5 Structs

Nur Nabilah Abu Mangshor

Objectives

- Learn about records (`structS`)
- Examine various operations on a `struct`
- Explore ways to manipulate data using a `struct`
- Learn about the relationship between a `struct` and functions
- Discover how arrays are used in a `struct`
- Learn how arrays are used in a `struct`
- Learn how to create an array of `struct` items

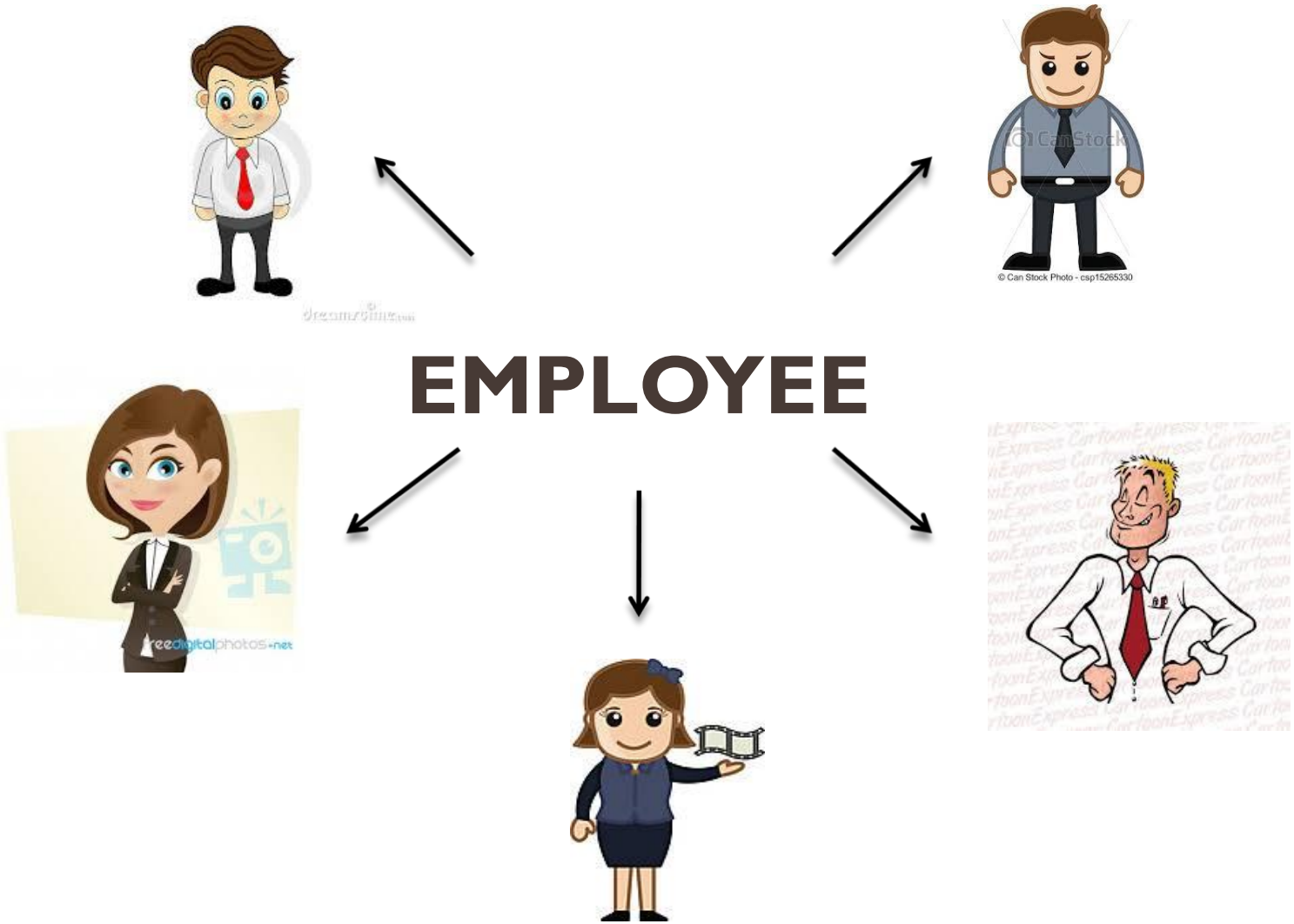
Structs

PART I

Contents

- Record definition
- Record variable declaration
- Accessing record members
- Record assignment
- Comparing record members

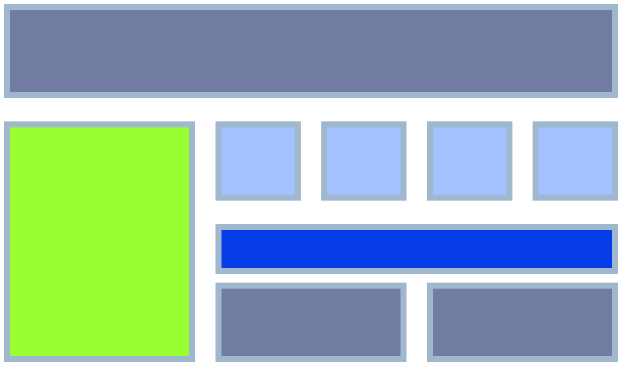
Introduction



Introduction

EMPLOYEE
firstName,
lastName,
salary,
address
empID,

Records (`structs`)



- A **Structure** is a collection of related data items, possibly of different types.
- A structure type in C++ is called **`struct`**
- A **`struct`** is heterogeneous in that it can be composed of data of different types

Records (`structs`)

- `struct` :A collection of a fixed number of components in which the components are accessed by name
- The components may be of different types
- The component of a `struct` are called the members of the `struct`

Records (`structs`)

- Structures hold data that belong **together**.
- Examples:
  - Student record: student id, name, major, gender, start year, …
  - Bank account: account number, name, currency, balance, …
  - Address book: name, address, telephone number, …
- In database applications, structures are called records

Records (structs)

- The general syntax of a **struct** in c++ is:

```
struct structName
{
  dataType1 identifier1;
  dataType2 identifier2;
  :
  dataTypen identifiern;
} ;
```

- **struct** is a reserved word

Records (structs)

- Example:

```
struct BankAccount{
    char Name[15];
    int AcountNo[10];
    double balance;
    Date Birthday;
};
```

The "BankAcount" structure has simple, array and structure types as members.

- Example:

```
struct StudentRecord{
    char Name[15];
    int Id;
    char Dept[5];
    char Gender;
};
```

The "StudentRecord" structure has 4 members.

Records (structs)

- Example:

```
struct StudentInfo{
    int Id;
    int age;
    char Gender;
    double CGA;
};
```

The "StudentInfo" structure has 4 members of different types.

- Example:

```
struct StudentGrade{
    char Name[15];
    char Course[9];
    int Lab[5];
    int Homework[3];
    int Exam[2];
};
```

The "StudentGrade" structure has 5 members of different array types.

Records (structs)

- The members of a **struct**, even though enclosed in braces are not considered to form a compound statement
- Thus, a semicolon (after the right brace) is used to end the **struct** statement

```
struct employeeType
{
  string firstName;
  string lastName;
  string address;
  float salary;
  int empID;
} ; //semicolon to end struct statement
```

Records (structs)

- Once the data type is defined, you can declare variables of that type
- Consider a **struct** type, `studentType`

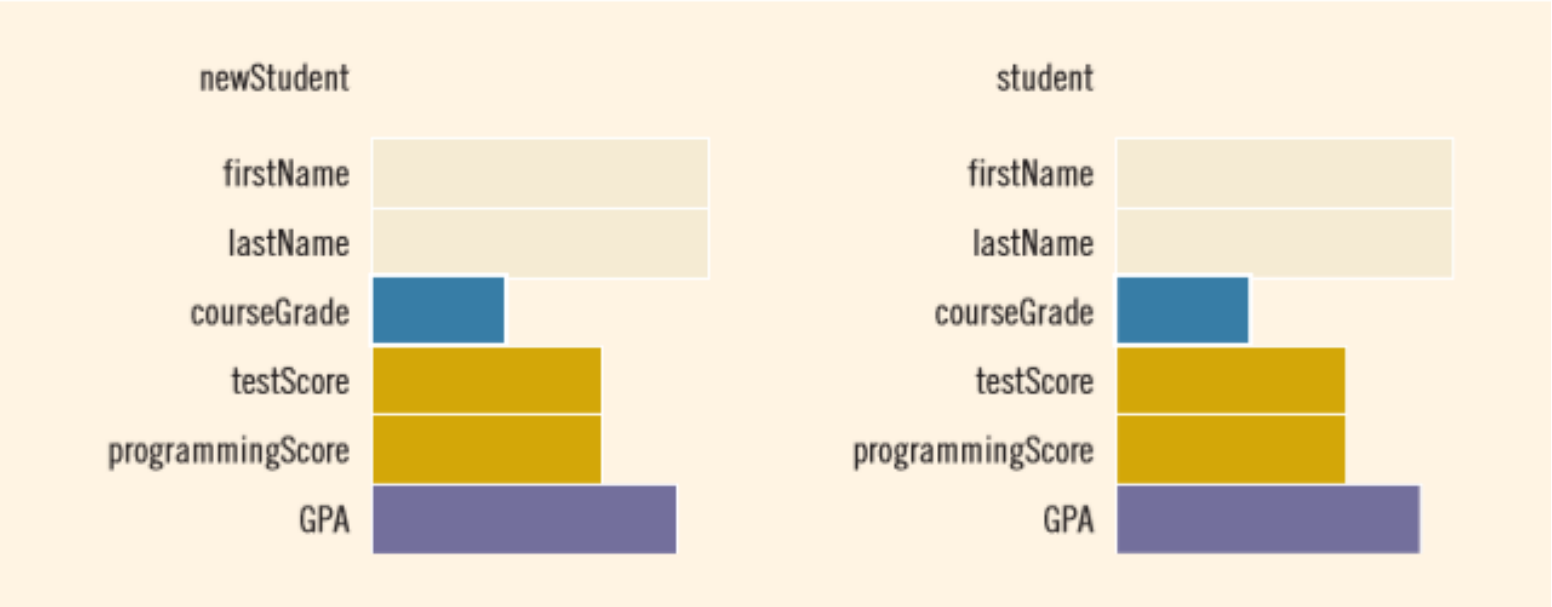
```
struct studentType
{
  string firstName;
  string lastName;
  char courseGrade;
  int testScore;
  int programmingScore;
  double GPA;
} ;
```

Records (structs)

```
//variable declaration
studentType newStudent;
studentType student;
```

- These statements declare two **struct** variable, newStudent and student, of type studentType
- The memory allocated is large enough to store firstName, lastName, courseGrade, testScore, programmingScore and GPA

Records (`structs`)



`struct` `newStudent` and `student`

Accessing `struct` Members

- To access structure member (component), you use the **`struct`** variable name together with the member name
- These names are separated by a dot (period)
- The syntax for accessing a **`struct`** member is:

```
structVariableName.memberName
```

- The `structVariableName.memberName` is just like any other variable

Accessing `struct` Members

- In C++, the dot (.) is an operator called the member access operator
- Suppose you want to initialize the member `GPA` of `newStudent` to `0.0`. The statement to accomplishes this task is:

```
newStudent.GPA = 0.0;
```

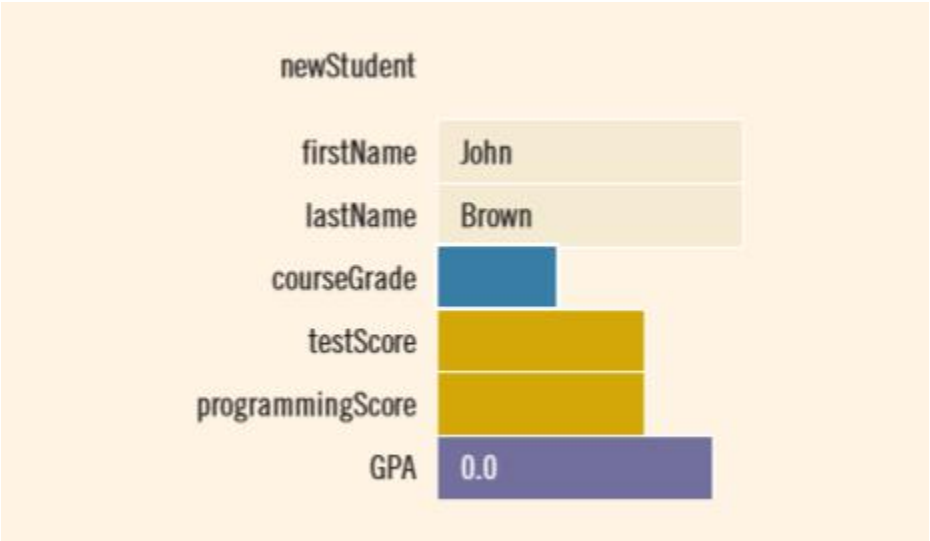
- Similarly, the statements:

```
newStudent.firstName = "John";
newStudent.lastName = "Brown";
```

Stores `"John"` in the member `firstName` and `"Brown"` in the member `lastName` of `newStudent`

Accessing `struct` Members

- After the preceding three assignment statements execute, `newStudent` is as shown as follows:



`struct newStudent`

Accessing `struct` Members

- The statement:

```
cin >> newStudent.firstName;
```

- Reads the next string from the standard input device and stores it in:

```
newStudent.firstName;
```

- The statement:

```
cin >>newStudent.testScore>>newStudent.programmingScore;
```

- Reads two integer values from the keyboard and stores them in `newStudent.testScore` and `newStudent.programmingScore` respectively

Accessing `struct` Members

- Suppose that score is a variable of type int.

```
score = (newStudent.testScore +
  newStudent.programmingScore) / 2;
```

- The above statement assigns the average of `newStudent.testScore` and `newStudent.programmingScore` to `score`
- The following statements determine the course grade and stores it in `newStudent.courseGrade`:

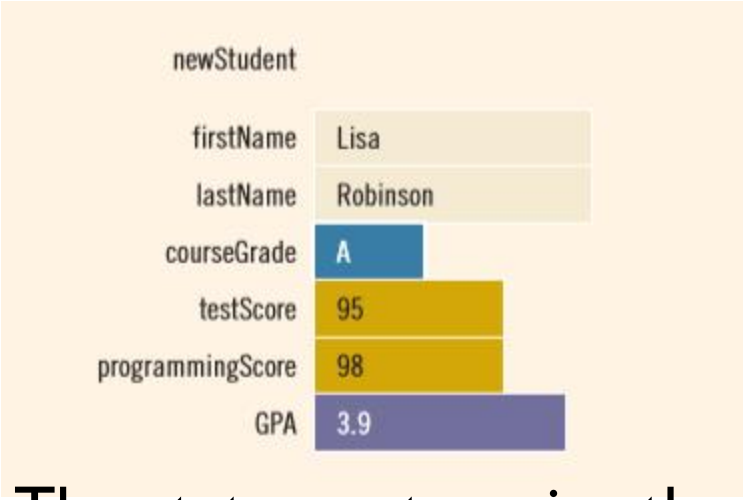
Accessing `struct` Members

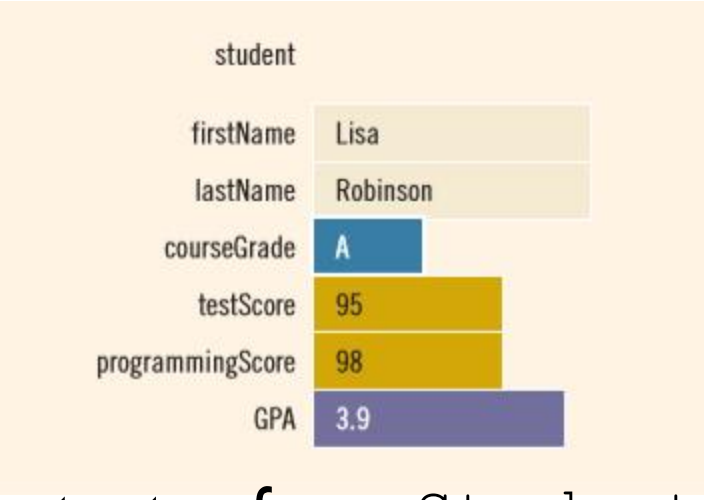
- The following statements determine the course grade and stores it in `newStudent.courseGrade`:

```
if (score >= 90)
  newStudent.courseGrade = 'A';
else if (score >= 80)
  newStudent.courseGrade = 'B';
else if (score >= 70)
  newStudent.courseGrade = 'C';
else if (score >= 60)
  newStudent.courseGrade = 'D';
else
  newStudent.courseGrade = 'E';
```

Assignment

- We can assign the value of one `struct` variable to another `struct` variable of the same type by using assignment statement
- The statement : `student = newStudent;`





- The statement copies the contents of `newStudent` into `student` as shown above

Assignment (cont)

- In fact the assignment statement:

```
student = newStudent;
```

is equivalent to the following statements:

```
student.firstName = newStudent.firstName;
student.lastName = newStudent.lastName;
student.courseGrade = newStudent.courseGrade;
student.testScore = newStudent.testGrade;
student.programmingScore =newStudent.programmingScore;
student.GPA = newStudent.GPA;
```

Comparison (Relational Operator)

- To compare `struct` variables, you compare them member-wise
- To compare the values of `student` and `newStudent`, you must compare them as follows:

```
if (student.firstName == newStudent.firstName
  && student.lastName == newStudent.lastName)
:
```

```
if (student == newStudent) //illegal
```

Structs

PART II

Contents

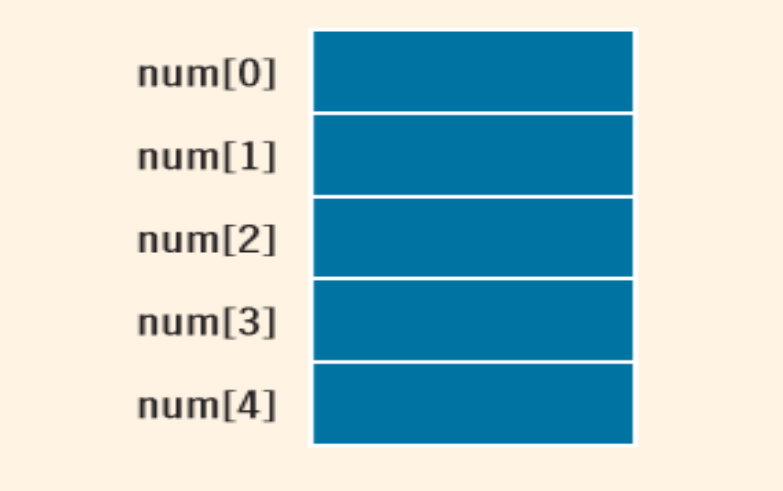
- Arrays
- Arrays in `structs`
- `structs` in Arrays
- `structs` within `structs`

Recap (Array)

- What is array?
- The statement to declares an array num of five components:

```
int num [5];
```

- Each component of type int. The components are `num[0], num[1]… num[4]`



Arrays versus `structs`

Aggregate Operation	Array	`struct`
Arithmetic	No	No
Assignment	No	Yes
Input/ Output	No (except strings)	No
Comparison	No	No
Parameter passing	By reference only	By value or by reference
Function returning a value	No	Yes

Arrays in structs

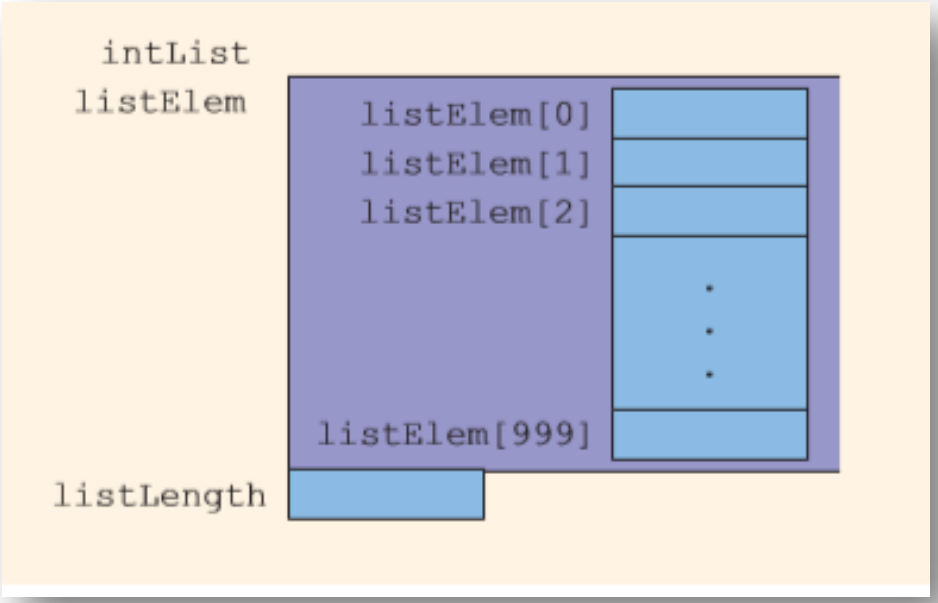
- A list is a set of elements of the same type
- There are two things associated with it
  - value → elements
  - length
- structs contain both value and length

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

Arrays in `structs` (cont)

- The following statement declares `intList` to be a **`struct`** variable of type `listType`

```
listType intList;
```



`struct` variable `intList`

Arrays in `struct`s (cont)

- The variable `intList` has two members:
  - **`listElem`** → an array of 1000 components of type `int`
  - **`listLength`** → type `int`
- `intList.listElem` accesses the member **`listElem`** and `intList.listLength` accesses the member **`listLength`**

Arrays in `structs` (cont)

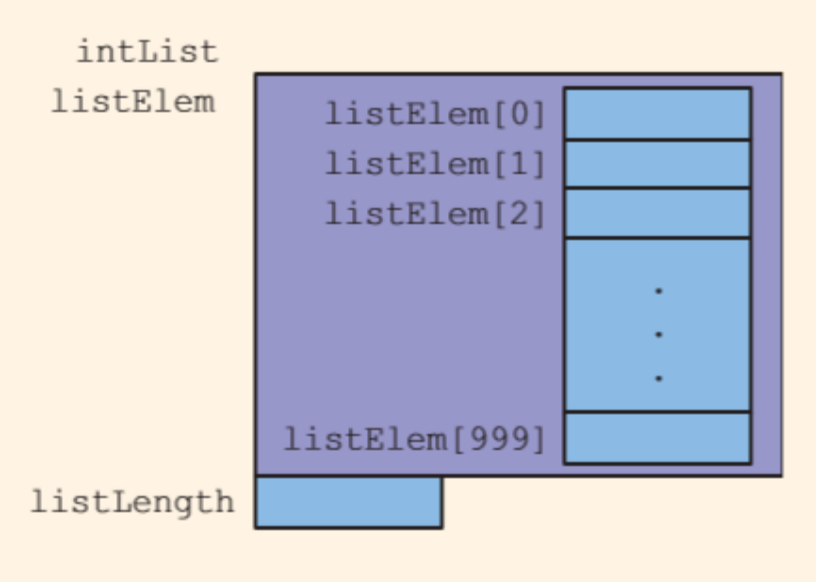
- Consider the following statements:

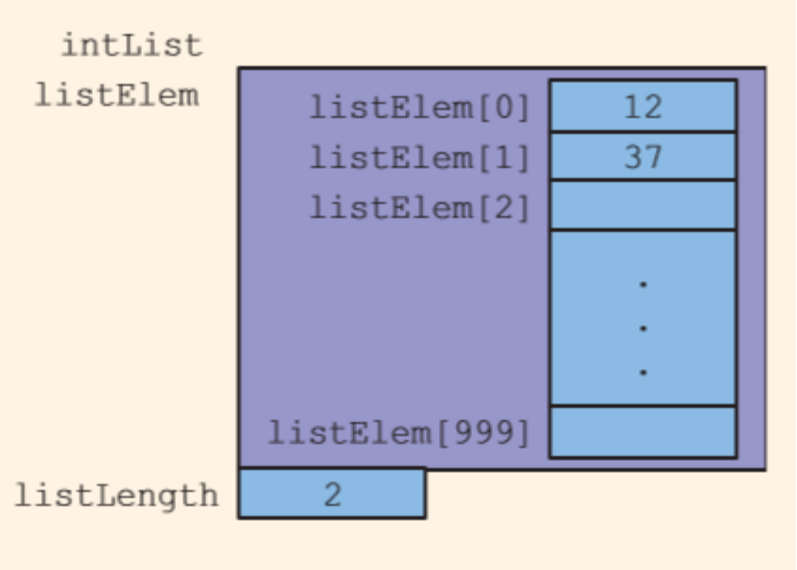
```
intList.listLength = 0     //Line 1
intList.listElem[0] = 12;  //Line 2
intList.listLength++;      //Line 3
intList.listElem[1] = 37;  //Line 4
intList.listLength++;      //Line 5
```

- The statement in `line 1` sets the value of the member listLength to 0
- The statement in `line 2` stores 12 in the first component of the array `listElem`
- The statement in `line 3` increments the value of `listLength` by 1

Arrays in `structs` (cont)

- `intList` is as shown below after those statements are execute:





`struct` variable `intList`

`intList` after the 5 statements execute

Arrays in `structs` (cont)

- Next, by using sequential search algorithm, determine whether specific item, **`searhItem`** is in the list

```
int seqSearch(const listType& list, int searchItem)
{
  int loc; bool found = false;
  for (loc = 0; loc < list.listLength; loc++)
      if(list.listElem[loc] == searchItem)
      {      found = true;
             break; }
      if (found)
      return loc;
      else
      return -1;
}
```

structs in Arrays

- Lets consider a company has 50 employees
- We want to do this task:
  1. To print their monthly paychecks
  2. To keep track company's total payment to in the year-to-date
- First, lets define an employee's record:

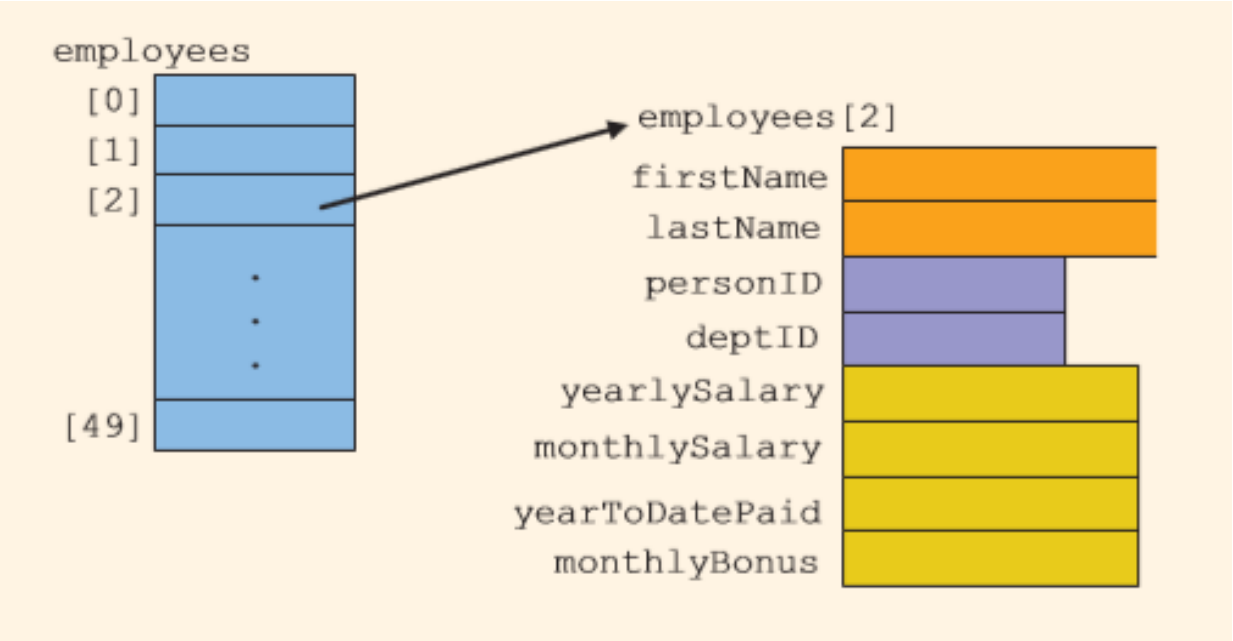
```
struct employeeType
{ string firstName;
  string lastName;
  int personID;
  string deptID;
  double yearlySalary;
  double monthlySalary;
  double yearToDatePaid;
  double monthlyBonus };
```

structs in Arrays (cont)

- Each employee has the following members (components): firstName, lastName, personID…..

```
employeeType employees [50];
```

- The above statement declares an array employees of 50 components of type employeeType



structs in Arrays (cont)

- Consider we have this declaration: `int count;`
- Assume that we ask input from employee:

```
for (count = 0; count <50; cout++)
{
  cin >> employees[count].firstName;
  cin >> employees[count].lastName;
  cin >> employees[count].personID;
  cin >> employees[count].deptID;
  cin >> employees[count].monthlySalary;
  employees.yearlySalary[count]=
  employees.monthlySalary*12;
  cin >> employees[count].monthlyBonus;
}
```

structs within structs

- Lets us consider this **struct**
- In this section, we will discover how to organize data in **struct** using another **struct**

```
struct  employeeType
{
        string firstname;
        string middlename;
        string lastname;
        string empID;
        string address1;
        string address2;
        string city;
        string state;
        string zip;
        int hiremonth;
        int hireday;
        int hireyear;
        int quitmonth;
        int quitday;
        int quityear;
        string phone;
        string cellphone;
        string fax;
        string pager;
        string email;
        string deptID;
        double salary;
};
```

`structs` within `structs` (cont)

- As you can see, a lot of information is packed into a single **`struct`** name `employeeType`
- `employeeType` has 22 members (components)
- Some members in this **`struct`** will be accessed more frequently than others
- Some members will have some underlying structure

structs within structs (cont)

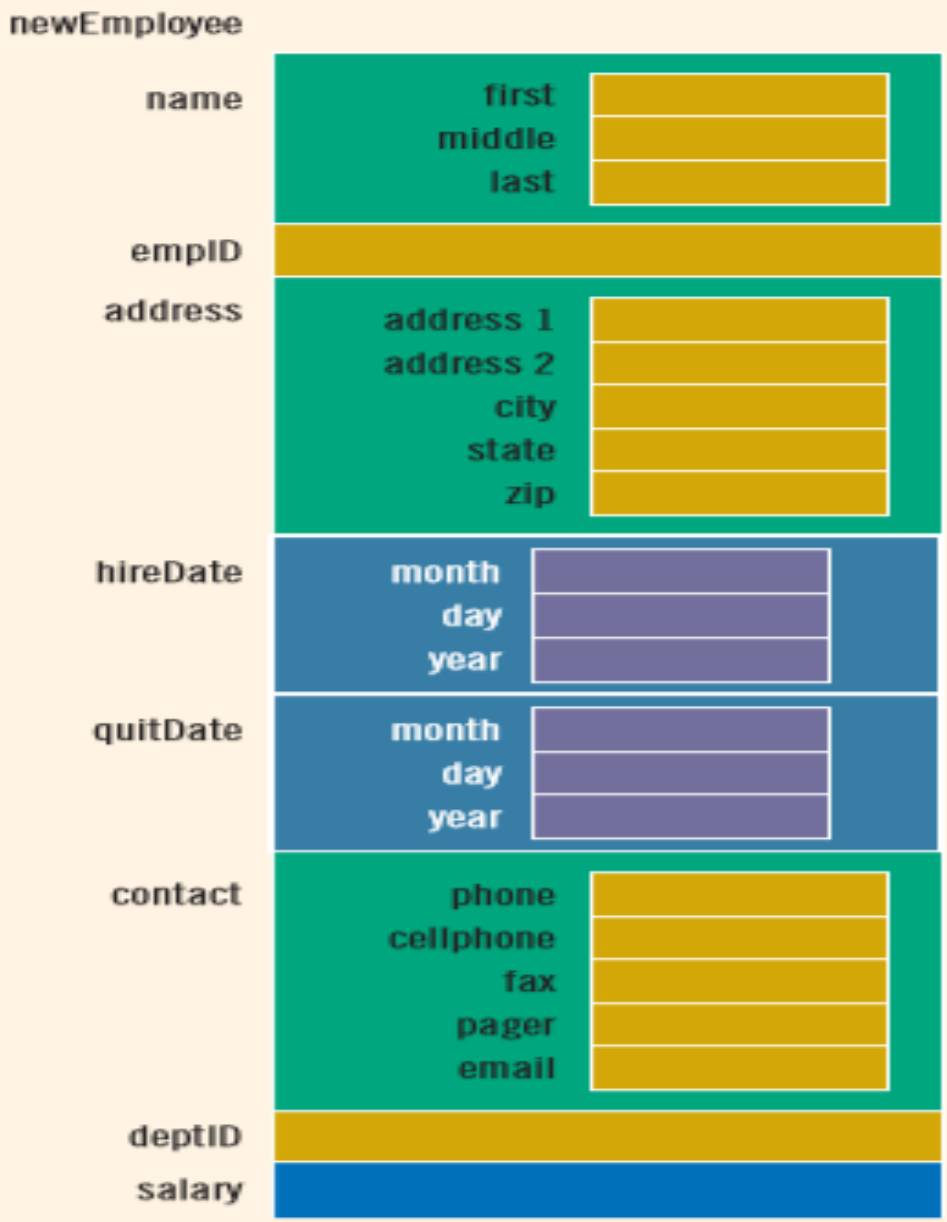
- Lets organize this **struct** as follows:

```
struct nameType
{
    string first;
    string middle;
    string last;
};

struct addressType
{
    string address1;
    string address2;
    string city;
    string state;
    string zip;
};
```

```
struct dateType
{
    int month;
    int day;
    int year;
};

struct contactType
{
    string phone;
    string cellphone;
    string fax;
    string pager;
    string email;
};
```


newEmployee
name
first
middle
last
empID
address
address 1
address 2
city
state
zip
hireDate
month
day
year
quitDate
month
day
year
contact
phone
cellphone
fax
pager
email
deptID
salary

Review Questions

- What is structs?
- What is the difference between structs and arrays?

Reference

- D. S. Malik. 2009. C++ Programming From Problem Analysis to Program Design.