Structs

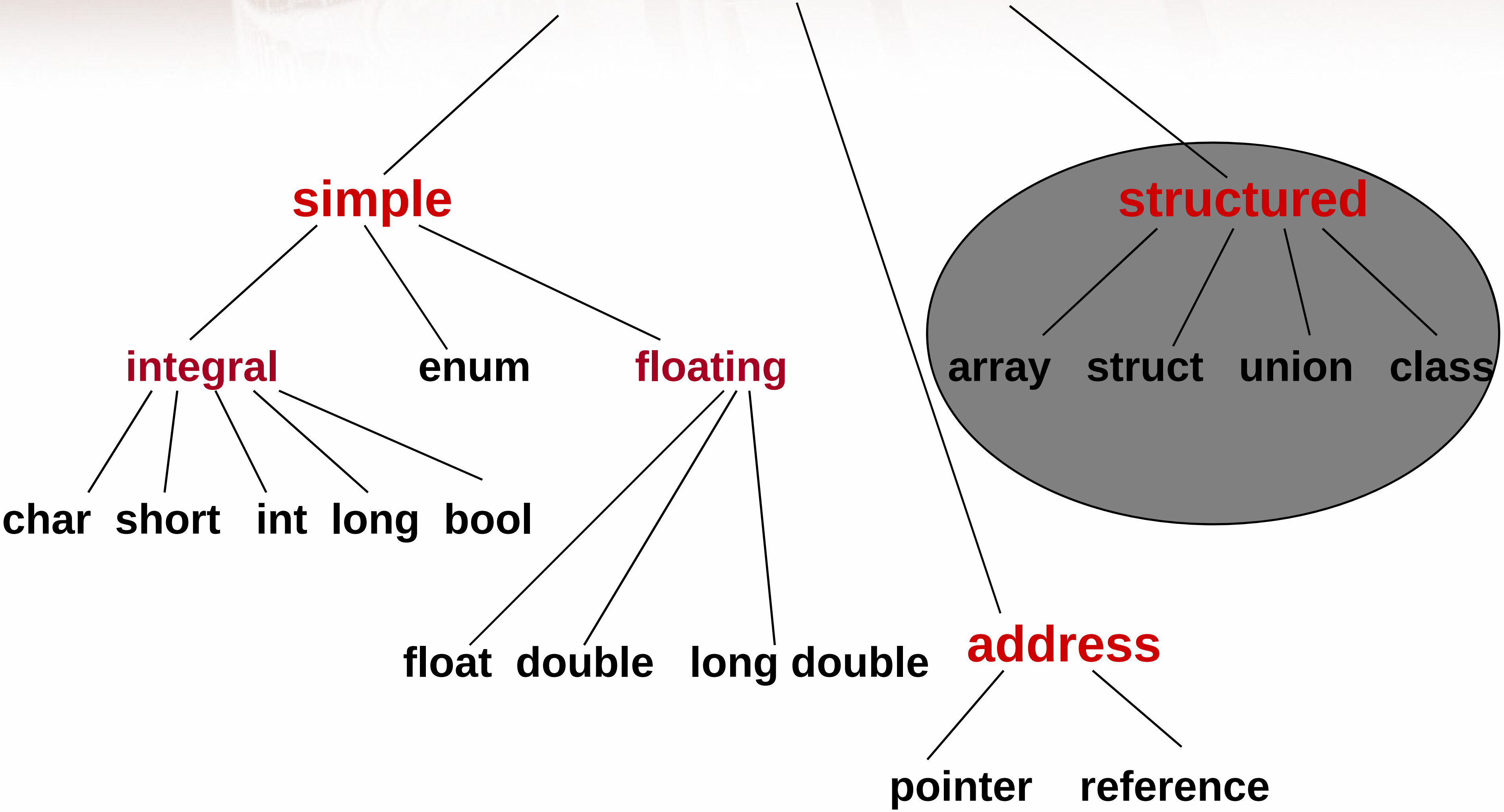
MCS2514

Yin Wang

First exam

- Oct 10th, Wed, 1:00PM-2:50PM
- Cover all you have learned so far + structs+cstring+string

C++ Data Types



simple

integral

char short int long bool

enum

floating

float double long double

address

pointer reference

structured

array struct union class

C++ Data Types

- There are simple data types that hold only one value
- There are structured data types that hold multiple values
- The array was the first example of a structured data type that can hold multiple values
- The structure is the second example

C++ Structs

- often we have related information of various types that we'd like to store together for convenient access under the same identifier…
- Why not use simple data types or arrays?

The **struct** Data Type

- **struct** used to store a collection of related data items
- Individual components of the **struct** are called its members
- Each member can be a different type of data

oneAnimal

	5000
.id	2037581
.name	“giant panda”
.genus	“Ailuropoda”
.species	“melanoluka”
.country	“China”
.age	18
.weight	234.6
.health	Good

anotherAnimal

	6000
.id	5281003
.name	“llama”
.genus	“Lama”
.species	“peruana”
.country	“Peru”
.age	7
.weight	278.5
.health	“Excellent”

Using the **struct** Data Type

- Three steps to use
  - declaration of struct type
  - use of struct type to define a new variable
  - access of members of the struct variable(s)
- Typically declare **struct** type globally

Struct Type Declaration

```
struct  AnimalType              // declares a  struct data type
{                               //  does not allocate any memory
    long            id ;
    string          name ;
    string          genus ;
    string          species ;
    string          country ;
    int             age ;
    float           weight ;
    string          health ;
} ;  // NOTE THE SEMICOLON
```

struct members

`struct` type Declaration

The struct declaration creates a type and names the members of that type.

It does not allocate memory for any variables of that type!

You still need to declare your struct variables to use them!

Struct Variable Declaration

```
int main()
{
// declare  variables of AnimalType
AnimalType    oneAnimal ;
AnimalType    anotherAnimal ;
...
}
```

Memory allocation for struct

- Practice:

```
struct A{
        char i;
        int j;
};
A a1;
```

What is the size of a1?

How about other type of structs?

More about `struct` declarations

If the struct type declaration precedes all functions it will be visible throughout the rest of the file. If it is placed within a function, only that function can use it.

It is common to place struct type declarations within a (.h) header file and #include that file (more on this later).

It is possible for members of different struct types to have the same identifiers. Also a non-struct variable may have the same identifier as a structure member.

Accessing `struct` Members

Dot (period) is the member selection operator.

After the struct type declaration, the various members can be used in your program only when they are preceded by a struct variable name and a dot.

EXAMPLES

```
oneAnimal.weight
anotherAnimal.country
```

Valid operations on a struct member depend only on its type

```
oneAnimal.age  =  18;

oneAnimal.id     =  2037581;

cin  >>  oneAnimal.weight;

getline ( cin, oneAnimal.species );

oneAnimal.name = “giant panda”;

oneAnimal.genus[ 0 ] = toupper (oneAnimal.genus[ 0 ] ) ;

oneAnimal.age++;
```

Aggregate Operation

- **is an operation on a data structure as a whole, as opposed to an operation on an individual component of the data structure**

Aggregate `struct` Operations

I/O, arithmetic, and comparisons of entire struct variables are NOT ALLOWED!

Operations valid on an entire struct type variable:

- **assignment to another struct variable of same type,**
- **pass to a function as argument (by value or by reference),**
- **return as value of a function**

Assignment

- **struct** copy or assignment

```
anotherAnimal = oneAnimal;
```

- Copies each member from oneAnimal **struct** to anotherAnimal **struct**

Passing as a “By-Value” Argument

- Actual and formal parameters must be same type **struct**
- Using structs as arguments can shorten argument list
- Passing **structs** by value can be inefficient, since it duplicates values of all members

```
void  WriteOut(AnimalType   thisAnimal)
// Prints out values of all members of thisAnimal
// Precondition:     all members of thisAnimal are assigned
// Postcondition:     all members have been written out
{
    cout   <<  “ID # “  <<  thisAnimal.id  <<  thisAnimal.name  << endl ;

    cout   <<  thisAnimal.genus   <<  thisAnimal.species << endl ;

    cout   <<  thisAnimal.country << endl ;

    cout   <<  thisAnimal.age   << “ years “  << endl ;

    cout   <<  thisAnimal.weight << “ lbs. “  << endl ;

    cout   << “General health : “ ;

    WriteWord ( thisAnimal.health ) ;
}
```

Passing **struct** by Reference

- Same as other reference parameters
  - use **&** to identify in parameter list

Passing a `struct` by Reference

```
void   IncrementAge (AnimalType&   thisAnimal )

// Adds 1 to age
// Precondition:      thisAnimal.age is assigned
// Postcondition:    thisAnimal.age == thisAnimal.age@entry + 1

{

     thisAnimal.age++ ;

}
```

Code efficiency about using structs

- Question: ……..

Passing **struct** by Reference with const

- Can also use constant reference to avoid copying large structs but keep function from modifying contents:
  - precede parameter with reserved word **const**

```
void WriteOut(const AnimalType& thisAnimal)
```

struct as a Return Value

- You can also return an entire struct variable from a function.

Comparison (Relational Operators)

- Compare struct variables member-wise (NOT THE WHOLE STRUCTURE)
- What does it mean for one animal to be greater than another?
- It makes more sense for one animal’s age to be greater than another’s age…

```
if(oneAnimal.age > anotherAnimal.age)
   …
```

Input/Output

- No aggregate input/output operations on a struct variable

```
cout<<oneAnimal; //does NOT work

cin>>anotherAnimal; //does NOT work!!!
```

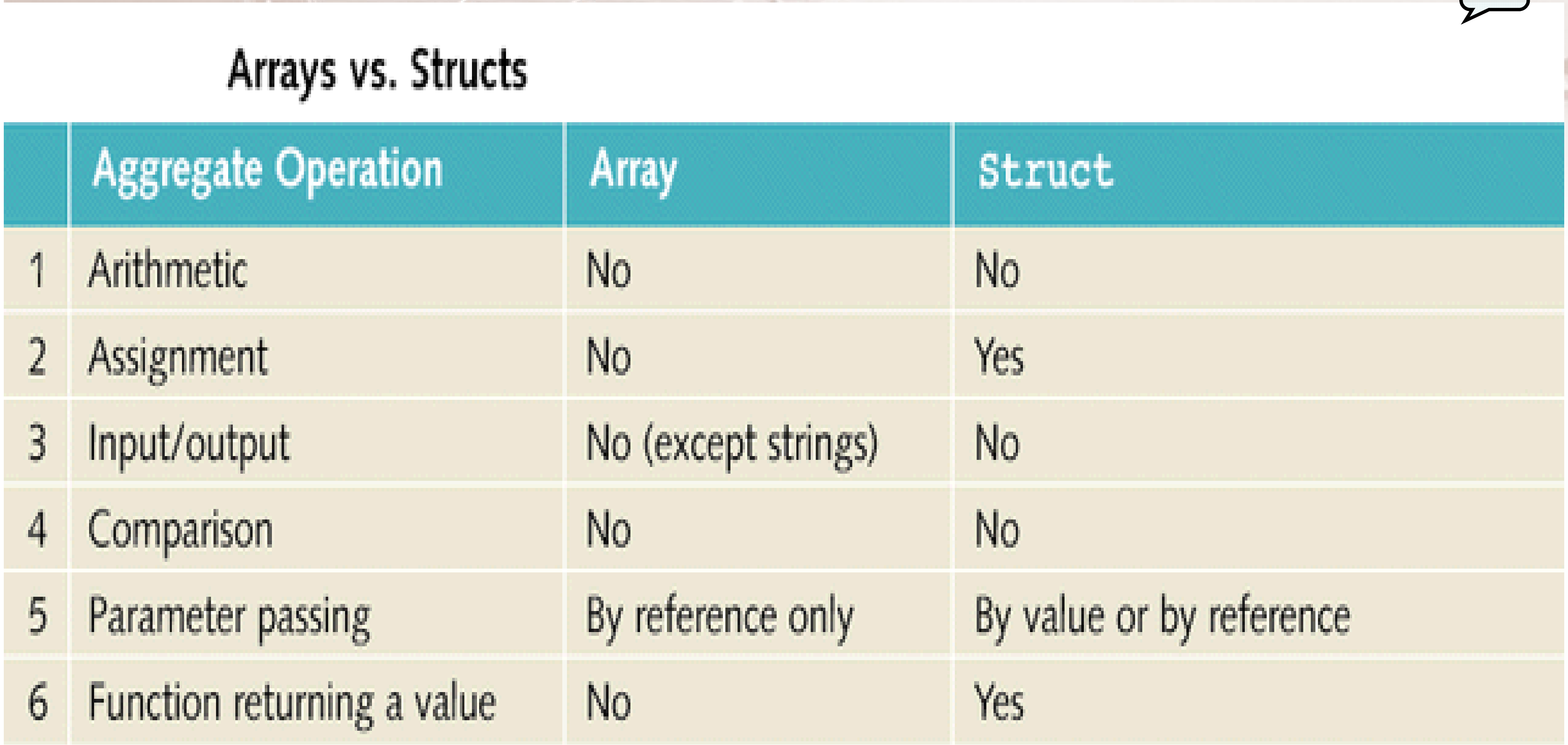
Input/Output

- Data in a struct variable must be read one member at a time

```
cin>>oneAnimal.id;
```

- The contents of a struct variable must be written one member at a time

```
cout<<oneAnimal.age;
```



Arrays vs. Structs

	Aggregate Operation	Array	Struct
1	Arithmetic	No	No
2	Assignment	No	Yes
3	Input/output	No (except strings)	No
4	Comparison	No	No
5	Parameter passing	By reference only	By value or by reference
6	Function returning a value	No	Yes

Also Note that arrays CANNOT store member components of multiple types, structures can…

Using Functions with Structs

- You can write utility functions to accompany your struct data types
  - Read struct data from keyboard/file
  - Write struct data to a specified stream
  - Etc…

```
void GetAnimalData (AnimalType& newAnimal)

// Obtains all information about an animal from
   keyboard

// Postcondition: AnimalType members filled
{


cout<<“\nPlease Enter the Animal’s ID:”
cin>>newAnimal.id;


//… continue until all data is filled


}
```

Hierarchical Structures

The type of a struct member can be another struct type. This is called nested or hierarchical structures.

Hierarchical structures are very useful when there is much detailed information in each record.

FOR EXAMPLE . . .

struct MachineRec

Information about each machine in a shop contains:

an idNumber,

a written description,

the purchase date,

the cost,

and a history (including failure rate, number of days down, and date of last service).

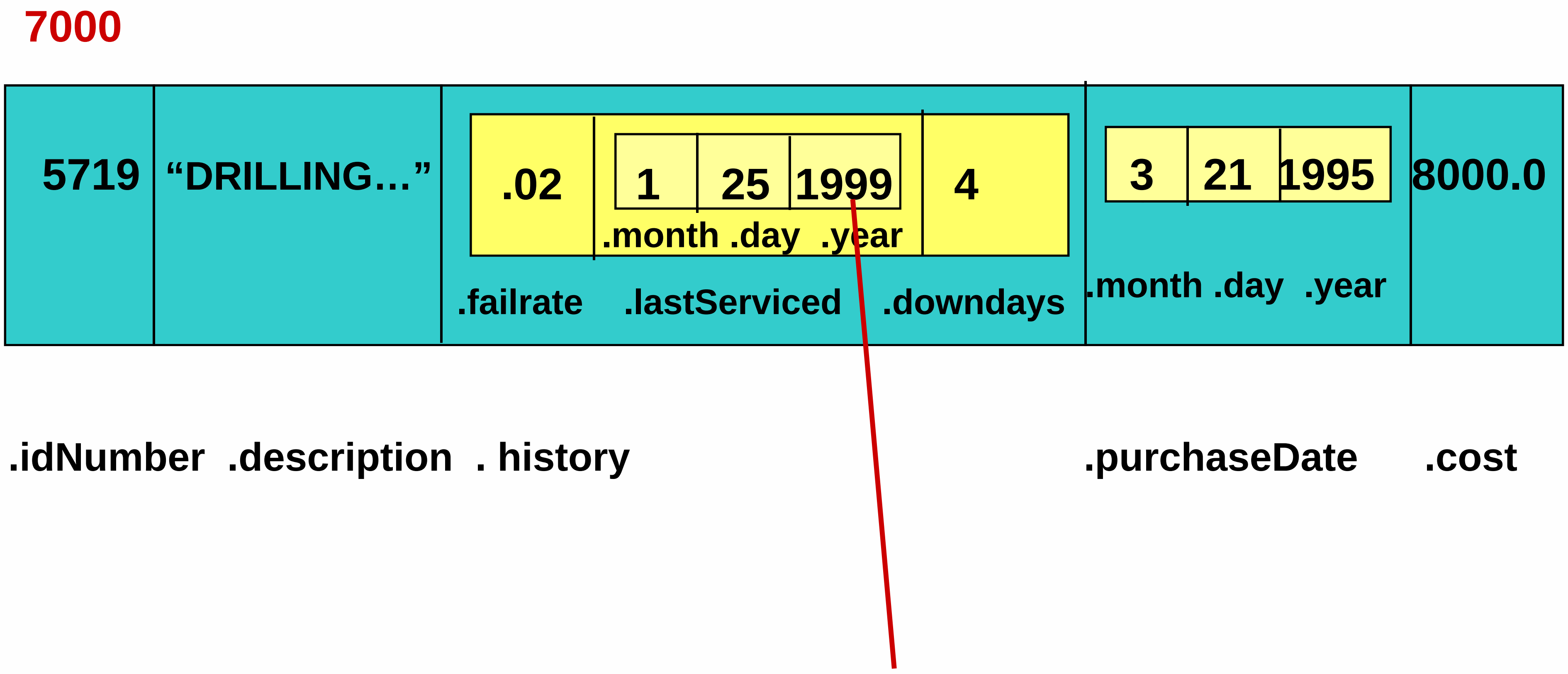
```
struct  DateType
{   int   month ;               // Assume 1 . . 12
    int   day ;                 // Assume  1 . . 31
    int   year ;                // Assume 1900 . . 2050
};

struct  StatisticsType
{   float           failRate ;
    DateType        lastServiced ;           // DateType is a struct type
    int             downDays ;
} ;

struct MachineRec
{   int                idNumber ;
    string             description ;
    StatisticsType  history ;       // StatisticsType is a struct type
    DateType           purchaseDate ;
    float              cost ;
} ;

MachineRec     machine ;
```

struct type variable machine



7000

5719 “DRILLING…” .02 1 25 1999 4 3 21 1995 8000.0

.month .day .year

.failrate .lastServiced .downdays

.month .day .year

.idNumber .description . history .purchaseDate .cost

machine.history.lastServiced.year has value 1999

Arrays **IN** structs

- Two key items are associated with a list:
  - Values (elements)
  - Length of the list
- Define a struct containing both items:

```
const int CAPACITY = 1000;
struct listType
{
  int listElem[CAPACITY];   //array containing the list
  int listLength;           //current length of the list
};
```

Arrays OF structs

- It is possible to declare an array of structs, after all, once declared your struct type is another recognized data type

- The following line of code declares an array of AnimalType structs.

AnimalType zooAnimals[NUM_ANIMALS];

- How do we access a member of one of these structures?

Summary

- Struct: collection of a fixed number of components
- Components can be of different types
- struct is a reserved word
- No memory is allocated for a struct; memory is allocated for struct variables when declared
- Components of a struct are called members

Summary

- struct components are accessed by name
- Dot (.) operator is called the member access operator
- Members of a struct are accessed using the dot (.) operator
- The only built-in operations on a struct are the assignment and member access

Summary

- Neither arithmetic nor relational operations are allowed on the entire structure
- structures can be passed by value or reference
- A function can return a structure
- A structure can be a member of another structure

Programming Example

- A company has six salespeople
- Every month they go on road trips to sell the company’s product
- At the end of each month, the total sales for each salesperson, salesperson’s ID, and the month, are recorded in a file
- At the end of each year, the manager of the company asks for a report

Programming Example

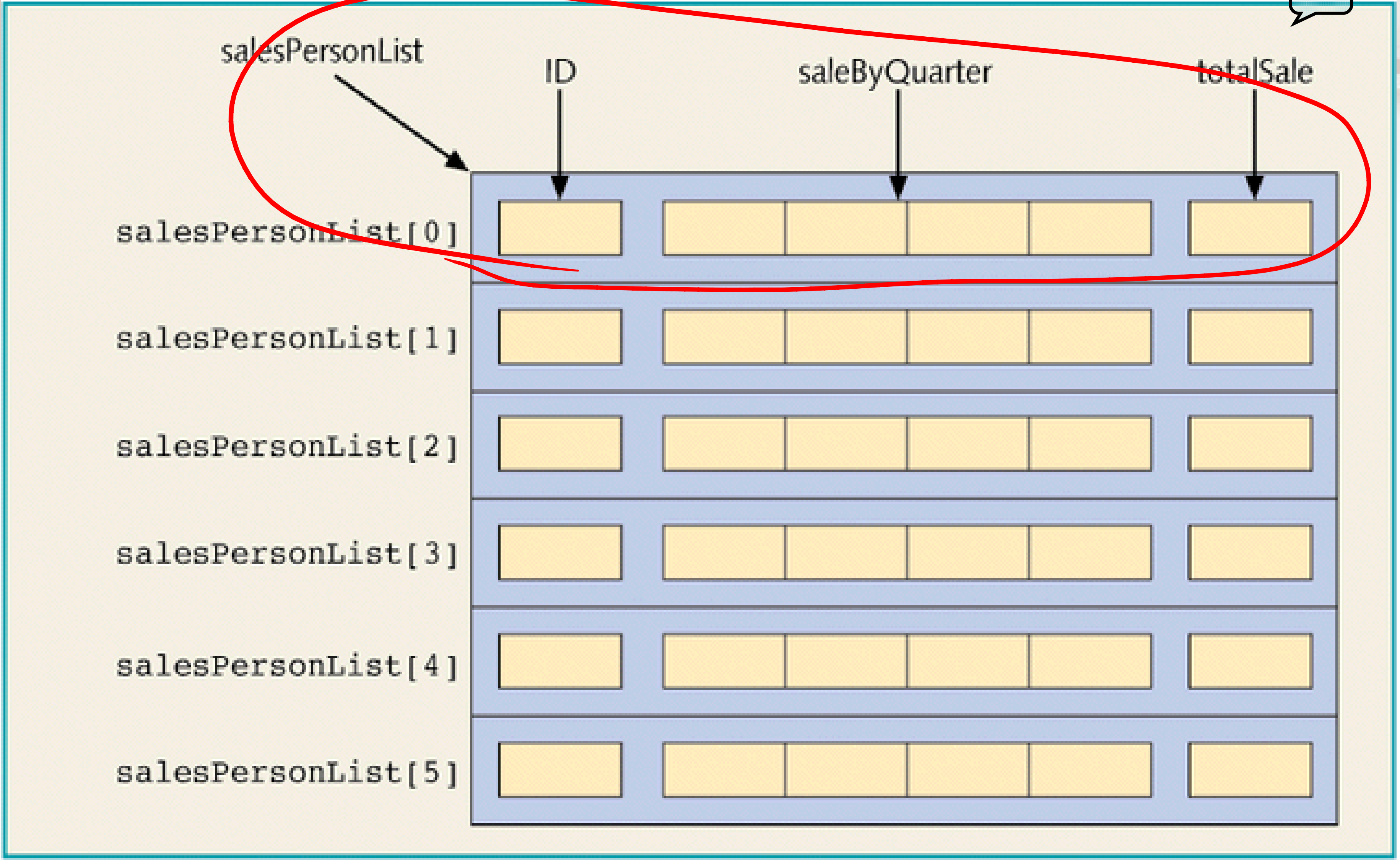
- The data in the file are in following format.

```
ID1 67
ID2 78
….
….
ID1 89
ID2 98
…
```

- Sales data are not ordered by ID, but every 6 lines represent the sales record for a month.

Data Design

- Design an appropriate scheme for organizing your input data.
- Once the data is properly organized, you can design algorithms to operate on that data to perform the necessary tasks.



Array salesPersonList

Output Format

```
-----------  Annual Sales Report -------------

  ID         QT1        QT2      QT3       QT4     Total

12345      1892.00      0.00    494.00    322.00   2708.00
32214       343.00    892.00   9023.00      0.00  10258.00
23422      1395.00   1901.00      0.00      0.00   3296.00
57373       893.00    892.00   8834.00      0.00  10619.00
35864      2882.00   1221.00      0.00   1223.00   5326.00
54654       893.00      0.00    392.00   3420.00   4705.00
~~Total      8298.00   4906.00  18743.00   4965.00~~

Max Sale by SalesPerson: ID = 57373, Amount = $10619.00
Max Sale by Quarter: Quarter = 3, Amount = $18743.00
```

QT1 stands for quarter 1 (months 1 to 3), QT2 for quarter 2 (months 4 to 6), QT3 for quarter 3 (months 7 to 9) and QT4 for quarter 4 (months 10 to 12)