# CSE 143

## Class Constructors

[Chapter 3, pp. 127-131]



## 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;
char InstructorName[] = "I. M. Boring";
```

- Input might do it

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



## 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++ local variables are not, not, not initialized automatically!
  - **But MSVC does so in "debug" mode (?)**

    **Highlights the difference between the C++ language and a particular C++ system.**
  - ***Useful advice: Always test your program in "release" mode before turning in!***



## 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



## One Solution

- Programmer-defined `init` function

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

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

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



## Better Solution

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

## A Better Bank Account

```
//in BankAccount.h:

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

```
//in BankAccount.cpp:

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



## 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 "balance" variable.



## 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 take arguments
  - allows us to pass in meaningful values for initialization.

```
class BankAccount {
public:
   BankAccount(char name[]);
   . . .
};
```



## 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



## An Even Better Bank Account

- Specification

```
class BankAccount {
public:
  BankAccount();
  BankAccount(char name[]);
  BankAccount(double v, char name[]);
. . .
};
```



## An Even Better Bank Account

- Implementation

```
BankAccount::BankAccount() {
   balance = 0.0;
   strcpy(owner, "");
}
BankAccount::BankAccount(char name[]) {
   balance = 0.0;
   strcpy(owner, name);
}
BankAccount::BankAccount(double v, char name[]) {
   balance = v;
   strcpy(owner, name);
}
```

## 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(char[])
BankAccount a2("Bob");

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



## "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



## Default Constructor Pitfall

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



## 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
- What if you want to invoke one of the other constructors, e.g., **BankAccount**(double v, char name[]);
  - Answer: Sorry, no way.



## Puzzler

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

```
BankAccount myaccount ("Martin"), youraccount;
BankAccount otheraccounts [100];
...
myaccount.GiveAwayMyMoney (otheraccounts, 100);

if (myaccount.IamRicher (youraccount))
  cout << "I win!!" ;
```



## Methods for Puzzler

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

//returns 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
```

## Constructors: Review

- Purpose: provide (automatic) initialization
- A constructor cannot return a value
- A class may provide multiple constructors
  - Compiler will choose appropriate one, depending on arguments.
- Invoking a constructor differs from invoking other methods
  - Happens automatically
  - Syntax a little weird



## Invocation: Review

```
//default constructor: no ( )
BankAccount b1;        //NOT BankAccount b1( );

//constructor with arguments
BankAccount b2(10.0, "Bob");

//initialization by copy
BankAccount b3 = BankAccount("Susan");

//assignment from a temporary constructed object
BankAccount b4; //default applied
b4 = BankAccount("Susan"); //new object assigned to old

//not allowed to use explicit . notation
BankAccount b5;  //default applied
b5.BankAccount ("Susan"); //NOT ALLOWED!!
```



## 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');
```



## Transcript Item

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

class {
public:
 TranscriptItem(int, QuarterType, char[], double);
 TranscriptItem(int, QuarterType, char[], char);
private:
  int year;
  QuarterType quarter;
  char courseName[10];
  int grade;     // 0 .. 40 - numerical
                 // 41+ letter grades, -1 invalid
};
```



## Exercise II

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