Structures

Department of Computer Science,
National University of Computer & Emerging Sciences,
Islamabad Campus

Structure

- A **Structure** is a **container**, it can **hold a bunch of *things***.
  - These **things** can be of **any type**.

- **Structures** are used to **organize** **related data** (variables) into a **nice neat package**.

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**.

Defining the Structure

- Structure definition tells how the structure is organized: it specifies what members the structure will contain.

Syntax & Example →

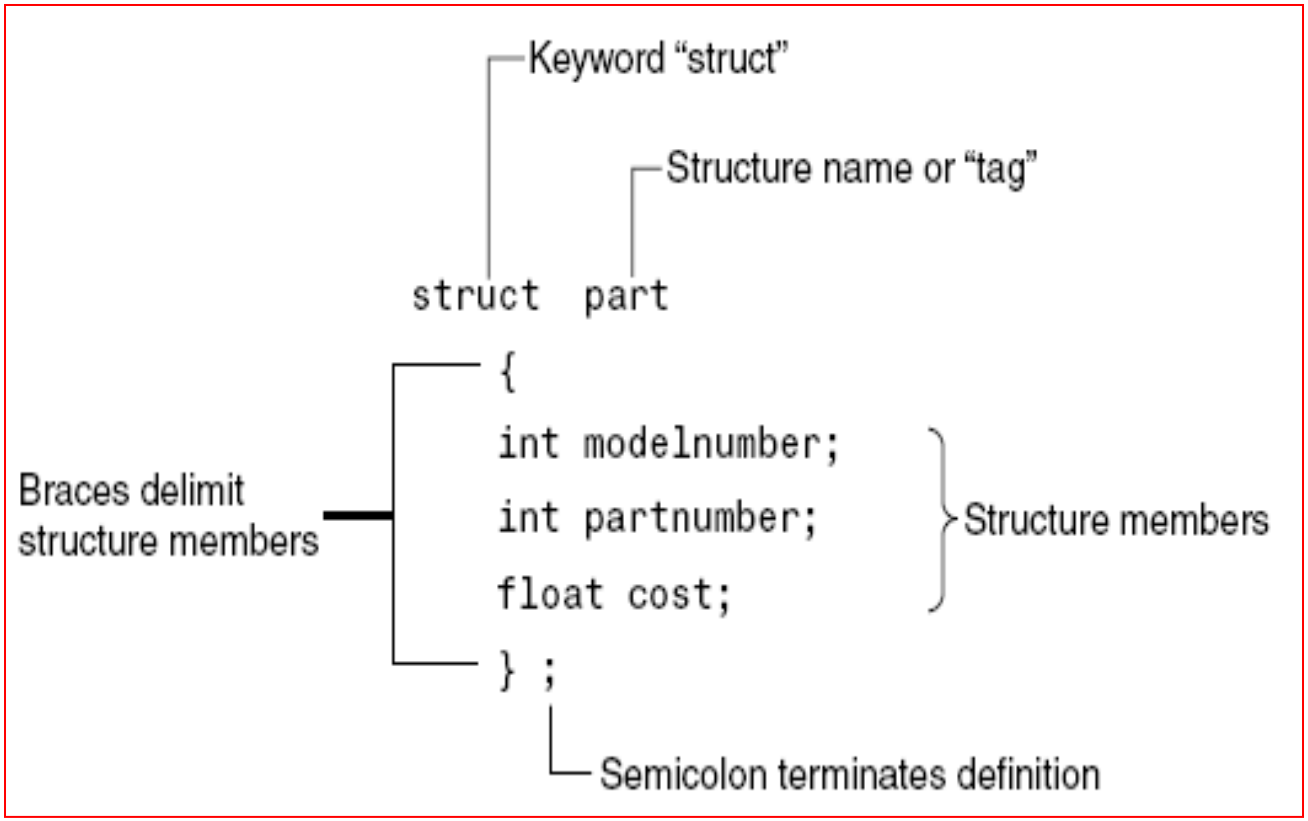
Syntax:

```
struct  StructName
{
    DataType1    Identifier1;
    DataType2    Identifier2;
    .
    .
    .
};
```

Example:

```
struct   Product
{
        int       ID;
        string    name;
        float     price;
};
```

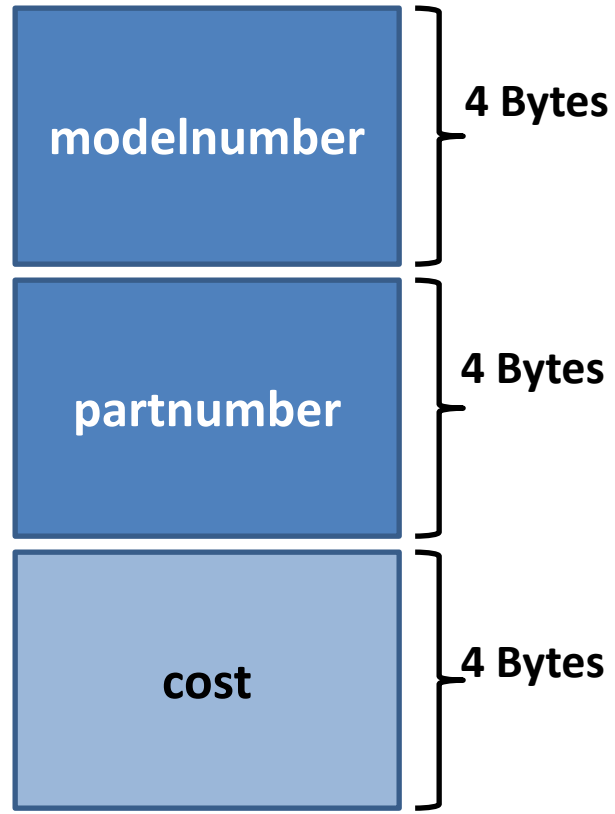
Structure Definition Syntax



Structure members in memory

Memory is only allocated when we create struct type variables

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



Declaring a Structure variable

- **Structure variable** can be **created** after the **definition** of a **structure**. The **syntax** of the **declaration** is:

```
StructName    Identifier;
```

Example:

```
part    processor;
part    keyboard;
```

Another way of Declaring a Structure variable

- You can also **declare struct variables** during **definition** of the **struct**. For example:

```
struct   part
{
    int       modelnumber;
    int       partnumber;
    float   cost;
} part1;
```

- These statements define the struct named **part** and also declare **part1** to be a variable of type **part**.

Examples

```
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 Structure Variables

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

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

Structure Variable Initialization with Declaration

```
struct   Student
{
      string    firstName;
      string    lastName;
      char      courseGrade;
      int       marks;
};


void main( )
{
      Student s1= {"M", "Umar", 'A', 94} ;
}
```

Note: Values should be written in the same sequence in which they are specified in structure definition.

Assigning Values to Structure Variables

- After creating structure variable, values to structure members can be assigned using **dot (.) operator**

- The syntax is as follows:

```
student s1;

s1.firstName  =  "ABC";
s1.lastName  =  "XYZ";
s1.courseGrade  =  'A';
s1.marks =  93;
```

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;
```

Assigning one Structure Variable to another

- 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:

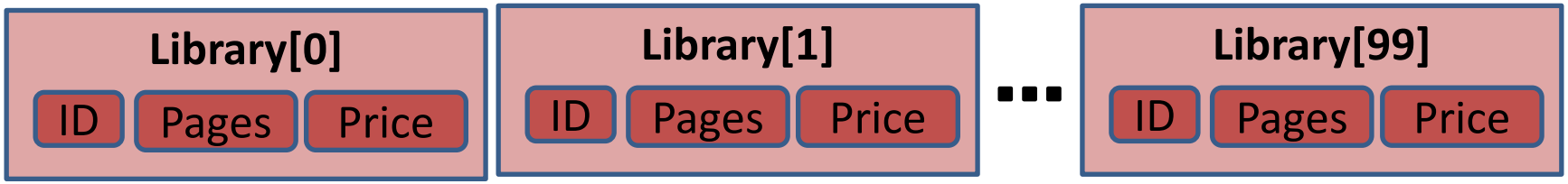
Example:

```
 studentType   Student1 = {"Amir", "Ali", 'A', 98} ;
studentType   student2 = Student1;
```

Array of Structures

An array of **structure** is a **type of array** in which **each element** contains a **complete structure**.

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



Initialization of Array of Structures

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

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

- Or it can be **assigned values later** in the **program**:

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

- Or **user** can **use *cin*** to get **input directly**:

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

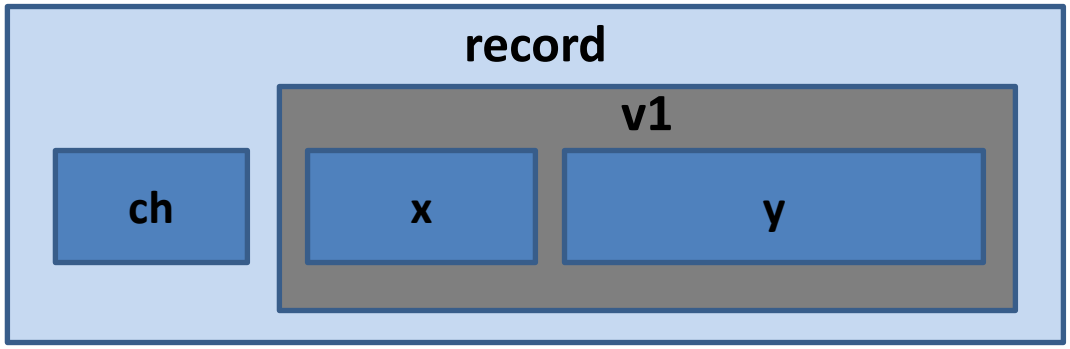
Nested Structure

- A structure can be a member of another structure: called nesting of structure.

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

struct   B
{
     char    ch;
     A       v1;
};

B  record;
```



Initializing/Assigning to 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;
}
```

Accessing Structures with Pointers

- **Pointer variables** can be **used** to **point** to **structure** type variables too.
- The pointer variable should be of same type, for example: structure type

```
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* rectPtr = &rect1;
            cout<< (*rectPrt).width<<(*rectPrt).height;
}
```

Accessing Structures with Pointers

- **Is there some easier way also?**
  - Use **arrow operator** (**->**)

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

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

Anonymous Structure

- **Structures** can be **anonymous**:

```
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***)

- A C++ ***class*** is **very similar** to a **structure**, we will focus on **classes**.
  - can have (**data**) **members**
  - can have **member functions.**
  - can also ***hide*** some of the **members** (**functions** and **data**).

Quick Example

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

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

Using the member function

```
StudentRecord stu;

…  // set values in the structure

  stu.print_ave( );
```

Structures and Functions

- **Structures can be passed in a function:**
  1. **Pass-by-value**
  2. **Pass-by-reference**
  3. **Pass-by-reference (using pointers)**

Structures and Functions – By Value

```
struct InventoryItem
{
    int partNum;                    // Part number
    string description;             // Item description
    int onHand;                     // Units on hand
    double price;                   // Unit price
};

// Function Prototypes
void getItem(InventoryItem&);       // Argument passed by reference
void showItem(InventoryItem);       // Argument passed by value

int main()
{
    InventoryItem part;

    getItem(part);
    showItem(part);
    return 0;
}
```

```
void getItem(InventoryItem &p)  // Uses a reference parameter
{
    // Get the part number.
    cout << "Enter the part number: ";
    cin >> p.partNum;

    // Get the part description.
    cout << "Enter the part description: ";
    cin.ignore();  // Ignore the remaining newline character
    getline(cin, p.description);

    // Get the quantity on hand.
    cout << "Enter the quantity on hand: ";
    cin >> p.onHand;

    // Get the unit price.
    cout << "Enter the unit price: ";
    cin >> p.price;
}
void showItem(InventoryItem p)
{
    cout << fixed << showpoint << setprecision(2);
    cout << "Part Number: " << p.partNum << endl;
    cout << "Description: " << p.description << endl;
    cout << "Units On Hand: " << p.onHand << endl;
    cout << "Price: $" << p.price << endl;
}
```

Structures and Functions – By Value

- **Problem**: <u>**copy of a large structure takes time and a lot of memory**</u>

- **Solution**: <u>**Pass-by reference (conserves the memory)**</u>
  - **Problem**: **Function can update/change original values…**
  - <u>**Solution: constant reference**</u>

```
void showItem(const InventoryItem &p)
{
    cout << fixed << showpoint << setprecision(2);
    cout << "Part Number: " << p.partNum << endl;
    cout << "Description: " << p.description << endl;
    cout << "Units on Hand: " << p.onHand << endl;
    cout << "Price: $" << p.price << endl;
}
```

Structures and Functions

- **Structures can be returned by a function**

```
Circle getInfo()
{
    Circle tempCircle;  // Temporary structure variable

    // Store circle data in the temporary variable.
    cout << "Enter the diameter of a circle: ";
    cin >> tempCircle.diameter;
    tempCircle.radius = tempCircle.diameter / 2.0;

    // Return the temporary variable.
    return tempCircle;
}
```

Structures Vs. Classes

- **Members** in **Structures** are **default public**
- **Members** in **Classes** are **default private**

- **Traditionally**, **structures** are used in **'C' language** for **holding records** consisting of **many data values**

- **Classes** are **meant to describe objects** having **both private state/data** and **public member functions**

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 500000 in price.

Practice Question 2

- Write a program that implements the following using C++ struct. The program should finally displays the values stored in a phone directory (for 10 people)