1. Lecture 4 – C++ Structs
   1. Today’s Topics
      1. C++ Structs
         1. C (basic) Structs
         2. C++ Context
         3. Struct vs Class
      2. Structs and Arrays
      3. Structs and Functions
   2. Description
      1. A “Structure” is a collection of related data items, possibly of different types.
         1. ⮚ A structure type in C++ is called **struct**. A **struct** is *heterogeneous*:
         2. ⮚ It can be composed of data of different types.
      2. An array is *homogeneous*:
         1. ⮚ It can contain only data of the same type.
      3. Structures are used to hold data that *belong* together. Examples:
         1. Student record: student id, name, major, gender, start year,
         2. Bank account: account number, name, currency, balance, ...
         3. Address book contact: name, address, telephone number, ...
      4. In database applications, structures are called records.
   3. Members
      1. Struct Members (or Fields):
         1. ⮚ Individual components of a **struct** type.
      2. Versatility:
         1. Struct Members can be of different types:
            1. ⮚ Simple
            2. ⮚ Array
            3. ⮚ Another **struct**
      3. Naming – Resolution:
         1. ⮚ A **struct** is named as a whole.
         2. ⮚ Individual Struct Members are named using *field identifiers*.
      4. Versatility:
         1. Complex data structures can be formed by defining arrays of **struct**s.
   4. Type Declaration
      1. struct type\_name {  
         member\_type1 member\_name1;  
         member\_type2 member\_name2;  
         member\_type3 member\_name3;  
         .  
         .  
         } object\_names;
      2. struct product {  
         int weight;  
         double price;  
         } apple, banana, melon;
   5. Initialization
      1. ***StudentRecord student1 =* {"John Doe", 123, "CSE", 'M'};** 
         1. ⮚ Heavily depends on **struct** type definition. Compromised maintainability.
         2. ⮚ Might break (type mismatch).
         3. ⮚ Might work but mess up (wrong value assignment).
   6. Assignment
      1. The values contained in one **struct** type variable can be assigned to another variable of the same **struct** type.
         1. ⮚ This involves *Data Copy* operations.
      2. Example:
         1. **strcpy(*student1.*name, "John Doe"); *student1.*id = 123; strcpy(*student1.*dept, "CSE"); *student1.*gender = 'M';**
   7. Nested Structures
      1. A **struct** type can be a member of another **struct**.
         1. ⮚ Program design w.r.t. inherent attributes.
         2. NOTE: Cannot have recursion here !
   8. Arrays of Structs
      1. Arrays are homogenous (one data type):
         1. Regular data type
      2. Supported type can be struct
      3. All aforementioned operations take place as usual:
   9. Structs and Functions
      1. Supported type for Function Parameters can be **struct &/\*/[]**:
         1. **struct *Point*{ double x, y; }; // need declaration before any mention // of Point can be made in the program**
      2. ⮚ Pass-By-Value
         1. **double points\_distance(*Point p1*, *Point p2*){**   
             **return sqrt((*p1.*x *p2.*x)\**(p1.*x*-p2.*x)+(*p1.*y**  
             ***p2.*y)\**(p1.*y*-p2.*y); }**   
            ***Point p1*, *p2*;  
            double p12\_distance = points\_distance(*p1*, *p2);***
2. Remember: Procedural vs Object-Oriented
   1. Procedural
      1. Focused on the question: “What should the program do next?” Structure program by:
         1. ⮚ Splitting into sets of tasks and subtasks.
         2. ⮚ Make functions for tasks.
         3. ⮚ Perform them in sequence (computer). Large amount of data and/or tasks makes projects/programs unmaintainable.
   2. **Object-Oriented (OO)** 
      1. Package-up self-sufficient modular pieces of code. The world is made up of interacting objects. Pack away details into boxes (objects) keep them in mind in their abstract form.  
         Focus on (numerous) interactions.
         1. ⮚ Encapsulation
         2. ⮚ Inheritance
         3. ⮚ Polymorphism
3. Remember: Classes
   1. Class
      1. C++ Classes are very similar to C Structs in that they both include user-defined sets of data items, which collectively describe some entity such as a Student, a Book, an Airplane, or a data construct such as a String, a ComplexNumber, etc...
4. C++ Structs
   1. Structs in C++
      1. **structs** related data.
         1. ⮚  Member variables maintain each object’s state.
         2. ⮚  All member “parts” *by default* are **public**ly accessible.
            1. (later: Class members *by default* are **private** – internally accessible for a specific Object from own methods, i.e. functions)
      2. When to use a **struct** (for now) :
         1. ⮚ For things that are mostly data-oriented.
         2. ⮚ Are there data-only limitations?  
            Data sanity checking might be necessary!
   2. **structs** can have methods (i.e. functions).
      1. ⮚ Actually in C++ **struct** and Class are very similar.
      2. ⮚ Default access level (**public** *vs* **private**) is the difference of significance from what we know so far.
   3. **structs** can have: (– *Note*: like Classes do) ⮚ Member variables
      1. ⮚ Methods (i.e. Functions)
      2. ⮚ Constructors, Destructors, etc. (more on these later)
      3. ⮚ **public**, **private**, and **protected** attributes (more on these later)
      4. ⮚ **virtual** functions (more on these later)