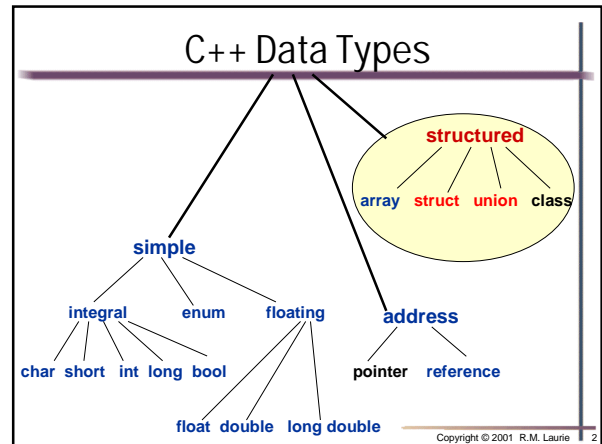


Data Record Definitions

- **Arrays**
 - *Homogeneous* data of the same type
 - Accessed by in index (element) number
 - Very efficient in a **for** loop process.
- **Records**
 - *Non-homogeneous* structured data type.
 - Records are accessed by name not index
 - In programming implemented as structure
 - C++ uses the keyword struct to define
- **Fields** are data components of a Record
 - Do not need to be of the same data type

Copyright © 2001 R.M. Laurie 1



Data Record Syntax

```

struct StructureName
{
    datatype1 FieldName1;
    datatype2 FieldName2;
};
    
```

**struct
type Declaration**

```

sStructureName RecordName1;
sStructureName RecordName2;
    
```

**Member List
Declaration**

Copyright © 2001 R.M. Laurie 3

Data Member Syntax

Use the **dot** operator between the Record Name (Structure Name) and the Field Name (Data Member) to choose a specific field

```

RecordName1.FieldName1;
RecordName1.FieldName2;
    
```

Copyright © 2001 R.M. Laurie 4

Data Record Declaration

```

#include <iostream>
#include <cstring>
using namespace std;
struct PersonalData
{
    char szName[80], szPhoneNumber[80], cSex;
    int nAge, nHeight;
    float fSalary;
};
    
```

Copyright © 2001 R.M. Laurie 5

Data Record Access

```

int main(void)
{
    PersonalData sPresident;
    strcpy(sPresident.szName, "Mr. Wizard");
    strcpy(sPresident.szPhoneNumber, "671-399-4784");
    sPresident.nAge = 39;
    cout << sPresident.szName << endl
         << sPresident.szPhoneNumber << endl
         << sPresident.nAge << endl
         << sizeof(sPresident.szName) << " "
         << sizeof(sPresident.nAge) << " "
         << sizeof(sPresident) << endl;
    return(0);
}
    
```

```

Mr. Wizard
671-399-4784
39
80 4 176
    
```

Copyright © 2001 R.M. Laurie 6

Related Items (Zoo Example)

sCageA	
.id	2037581
.name	"giant panda"
.genus	"Ailuropoda"
.species	"melanoluka"
.country	"China"
.age	18
.weight	234.6
.health	Good

Copyright © 2001 R.M. Laurie 7

Related Items (Zoo Example)

sCageB	
.id	5281003
.name	"llama"
.genus	"Lama"
.species	"peruana"
.country	"Peru"
.age	7
.weight	278.5
.health	Excellent

Copyright © 2001 R.M. Laurie 8

struct AnimalType

```
enum HealthType { Poor, Fair, Good, Excellent };
struct AnimalType // declares a struct data type
                  // does not allocate memory
{
    long    id;
    char    name[40];
    char    genus[40];
    char    species[40]; // struct members
    char    country[40];
    int     age;
    float   weight;
    HealthType health;
};
AnimalType sCageA; // declare variables of AnimalType
AnimalType sCageB;
```

Copyright © 2001 R.M. Laurie 9

struct type Declaration

- The struct declaration names a type and names the members of the struct.
- It does not allocate memory for any variables of that type!
- You still need to declare your struct variables which allocates memory
- You can declare arrays of records using a specific structure, just like any other array type using an index (element) number

Copyright © 2001 R.M. Laurie 10

Arrays of Records Access

```
#include <iostream>
#include <cstring>
using namespace std;
struct PersonalData
{
    char szName[80], szPhoneNumber[80], cSex;
    int  nAge, nHeight;
    float fSalary;
};
```

Copyright © 2001 R.M. Laurie 11

Arrays of Records Access

```
int main()
{
    PersonalData sPresident, sWorker[10];
    strcpy(sPresident.szName, "George Washington");
    strcpy(sWorker[4].szName, "Sneezy");
    strcpy(sWorker[5].szName, "Sleepy");
    sWorker[5].fSalary = 23333.33;
```

Copyright © 2001 R.M. Laurie 12

Arrays of Records Access

```
cout << sWorker[5].szName << endl
    << sWorker[5].szName[5] << endl
    << "$" << sWorker[5].fSalary << endl
    << sizeof(sWorker[5].szName) << " "
    << sizeof(sWorker[5].nAge) << " "
    << sizeof(sWorker[5]) << endl
    << sizeof(sWorker) << endl;

return 0;
}
```

```
Sl eepy
y
$23333.3
80 2 176
1760
```

Copyright © 2001 R.M. Laurie 13

Alternative Record Declaration Syntax

```
struct StructureName
{
    datatype1 FieldName1;
    datatype2 FieldName2;
}sRecordName1, sRecordName2;
```

However, it is far better to use the original form so that you can control the scope and life time of structure variables

```
struct StructureName
{
    datatype1 FieldName1;
    datatype2 FieldName2;
};
StructureName sRecordName1;
StructureName sRecordName2;
```

Copyright © 2001 R.M. Laurie 14

Hierarchiacal Records

Definition: Records that contain as a component (field) another record

```
struct PhoneNumbers
{
    char szPhoneHome[80];
    char szPhoneWork[80];
    int nExtension;
}
```

Copyright © 2001 R.M. Laurie 15

Hierarchiacal Records

• **Definition: Records that contain as a component (field) another record**

```
struct PersonalData
{
    char szName[80], cSex;
    PhoneNumbers Telephone;
    int nAge;
    float fSalary;
};
struct PhoneNumbers
{
    char szPhoneHome[80];
    char szPhoneWork[80];
    int nExtension;
};
```

Copyright © 2001 R.M. Laurie 16

Hierarchiacal Records Access

```
sPresident.Telephone.PhoneWork = "347-87";
```

```
cin >> sWorker[3].Telephone.nExtension;
```

```
sWorker[5].cSex = 'Y';
```

Copyright © 2001 R.M. Laurie 17

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 from a function

```
anotherAnimal = thisAnimal; // assignment
WriteOut(thisAnimal); // value parameter
ChangeWeightAndAge(thisAnimal); // reference
thisAnimal = GetAnimalData( ); // return value
```

Copyright © 2001 R.M. Laurie 18

Pass by Value struct Parameter

```
void WriteOut( /* in */ 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 ) ;
}
}
```

Copyright © 2001 R.M. Laurie 19

Pass by Reference struct Parameter

```
void ChangeAge ( /* inout */ AnimalType& thisAnimal)
// Adds 1 to age
// Precondition:  thisAnimal.age is assigned
// Postcondition:  thisAnimal.age ==
                  thisAnimal.age@entry + 1
{
    thisAnimal.age++ ;
}
}
```

Copyright © 2001 R.M. Laurie 20

Pass by Reference struct Parameter

```
AnimalType GetAnimalData ( void )
// Obtains all information about an animal from keyboard
// Postcondition:
// Function value == AnimalType members entered at kbd
{
    AnimalType thisAnimal ;
    char        response ;
    do {
        // have user enter all members until they are correct
        .
        .
        .
    } while (response != 'Y') ;
    return thisAnimal ;
}
}
```

Copyright © 2001 R.M. Laurie 21

Unions

- Definition: Structure that holds only one of its members at a time during program execution.
- Purpose: Used to conserve memory by allowing data of different types to be stored at same location
- See p. 589

```
union WeighType
{
    long wtOunces;
    int  wtPounds;
    float wtTons;
}
}
```

Copyright © 2001 R.M. Laurie 22

Data Abstraction

- Definition: Separation of a data type's logical properties from its implementation
- Purpose: Hides implementation details from user.

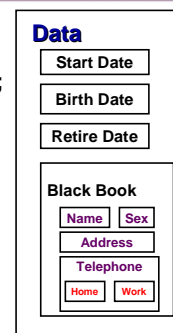
```
struct DateType      struct DateType
{
    int nDay;         char cDay[2];
    int nMonth;      char cMonth[3];
    int nYear;       char cYear[4];
}
}
```

Copyright © 2001 R.M. Laurie 23

Data Abstraction

```
struct EmployeeData
{
    PersonalData BlackBook;
    DateType StartDate;
    DateType BirthDate;
    DateType RetireDate;
};

sEmployeeNumber[1000];
```



Copyright © 2001 R.M. Laurie 24