

The C++ Language

Structured
Information

Data Types

- Recall the data types we have at our disposal
 - Basic types: `int`, `double`, `char`, `bool`
 - Integers, floating points, characters, booleans
 - Complex types: `string`, `ifstream`, `ofstream`
 - Aggregate types: arrays, parallel arrays, etc.
 - User defined types: `typedef`, `enum`

Aggregate Information

- Arrays
 - Store information of the same type
- What do we do if we want to store related information of different types?
 - Parallel arrays

3

Struble - Structured Information

Aggregate Information

- Store information about employees
 - First name, last name, age, hourly wage
 - Use an integer ID number for an index

ID	fnames	lnames	ages	hwages
0	"Mary"	"Jones"	27	14.25
1	"Eric"	"Allen"	43	18.72
2	"Nancy"	"Lane"	51	21.33
3	"John"	"Steele"	32	15.24
4	"Jacob"	"Brown"	24	13.56

4

Struble - Structured Information

Aggregate Information

```
// Prototypes
void InitStringArray(string array[], int usage, const string& val);
void InitIntArray(int array[], int usage, int val);
void InitDoubleArray(double array[], int usage, double val);
int ReadData(istream& In, string fnames[], string lnames[],
             int ages[], double hwages[]);

// Constants
const int NUM_EMPL = 5;
const string EMPTY_STRING = "";

// Main program
void main() {
    string fnames[NUM_EMPL], lnames[NUM_EMPL];
    int ages[NUM_EMPL];
    double hwages[NUM_EMPL];
    int usage = 0;
```

5

Struble - Structured Information

Aggregate Information (Main Cont.)

```
// Initialize your arrays
InitStringArray(fnames, NUM_EMPL, EMPTY_STRING);
InitStringArray(lnames, NUM_EMPL, EMPTY_STRING);
InitIntArray(ages, NUM_EMPL, 0);
InitDoubleArray(hwages, NUM_EMPL, 0.0);

usage = ReadData(cin, fnames, lnames, ages, hwages);
cout << "Read in " << usage << " employees." << endl;

// rest of the program computes with parallel arrays
...
} // end of main()
```

6

Struble - Structured Information

Aggregate Information

```
int ReadData(istream& In, string fnames[], string lnames[],
            int ages[], double hwages[])
{
    string fname, lname;    // temp first, last name
    int age, ID = 0;        // temp age, real employee ID
    double hwage;          // temp wage

    In >> fname >> lname >> age >> hwage;    // priming read
    while (In && ID < NUM_EMPL) {
        fnames[ID] = fname;                    // store information
        lnames[ID] = lname;                    // in parallel arrays
        ages[ID] = age;
        hwages[ID] = hwage;
        ID++;
        In >> fname >> lname >> age >> hwage; // next data set
    }
    return ID;
}
```

7

Struble - Structured Information

Analysis of Parallel Arrays

- Must declare several arrays
- Can be difficult to keep track of information for one item
 - e.g., have to access four arrays to read or store employee information
- Functions dealing with parallel arrays take many parameters
 - at least one for each array, plus one for usage

8

Struble - Structured Information

Wouldn't it be nice if...

- What if we could bundle all the information for an employee up?

<u>Employee</u>	anEmployee
fname	"John"
lname	"Steele"
age	32
hwage	15.23

9

Struble - Structured Information

Wouldn't it be nice if...

- If we could bundle it up, maybe we can
 - copy one employee to another
 - pass an employee as a parameter
 - have an array of employees
 - return an employee as the result of a function

10

Struble - Structured Information

Structures

- C++ provides a way to create bundles called *structures*
- A structure is an aggregate data type
 - stores related information of different types
- A structure is a user defined type
 - a type that you define for use in your program
 - like `typedef` or `enum`

11

Struble - Structured Information

Structures

- Syntax

```
struct StructIdentifier {  
    DataType Identifier ;  
    DataType Identifier ;  
    ...  
};
```

- *StructIdentifier* becomes the name of a type you can use in your program

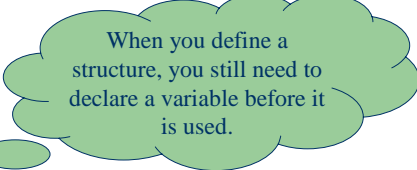
12

Struble - Structured Information

Structures (Example)

```
// Structure definition
struct Employee {
    string fname;
    string lname;
    int age;
    double hwage;
};

void main() {
    Employee anEmployee; // variable declaration
```



When you define a structure, you still need to declare a variable before it is used.

13

Struble - Structured Information

Terminology

- Structures are also called *records*
- Each piece of information is called a *field* or *member (variable)*
 - Although remember structures are NOT variables
- Each field is identified by its *field name* or *field identifier*

14

Struble - Structured Information

Accessing Structure Elements

- Structures allow us to bundle together several pieces of information of different types
- Need to access an individual piece of information
 - Use the dot (.) operator
- Syntax

```
StructureVariable.FieldName
```

15

Struble - Structured Information

Accessing Structure Elements (Example)

- Consider assigning values to each piece of information in `anEmployee`

```
anEmployee.fname = "Mary";  
anEmployee.lname = "Jones";  
anEmployee.age = 27;  
anEmployee.hwage = 14.25;
```

16

Struble - Structured Information

Accessing Structure Elements (Example)

- Read information for an employee

```
ifstream In("Employees.txt");  
  
In >> anEmployee.fname >> anEmployee.lname  
    >> anEmployee.age >> anEmployee.hwage;
```

17

Struble - Structured Information

Accessing Structure Elements (Example)

- Printing out the contents of an employee

```
ofstream Out("Report.txt");  
  
Out << "Name: " << anEmployee.fname << " "  
    << anEmployee.lname << endl;  
  
Out << "Age: " << anEmployee.age << endl;  
Out << "Hourly Wage: " << anEmployee.hwage << endl;
```

18

Struble - Structured Information

Accessing Structure Elements

- Once an element in a structure typed variable is accessed, the element acts just like a variable of the defined type.
- Example

```
Employee anEmployee2;  
int minAge;  
anEmployee2.age = 33;  
  
// minAge will be assigned 27, the minimum of the two ages  
minAge = Minimum(anEmployee.age, anEmployee2.age);
```

19

Struble - Structured Information

Accessing Structure Elements

- We can even pass two elements by reference.

```
void Swap(int& a, int& b) {  
    int temp;  
    temp = a;  
    a = b;  
    b = temp;  
}  
  
// swaps just the ages of the first employee  
// with the second. Mary Jones will now be 33  
// years old!  
Swap(anEmployee.age, anEmployee2.age);
```

20

Struble - Structured Information

Aggregate Operations

- So what can we do with structures?

```
Employee anEmployee1;
Employee anEmployee2;

anEmployee1 = anEmployee2;    // YES, assignment works
anEmployee1 == anEmployee2;   // NO, cannot compare
cout << anEmployee1;         // NO, must write own func
InitEmployee(anEmployee1);    // YES, value or reference
anEmployee1 = EmptyEmployee(); // YES, return val. OK
anEmployee1 + anEmployee2;    // NO, what would it mean?
```

21

Struble - Structured Information

Aggregate Operations (Equality)

```
bool EqualEmployees(const Employee& emp1,
                   const Employee& emp2)
{
    if (emp1.fname != emp2.fname)
        return false;

    if (emp1.lname != emp2.lname)
        return false;

    if (emp1.age != emp2.age)
        return false;

    if (emp1.hwage != emp2.hwage)
        return false;

    return true;
}
```

Pass by
constant
reference for
efficiency.

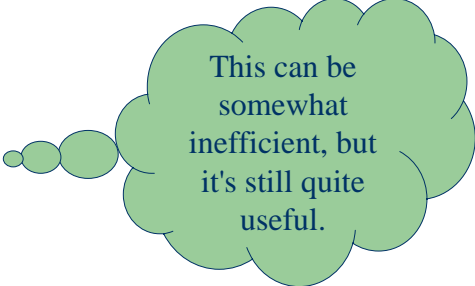
Must compare
each individual
field.

22

Struble - Structured Information

Aggregate Operations (Return Value)

```
Employee EmptyEmployee() {  
    Employee empty;  
  
    empty.fname = "";  
    empty.lname = "";  
    empty.age = 0;  
    empty.hwage = 0.0;  
  
    return empty;  
}
```



This can be somewhat inefficient, but it's still quite useful.

```
Employee anEmployee = EmptyEmployee();
```

23

Struble - Structured Information

Exercise

- Write a function that swaps two employee structure variables.

24

Struble - Structured Information

Arrays of Structures

- We can create arrays of structures just like we did with arrays of other types
- To access a field in an array element, first we index, then use the . operator

```
Employee employees[NUM_EMPL];

// Assign information to the fourth employee
employees[3].fname = "John";
employees[3].lname = "Steele";
employees[3].age = 32;
employees[3].hwage = 15.24;
```

25

Struble - Structured Information

Revisiting Earlier Code

- What about rewriting the code from slides 5-7?

```
// Employee Structure
struct Employee {
    string fname;    // Employee's first name
    string lname;   // Employee's last name
    int age;        // Employee's age
    double hwage;   // Employee's hourly wage
};

// Prototypes
void InitEmplArray(Employee array[], int usage, const Employee& val);
Employee EmptyEmployee();
int ReadData(istream& In, Employee array[]);

// Constants
const int NUM_EMPL = 5;
```

26

Struble - Structured Information

Rewriting Earlier Code

```
void main() {
    Employee employees[NUM_EMPL];
    Employee initVal = EmptyEmployee();
    int usage = 0;

    InitEmplArray(employees, NUM_EMPL, initVal);
    usage = ReadData(cin, employees);
    cout << "Read in " << usage << " employees." << endl;

    // do whatever else you want to do
    ...
}
```

27

Struble - Structured Information

Rewriting Earlier Code

```
int ReadData(istream& In, Employee array[]) {
    Employee employee;
    int ID = 0;

    In >> employee.fname >> employee.lname
    >> employee.age >> employee.hwage;
    while (In && ID < NUM_EMPL) {
        array[ID] = employee;
        ID++;
        In >> employee.fname >> employee.lname
        >> employee.age >> employee.hwage;
    }
    return ID;
}
```

28

Struble - Structured Information

Exercise

- Write the `InitEmplArray` function.

29

Struble - Structured Information

Hierarchical Structures

- Structures can contain fields with other structures
- Example
 - Add a birthdate to employees, without hierarchical structures

```
struct Employee {  
    string fname;           // first name  
    string lname;          // last name  
    int age;                // age  
    double hwage;          // hourly wage  
    Month birthMonth;      // Assume Month is an enumerated type  
    int birthDate;         // From 1 to 31  
    int birthYear;         // From 1900 onward  
};
```

30

Struble - Structured Information

Hierarchical Structures

- However, dates are bundles of information that we can act on too. Let's separate them out...

```
struct Date {
    Month month;    // assume month is an enumerated type
    int date;      // from 1 to 31
    int year;      // from 1900 onward
};

struct Employee {
    string fname;    // first name
    string lname;   // last name
    int age;         // age
    double hwage;   // hourly wage
    Date  birthDate; // the employee's birthdate
};
```

31

Struble - Structured Information

Accessing Hierarchical Information

- We use the dot operator multiple times to access hierarchical information

```
Employee anEmployee;

// Birthday for someone born to be an IRS agent
anEmployee.birthDate.month = APRIL;
anEmployee.birthDate.date = 15;
anEmployee.birthDate.year = 1973;
```

32

Struble - Structured Information

Hierarchical Information

anEmployee		
fname		
lname		
age		
hwage		
birthDate	month	APRIL
	date	15
	year	1973

33

Struble - Structured Information

Arrays In Structures

- You can place arrays in structures too

```
struct Course {
    string name;           // name of the course
    int test1[NUM_STUDENTS]; // test1 grades for course
    Student students[NUM_STUDENTS]; // student info
};
```

```
Course cs1044;
// Set the test grade for the 43rd student
cs1044.test1[42] = 84;
```

34

Struble - Structured Information

Structure Initialization

- Structures do support initialization at declaration time
 - Looks similar to array initialization
 - Doesn't work for strings however...
- Example

```
Date taxDay = {APRIL, 16, 2001};
```

	taxDay
month	APRIL
date	16
year	2001