Structures

Dr. Raman Adaikkalavan and Hakimzadeh
CS & Informatics, IUSB
Learning Objectives

• Structures
  – Creating a new Structure types
  – Creating a variable of this new type
  – Structures as function arguments
  – Creating an array of structure variables
Why do we need Structures?

- A **structure** provides the means to group together related data elements.

- A **structure** is heterogeneous aggregate of data elements.

- C++ allows programmers to define their own user-defined data types using the so called ‘Struct’ statement.
Structures

- Syntax:

```
Struct StructureName
{
  Datatype Field1;
  Datatype Field2; ...
}
```

- The name of the new structure
- Elements of the structure or member variables
Defining a new Structure

To define a new data type called Employee.

```c
struct Employee
{
    string Name;
    double HourlyWage;
    double HoursWorked;
    double GrossPay;
}; // the semicolon is necessary
```
Note:

- Defining a new structure does not actually create a variable. It just create a new user-defined data type!

- We need to declare a variable of this type to get a variable and allocate memory!
Declaring a variable using the new Employee Structure

Employee employee1;

A new variable of type Employee is created
Accessing the new Structure

- Fields are referenced (accessed) using the dot notation

```c
employee1.Name = "John Smith"
employee1.HourlyWage = 8.75
employee1.HoursWorked = 38
```

Now we can store values into the elements of the structure.
Example 1

Employee Structure
Example 1

```c
struct Employee
{
    string Name;
    double HourlyWage;
    double HoursWorked;
    double GrossPay;
}; // the semicolon is necessary
```

```c
void main()
{
    Employee employee1;
    employee1.Name = "John Smith";
    employee1.HourlyWage = 8.75;
    employee1.HoursWorked = 38;
    cout << "Employee Name: " << employee1.Name << endl;
    cout << "Hourly Wage  : " << employee1.HourlyWage << endl;
    cout << "Hourly Worked: " << employee1.HoursWorked << endl;
}
```

Defining the Employee structure
Declaring an Employee
Manipulating the Employee fields

(c) Copyright 2007, H. Hakimzadeh
Creating an Array of Structures

Let’s create an array of 5 employees:

Employee WorkGroup[5];
Storing employee information in the array of employees:

Employee WorkGroup[5];

WorkGroup[0].Name = "John Smith";
WorkGroup[0].HourlyWage = 8.75;
WorkGroup[0].HoursWorked = 38;
WorkGroup[0].GrossPay = WorkGroup[0].HourlyWage * WorkGroup[0].HoursWorked;
Storing employee information in the array of employees:

- WorkGroup[0].name = "John Smith"
- WorkGroup[0].HourlyWages = 8.75
- WorkGroup[0].HoursWorked = 38
- WorkGroup[0].GrossPay = WorkGroup[0].HoursWorked * WorkGroup[0].HourlyWages

- WorkGroup[1].name = "Mary Thomas"
- WorkGroup[1].HourlyWages = 12.75
- WorkGroup[1].HoursWorked = 20

....

....
Operation on Employee Structure

What are some common operations that one may perform on an employee structure?

1. Put information into the structure (e.g. insert/update)
2. Print the employee information
3. Calculate the Gross Pay for the employee
Reading Information Into a Structure:

```cpp
void ReadEmployeeInfo(Employee& AnEmployee)
{
    cout << "Enter Employee Name: ";
    cin >> AnEmployee.Name;

    cout << "Enter Hourly Wage: ";
    cin >> AnEmployee.HourlyWage;

    cout << "Enter Hours Worked: ";
    cin >> AnEmployee.HoursWorked;
}
```

Why pass it as a Reference Variable?
Print Function

```cpp
void PrintEmployee(Employee AnEmployee)
{
    cout << "Employee Name: " << AnEmployee.Name << endl;
    cout << "Hourly Wage : " << AnEmployee.HourlyWage << endl;
    cout << "Hourly Worked: " << AnEmployee.HoursWorked << endl;
}
```

Why NOT pass it as a Reference Variable?
Calculate the Gross Pay:

void CalculateGrossPay(Employee& AnEmployee)
{
}

Why pass it by Reference?
Putting it all together....

- Next we will create a complete program using structures
Main Program:

```c
struct Employee
{
    string Name;
    double HourlyWage;
    double HoursWorked;
    double GrossPay;
}; // the semicolon is necessary

void CreateAnEmployee(void);
void CreateArrayOfEmployees(void);
void ReadEmployeeInfo(Employee& AnEmployee);
void PrintEmployee(Employee AnEmployee);
void CalculateGrossPay(Employee& AnEmployee);

void main()
{
    CreateAnEmployee();
    CreateArrayOfEmployees();

    // Passing and employee structure to a function
    Employee employee2;
    ReadEmployeeInfo(employee2);
    PrintEmployee(employee2);
    CalculateGrossPay(employee2);
    PrintEmployee(employee2);
}
```
Additional things you can do!

- You can use Assignment with structs:

```c
employee3 = employee2;
```
Summary 1

• Structure is collection of different types, which makes a new user-defined type.

• You must first declare a variable of this new type, before you can use it.

• You use the “dot” notation to access the member variables of a struct.

• You can pass structs by value or by reference.

• You can create an array of structs and use it just like any other array!
Additional things you can do!

- You can use Assignment with structs:

  ```
  employee3 = employee2;
  ```
Additional things you can do!

You can `return` struct from a function:

```c
Main()
{
    Employee employee4;
    employee4 = ReturnAnEmployee();
}

Employee ReturnAnEmployee(void)
{
    Employee employee1;
    employee1.Name = "New Employee";
    employee1.HourlyWage = 18.75;
    employee1.HoursWorked = 25;
    return(employee1);
}
```
Additional things you can do!

- You can initialize a structure using the `{}` syntax.

```cpp
Employee employee5 = { "Emp 5", 10, 32, 10 * 32 };

employee5 = { "Emp 5", 20, 42, 20 * 42 };
```
Things you can’t do!

- Can’t compare two structures!

```java
if (employee4 == employee5)
    // do something
```

// won't work since the "==" operator is not defined for employees!!