Structures in C++
Introduction to Linked Lists

CS 16: Solving Problems with Computers I
Lecture #14

Ziad Matni
Dept. of Computer Science, UCSB

Lecture Outline

Structures (Ch. 10.1)

- Defining structures
- Member variables and functions
- Structures in functions
- Hierarchy in structures
- Initializing structures

Linked Lists (Ch. 13.1)

- We will cover everything in this section
  - We are not covering **Ch. 13.2** section!

First… What Is a Class?

- A ***class*** is a data type whose variables are called ***objects***
- Some pre-defined data types you have used are: **int**, **char**, **double**
- Some pre-defined classes you have used are: **ifstream**, **string**, **vector**
- You can also define your own classes as well

Class Definitions

- To define a "class", we need to…
  - Describe the **kinds of values** the variable can hold
    - Numbers? Characters? *Both*? Something else?
  - Describe the **member functions**
    - What can we do with these values?

- We will start by defining ***structures*** as a first step toward defining classes

STRUCTURES

Structures

- A structure’s use can be viewed as an **object**
- Let’s say it does not contain any member functions (for now…)
- It does contain multiple values of possibly different types
- We’ll call these **member variables**

Structures for Data

- These multiple values are logically related to one another and come together as a single item
  - Examples:
    A bank Certificate of Deposit (CD) which has the following values:

    a balance
    an interest rate
    a term (how many months to maturity)

    What kind of values should these individually be?!

  - A student record which has the following values:

    the student’s ID number
    the student’s last name
    the student’s first name
    the student’s GPA

    What kind of values should these individually be?!

The CD Structure Example: Definition

- The Certificate of Deposit structure can be defined as

```
struct CDAccount
{
    double balance;         // a dollar amount
    double interest_rate;   // a percentage
    int term;               // a term amount in months
} ;
```

Remember this semicolon!

- Keyword **`struct`** begins a structure definition
- **`CDAccount`** is the structure *tag* – this is the structure’s **type**
- Member names are *identifiers* declared in the braces

Using the Structure

- Structure definition should be placed ***outside*** any function definition
  - Including outside of **main( )**
  - This makes the structure type available to all code that follows the structure definition (i.e. global)

- To declare two variables of type **`CDAccount`**:

```
CDAccount  my_account, your_account;
```

`my_account` and `your_account`
contain distinct member variables **`balance`**, **`interest_rate`**, and **`term`**

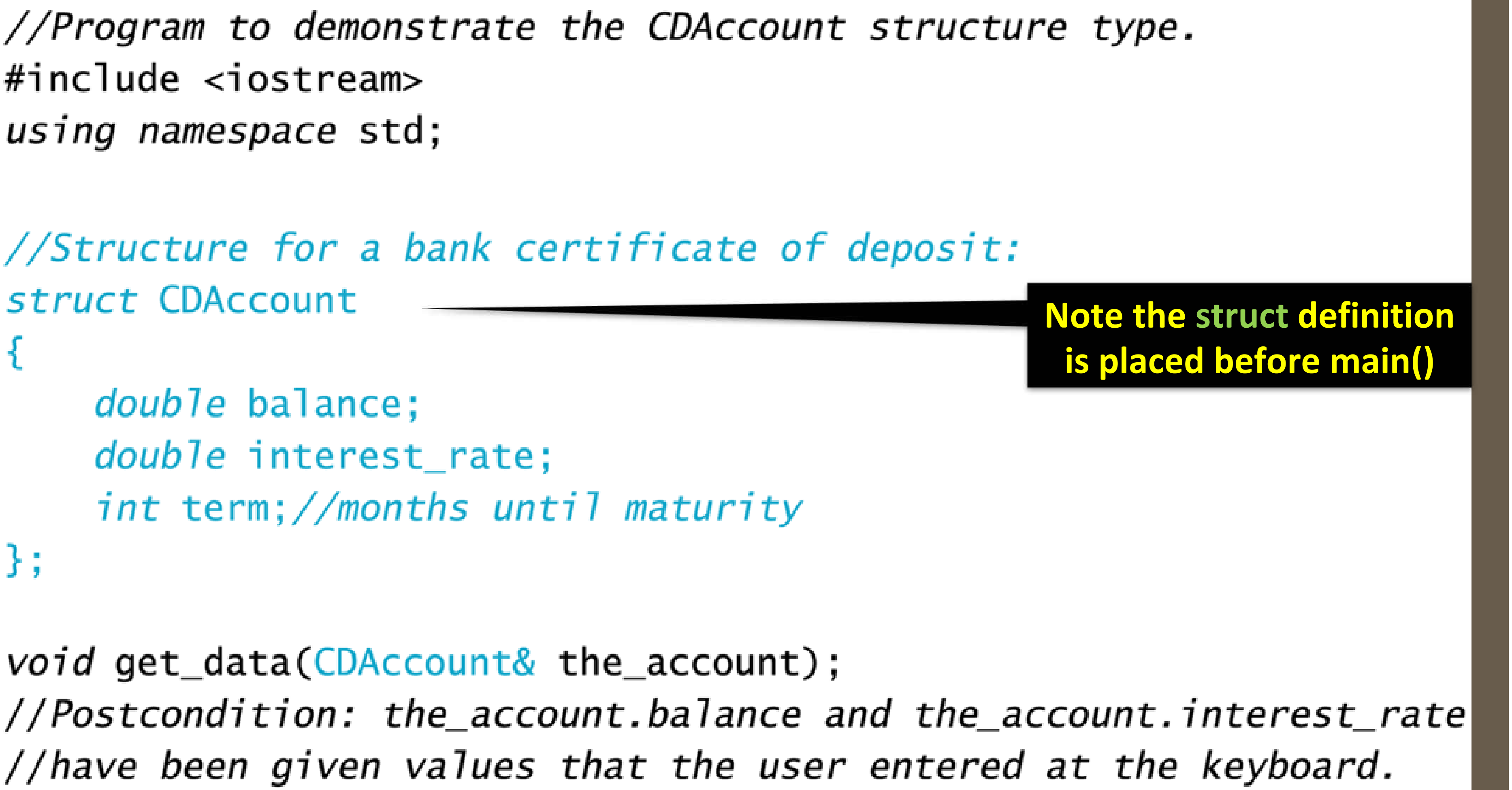
Specifying Member Variables

- Member variables are specific to the structure variable in which they are declared

- Syntax to specify a member variable (note the '.')
  Structure_Variable_Name . Member_Variable_Name

- Given the declaration:
```
CDAccount  my_account, your_account;
```

- Use the **dot operator** to specify a member variable, e.g.
```
my_account.balance              is a double
my_account.interest_rate        is a double
my_account.term                 is an int
```

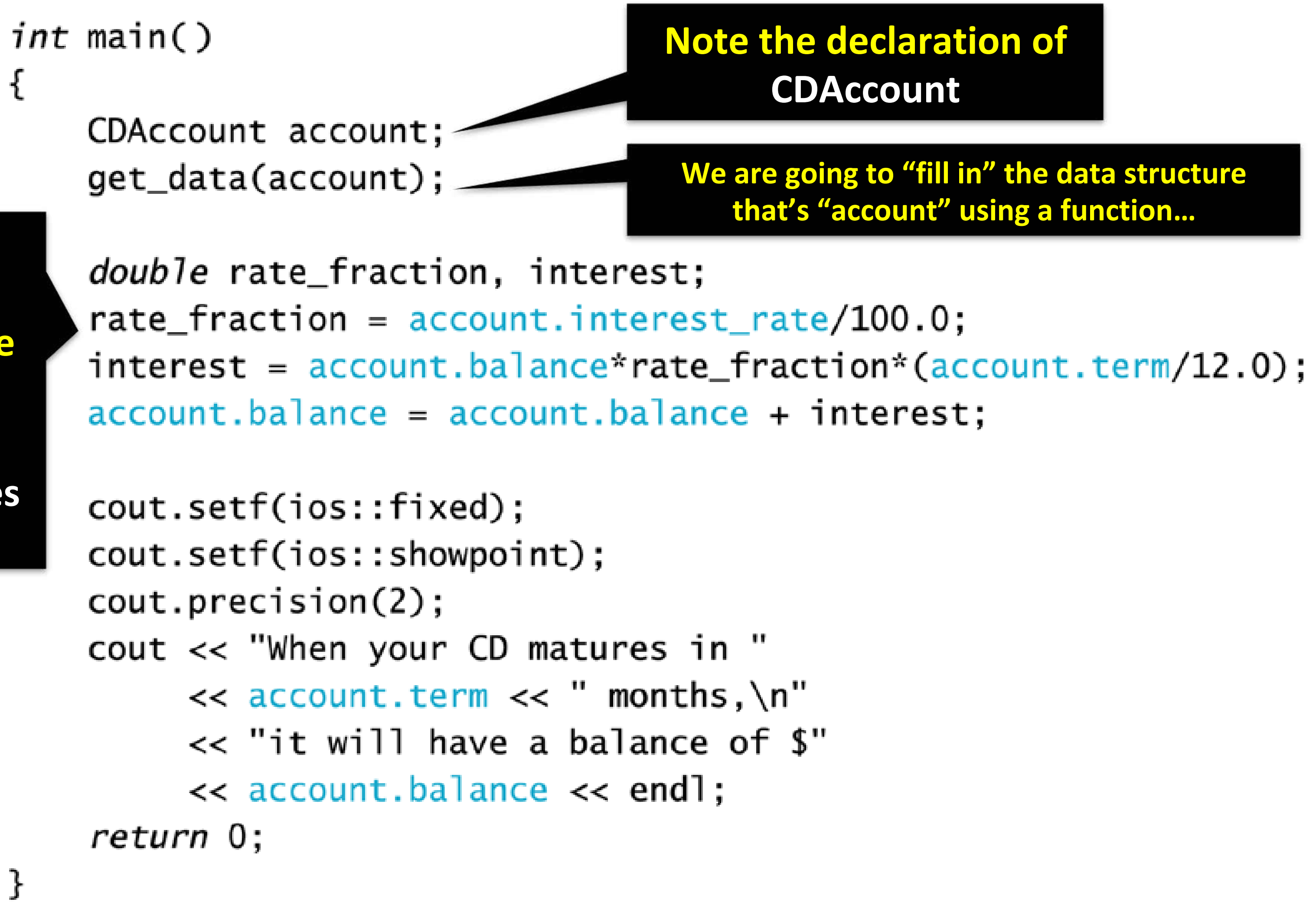


```
//Program to demonstrate the CDAccount structure type.
#include <iostream>
using namespace std;

//Structure for a bank certificate of deposit:
struct CDAccount
{
    double balance;
    double interest_rate;
    int term;//months until maturity
};

void get_data(CDAccount& the_account);
//Postcondition: the_account.balance and the_account.interest_rate
//have been given values that the user entered at the keyboard.
```

Note the struct definition is placed before main()



```
int main()
{
    CDAccount account;
    get_data(account);

    double rate_fraction, interest;
    rate_fraction = account.interest_rate/100.0;
    interest = account.balance*rate_fraction*(account.term/12.0);
    account.balance = account.balance + interest;

    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(2);
    cout << "When your CD matures in "
         << account.term << " months,\n"
         << "it will have a balance of $"
         << account.balance << endl;
    return 0;
}
```

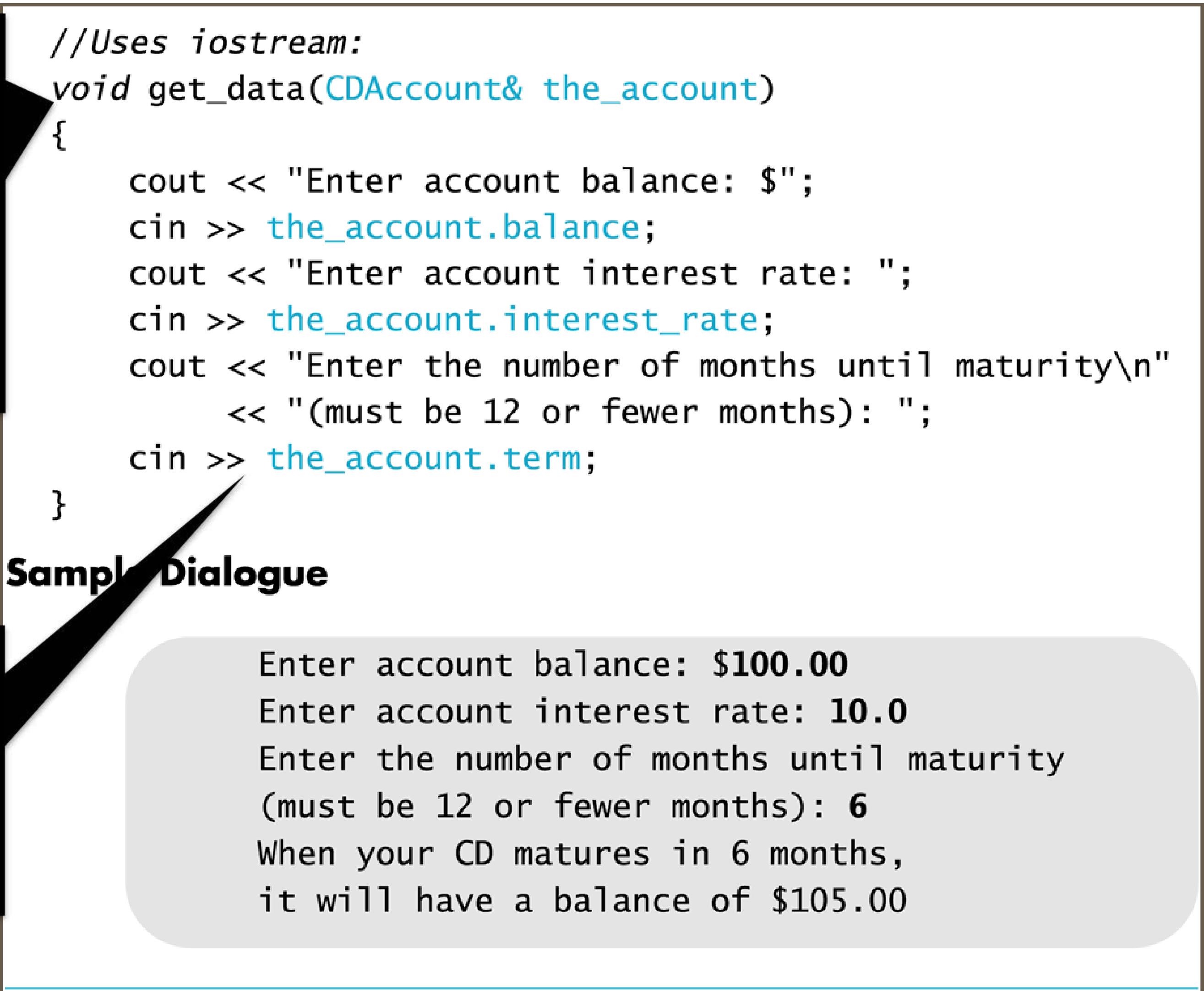
Note the declaration of CDAccount

We are going to “fill in” the data structure that’s “account” using a function…

Note the calculations done with the structure’s member variables

Note that the structure is passed into the function as call-by-reference.

You can also pass a structure call-by-value.



```
//Uses iostream:
void get_data(CDAccount& the_account)
{
    cout << "Enter account balance: $";
    cin >> the_account.balance;
    cout << "Enter account interest rate: ";
    cin >> the_account.interest_rate;
    cout << "Enter the number of months until maturity\n"
         << "(must be 12 or fewer months): ";
    cin >> the_account.term;
}
```

Sample Dialogue

```
Enter account balance: $100.00
Enter account interest rate: 10.0
Enter the number of months until maturity
(must be 12 or fewer months): 6
When your CD matures in 6 months,
it will have a balance of $105.00
```

Note the use of the structure's member variables with an input stream.

Duplicate Names

- Member variable names duplicated between structure types are **not** a problem

```
struct FertilizerStock
{
    double quantity;
    double nitrogen_content;
};

FertilizerStock  super_grow;
```

```
struct CropYield
{
   int quantity;
   double size;
};

CropYield  apples;
```

- This is because we have to use the dot operator
- **super_grow.quantity** and **apples.quantity** are different variables stored in different locations in computer memory

Structures as Return Function Types

- Structures **can also** be the type of a value ***returned*** by a function

Example:

```
CDAccount shrink_wrap
           (double the_balance, double the_rate, int the_term)
{
    CDAccount temp;
    temp.balance = the_balance;
    temp.interest_rate = the_rate;
    temp.term = the_term;
    return temp;
}
```

What is this function doing?

Example: Using Function **shrink_wrap**

- **shrink_wrap** builds a complete structure value in the structure **temp**, which is returned by the function

- We can use **shrink_wrap** to give a variable of type **CDAccount** a value in this way:

```
CDAccount  new_account;
new_account = shrink_wrap(1000.00, 5.1, 11);
```

Assignment and Structures

- The assignment operator (=) can also be used to give values to structure types
- Using the CDAccount structure again for example:

```
CDAccount my_account, your_account;
my_account.balance = 1000.00;
my_account.interest_rate = 5.1;
my_account.term = 12;
your_account = my_account;
```

- Note: This last line assigns *all member variables* in **your_account** the corresponding values in **my_account**

Hierarchical Structures

- Structures **can** contain member variables that are **also structures**

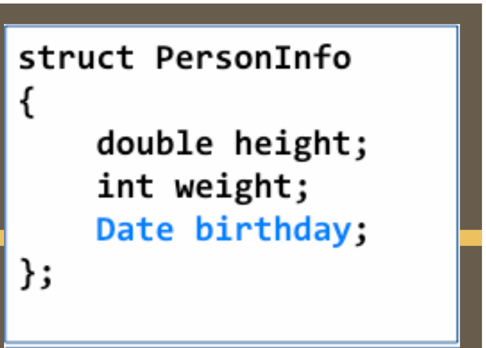
```
struct Date
{
   int month;
   int day;
   int year;
};
```

```
struct PersonInfo
{
    double height;
    int weight;
    Date birthday;
};
```

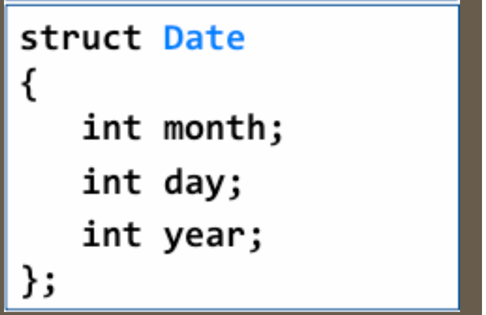
- struct **PersonInfo** contains a **Date** structure

Using **PersonInfo**

*An example on **"."** **operator** use*



```
struct PersonInfo
{
    double height;
    int weight;
    Date birthday;
};
```



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

- A variable of type **PersonInfo** is declared:

```
PersonInfo person1;
```

- To display the birth year of **person1**,
  first access the birthday member of person1

```
cout <<  person1.birthday…(wait! not complete yet!)
```

- But we want the ***year***, so we now specify the year member of the birthday member

```
cout << person1.birthday.year;
```

Initializing Structures

- A structure can be initialized when declared

Example:

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

month day year

- Can be initialized in this way – watch out for the order!:

```
Date due_date = {4, 20, 2018};
Date  birthday = {12, 25, 2000};
```

Application of Structures

Linked Lists!

Pointers and Linked Lists

- **Definition of Linked Lists:**
  Linear collection of data elements, called ***nodes***, each pointing to the *next* node by means of a pointer

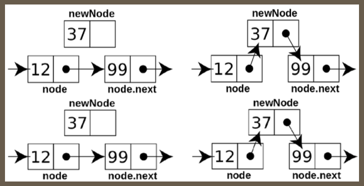
- List elements can easily be **inserted** or **removed** *without* reorganization of the entire structure (unlike arrays)

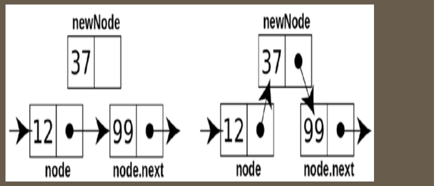
- Data items in a linked list do not have to be stored in one large memory block (again, unlike arrays)

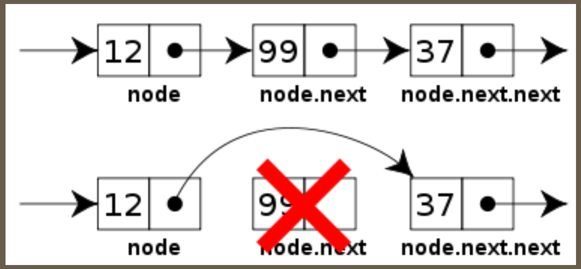
Linked Lists

- You can build a list of “nodes” which are made up of variables and pointers to create a chain.

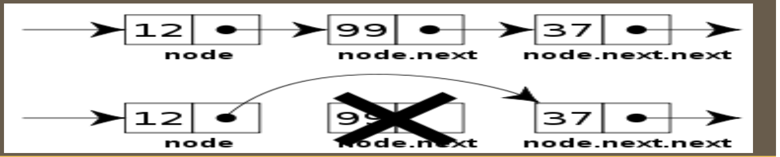
- Adding and deleting nodes in the link can be done by “re-routing” pointer links.







Nodes

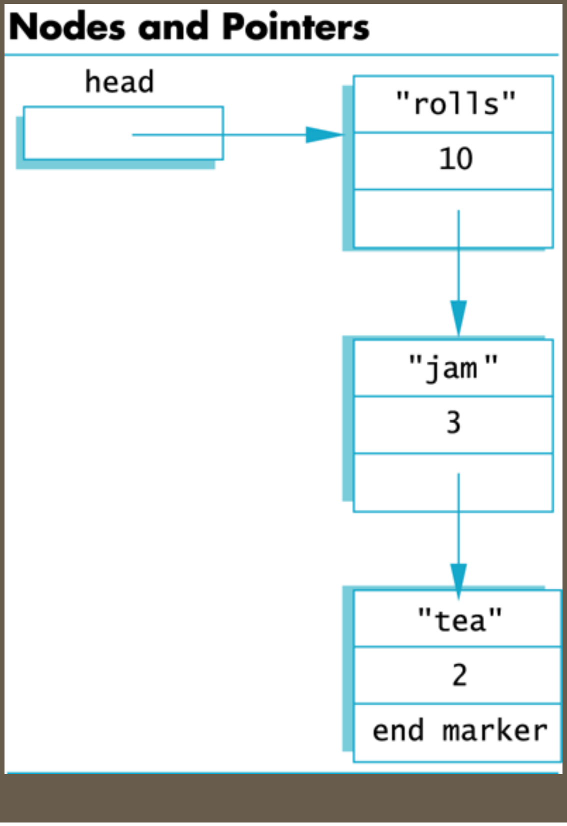


- The boxes in the previous drawing represent the **nodes** of a linked list
  - Nodes contain the data item(s) **and** a pointer that can point to another node of the same type
  - The pointers **point to an entire node**, not an individual item that might be in the node

- The arrows in the drawing represent pointers

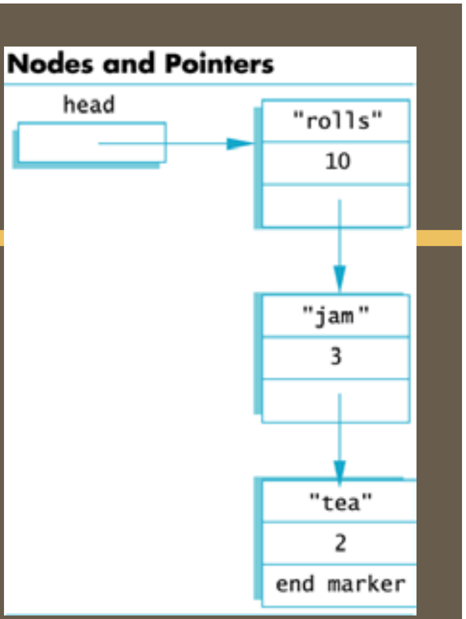
Nodes and Pointers – An Illustrated Example

(shown as Display 13.1 in the textbook)



Nodes and Pointers

Implementing Nodes



- Nodes are implemented in C++ as **structs** or **classes**
- *Example*: A structure to store two data items and a pointer to another node of the same type, along with a type definition might be:

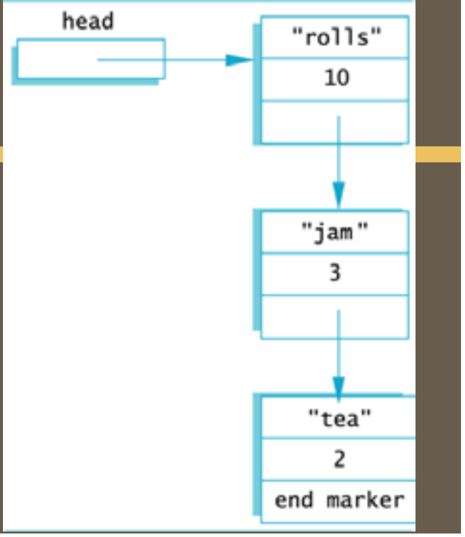
```
struct ListNode
{
   string item;
   int count;
   ListNode *link;
};

typedef ListNode* ListNodePtr;
```

This circular definition is allowed in C++

The **head** of a List

Nodes and Pointers



- The box labeled head, in Display 13.1, is not a node, but simply a **pointer variable** that points to a node

- Pointer variable head is declared as:

```
ListNodePtr head;
```

```
struct ListNode
{
    string item;
    int count;
    ListNode *link;
};
typedef ListNode* ListNodePtr;
ListNodePtr head;
```

Creating a Linked List

- First create the node(s)

```
ListNode myNode1, myNode2;
myNode1.item = “Thingamajiggie”;
myNode1.count = 5;  // etc…
```

- Then link the head pointer to the 1st node in the list

```
head = new ListNode;
*head = myNode1;
// i.e. “what head links to is myNode1”
```

- Then link all the other nodes to each other

```
*(myNode1.link) = myNode2;  // etc…
```

Check out demo:
linkedList.cpp

YOUR TO-DOs

- ❑ Turn in Lab 8 on Monday
- ❑ Do HW14 by **Tuesday**

- ❑ Visit TAs‘ office hours if you need help!
  - ❑ Prof. will not have office hours next Monday (University holiday)

- ❑ Enjoy the long weekend! ☺

</LECTURE>