CS 31: Introduction To Computer Science I

Howard A. Stahl

Agenda

- Structures
- Introducing Object-Oriented Programming
- Objects and Classes

Structures

- An “Aggregate” Data Type
  - Combining Together Different Types Of Data Into A Newly Defined Type

Structures

- An “Aggregate” Data Type
  - Combining Together Different Types Of Data Into A Newly Defined Type

Unlike An Array, We Have To Declare A `struct` Before We Use Them…

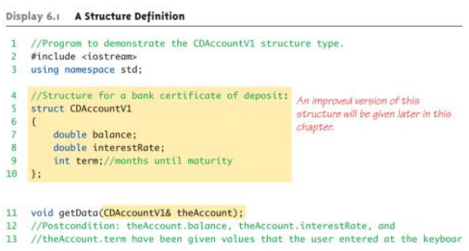
Structures

- An “Aggregate” Data Type
  - Combining Together Different Types Of Data Into A Newly Defined Type

Unlike An Array, We Have To Declare A `struct` Before We Use Them…

ike An Array, But An Array Is A Set Of The Same Type…

Textbook Example

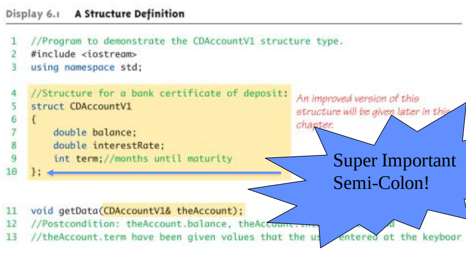


Display 6.1 **A Structure Definition**

```
//Program to demonstrate the CDAccountV1 structure type.
#include <iostream>
using namespace std;

//Structure for a bank certificate of deposit:
struct CDAccountV1
{
    double balance;
    double interestRate;
    int term;//months until maturity
};


void getData(CDAccountV1& theAccount);
//Postcondition: theAccount.balance, theAccount.interestRate, and
//theAccount.term have been given values that the user entered at the keyboar
```

An improved version of this structure will be given later in this chapter.

Textbook Example



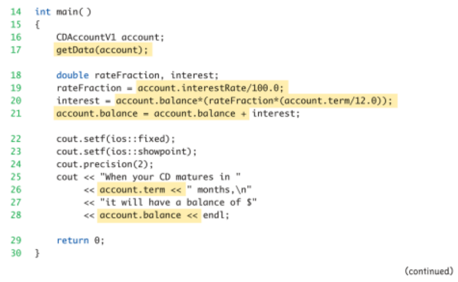
Display 6.1 **A Structure Definition**

```
//Program to demonstrate the CDAccountV1 structure type.
#include <iostream>
using namespace std;

//Structure for a bank certificate of deposit:
struct CDAccountV1
{
    double balance;
    double interestRate;
    int term;//months until maturity
};

void getData(CDAccountV1& theAccount);
//Postcondition: theAccount.balance, theAcc
//theAccount.term have been given values that the u  entered at the keyboar
```

An improved version of this structure will be given later in this chapter.

Super Important Semi-Colon!

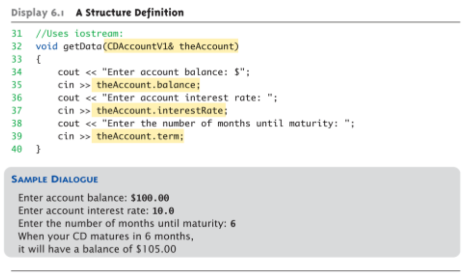
Textbook Example



```
int main( )
{
    CDAccountV1 account;
    getData(account);

    double rateFraction, interest;
    rateFraction = account.interestRate/100.0;
    interest = account.balance*(rateFraction*(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;
}
```

(continued)

Textbook Example



Display 6.1 **A Structure Definition**

```
//Uses iostream:
void getData(CDAccountV1& theAccount)
{
    cout << "Enter account balance: $";
    cin >> theAccount.balance;
    cout << "Enter account interest rate: ";
    cin >> theAccount.interestRate;
    cout << "Enter the number of months until maturity: ";
    cin >> theAccount.term;
}
```

Sample Dialogue

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

`struct` Details

- Defining A `struct` Tells C++ What It "Looks Like"
- No Memory Is Actually Allocated Until You Declare A Variable Of That Type
- Every `struct` Has A
  - Name Data Member Variables

`struct` Details

- Defining A `struct` Tells C++ What It "Looks Like"
- No Memory Is Actually Allocated Until You Declare A Variable Of That Type
- Every `struct` Has A
  - Name ← Essentially A Brand New Type!
  - Data Member Variables

`struct` Details

- Defining A `struct` Tells C++ What It "Looks Like"
- No Memory Is Actually Allocated Until You Declare A Variable Of That Type
- Every `struct` Has A
  - Name ← Essentially A Brand New Type!
  - Data Member Variables ← Parts Of It!

Accessing A `struct`

- Use . Syntax To Access Data Members
  - `account.balance`
  - `account.term`
  - `account.interestRate`
- Member Variables Can Have The Same Name As The `struct` Itself
  - No Conflict, Maybe Confusing, But Legal

All Other Laws Of Physics Apply

- `struct` Can Be Declared And Initialized

  -
    ```
    struct Date
     {
         int month;
         int day;
         int year;
     };
     Date dueDate = {12, 31, 2003};
    ```

All Other Laws Of Physics Apply

- `struct` Can Work With Functions
  - Passed By Value
  - Passed By Reference
  - Passed By Constant Reference
  - Can Be Returned By A Function To The Caller
    - The `return` Statement Would Need To Pass By A Variable Of The Correct `struct` Type

All Other Laws Of Physics Apply

- `struct` Can Work With Functions
  - Passed By Value
  - Passed By Reference
  - Passed By Constant Reference
  - Can Be Returned By A Function To The Caller
    - The `return` Statement Would Need To Pass By A Variable Of The Correct `struct` Type

Just Like
Any Other
Simple Type

Function-Oriented Programming

- Up Until Now, Everything We Have Learned Is Closely Related To C
- Programs Are Collections Of Functions With Controlling Drivers
- Program Structure Decomposes Algorithms Into Isolated Functions
- *functions*, *procedures* And *subroutines* Are The Primary Program Structure

Object-Oriented Programming

- Object-Orientation Is Where C++ Differs From C
- Programs Viewed As A Collections Of Collaborating Objects
- Closely Models The Real World
- Program Structure Implemented Via *classes* And *objects*

Objects

- Consider My Car:

PROPERTIES
Make: Honda
Model: Prelude

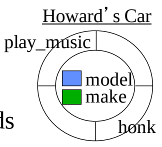
FUNCTIONALITY
play_music
toggle_left_blinker
honk

An Object Has...

- State Described Via Attributes
  - every car has a make and a model
- Behavior Described Via Methods
  - every car can honk its horn
- Identity Described Via Instances
  - from the sea of all Honda Preludes, I can identify the one that is mine

An Object Has...

- State Described Via Attributes
  - every car has a make and a model
- Behavior Described Via Methods
  - every car can honk its horn
- Identity Described Via Instances
  - from the sea of all Honda Preludes, I can identify the one that is mine



Classes

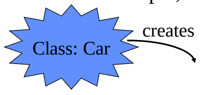
- Describe Similar Kinds Of Things
  - for example, consider the class of all `int`'s
- Programs Let Us Declare An Instance Of This Type
  - for example, `int i,j,k;`

Classes

- Describe Similar Kinds Of Things
  - for example, consider the class of all `int`'s
- Programs Let Us Declare An Instance Of This Type
  - for example, `int i,j,k;`

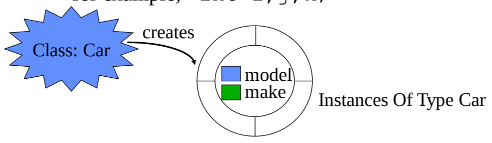
Classes

- Describe Similar Kinds Of Things
  - for example, consider the class of all `int`'s
- Programs Let Us Declare An Instance Of This Type
  - for example, `int i,j,k;`



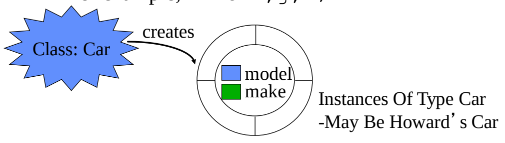
Classes

- Describe Similar Kinds Of Things
  - for example, consider the class of all `int`'s
- Programs Let Us Declare An Instance Of This Type
  - for example, `int i,j,k;`



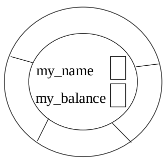
Classes

- Describe Similar Kinds Of Things
  - for example, consider the class of all `int`'s
- Programs Let Us Declare An Instance Of This Type
  - for example, `int i,j,k;`

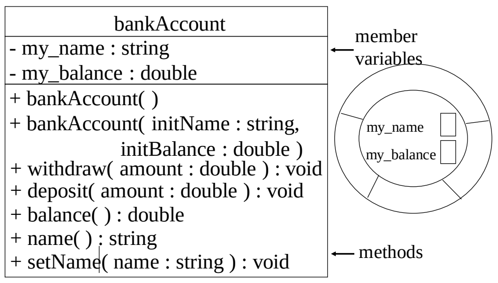


Example:Bank Account

bankAccount
- my_name : string - my_balance : double
+ bankAccount( ) + bankAccount(initName : string, initBalance : double) + withdraw(amount : double) : void + deposit(amount : double) : void + balance( ) : double + name( ) : string + setName(name : string) : void



Example:Bank Account



```
bankAccount
- my_name : string
- my_balance : double
+ bankAccount( )
+ bankAccount( initName : string,
               initBalance : double )
+ withdraw( amount : double ) : void
+ deposit( amount : double ) : void
+ balance( ) : double
+ name( ) : string
+ setName( name : string ) : void
```

Instantiation

- Like Any Other Variable, Instances Must Be Declared Before They Are Used

Instantiation

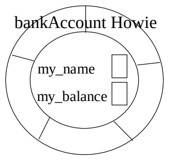
- Like Any Other Variable, Instances Must Be Declared Before They Are Used

```
bankAccount Howie;
```

Instantiation

- Like Any Other Variable, Instances Must Be Declared Before They Are Used

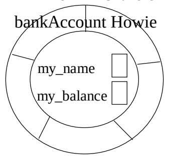
```
bankAccount Howie;
```



Interacting With Objects

- Like Any Other Variable, Instances Must Be Declared Before They Are Used

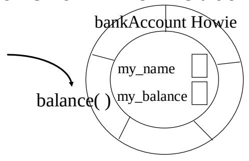
```
bankAccount Howie;
double d = Howie.balance();
```



Interacting With Objects

- Like Any Other Variable, Instances Must Be Declared Before They Are Used

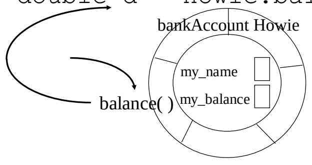
```
bankAccount Howie;
double d = Howie.balance();
```



Interacting With Objects

- Like Any Other Variable, Instances Must Be Declared Before They Are Used

```
bankAccount Howie;
double d = Howie.balance();
```



Time For Our First Demo!

- Banker.cpp

(See Handout For Example 1)

Summarizing Our First Demo!

- `#include "filename.h"` makes the preprocessor acquire definitions for any non-system classes
- Instances Are Declared Like Any Other Variable
- Dialog With Instances By Using Public Interface
- Messages To Instances Use `.` Operator And Work Like Any Other Function Call

Why Classes And Objects?

- Consider Our `bankAccount` Example?
  - What Do We Need To Know To Use It?

Why Classes And Objects?

- Consider Our `bankAccount` Example?
  - What Do We Need To Know To Use It?
- Information Hiding Makes Complex Things Much Simpler
- Different Audiences Need Different Levels of Detail
  - consumers of a class know very little about how it does what it does
  - suppliers of a class are far more in-the-know

Why Classes And Objects?

- Consider My Car:

PROPERTIES

Make: Honda
Model: Prelude

FUNCTIONALITY

play_music
toggle_left_blinker
honk

Why Classes And Objects?

- Consider My Car:

PROPERTIES

Make: Honda
Model: Prelude

FUNCTIONALITY

I don’t know anything about electronics, but I can use all these things → play_music
toggle_left_blinker
honk

Why Classes And Objects?

- Consider My Car:

PROPERTIES

Make: Honda
Model: Prelude

FUNCTIONALITY

My Mechanic can use this & other interfaces when working with my car → play_music
toggle_left_blinker
honk

Why Classes And Objects?

- Consider My Car:

PROPERTIES

Make: Honda
Model: Prelude

FUNCTIONALITY

My Mechanic can use this & other interfaces when working with my car → play_music
toggle_left_blinker
honk

Object-Oriented Programming Offers The Same Benefits!!!

Dot And Scope Resolution Operators

- Dot Operator .
  - Both Visual Studio And Xcode Are Very Sensitive When You Type `object.something`
  - Press TAB To Auto-Complete
- Scope Resolution Operator ::
  - Specifies What Class The Method Definition Comes From

Public And Private Example

-
  ```
  class DayOfYear
  {
  public:
     void input();
     void output();
  private:
     int month;
     int day;
  };
  ```

Public And Private Example

-
  ```
  DayOfYear today;
  today.input( );
  today.output( );
  ```

Public And Private Example

-
  ```
  DayOfYear today;
  today.input( );
  today.output( );
  ```

-
  ```
  cin >> today.month;
  cout << today.day;
  ```

Public And Private Example

-
  ```
  DayOfYear today;
  today.input( );
  today.output( );
  ```
- ~~`cin >> today.month;`~~
  ~~`cout << today.day;`~~

Summary

- Introducing Object-Oriented Programming
- Objects and Classes