CSE 143

Class Constructors

[Chapter 3, pp. 127-131]

4/4/2001 F-1

Initialization: Review

- Variables *must* be initialized before 1st use

```
int sum;
for (int i = 0; i < 10; i++)
    sum = sum + i;                    //whoops!
```

- Simple types can be initialized at declaration

```
int x = 23;
string InstructorName = "I. M. Boring";
```

- Input might do it

```
int num;
cin >> num;
```

4/4/2001 F-2

Initialization: Other Cases

- Parameter: maybe

```
int angle;
modifyTriangle (angle);
  //is this or is it not initializing "angle"?
```

- If a variable is not initialized somehow, it is an error.
  - What kind of error?
- C++ variables are not, not, not initialized automatically!
  - **But some C++ implementations do (trying to be "helpful")**
    - **C++ language != a particular C++ system.**
  - ***Useful advice: Always test your program using different compilers/configurations (release vs debug mode in MSVC, for example)***

4/4/2001 F-3

Initialization of Instances

- When declaring an instance of a class, its data members are all uninitialized
  - no surprise, consistent with C philosophy

```
BankAccount a1;  // what is "name"?  "balance"?
a1.deposit(20.0);
cout << a1.amount(); //what's the result?
```

- Need a way to "construct" and initialize new objects

4/4/2001 F-4

One Solution: Programmer-defined `init` function

```
class BankAccount {
public:
   void init(string name, double initBalance);
  . . .
};

BankAccount anAccount;
anAccount.init("Bob", 200.0);
```

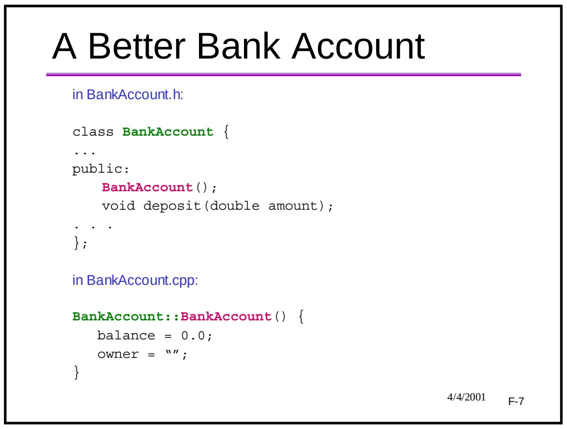
- Drawback: What if the client doesn't call `init`?

4/4/2001 F-5

Constructors

- In C++, a constructor is a special function (method) *automatically* called when a class instance is created (declared)
- Three Weirdnesses:
  - Constructor's name is class name
  - No explicit return type, not even *void*...
  - Invocation is automatic: can't disable; can't call explicitly

4/4/2001 F-6



A Better Bank Account

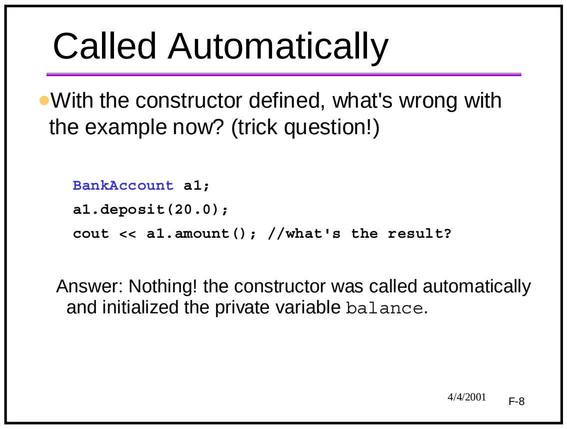
in BankAccount.h:

```
class BankAccount {
...
public:
    BankAccount();
    void deposit(double amount);
. . .
};
```

in BankAccount.cpp:

```
BankAccount::BankAccount() {
   balance = 0.0;
   owner = "";
}
```

4/4/2001 F-7



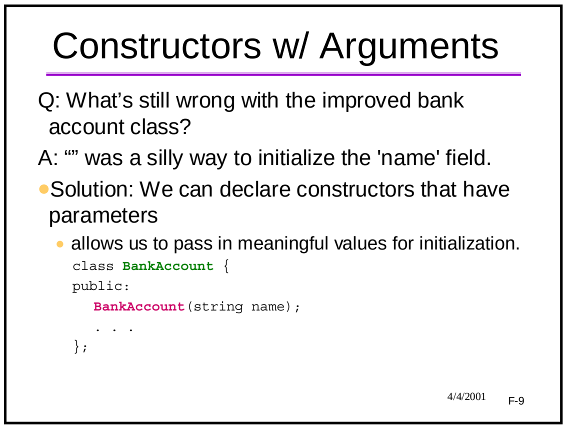
Called Automatically

- With the constructor defined, what's wrong with the example now? (trick question!)

```
BankAccount a1;
a1.deposit(20.0);
cout << a1.amount(); //what's the result?
```

Answer: Nothing! the constructor was called automatically and initialized the private variable `balance`.

4/4/2001 F-8



Constructors w/ Arguments

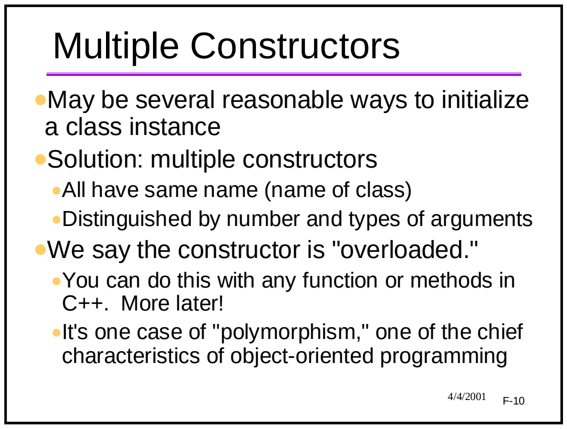
Q: What's still wrong with the improved bank account class?

A: "" was a silly way to initialize the 'name' field.

- Solution: We can declare constructors that have parameters
  - allows us to pass in meaningful values for initialization.

```
class BankAccount {
public:
   BankAccount(string name);
   . . .
};
```

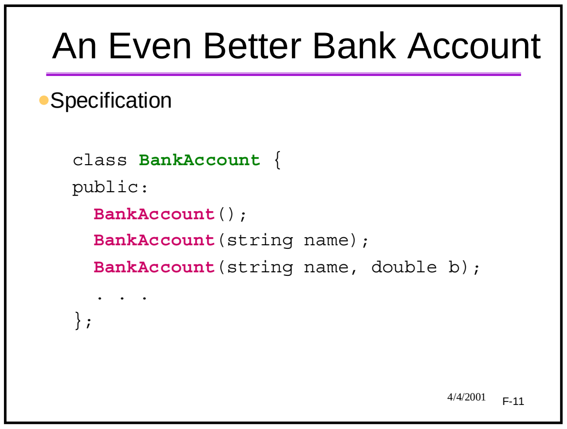
4/4/2001 F-9



Multiple Constructors

- May be several reasonable ways to initialize a class instance
- Solution: multiple constructors
  - All have same name (name of class)
  - Distinguished by number and types of arguments
- We say the constructor is "overloaded."
  - You can do this with any function or methods in C++. More later!
  - It's one case of "polymorphism," one of the chief characteristics of object-oriented programming

4/4/2001 F-10

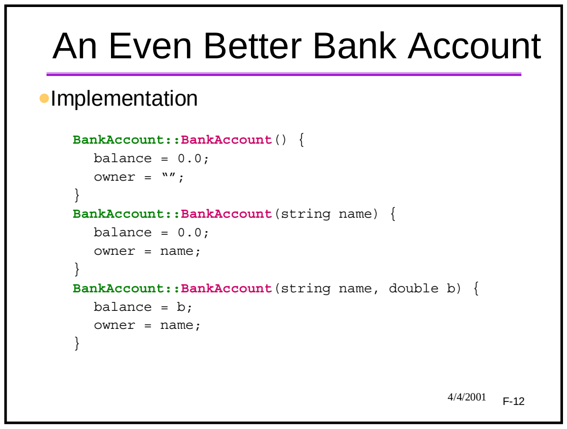


An Even Better Bank Account

- Specification

```
class BankAccount {
public:
   BankAccount();
   BankAccount(string name);
   BankAccount(string name, double b);
   . . .
};
```

4/4/2001 F-11



An Even Better Bank Account

- Implementation

```
BankAccount::BankAccount() {
   balance = 0.0;
   owner = "";
}
BankAccount::BankAccount(string name) {
   balance = 0.0;
   owner = name;
}
BankAccount::BankAccount(string name, double b) {
   balance = b;
   owner = name;
}
```

4/4/2001 F-12

Invoking a Constructor

- A constructor is never invoked using the dot notation
- A constructor is invoked (automatically) whenever a class instance is created:

```
// implicit invocation of BankAccount()
BankAccount a1;

// implicit invocation of BankAccount(string)
BankAccount a2("Bob");

// explicit invocation of BankAccount(string)
BankAccount a3 = BankAccount("Bob");
  //This is NOT an assignment statement!
```

4/4/2001 F-13

"Default" Constructors

- A constructor with 0 arguments is called a *default constructor*.
  - It is invoked in the variable declaration without () -- another weirdness
- If no explicit constructors are given, a default is supplied by compiler
  - Takes no arguments, does nothing
  - Not guaranteed to perform **any** initialization
  - Invisible

4/4/2001 F-14

Default Constructor Pitfall

- If a class has one or more “non-default” constructors:
  - then NO compiler-generator default constructor will be supplied
- Can cause subtle errors
- Wise advice: *always define your own default constructor*

4/4/2001 F-15

Constructors and Arrays

- BankAccount AccountList [10000];
- How many objects are being created?
- Is a constructor called? How many times? Which constructor?
- Answer: in an array of class instances, the default constructor is called for each array element
  - If there is one
- What if you want to invoke one of the other constructors, e.g., **BankAccount**(string name, double b);
  - Answer: Sorry, no way.

4/4/2001 F-16

Puzzler

- *How many times is a constructor called in this code?*

```
BankAccount anAccount (“Dilbert"), anotherAccount;
BankAccount otherAccounts [100];
...
anAccount.GiveAwayMyMoney (otheraccounts, 100);

if (anAccount.IamRicher (anotherAccount)) {
  cout << “Dilbert wins!!" ;
}
```

4/4/2001 F-17

Methods for Puzzler

```
//Takes all the money from my account and gives it to
//the poor
void BankAccount::GiveAwayMyMoney
                    (BankAccount  them [ ], int num);

//return true iff this account has more money than
//second one (the argument)
bool BankAccount::IamRicher (BankAccount  b);
            //A "copy constructor" is involved
            //more about that another day
```

4/4/2001 F-18

Constructors: Review

- A constructor cannot return a value
  - so it must be declared without a return type
- A class may provide multiple constructors
  - Compiler will choose appropriate one, depending on context.
- Syntax for invoking a constructor

```
BankAccount a1;        //NOT BankAccount a1( );
BankAccount a2("Bob", 10.0);
BankAccount a3 = BankAccount("Susan");
```

But not this:

```
BankAccount a3;
a3 = BankAccount("Susan");
```

4/4/2001 F-19

Exercise I

- Design a TranscriptItem class
  - Quarter
  - Course name
  - Grades
  - UW style grades - numerical + letters (I, X, N,…)
- Function overloading - same function may take different types of arguments

```
ti.SetGrade(3.9);
ti.SetGrade('X');
```

4/4/2001 F-20

Transcript Item

```
enum QuarterType{WINTER, SPRING, SUMMER,AUTUMN};
enum GradeType{X_GRADE = 41, I_GRADE, N_GRADE};

class {
public:
 TranscriptItem(int, QuarterType, string, double);
 TranscriptItem(int, QuarterType, string, char);
private:
  int year;
  QuarterType quarter;
  string courseName;
  int grade;     // 0 .. 40 - numerical
                 // 41+ letter grades, -1 invalid
};
```

4/4/2001 F-21

Exercise II

- Design a Transcript class
  - How is the data represented?
  - What are the public methods?
  - Are there any private methods?

4/4/2001 F-22