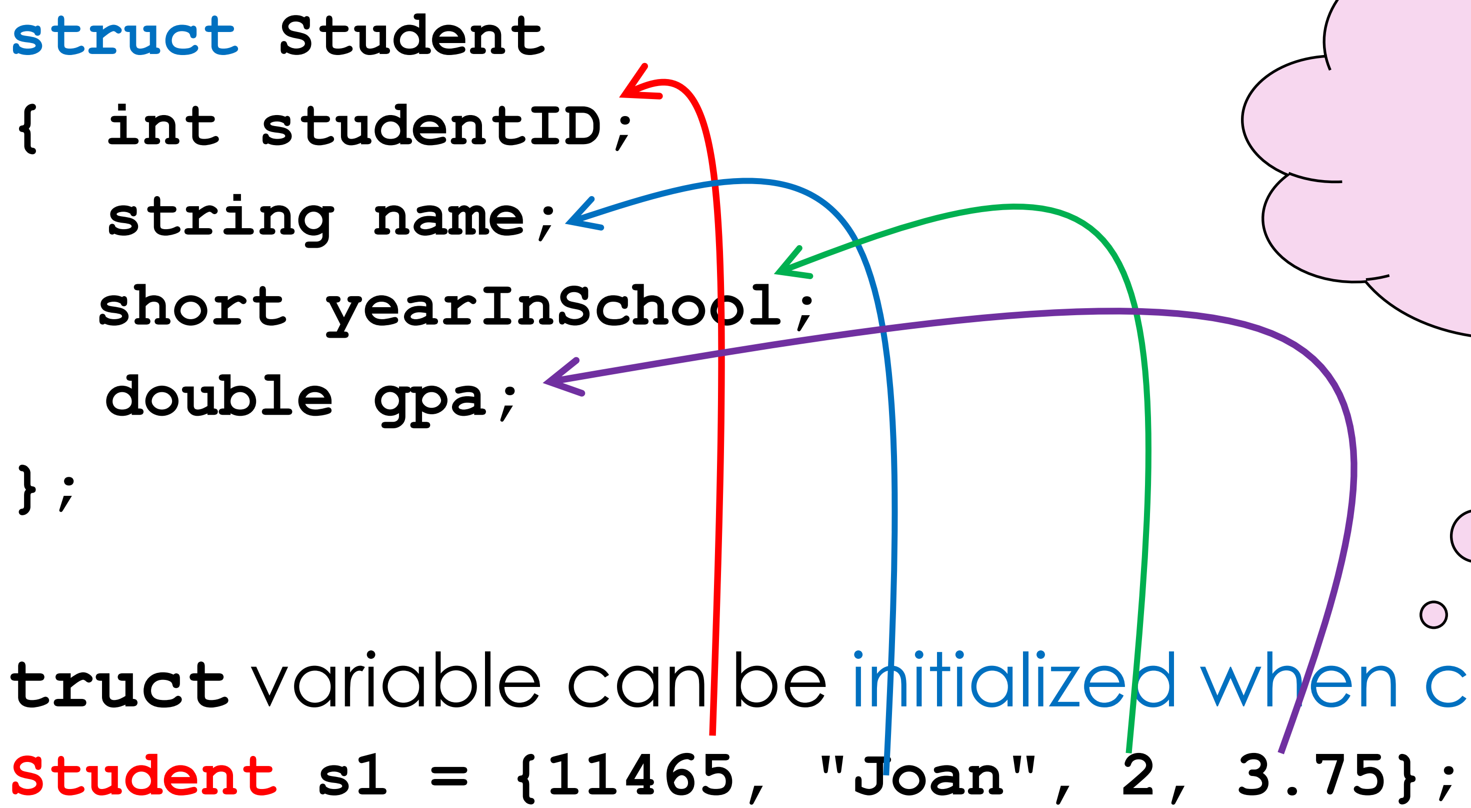
```
struct   Student
{
    string      firstName;
    string      lastName;
    char        courseGrade;
    int         Score;
    double     CGPA;

} s1, s2;
```

Initializing a Structure

- The syntax of **initializing structure** is:

```
StructName  struct_identifier  =  {Value1, Value2, …};
```

Initializing a 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};
```

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

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

Assigning Values to Structure Variables

- After creating structure variable, values to structure members can be assigned using ***cin***
- Output to screen using ***cout***

```
 student s1;

cin>>s1.firstName;
cin>>s1.lastName;
cin>>s1.courseGrade;
cin>>s1.marks ;

cout<<s1.firstName<<s1.lastName;
```

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;
```

Assigning one Structure Variable to another

- **structure variable** can be **assigned** to **another structure variable** ***<u>only if both are of same type</u>***
- A **structure variable** can be **initialized** by **assigning** another **structure variable** to it by **using** the **assignment operator** as follows:

Example:

```
studentType    Student1 = {“Amir”, “Ali”, ‘A’, 98} ;
studentType    student2 = Student1;
```

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)
```

Practice Question

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

- Create a variable of type Car called c1

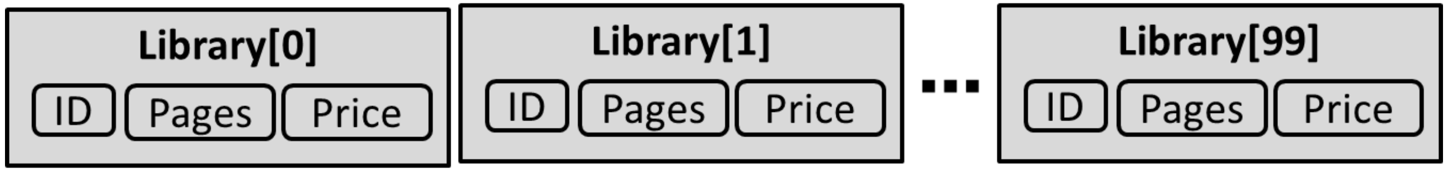
- Get input for all structure members from the user

- Then the program should display complete information (***Model***, ***Year***, ***Price***) of c1

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
```



Initialization of Array of Structures

- Can be used in place of parallel arrays

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

Initialization of Array of Structures

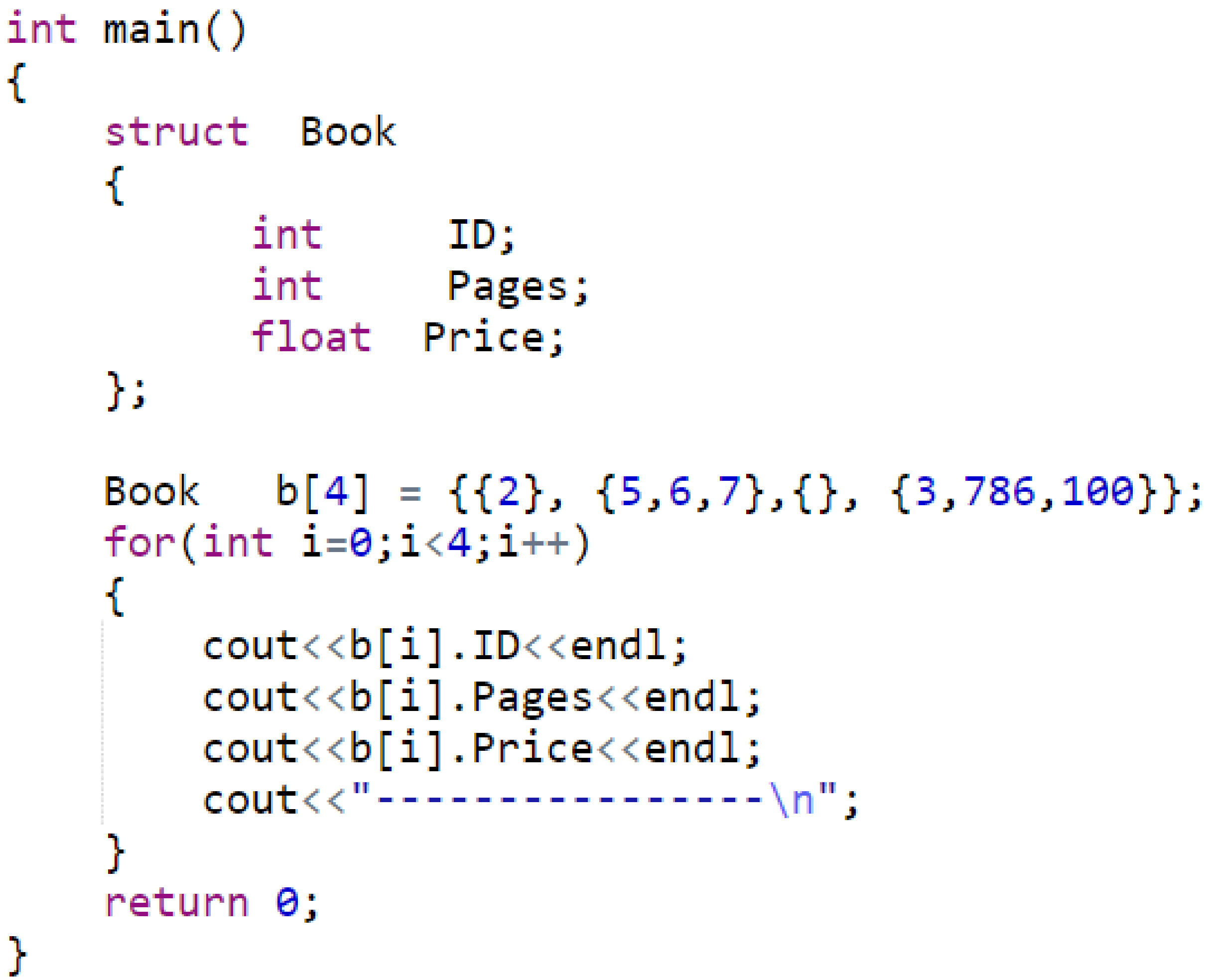
- **Initializing can be at the time of declaration**

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

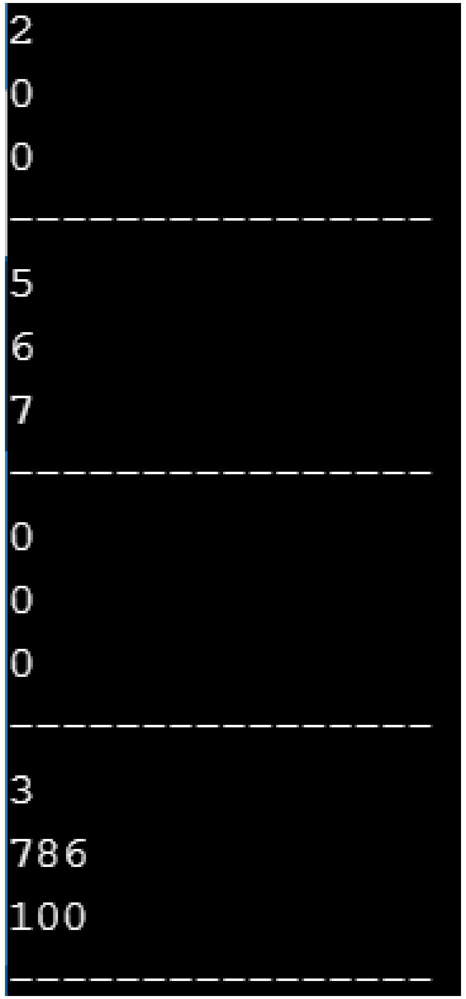
- Or can be assigned values using ***cin***:

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

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
----------------
```

Practice Question

- Define a structure called **“Car”** in global scope. The member elements of the car structure are:
  - string Model;
  - int Year;
  - float Price
- Create an array of 30 cars called showroom. 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

```
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;
                }
        }
}
```

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   S   = {1,   {70.0, 90.0, 97.0}  };
```

Array as Member of Structures

- Can also assigned values later in the program:

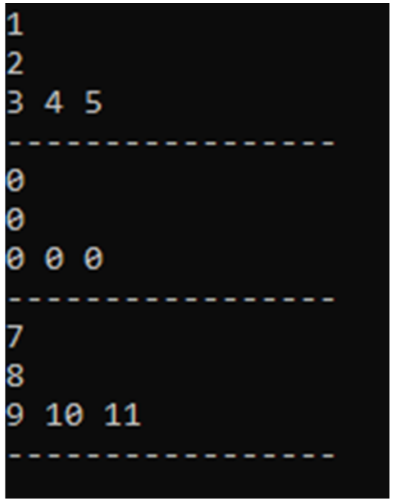
```
Student   s1;
s1.RollNo = 1;
s1.Marks[0] = 70.0;
s1.Marks[1] = 90.0;
s1.Marks[2] = 97.0;
```

- 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

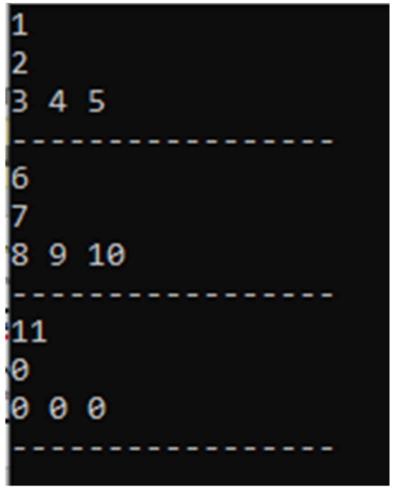
```
struct Student {
      int age;
      int rollNum;
      int marks[3];
};
void main() {
Student s[3] = { 1,2,3,4,5,{},7,8,9,10,11 };
for (int i = 0; i < 3; i++) {
      cout << s[i].age << endl;
      cout << s[i].rollNum << endl;
      cout << s[i].marks[0] << "  " << s[i].marks[1]
<< "   " << s[i].marks[2]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



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

Array as Member of Structures

```
struct Student {
      int age;
      int rollNum;
      int marks[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].rollNum << endl;
      cout << s[i].marks[0] << " " << s[i].marks[1]
<< "   " << s[i].marks[2] << endl;;
      cout<<“------------”<<endl;   }
}
```

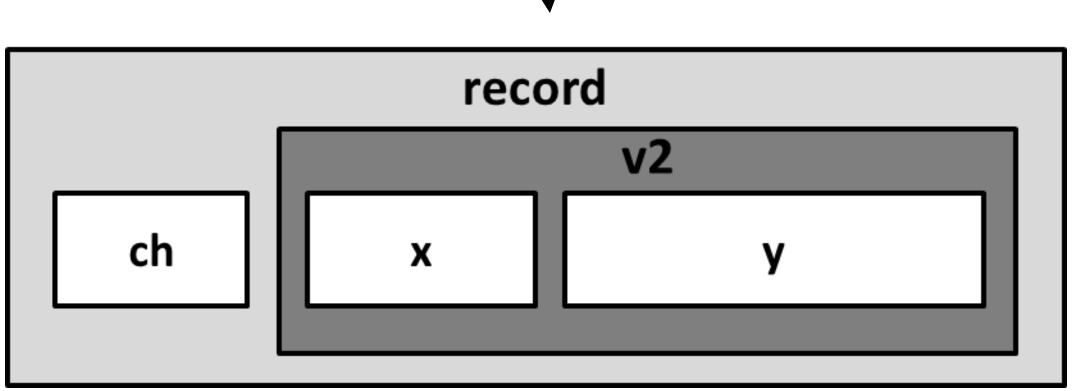


```
1
2
3 4 5
----------------
6
7
8 9 10
----------------
11
0
0 0 0
----------------
```

Nested Structure

- A structure variable 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;
};
```

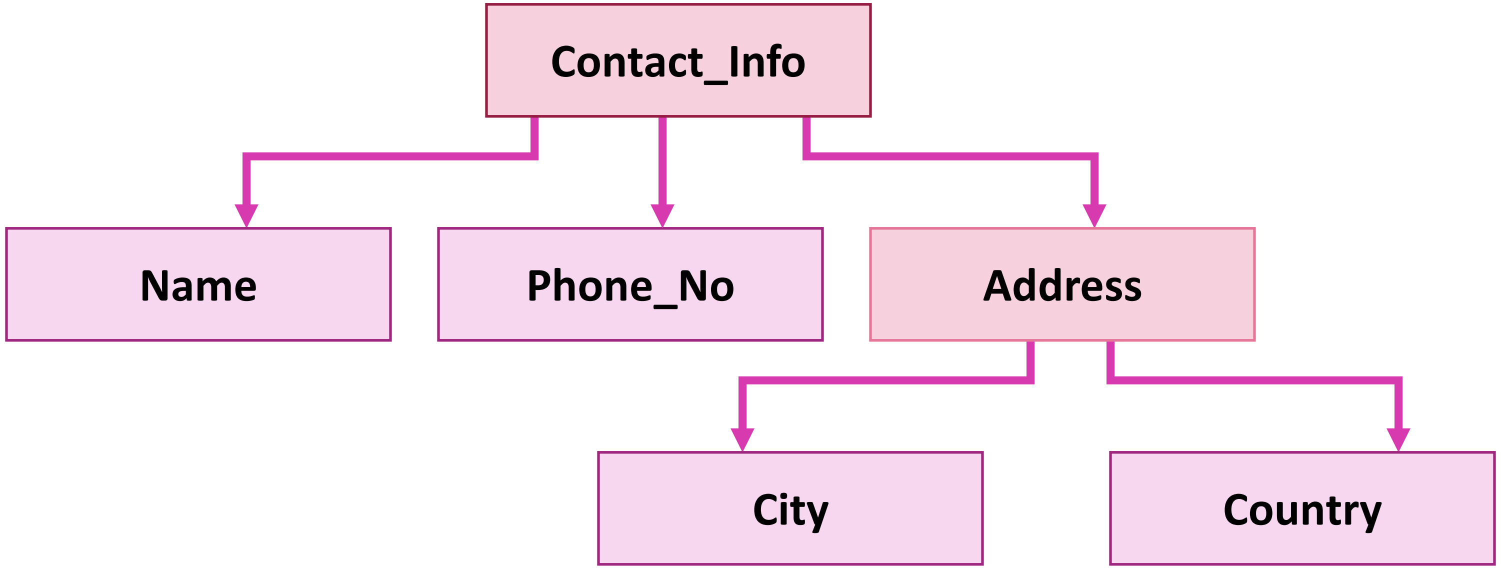
```
void main() // Input
{
      B   record;
      cin>>record.ch;
      cin>>record.v2.x;
      cin>>record.v2.y;
}
```

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

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

Practice Question

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



Practice Question

```
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;;
}       }
```

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;
}
```

Dynamic Memory Allocation

- The pointer variable can be used to dynamically allocate memory for a structure variable:

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

void main( )
{
Rectangle* rect1Ptr = new Rectangle;
rect1Ptr->width=22;
rect1Ptr->height=33;
}
```

Pointer as Structure Member

- Pointers can also be a member of a structure

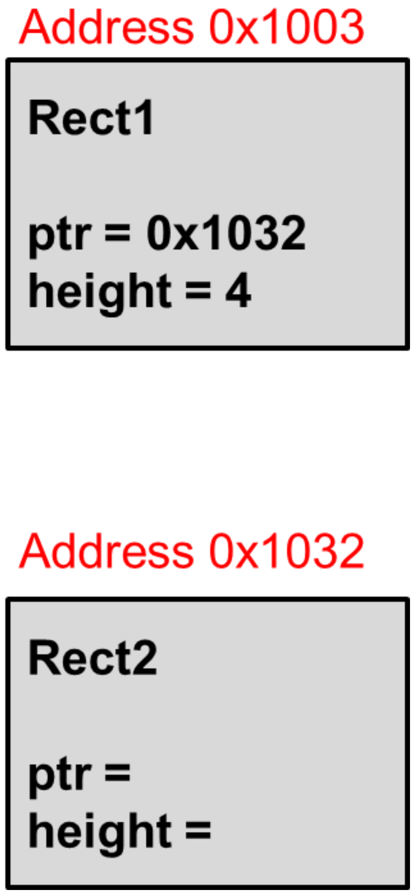
```
struct Rectangle {
    int *width;
    int height;
};

void main( )
{    int w = 3;
     Rectangle rect1 = {&w,5}
     *rect1.width = 2; //dot has higher precedence
     rect1.height = 4;
}
```

Address of variable **w** is assigned to the **pointer width**

Pointer as Structure Member

- Pointers can also be a member of a structure

```
struct Rectangle {
    Rectangle *ptr;
    int height;
};

void main( )
{     int w = 3;
      Rectangle rect1,rect2;
      rect1.height = 4;
      rect1.ptr = &rect2;
}
```



Address 0x1003

Rect1

ptr = 0x1032
height = 4

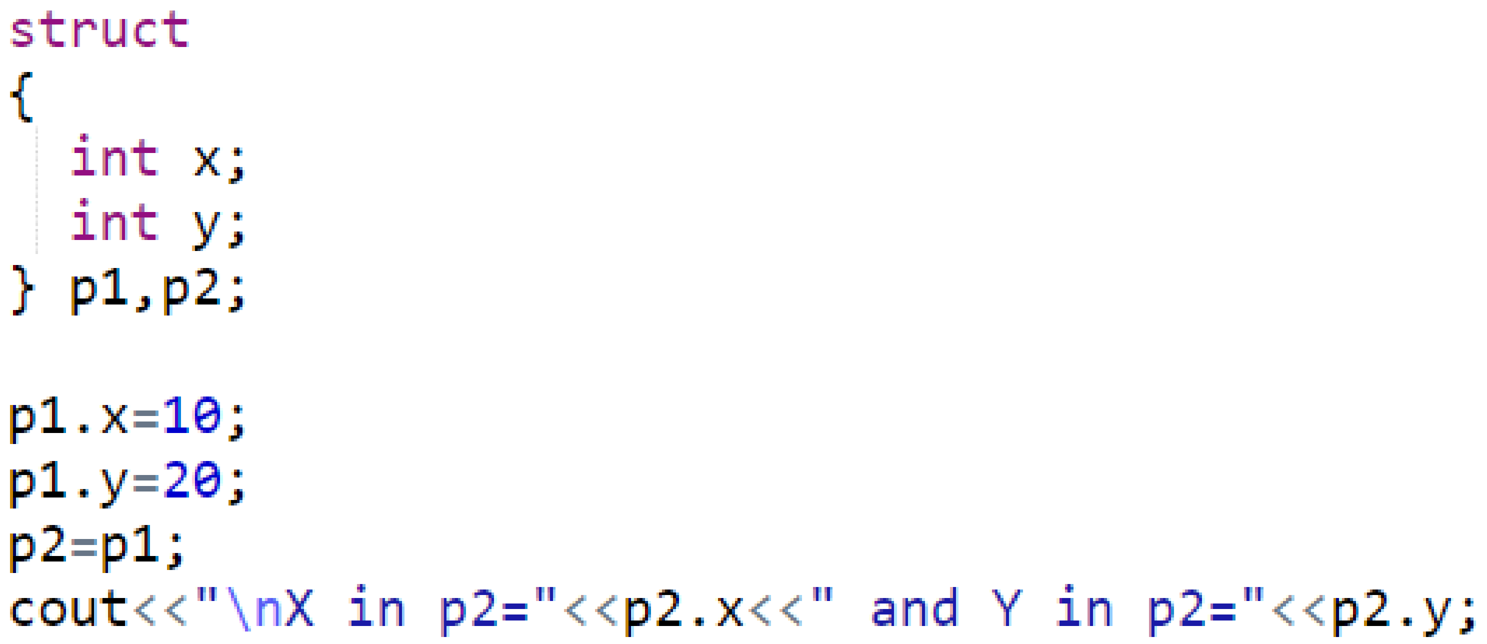
Address 0x1032

Rect2

ptr =
height =

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;
    }
};
```