11.1 Mutators, accessors, and private helpers

Mutators and accessors

A class' public functions are commonly classified as either mutators or accessors.

- A ***mutator*** function may modify ("mutate") a class' data members.
- An ***accessor*** function accesses data members but does not modify a class' data members.

Commonly, a data member has two associated functions: a mutator for setting the value, and an accessor for getting the value, known as a ***setter*** and ***getter*** function, respectively, and typically with names starting with set or get. Other mutators and accessors may exist that aren't associated with just one data member, such as the Print() function below.

Accessor functions usually are defined as const to make clear that data members won't be changed. The keyword ***const*** after a member function's name and parameters causes a compiler error if the function modifies a data member. If a const member function calls another member function, that function must also be const.

Figure 11.1.1: Mutator and accessor public member functions.

```
#include <iostream>
#include <string>
using namespace std;

class Restaurant {
   public:
      void   SetName(string restaurantName); // Mutator
      void   SetRating(int userRating);      // Mutator
      string GetName() const;                // Accessor
      int    GetRating() const;              // Accessor
      void   Print() const;                  // Accessor

   private:
      string name;
      int rating;
};

void Restaurant::SetName(string restaurantName) {
   name = restaurantName;
}

void Restaurant::SetRating(int userRating) {
   rating = userRating;
}

string Restaurant::GetName() const {
   return name;
}

int Restaurant::GetRating() const {
   return rating;
}

void Restaurant::Print() const {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant myPlace;

   myPlace.SetName("Maria's Diner");
   myPlace.SetRating(5);

   cout << myPlace.GetName() << " is rated ";
   cout << myPlace.GetRating() << endl;

   return 0;
}
```

```
Maria's Diner is rated 5
```


PARTICIPATION ACTIVITY 11.1.1: Mutators and accessors.

1) A mutator should not change a class' private data members.

- ○ True
- ○ False

2) An accessor should not change a class' private data members.

- True
- False

3) A private data member sometimes has a pair of associated set and get functions.

- True
- False

4) Accessor functions are required to be defined as const.

- True
- False

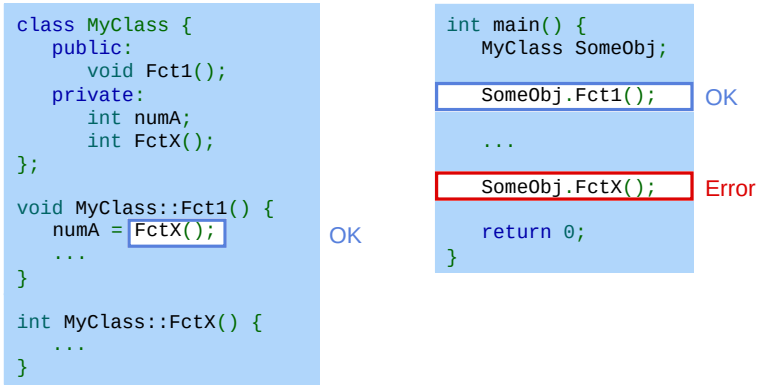
5) A const accessor function may call a non-const member function.

- True
- False

Private helper functions

A programmer commonly creates private functions, known as ***private helper functions***, to help public functions carry out tasks.

PARTICIPATION ACTIVITY 11.1.2: Private helper member functions.



Animation content:

Static figure: A code block is displayed.
Begin C++ code:
class MyClass {
public:
void Fct1();
private:
int numA;

```
    int FctX();
};
void MyClass::Fct1() {
  …
}

int MyClass::FctX() {
  ...
}
```


End C++ code.

Step 1: In addition to public member functions, a class may define private member functions. The lines of code, int FctX();, int MyClass::FctX() {, …, }, are highlighted.

Step 2: Any member function (public or private) may call a private member function. The lines of code, void MyClass::Fct1() {, …, }, are replaced with the code, void MyClass::Fct1() {, numA = FctX();, …, }. The code, FctX();, is highlighted and the text, OK, appears next to the highlighted code.

Step 3: A user of the class can call public member functions, but a user can not call private member functions (which would yield a compiler error). A new code block is displayed.

Begin C++ code:

```
int main() {
  MyClass SomeObj;

  SomeObj.Fct1();

  ...

  SomeObj.FctX();

  return 0;
}
```

End C++ code.

The line of code, SomeObj.Fct1();, is highlighted and the text, OK, appears next to the highlighted code. The line of code, SomeObj.FctX();, is highlighted and the text, Error, appears next to the highlighted code.

Animation captions:

1. In addition to public member functions, a class may define private member functions.
2. Any member function (public or private) may call a private member function.
3. A user of the class can call public member functions, but a user can not call private member functions (which would yield a compiler error).

PARTICIPATION ACTIVITY 11.1.3: Private helper functions.

1) A class' private helper function can be called from main().
   - ○ True
   - ○ False

2) A private helper function typically helps public functions carry out their tasks.

- True
- False

3) A private helper function cannot call another private helper function.

- True
- False

4) A public member function may not call another public member function.

- True
- False

CHALLENGE ACTIVITY 11.1.1: Mutators, accessors, and private helpers.

519134.1787752.qx3zqy7

Start

Type the program's output

```
#include <iostream>
using namespace std;

class Dog {
   public:
      void SetAge(int monthsToSet);
      string GetStage() const;
   private:
      int months;
};

void Dog::SetAge(int monthsToSet) {
   months = monthsToSet;
}

string Dog::GetStage() const {
   string stage;
   if (months < 11) {
      stage = "Puppy";
   }
   else if (months < 13) {
      stage = "Adolescence";
   }
   else if (months < 70) {
      stage = "Adulthood";
   }
   else {
      stage = "Senior";
   }

   return stage;
}

int main() {
   Dog buddy;

   buddy.SetAge(7);

   cout << buddy.GetStage();
   return 0;
}
```

Puppy

1 2 3 4

Check Next

CHALLENGE ACTIVITY 11.1.2: Mutators, accessors, and private helpers.

519134.1787752.qx3zqy7

Start

Define the Voicemail class's SetGreeting() mutator that sets data member greeting to "Please leave a message for ", followed voicemailGreeting, and the SetAreaCode() mutator that sets data member areaCode to voicemailAreaCode.

Ex: If the input is `Mia 466`, then the output is:

```
Voicemail: Please leave a message for Mia
Area Code: 466
```

```
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

class Voicemail {
   public:
      void SetGreeting(string voicemailGreeting);
      void SetAreaCode(int voicemailAreaCode);
      void Print() const;
   private:
      string greeting;
      int areaCode;
};

/* Your code goes here */

void Voicemail::Print() const {
   cout << "Voicemail: " << greeting << endl;
   cout << "Area Code: " << areaCode << endl;
}
```

1 2 3 4

Check Next level

11.2 Initialization and constructors

A good practice is to initialize all variables when declared. This section deals with initializing the data members of a class when a variable of the class type is declared.

Data member initialization (C++11)

Since C++11, a programmer can initialize data members in the class definition. Any variable declared of that class type will initially have those values.

Figure 11.2.1: A class definition with initialized data members.

```
#include <iostream>
#include <string>
using namespace std;

class Restaurant {
   public:
      void SetName(string restaurantName);
      void SetRating(int userRating);
      void Print();

   private:
      string name = "NoName";  // NoName indicates name was not
set
      int rating = -1;         // -1 indicates rating was not
set
};

void Restaurant::SetName(string restaurantName) {
   name = restaurantName;
}

void Restaurant::SetRating(int userRating) {
   rating = userRating;
}

void Restaurant::Print() {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant favLunchPlace;  // Initializes members with
values in class definition

   favLunchPlace.Print();

   favLunchPlace.SetName("Central Deli");
   favLunchPlace.SetRating(4);

   favLunchPlace.Print();

   return 0;
}
```

```
NoName -- -1
Central Deli
-- 4
```


PARTICIPATION ACTIVITY 11.2.1: Initialization.

Consider the example above.

1) When favLunchPlace is initially declared, what is the value of favLunchPlace's rating?

Check Show answer

2) After the call to SetRating(), what is the value of favLunchPlace's rating?

Check Show answer

Constructors

C++ has a special class member function, a ***constructor***, called *automatically* when a variable of that class type is declared, and which can initialize data members. A constructor callable without arguments is a ***default constructor***, like the Restaurant constructor below.

A constructor has the same name as the class. A constructor function has no return type, not even void. Ex: `Restaurant::Restaurant() {...}` defines a constructor for the Restaurant class.

If a class has no programmer-defined constructor, then the compiler *implicitly* defines a default constructor having no statements.

Figure 11.2.2: Adding a constructor member function to the Restaurant class.

```
#include <iostream>
#include <string>
using namespace std;

class Restaurant {
   public:
      Restaurant();
      void SetName(string restaurantName);
      void SetRating(int userRating);
      void Print();
   private:
      string name;
      int rating;
};

Restaurant::Restaurant() {  // Default constructor
   name = "NoName";         // Default name: NoName indicates name was not
set
   rating = -1;             // Default rating: -1 indicates rating was not
set
}

void Restaurant::SetName(string restaurantName) {
   name = restaurantName;
}

void Restaurant::SetRating(int userRating) {
   rating = userRating;
}

// Prints name and rating on one line
void Restaurant::Print() {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant favLunchPlace;  // Automatically calls the default constructor

   favLunchPlace.Print();

   favLunchPlace.SetName("Central Deli");
   favLunchPlace.SetRating(4);
   favLunchPlace.Print();

   return 0;
}
```

```
NoName -- -1
Central Deli -- 4
```


PARTICIPATION ACTIVITY 11.2.2: Default constructors.

Assume a class named Seat.

1) A default constructor declaration in class Seat { ... } is:

```
class Seat {
   ...
   void Seat();
}
```

- ○ True
- ○ False

2) A default constructor definition has this form:

```
Seat::Seat() {
   ...
}
```

- True
- False

3) Not defining any constructor is essentially the same as defining a constructor with no statements.

- True
- False

4) The following calls the default constructor once:

```
Seat mySeat;
```

- True
- False

5) The following calls the default constructor once:

```
Seat seat1;
Seat seat2;
```

- True
- False

6) The following calls the default constructor 5 times:

```
vector<Seat> seats(5);
```

- True
- False

Note: Since C++11, data members can be initialized in the class definition as in `int price = -1;`, which is usually preferred over using a constructor. However, sometimes initializations are more complicated, in which case a constructor is needed.

Exploring further:

- Constructors from msdn.microsoft.com

CHALLENGE ACTIVITY 11.2.1: Enter the output of classes.

519134.1787752.qx3zqy7

Start

Type the program's output

```
#include <iostream>
#include <string>
using namespace std;

class Bicycle {
   public:
      void SetType(string bicycleType);
      void SetYear(int bicycleYear);
      void Print();
   private:
      string type = "NoType";  // NoType indicates brand was not set
      int year = -1;           // -1 indicates year was not set
};

void Bicycle::SetType(string bicycleType) {
   type = bicycleType;
}

void Bicycle::SetYear(int bicycleYear) {
   year = bicycleYear;
}

void Bicycle::Print() {
   cout << type << " " << year << endl;
}

int main() {
   Bicycle commuterBike;

   commuterBike.Print();

   commuterBike.SetType("fitness");
   commuterBike.SetYear(1980);

   commuterBike.Print();

   return 0;
}
```



```
NoType -1
fitness 1980
```

1 2

Check Next

CHALLENGE ACTIVITY 11.2.2: Initialization and constructors.

519134.1787752.qx3zqy7

Start

In the class definition, initialize the data members, integer rating, string name, and integer numEmployees, with the default va -1, "Unknown", and 0, respectively.

Ex: If the input is `5 Gil 14`, then the output is:

```
Rating: -1, Name: Unknown's Bakery, Number of employees: 0
Rating: 5, Name: Gil's Bakery, Number of employees: 14
```

Note: The class's print function is called first after the default constructor, then again after the inputs are passed to the setter


```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Eatery {
6     public:
7        void SetRating(int eateryRating);
8        void SetName(string eateryName);
9        void SetNumEmployees(int eateryNumEmployees);
10       void Print();
11
12    private:
13
14       /* Your code goes here */
15
```

```
};

void Eatery::SetRating(int eateryRating) {
   rating = eateryRating;
}

```

1 2

Check Next level

11.3 Classes and vectors/classes

Vector of objects: A reviews program

Combining classes and vectors is powerful. The program below creates a Review class (reviews might be for a restaurant, movie, etc.), then manages a vector of Review objects.

Figure 11.3.1: Classes and vectors: A reviews program.

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Review {
   public:
      void SetRatingAndComment(int revRating, string
revComment) {
         rating = revRating;
         comment = revComment;
      }
      int GetRating() const { return rating; }
      string GetComment() const { return comment; }

   private:
      int rating = -1;
      string comment = "NoComment";
};

int main() {
   vector<Review> reviewList;
   Review currReview;
   int currRating;
   string currComment;
   unsigned int i;

   cout << "Type rating + comments. To end: -1" <<
endl;
   cin >> currRating;
   while (currRating >= 0) {
      getline(cin, currComment); // Gets rest of line
      currReview.SetRatingAndComment(currRating,
currComment);
      reviewList.push_back(currReview);
      cin >> currRating;
   }

   // Output all comments for given rating
   cout << endl << "Type rating. To end: -1" << endl;
   cin >> currRating;
   while (currRating != -1) {
      for (i = 0; i < reviewList.size(); ++i) {
         currReview = reviewList.at(i);
         if (currRating == currReview.GetRating()) {
            cout << currReview.GetComment() << endl;
         }
      }
      cin >> currRating;
   }

   return 0;
}
```

```
Type rating +
comments. To end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

Type rating. To end:
-1
5
 Great place!
 Loved the food.
1
4
 New owners are nice.
 What a gem.
-1
```


PARTICIPATION ACTIVITY 11.3.1: Reviews program.

Consider the reviews program above.

1) How many member functions does the Review class have?

Check Show answer

2) When currReview is declared, what is the initial rating?

Check Show answer

3) As rating and comment pairs are read from input, what function adds them to vector reviewList? Type the name only, like: append.

Check Show answer

4) How many comments were output for reviews having a rating of 5?

Check Show answer

A class with a vector: The Reviews class

A class' private data often involves vectors. The program below redoes the example above, creating a Reviews class for managing a vector of Review objects.

The Reviews class has functions for reading reviews and printing comments. The resulting main() is clearer than above.

The Reviews class has a "getter" function returning the average rating. The function computes the average rather than reading a private data member. The class user does not need to know how the function is implemented.

Figure 11.3.2: Improved reviews program with a Reviews class.

```
Type ratings +
comments. To end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

Average rating: 3

Type rating. To end:
-1
5
 Great place!
 Loved the food.
1
4
 New owners are nice.
 What a gem.
-1
```

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Review {
   public:
      void SetRatingAndComment(int revRating, string
revComment) {
         rating = revRating;
         comment = revComment;
      }
      int GetRating() const { return rating; }
      string GetComment() const { return comment; }

   private:
      int rating = -1;
      string comment = "NoComment";
};
// END Review class


class Reviews {
   public:
      void InputReviews();
      void PrintCommentsForRating(int currRating) const;
      int GetAverageRating() const;

   private:
      vector<Review> reviewList;
};

// Get rating comment pairs, add each to list. -1 rating
ends.
void Reviews::InputReviews() {
   Review currReview;
   int currRating;
   string currComment;

   cin >> currRating;
   while (currRating >= 0) {
      getline(cin, currComment); // Gets rest of line
      currReview.SetRatingAndComment(currRating,
currComment);
      reviewList.push_back(currReview);
      cin >> currRating;
   }
}

// Print all comments for reviews having the given
rating
void Reviews::PrintCommentsForRating(int currRating)
const {
   Review currReview;
   unsigned int i;

   for (i = 0; i < reviewList.size(); ++i) {
      currReview = reviewList.at(i);
      if (currRating == currReview.GetRating()) {
         cout << currReview.GetComment() << endl;
      }
   }
}

int Reviews::GetAverageRating() const {
   int ratingsSum;
   unsigned int i;

   ratingsSum = 0;
   for (i = 0; i < reviewList.size(); ++i) {
      ratingsSum += reviewList.at(i).GetRating();
   }
   return (ratingsSum / reviewList.size());
```

```
}
// END Reviews class

int main() {
   Reviews allReviews;
   string currName;
   int currRating;

   cout << "Type ratings + comments. To end: -1" <<
endl;
   allReviews.InputReviews();

   cout << endl << "Average rating: ";
   cout << allReviews.GetAverageRating() << endl;

   // Output all comments for given rating
   cout << endl << "Type rating. To end: -1" << endl;
   cin >> currRating;
   while (currRating != -1) {
      allReviews.PrintCommentsForRating(currRating);
      cin >> currRating;
   }

   return 0;
}
```


PARTICIPATION ACTIVITY 11.3.2: Reviews program.

Consider the reviews program above.

1) The first class is named Review. What is the second class named?
   - Reviews
   - reviewList
   - allReviews

2) How many private data members does the Reviews class have?
   - 0
   - 1
   - 2

3) Which function reads all reviews?
   - GetReviews()
   - InputReviews()

4) What does PrintCommentsForRating() do?
   - Prints reviews sorted by rating level.
   - Print all reviews above a rating level.
   - Print all reviews having a particular rating level.

5) Does main() declare a vector?

- Yes
- No

Using Reviews in the Restaurant class

Programmers commonly use classes within classes. The program below improves the Restaurant class by having a Reviews object rather than a single rating.

Figure 11.3.3: Improved reviews program with a Restaurant class.

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// Review and Reviews classes omitted from figure
// ...


class Restaurant {
   public:
      void SetName(string restaurantName) {
         name = restaurantName;
      }
      void ReadAllReviews();
      void PrintCommentsByRating() const;

   private:
      string name;
      Reviews reviews;
};

void Restaurant::ReadAllReviews() {
   cout << "Type ratings + comments. To end: -1"
<< endl;
   reviews.InputReviews();
}

void Restaurant::PrintCommentsByRating() const {
   int i;

   cout << "Comments for each rating level: " <<
endl;
   for (i = 1; i <= 5; ++i) {
     cout << i << ":" << endl;
     reviews.PrintCommentsForRating(i);
   }
}

int main() {
   Restaurant ourPlace;
   string currName;

   cout << "Type restaurant name: " << endl;
   getline(cin, currName);
   ourPlace.SetName(currName);
   cout << endl;

   ourPlace.ReadAllReviews();
   cout << endl;

   ourPlace.PrintCommentsByRating();

   return 0;
}
```

```
Type restaurant name:
Maria's Healthy Food

Type ratings + comments. To
end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

Comments for each rating
level:
1:
2:
 Pretty bad service.
 Yuk!!!
3:
4:
 New owners are nice.
 What a gem.
5:
 Great place!
 Loved the food.
```


PARTICIPATION ACTIVITY 11.3.3: Restaurant program with reviews.

Consider the Restaurant program above.

1) How many private data members does the Restaurant class have?

- 0
- 1
- 2

2) Which Restaurant member function reads all reviews?

- GetReviews()
- InputReviews()
- ReadAllReviews()

3) What does PrintCommentsByRating() do?

- Prints comments sorted by rating level.
- Print all reviews having a particular rating level.

4) Does main() declare a Reviews object?

- Yes
- No

CHALLENGE ACTIVITY 11.3.1: Enter the output of classes and vectors.

519134.1787752.qx3zqy7

Start

Type the program's output

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

class Product {
   public:
      void SetPriceAndName(int productPrice, string productName) {
         price = productPrice;
         name = productName;
      };
      int GetPrice() const { return price; };
      string GetName() const { return name; };
   private:
      int price; // in dollars
      string name;
};

int main() {
   vector<Product> productList;
   Product currProduct;
   int currPrice;
   string currName;
   unsigned int i;
   Product resultProduct;

   cin >> currPrice;
   while (currPrice > 0) {
      cin >> currName;
      currProduct.SetPriceAndName(currPrice, currName);
      productList.push_back(currProduct);
      cin >> currPrice;
   }

   resultProduct = productList.at(0);
   for (i = 0; i < productList.size(); ++i) {
      if (productList.at(i).GetPrice() > resultProduct.GetPrice()) {
         resultProduct = productList.at(i);
      }
   }

   cout << "$" << resultProduct.GetPrice() << " " << resultProduct.GetName() << endl;

   return 0;
}
```


Input

```
5 Berries
8 Foil
13 Shirt
-1
```

Output

```
$13 Shirt
```

1 2 3

Check Next

CHALLENGE ACTIVITY 11.3.2: Writing vectors with classes.

519134.1787752.qx3zqy7

Start

Write code to assign x and y coordinates to currCoord, and store currCoord in criticalPoints.
Input first receives an x value, then a y value. Input example: `12 32 88 2 -1 -1`

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  class Coordinate {
6     public:
7        void SetXAndY(int coordinateX, int coordinateY) {
8           x = coordinateX;
9           y = coordinateY;
10       }
11       void PrintCoordinate() const {
12          cout << x << " - " << y << endl;
13       }
14       int GetX() const { return x; }
15       int GetY() const { return y; }
16
17    private:
```

```
18      int x;
19      int y;
20  };
21
22  int main() {
23     vector<Coordinate> criticalPoints;
24     Coordinate currCoord;
25     int currX;
26     int currY;
27     unsigned int i;
28
29     cin >> currX;
30     cin >> currY;
31     while ((currX >= 0) && (currY >= 0)) {
32
33        /* Your code goes here */
34
35        cin >> currX;
```


1 2 3

Check Next

CHALLENGE ACTIVITY 11.3.3: Classes and vectors/classes.

519134.1787752.qx3zqy7

Start

The program first reads integer lectureCount from input, representing the number of pairs of inputs to be read. Each pair has string and a character, representing the lecture's topic and discount, respectively. One Lecture object is created for each pair added to vector lectureList. If a Lecture object's discount status is equal to 'N', call the Lecture object's Print() function.

Ex: If the input is:

```
4
Hobbies Y Sports N Marketing Y Environment N
```

then the output is:

```
Lecture: Sports, Discount: N
Lecture: Environment, Discount: N
```

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  class Lecture {
6     public:
7        void SetTopicAndDiscount(string newTopic, char newDiscount);
8        char GetDiscount() const;
9        void Print() const;
10    private:
11       string topic;
12       char discount;
13 };
14
15 void Lecture::SetTopicAndDiscount(string newTopic, char newDiscount) {
16    topic = newTopic;
17    discount = newDiscount;
18 }
19
20 char Lecture::GetDiscount() const {
21    return discount;
```


1 2 3 4

Check Next level

11.4 Choosing classes to create

Decomposing into classes

Creating a program may start by a programmer deciding what "things" exist, and what each thing contains and does.

Below, the programmer wants to maintain a soccer team. The programmer realizes the team will have people, so decides to sketch a Person class. Each Person class will have private (shown by "-") data like name and age, and public (shown by "+") functions like get/set name, get/set age, and print. The programmer then sketches a Team class, which uses Person objects.

PARTICIPATION ACTIVITY 11.4.1: Creating a program by first sketching classes.

My program
Will have a soccer team
The team will have a head coach, assistant coach, list of players, name, etc.
Each coach and player will have a name, age, phone, etc.

I need a class for a "person" (coaches, players)

Person
-name : string -age : int
+get/set name +get/set age +print

And for a "team"

Team
-head coach : Person -asst coach : Person
+get/set head coach +get/set asst coach +print

More to come (list of players, name, etc.)

Animation content:

Programmer decides what "things" exist:
My program
Will have a soccer team
The team will have a head coach, assistant coach, list of players, name, etc.
Each coach and player will have a name, age, phone, etc.

Programmer sketches a Person class:
I need a class for a "person" (coaches, players)
Person
-name : string
-age : int
+get/set name
+get/set age
+print

Programmer sketches a Team class:

And for a "team"
Team
-head coach : Person
-asst coach : Person
+get/set head coach
+get/set asst coach
+print
More to come (list of players, name, etc.)

Animation captions:

1. A programmer thinks of what "things" a program may involve. The programmer decides one thing is a Team, and another thing is a Person.
2. The programmer sketches a Person class. Private items (shown by "-") are name and age. Public items (shown by "+") are getters/setters and print.
3. The programmer then sketches a Team class. Private items are head coach and asst coach, both of Person type. Public items are getters/setters and print.

PARTICIPATION ACTIVITY 11.4.2: Decomposing a program into classes.

Consider the example above.

1) Only one way exists to decompose a program into classes.
   - True
   - False

2) The - indicates a class' private item.
   - True
   - False

3) The + indicates additional private items.
   - True
   - False

4) The Team class uses the Person class.
   - True
   - False

5) The Person class uses the Team class.
   - True
   - False

Coding the classes

A programmer can convert the class sketches above into code. The programmer likely would first create and test the Person class, followed by the Team class.

Figure 11.4.1: SoccerTeam and TeamPerson classes.

TeamPerson.h

```
#ifndef TEAMPERSON_H
#define TEAMPERSON_H

#include <string>
using namespace std;

class TeamPerson {
   public:
      void   SetFullName(string
firstAndLastName);
      void   SetAgeYears(int ageInYears);
      string GetFullName() const;
      int    GetAgeYears() const;
      void   Print() const;

   private:
      string fullName;
      int    ageYears;
};

#endif
```

TeamPerson.cpp

```
#include <iostream>
#include <string>
using namespace std;

#include "TeamPerson.h"

void TeamPerson::SetFullName(stri
firstAndLastName) {
   fullName = firstAndLastName;
}

void TeamPerson::SetAgeYears(int
ageInYears) {
   ageYears = ageInYears;
}

string TeamPerson::GetFullName() c
{
   return fullName;
}

int TeamPerson::GetAgeYears() cons
   return ageYears;
}

void TeamPerson::Print() const {
   cout << "Full name: "   << ful
<< endl;
   cout << "Age (years): " << age
<< endl;
}
```

SoccerTeam.h

```
#ifndef SOCCERTEAM_H
#define SOCCERTEAM_H

#include "TeamPerson.h"

class SoccerTeam {
   public:
      void SetHeadCoach(TeamPerson
teamPerson);
      void SetAssistantCoach (TeamPerson
teamPerson);

      TeamPerson GetHeadCoach() const;
      TeamPerson GetAssistantCoach() const;

      void Print() const;

   private:
      TeamPerson headCoach;
      TeamPerson assistantCoach;
      // Players omitted for brevity
};

#endif
```

SoccerTeam.cpp

```
#include <iostream>
using namespace std;

#include "SoccerTeam.h"

void SoccerTeam::SetHeadCoach(Tea
teamPerson) {
   headCoach = teamPerson;
}

void
SoccerTeam::SetAssistantCoach(Tea
teamPerson) {
   assistantCoach = teamPerson;
}

TeamPerson SoccerTeam::GetHeadCoa
const {
   return headCoach;
}

TeamPerson
SoccerTeam::GetAssistantCoach() c
   return assistantCoach;
}

void SoccerTeam::Print() const {
   cout << "HEAD COACH: " << endl
   headCoach.Print();
   cout << endl;

   cout << "ASSISTANT COACH: " <<
   assistantCoach.Print();
   cout << endl;
}
```


main.cpp

```
#include <iostream>
using namespace std;

#include "SoccerTeam.h"
#include "TeamPerson.h"

int main() {
   SoccerTeam teamCalifornia;
   TeamPerson headCoach;
   TeamPerson asstCoach;

   headCoach.SetFullName("Mark Miwerds");
   headCoach.SetAgeYears(42);
   teamCalifornia.SetHeadCoach(headCoach);

   asstCoach.SetFullName("Stanley Lee");
   asstCoach.SetAgeYears(30);

teamCalifornia.SetAssistantCoach(asstCoach);

   teamCalifornia.Print();

   return 0;
}
```

```
HEAD COACH:
Full name: Mark Miwerds
Age (years): 42

ASSISTANT COACH:
Full name: Stanley Lee
Age (years): 30
```


PARTICIPATION ACTIVITY 11.4.3: Coding classes.

Consider the example above.

1) The programmer first sketched the desired classes, before writing the code seen above.
   - True
   - False

2) The programmer wrote one large file containing all the classes.
   - True
   - False

3) Good practice would be to first write the TeamPerson class and then test that class, followed by writing the SoccerTeam class and testing that class.
   - True
   - False

Included files

Above, note that each file only includes needed header files. SoccerTeam.h has a TeamPerson member so includes TeamPerson.h. SoccerTeam.cpp includes SoccerTeam.h. main.cpp declares objects of both types so also includes both .h files. *A common error is to include unnecessary .h files, which misleads the reader.*

Note that only .h files are included, never .cpp files.

PARTICIPATION ACTIVITY 11.4.4: Classes and includes.

Consider the earlier SoccerTeam and TeamPerson classes. Indicate which .h files should be included in each file.

1) TeamPerson.h
   - TeamPerson.h
   - SoccerTeam.h
   - No .h file needed

2) TeamPerson.cpp
   - TeamPerson.h
   - SoccerTeam.h
   - No .h file needed

3) SoccerTeam.h
   - TeamPerson.h
   - SoccerTeam.h
   - No .h file needed

4) SoccerTeam.cpp

- TeamPerson.h
- SoccerTeam.h
- TeamPerson.cpp
- TeamPerson.h and SoccerTeam.h

5) main.cpp

- main.h
- TeamPerson.h
- TeamPerson.h and SoccerTeam.h
- TeamPerson.cpp
- SoccerTeam.cpp

11.5 Constructor overloading

Basics

Programmers often want to provide different initialization values when creating a new object. A class creator can ***overload*** a constructor by defining multiple constructors differing in parameter types. A constructor declaration can have arguments. The constructor with matching parameters will be called.

PARTICIPATION ACTIVITY 11.5.1: Overloaded constructors.

```
class Restaurant {
   public:
      Restaurant();
      Restaurant(string initName, int initRating);

   ...
};

// Default constructor
Restaurant::Restaurant() {
   name = "NoName";
   rating = -1;
}

// Another constructor
Restaurant::Restaurant(string initName, int initRating) {
   name = initName;
   rating = initRating;
}

int main() {
   Restaurant foodPlace;               // Calls default constructor

   Restaurant coffeePlace("Joes", 5); // Calls another constructor

   ...
}
```

foodPlace
Name: NoName
Rating: -1

coffeePlace
Name: Joes
Rating: 5

Animation content:

Shows code having two constructors, one with no parameters, and one with two parameters. Then in main(), one declaration has no arguments, and another has two arguments.

Animation captions:

1. A declaration with no arguments calls the default constructor. In this case, the object gets initialized with NoName and -1.
2. This declaration's string and int arguments match another constructor, which is called instead. The object gets initialized with those argument values.

zyDE 11.5.1: Overloading a constructor.

Load default template... Run

```
#include <iostream>
#include <string>
using namespace std;

class Restaurant {
   public:
      Restaurant();
      Restaurant(string initN
      void Print();

   private:
      string name;
      int rating;
};

// Default constructor
Restaurant::Restaurant() {
   name = "NoName";
   rating = -1;
}
```

PARTICIPATION ACTIVITY 11.5.2: Overloaded constructors.

Given the three constructors below, indicate which will be called for each declaration.

```
class SomeClass {
   SomeClass();                    // A
   SomeClass(string name);         // B
   SomeClass(string name, int num); // C
}
```

1) `SomeClass myObj("Lee");`

- A
- B
- C
- Error

2) `SomeClass myObj();`

- A
- B
- Error

3) `SomeClass myObj;`

- A
- B
- Error

4) `SomeClass myObj("Lee", 5, 0);`

- C
- Error

5) `vector<SomeClass> myVect(5);`

- A
- Error

If any constructor defined, should define default

If a programmer defines any constructor, the compiler does not implicitly define a default constructor, so good practice is for the programmer to also explicitly define a default constructor so that a declaration like `MyClass x;` remains supported.

Figure 11.5.1: Error - The programmer defined a constructor, so the compiler does not automatically define a default constructor.

```
class Restaurant {
   public:
      Restaurant(string initName,
int initRating);

      // No other constructors
      ...
};

int main() {
   Restaurant foodPlace;
   ...
}
```

```
tmp1.cpp:37:15: error: no matching
constructor for initialization of
      'Restaurant'
   Restaurant foodPlace;
```

PARTICIPATION ACTIVITY 11.5.3: Constructor definitions.

Which of the following is OK as the entire set of constructors for class MyClass? Assume a declaration like `MyClass x;` should be supported.

1) `MyClass();`

- OK
- Error

2) `// None`

- OK
- Error

3) `MyClass();`
`MyClass(string name);`

- OK
- Error

4) `MyClass(string name);`

- OK
- Error

Constructors with default parameter values

Like any function, a constructor's parameters may be assigned default values.

If those default values allow the constructor to be called without arguments, then that constructor can serve as the default constructor.

The default values could be in the function definition, but are clearer to class users in the declaration.

Figure 11.5.2: A constructor with default parameter values can serve as the default constructor.

```
#include <iostream>
#include <string>
using namespace std;

class Restaurant {
   public:
      Restaurant(string initName = "NoName", int initRating =
-1);
      void Print();

   private:
      string name;
      int rating;
};

Restaurant::Restaurant(string initName, int initRating) {
   name = initName;
   rating = initRating;
}

// Prints name and rating on one line
void Restaurant::Print() {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant foodPlace;
   Restaurant coffeePlace("Joes", 5);

   foodPlace.Print();
   coffeePlace.Print();

   return 0;
}
```

```
NoName --
-1
Joes -- 5
```

PARTICIPATION ACTIVITY | 11.5.4: Constructor with default parameter values may serve as default constructor.

Which of the following is OK as the entire set of constructors for class YourClass? Assume a declaration like `YourClass obj;` should be supported.

1) `YourClass();`

- OK
- Error

2) `YourClass(string name, int num);`

- OK
- Error

3)
```
YourClass(string name = "", int
num = 0);
```

- OK
- Error

4)
```
YourClass();
YourClass(string name = "", int
num = 0);
```

- OK
- Error

CHALLENGE ACTIVITY | 11.5.1: Enter the output of the constructor overloading.

519134.1787752.qx3zqy7

Start

Type the program's output

```
#include <iostream>
#include <string>
using namespace std;

class Pet {
   public:
      Pet();
      Pet(string petName, int yearsOld);
      void Print();

   private:
      string name;
      int age;
};

Pet::Pet() {
   name = "NoName";
   age = -1;
}

Pet::Pet(string petName, int yearsOld) {
   name = petName;
   age = yearsOld;
}

void Pet::Print() {
   cout << name << ", " << age << endl;
}

int main() {
   Pet dog;
   Pet cat("Cleo", 7);

   cat.Print();
   dog.Print();

   return 0;
}
```



```
Cleo, 7
NoName, -1
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY 11.5.2: Constructor overloading.

519134.1787752.qx3zqy7

Start

The Animal class has a default constructor, a constructor with one parameter, and a constructor with three parameters. Decl the following objects:

- animal1 with no arguments
- animal2 with animalType as an argument
- animal3 with animalType, animalColor, and animalAge as arguments

Ex: If the input is `gecko blue 6`, then the output is:

```
Animal: None, Unspecified, 0
Animal: gecko, Unspecified, 0
Animal: gecko, blue, 6
```



```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Animal {
6     public:
7        Animal();
8        Animal(string animalType);
9        Animal(string animalType, string animalColor, int animalAge);
```

```
10      void Print();
11
12   private:
13      string type;
14      string color;
15      int age;
16 };
17
18 // Default constructor
19 Animal::Animal() {
20    type = "None";
21    color = "Unspecified";
```

1 2 3

Check Next level

11.6 Constructor initializer lists

A ***constructor initializer list*** is an alternative approach for initializing data members in a constructor, coming after a colon and consisting of a comma-separated list of variableName(initValue) items.

Figure 11.6.1: Member initialization: (left) Using statements in the constructor, (right) Using a constructor initializer list.

```
#include <iostream>
using namespace std;

class SampleClass {
   public:
      SampleClass();
      void Print() const;

   private:
      int field1;
      int field2;
};

SampleClass::SampleClass() {
   field1 = 100;
   field2 = 200;
}

void SampleClass::Print() const
{
   cout << "Field1: " << field1
<< endl;
   cout << "Field2: " << field2
<< endl;
}

int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
}
```

```
#include <iostream>
using namespace std;

class SampleClass {
   public:
      SampleClass();
      void Print() const;

   private:
      int field1;
      int field2;
};

SampleClass::SampleClass() :
field1(100), field2(200) {
}



void SampleClass::Print() const {
   cout << "Field1: " << field1 <<
endl;
   cout << "Field2: " << field2 <<
endl;
}

int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
}
```

`Field1: 100` `Field2: 200`	`Field1: 100` `Field2: 200`

PARTICIPATION ACTIVITY 11.6.1: Member initialization.

1) Convert this constructor to use a constructor initializer list.

```
MyClass::MyClass() {
   x = -1;
   y = 0;
}


MyClass::MyClass()
                  {
}
```

Check Show answer

The approach is important when a data member is a class type that must be explicitly constructed. Otherwise, that data member is by default constructed. Ex: If you have studied vectors, consider a data member consisting of a vector of size 2.

Figure 11.6.2: Member initialization in a constructor.

```
#include <iostream>
#include <vector>
using namespace std;

class SampleClass {
   public:
      SampleClass();
      void Print() const;

   private:
      vector<int> itemList;
};

SampleClass::SampleClass() {
   // itemList gets default
constructed, size 0
   itemList.resize(2);
}

void SampleClass::Print() const {
   cout << "Size: " <<
itemList.size() << endl;
   cout << "Item1: " <<
itemList.at(0) << endl;
   cout << "Item2: " <<
itemList.at(1) << endl;
}

int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
}
```

```
Size: 2
Item1: 0
Item2: 0
```

```
#include <iostream>
#include <vector>
using namespace std;

class SampleClass {
   public:
      SampleClass();
      void Print() const;

   private:
      vector<int> itemList;
};

SampleClass::SampleClass() :
itemList(2) {
   // itemList gets constructed
with size 2
}

void SampleClass::Print() const {
   cout << "Size: " <<
itemList.size() << endl;
   cout << "Item1: " <<
itemList.at(0) << endl;
   cout << "Item2: " <<
itemList.at(1) << endl;
}

int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
}
```

```
Size: 2
Item1: 0
Item2: 0
```


On the left, the constructor initially creates a vector of size 0, then resizes to size 2, where each element has the value 0. On the right, itemList(2) is provided in the SampleClass constructor initialization list, causing the vector constructor to be called with size 2 and each vector element to be initialized with the value 0. Using the initialization list avoids the inefficiency of constructing and then modifying an item.

Note: Since C++11, the data member could have been initialized in the class definition: `vector<int> itemList(2);`. However, initialization lists are still useful for other cases.

PARTICIPATION ACTIVITY 11.6.2: Constructor initializer list.

Consider the example above.

1) On the left, itemList is first constructed with size 0, then resized to size 2.
   - ○ True
   - ○ False

2) On the right, itemList is first constructed with size 0, then resized to size 2.

- True
- False

CHALLENGE ACTIVITY 11.6.1: Enter the output of constructor initializer lists.

519134.1787752.qx3zqy7

Start

Type the program's output

```
#include <iostream>
#include <string>
using namespace std;

class Tutor {
   public:
      Tutor();
      void Print() const;

   private:
      string name;
      string topic;
};

Tutor::Tutor() : name("NeedsName"), topic("NeedsTopic") {}

void Tutor::Print() const {
   cout << name << " tutors " << topic << endl;
}

int main() {
   Tutor myTutor;

   myTutor.Print();

   return 0;
}
```

```
NeedsName tutors NeedsTopic
```

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 11.6.2: Constructor initializer lists.

519134.1787752.qx3zqy7

Start

Add a constructor initializer list to the default Appointment constructor to initialize month with "Empty", date with -99, and weekday with '-'.

Ex: If the input is `Jul 14 M`, then the output is:

```
Appointment: Empty, Date: -99, Weekday: -
Appointment: Jul, Date: 14, Weekday: M
```

```
1 #include <iostream>
2 using namespace std;
3
4 class Appointment {
```

```
   public:
      Appointment();
      void SetData(string newMonth, int newDate, char newWeekday);
      void Print() const;
   private:
      string month;
      int date;
      char weekday;
};

Appointment::Appointment() /* Your code goes here */ {
}

void Appointment::SetData(string newMonth, int newDate, char newWeekday) {
   month = newMonth;
   date = newDate;
   weekday = newWeekday;
```


1 2 3

Check Next level

Exploring further:

- Classes from cplusplus.com, see "Member initialization in constructors" section.
- Constructors from msdn.microsoft.com, see "Member lists".

11.7 The 'this' implicit parameter

Implicit parameter

An object's member function is called using the syntax `object.Function()`. The object variable before the function name is known as an ***implicit parameter*** of the member function because the compiler converts the call syntax `object.Function(...)` into a function call with a pointer to the object implicitly passed as a parameter. Ex: `Function(object, ...)`.

Within a member function, the implicitly-passed object pointer is accessible via the name ***this***. In particular, a member can be accessed as `this->member`. The -> is the member access operator for a pointer, similar to the "." operator for non-pointers.

Using `this->` makes clear that a class member is being accessed and is essential if a data member and parameter have the same identifier. In the example below, `this->` is necessary to differentiate between the data member sideLength and the parameter sideLength.

Figure 11.7.1: Using 'this' to refer to an object's member.

```
#include <iostream>
using namespace std;

class ShapeSquare {
   public:
      void   SetSideLength(double sideLength);
      double GetArea() const;
   private:
      double sideLength;
};

void ShapeSquare::SetSideLength(double sideLength) {
   this->sideLength = sideLength;
   // Data member      Parameter
}

double ShapeSquare::GetArea() const{
   return sideLength * sideLength; // Both refer to data
member
}

int main() {
   ShapeSquare square1;

   square1.SetSideLength(1.2);
   cout << "Square's area: " << square1.GetArea() << endl;

   return 0;
}
```

```
Square's area:
1.44
```


PARTICIPATION ACTIVITY 11.7.1: The 'this' implicit parameter.

Given a class Spaceship with private data member numYears and public member function:
`void Spaceship::AddNumYears(int numYears)`

1) In AddNumYears(), which line assigns the data member numYears with 0?

   - `numYears = 0;`
   - `this.numYears = 0;`
   - `this->numYears = 0;`

2) In AddNumYears(), which line assigns the data member numYears with the parameter numYears?

   - `numYears = this->numYears;`
   - `this->numYears = numYears;`

3) In AddNumYears(), which line adds the parameter numYears to the existing value of data member numYears?

- `this->numYears = this->numYears + numYears;`
- `this->numYears = numYears + numYears;`
- `numYears = this->numYears + numYears;`

4) Given variable `Spaceship ss1` is declared in main(), which line assigns ss1's numYears with 5?

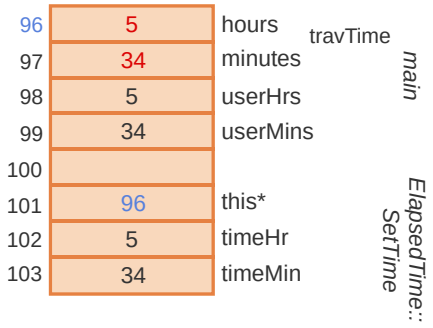
- `ss1.numYears = 5;`
- `ss1->numYears = 5;`
- `this->numYears = 5;`
- None of the above.

Using 'this' in class member functions and constructors

The animation below illustrates how member functions work. When an object's member function is called, the object's memory address is passed to the function via the implicit "this" parameter. An access in SetTime() to `this->hours` first goes to the object's address, then to the hours data member. If SetTime() instead had the assignment hours = timeHr, the compiler would use `this->hours` for hours because no other variable in SetTime() is named hours.

PARTICIPATION ACTIVITY 11.7.2: How a member function works.

```
#include <iostream>
using namespace std;

class ElapsedTime {
   private:
      int hours;
      int minutes;

   public:
      void SetTime(int timeHr, int timeMin);
};

void ElapsedTime::SetTime(int timeHr, int timeMin) {
   this->hours = timeHr;
   this->minutes = timeMin;
}

int main() {
   ElapsedTime travTime;
   int userHrs;
   int userMins;

   userHrs = 5;
   userMins = 34;

   travTime.SetTime(userHrs, userMins);

   return 0;
}
```



Animation content:

Static Figure:
Begin C++ code:

```
#include <iostream>
using namespace std;

class ElapsedTime {
  private:
    int hours;
    int minutes;

  public:
    void SetTime(int timeHr, int timeMin);
};

void ElapsedTime::SetTime(int timeHr, int timeMin) {
  this->hours = timeHr;
  this->minutes = timeMin;
}

int main() {
  ElapsedTime travTime;
  int userHrs;
  int userMins;

  userHrs = 5;
  userMins = 34;

  travTime.SetTime(userHrs, userMins);

  return 0;
}
```

End C++ code.
A box containing 8 memory addresses from 96 to 103, consecutively. The memory address 96 is labeled hours and contains “5”. The memory address 97 is labeled minutes and contains “35”. The memory address 98 is labeled userHrs and contains “5”. The memory address 99 is labeled userMins and contains “35”. Memory addresses 96 to 99 are labeled main and represents the object travTime.

The memory address 100 is not labeled and is empty. The memory address 101 is labeled this* and contains “96”. The memory address 101 is labeled this* and contains “96”. The memory address 102 is labeled timeHr and contains “5”. The memory address 103 is labeled timeMin and contains “34”. Memory addresses 96 to 99 are labeled ElapsedTime:: SetTime.

©zyBooks 04/10/24 16:32 893876
Anthony Hamlin
DMACCCIS161Spring2024

Step 1: travTime is an object of class type ElapsedTime.

A highlight box highlights the C++ code lines, ElapsedTime travTime;, int userHrs;, and int userMins;. The traveTime object is created, allocating four memory addresses 96 to 103 for variables hours, minutes, userHrs, and userMins.
The highlight box highlights C++ code lines userHrs = 5; and userMins = 34;. The variable userHrs is initialized with the value 5 in memory address 98. The variable userMins is initialized to 34 in memory address 99.

Step 2: When travTime's SetTime() member function is called, travTime's memory address is passed to the function via the implicit "this" parameter.

The highlight box highlights the C++ line travTime.SetTime(userHrs, userMins);. The SetTime function is called and another highlight box highlights the C++ code line void ElapsedTime::SetTime(int timeHr, int timeMin) {. 3 memory addresses 101 to 103 are allocated for the SetTime function variables. The first memory address 101 contains 96 which points to the beginning of the tracTime memory address. The memory address 102 contains the passed userHrs value, 5. The memory address 103 contains the passed userMins value, 34.
Step 3: The implicitly-passed object pointer is accessible within the member function via the name "this". Ex: this->hours first goes to travTime's address, then to the hours data member.

The highlight box highlights the C++ code lines this->hours = timeHr; and this->minutes = timeMin;. A copy of the value 5 from memory address 102 is added the memory address 96. A copy of the value 34 from memory address 103 is added the memory address 97.

Animation captions:

1. travTime is an object of class type ElapsedTime.
2. When travTime's SetTime() member function is called, travTime's memory address is passed to the function via the implicit "this" parameter.
3. The implicitly-passed object pointer is accessible within the member function via the name "this". Ex: this->hours first goes to travTime's address, then to the hours data member.

PARTICIPATION ACTIVITY 11.7.3: Using the 'this' pointer in member functions and constructors.

1) Complete the code to assign the value of mins to data member minutes using `this->` notation.

```
void
ElapsedTime::SetMinutes(int
mins) {
    [          ] =
mins;
}
```

Check Show answer

2) Complete the code to assign the value of parameter hours to data member hours using `this->` notation.

```
void ElapsedTime::SetHours(int
hours) {
    [          ];
}
```

Check Show answer

Exploring further:

- The 'this' pointer from msdn.microsoft.com.

CHALLENGE ACTIVITY 11.7.1: Enter the output of the function.

519134.1787752.qx3zqy7

Start

Type the program's output

```
#include <iostream>
using namespace std;

class Airplane {
   public:
      Airplane();
      void Print() const;
      void SetSpeed(int speed);
   private:
      int speed;
};

Airplane::Airplane() {
   speed = 0;
}

void Airplane::SetSpeed(int speed) {
   this->speed = speed;
}

void Airplane::Print() const {
   cout << speed << " MPH" << endl;
}

int main() {
   Airplane boeing747;

   boeing747.SetSpeed(850);
   boeing747.Print();

   return 0;
}
```

```
850 MPH
```

1 2 3

Check Next

CHALLENGE ACTIVITY 11.7.2: The this implicit parameter.

Define the missing member function. Use "this" to distinguish the local member from the parameter name.

Learn how our autograder works

519134.1787752.qx3zqy7

```
1 #include <iostream>
2 using namespace std;
3
4 class CablePlan{
5    public:
6       void SetNumDays(int numDays);
```

```
   void   SetNumDays(int numDays);
   int    GetNumDays() const;
private:
   int    numDays;
};

// FIXME: Define SetNumDays() member function, using "this" implicit parameter.
void CablePlan::SetNumDays(int numDays) {

   /* Your solution goes here  */

}

int CablePlan::GetNumDays() const {
   return numDays;
}
```


Run

View your last submission

11.8 Static data members and functions

Static data members

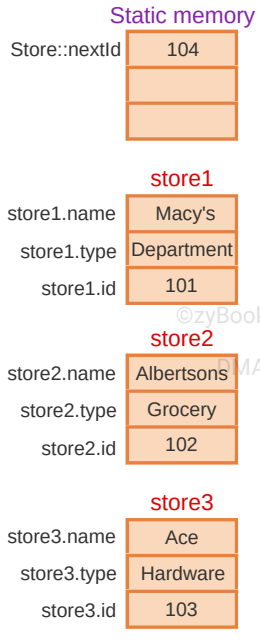
The keyword ***static*** indicates a variable is allocated in memory only once during a program's execution. Static variables are allocated memory once and reside in the program's static memory region for the entire program. Thus, a static variable retains a value throughout the program.

In a class, a ***static data member*** is a data member of the class instead of a data member of each class object. Thus, static data members are independent of any class object, and can be accessed without creating a class object.

A static data member is declared inside the class definition, but must also be defined outside the class declaration. Within a class function, a static data member can be accessed just by variable name. A public static data member can be accessed outside the class using the scope resolution operator: ClassName::variableName.

PARTICIPATION ACTIVITY 11.8.1: Static data member used to create object ID numbers.

```
class Store {
   public:
      Store(string storeName, string storeType);
      int getId();
      static int nextId;   // Declare static member variable
   private:
      string name = "None";
      string type = "None";
      int id = 0;
};

Store::Store(string storeName, string storeType) {
   name = storeName;
   type = storeType;
   id = nextId;   // Assign object id with nextId

   ++nextId;      // Increment nextId for next object to be created
}
...
int Store::nextId = 101; // Define and initialize static data member

int main() {

   Store store1("Macy's", "Department");
   Store store2("Albertsons", "Grocery");
   Store store3("Ace", "Hardware");
```



```
   cout << "Store 1's ID: "<< store1.getId() << endl;
   cout << "Store 2's ID: "<< store2.getId() << endl;
   cout << "Store 3's ID: "<< store3.getId() << endl;
   cout << "Next ID: " << Store::nextId << endl;

   return 0;
}
```

Console

```
Store 1's ID: 101
Store 2's ID: 102
Store 3's ID: 103
Next ID: 104
```

Animation content:

Static Figure:
Begin C++ code:

```
class Store {
  public:
    Store(string storeName, string storeType);
    int getId();
    static int nextId;   // Declare static member variable
  private:
    string name = "None";
    string type = "None";
    int id = 0;
};

Store::Store(string storeName, string storeType) {
  name = storeName;
  type = storeType;
  id = nextId;   // Assign object id with nextId

  ++nextId;     // Increment nextId for next object to be created
}
...
int Store::nextId = 101; // Define and initialize static data member

int main() {

  Store store1("Macy's", "Department");
  Store store2("Albertsons", "Grocery");
  Store store3("Ace", "Hardware");

  cout << "Store 1's ID: " << store1.getId() << endl;
  cout << "Store 2's ID: " << store2.getId() << endl;
  cout << "Store 3's ID: " << store3.getId() << endl;
  cout << "Next ID: " << Store::nextId << endl;

  return 0;
}
```

End C++ code.

The program in the static figure is executed and the steps and substeps of the program are described in the following three tables and five steps. The program uses four different regions of memory: Static memory, store1, store2, and store3. All ten memory locations, as well as the output console, are initially empty.

Step 1: The Store class' static data member nextId is declared in the Store class declaration.

Code executed	Memory label	Value stored	Region of memory
static int nextId;	Store::nextId		Static memory

Step 2: Store::nextId must be defined and initialized outside the class declaration. Only one instance of that variable will exist in memory.

Code executed	Memory label	Value stored	Region of memory
int Store::nextId = 101; // Define and initialize static data member	Store::nextId	101	Static memory

Step 3: When a Store object is created, memory is allocated for the object's name, type, and id data members, but not the static member nextId.

Code executed	Memory label	Value stored	Region of memory
Store store1("Macy's", "Department");			store1

Step 4: The constructor assigns an object's id with nextId, and then increments nextId. Each time an object is created, nextId is incremented, and each object has a unique id.

Code executed	Memory label	Value stored	Region of memory
name = storeName;	store1.name	Macy's	store1
type = storeType;	store1.type	Department	store1
id = nextId;	store1.id	101	store1
++nextId;	Store::nextId	102	Static memory
Store store2("Albertsons", "Grocery");	Three memory locations are allocated: store2.name, store2.type, and, store2.id		store2
name = storeName;	store2.name	Albertsons	store2
type = storeType;	store2.type	Grocery	store2
id = nextId;	store2.id	102	store2
++nextId;	Store::nextId	103	Static memory
Store store3("Ace", "Hardware");	Three memory locations are allocated: store3.name, store3.type, and, store3.id		store3

name = storeName;	store3.name	Ace	store3
type = storeType;	store3.type	Hardware	store3
id = nextId;	store3.id	103	store3
++nextId;	Store::nextId	104	Static memory

Step 5: Any class member function can access or mutate a static data member. nextId can also be accessed outside the class using the scope resolution operator (::).

The following code is executed:
cout << "Store 1's ID: " << store1.getId() << endl;
cout << "Store 2's ID: " << store2.getId() << endl;
cout << "Store 3's ID: " << store3.getId() << endl;
cout << "Next ID: " << Store::nextId << endl;

The output console now contains four lines of output:
Store 1's ID: 101
Store 2's ID: 102
Store 3's ID: 103
Next ID: 104

Animation captions:

1. The Store class' static data member nextId is declared in the Store class declaration.
2. Store::nextId must be defined and initialized outside the class declaration. Only one instance of that variable will exist in memory.
3. When a Store object is created, memory is allocated for the object's name, type, and id data members, but not the static member nextId.
4. The constructor assigns an object's id with nextId, and then increments nextId. Each time an object is created, nextId is incremented, and each object has a unique id.
5. Any class member function can access or mutate a static data member. nextId can also be accessed outside the class using the scope resolution operator (::).

PARTICIPATION ACTIVITY 11.8.2: Static data members.

1) Each constructed class object creates a new instance of a static data member.
   - ○ True
   - ○ False

2) All static data members can be accessed anywhere in a program.
   - ○ True
   - ○ False

3) Outside of the class where declared, private static data members can be accessed using dot notation.

- True
- False

Static member functions

A ***static member function*** is a class function that is independent of class objects. Static member functions are typically used to access and mutate private static data members from outside the class. Since static methods are independent of class objects, the `this` parameter is not passed to a static member function. So, a static member function can only access a class' static data members.

Figure 11.8.1: Static member function used to access a private static data member.

```
#include <iostream>
#include <string>
using namespace std;

class Store {
public:
   Store(string storeName, string storeType);
   int getId();
   static int getNextId();

private:
   string name = "None";
   string type = "None";
   int id = 0;
   static int nextId;   // Declare static member variable
};

Store::Store(string storeName, string storeType) {
   name = storeName;
   type = storeType;
   id = nextId;       // Assign object id with nextId

   ++nextId;         // Increment nextId for next object to be created
}

int Store::getId() {
   return id;
}

int Store::getNextId() {
   return nextId;
}

int Store::nextId = 101;   // Define and initialize static data member

int main() {
   Store store1("Macy's", "Department");
   Store store2("Albertsons", "Grocery");
   Store store3("Ace", "Hardware");

   cout << "Store 1's ID: "<< store1.getId() << endl;
   cout << "Store 2's ID: "<< store2.getId() << endl;
   cout << "Store 3's ID: "<< store3.getId() << endl;
   cout << "Next ID: " << Store::getNextId() << endl;

   return 0;
}
```

```
Store 1's ID: 101
Store 2's ID: 102
Store 3's ID: 103
Next ID: 104
```


PARTICIPATION ACTIVITY 11.8.3: Static member functions.

1) A static member function is needed to access or mutate a _____ static data member from outside of the class.

   - ○ public
   - ○ private

2) The `this` parameter can be used in a static member function to access an object's non-static data members.

- True
- False

CHALLENGE ACTIVITY 11.8.1: Enter the output with static members.

519134.1787752.qx3zqy7

Start

Type the program's output

```
#include <iostream>
#include <string>
using namespace std;

class FoodType {
   public:
      FoodType(string foodType);
      static int nextId;
      void Print();
   private:
      string type = "None";
      int id = 0;
};

FoodType::FoodType(string foodType) {
   type = foodType;
   id = nextId;

   nextId += 4;
}

void FoodType::Print() {
   cout << id << ": " << type << endl;
}

int FoodType::nextId = 60;

int main() {
   FoodType order1("Soup");
   FoodType order2("Cake");
   FoodType order3("Crab");

   order3.Print();

   return 0;
}
```

```
68: Crab
```

1 2

Check Next

11.9 C++ example: Salary calculation with classes

zyDE 11.9.1: Calculate salary: Using classes.

The program below uses a class, TaxTableTools, which has a tax table built in. The main function prompts for a salary, then uses a TaxTableTools function to get the tax rate. The program then calculates the tax to pay and displays the results to the user. Run the program with annual salaries of 10000, 50000, 50001, 100001 and -1 (to end the program) and note the output tax rate and tax to pay.

1. Modify the TaxTableTools class to use a setter function that accepts a new salary and tax rate table.
2. Modify the program to call the new function, and run the program again, noting the same output.

The program's two classes are in separate tabs at the top.

Note: The calculation is inaccurate to how taxes are formally assessed and is a simplification for educational purposes only.

Current file: IncomeTaxMain.cpp Load default template...

```
#include <iostream>
#include <limits>
#include <vector>
#include <string>
#include "TaxTableTools.h"

int GetInteger(const string userPrompt) {
   int inputValue;

   cout << userPrompt << ": " << endl;
   cin >> inputValue;

   return inputValue;
}

// ************************************************************

int main() {
   const string PROMPT_SALARY = "\nEnter annual salary (-1 to ex
   int annualSalary;
   double taxRate;
```

10000 50000 50001 100001 -1

Run

zyDE 11.9.2: Calculate salary: Using classes (solution).

A solution to the above problem follows.

Note that the program's two classes are in separate tabs at the top.

Current file: IncomeTaxMain.cpp Load default template...

```
#include <iostream>
#include <limits>
#include <vector>
#include <string>
#include "TaxTableTools.h"

int GetInteger(const string userPrompt) {
   int inputValue;

   cout << userPrompt << ": " << endl;
   cin >> inputValue;

   return inputValue;
}

// ************************************************************

int main() {
   const string PROMPT_SALARY = "\nEnter annual salary (-1 to ex
   int annualSalary;
   double taxRate;
```


10000 50000 50001 100001 -1

Run

zyDE 11.9.3: Salary calculation: Overloading a constructor.

The program below calculates a tax rate and tax to pay given an annual salary. The program uses a class, TaxTableTools, which has the tax table built in. Run the program with annual salaries of 10000, 50000, 50001, 100001 and -1 (to end the program) and note the output tax rate and tax to pay.

1. Overload the constructor.
   a. Add to the TaxTableTools class an overloaded constructor that accepts the base salary table and corresponding tax rate table as parameters.
   b. Modify the main function to call the overloaded constructor with the two tables (vectors) provided in the main function. Be sure to set the nEntries value, too.
   c. Note that the tables in the main function are the same as the tables in the TaxTableTools class. This sameness facilitates testing the program with the same annual salary values listed above.
   d. Test the program with the annual salary values listed above.
2. Modify the salary and tax tables
   a. Modify the salary and tax tables in the main function to use different salary ranges and tax rates.
   b. Use the just-created overloaded constructor to initialize the salary and tax tables.
   c. Test the program with the annual salary values listed above.

Note: The calculation is inaccurate to how taxes are formally assessed and is a simplification for educational purposes only.

Current file: IncomeTaxMain.cpp Load default template...

```
#include <iostream>
#include <limits>
#include <vector>
#include <string>
#include "TaxTableTools.h"
using namespace std;

int main() {
   const string PROMPT_SALARY = "\nEnter annual salary (-1 to ex
   int annualSalary;
   double taxRate;
   int taxToPay;
   vector<int> salaryBase(5);
   vector<double> taxBase(5);

   salaryBase.at(0) = 0;
   salaryBase.at(1) = 20000;
   salaryBase.at(2) = 50000;
   salaryBase.at(3) = 100000;
   salaryBase.at(4) = numeric_limits<int>::max();
```



```
10000
50000
50001
```

Run

zyDE 11.9.4: Salary calculation: Overloading a constructor (solution).

A solution to the above problem follows.

Note that the program's two classes are in separate tabs at the top.

Current file: IncomeTaxMain.cpp Load default template...

```
#include <iostream>
#include <limits>
#include <vector>
#include <string>
#include "TaxTableTools.h"
using namespace std;

int main() {
   const string PROMPT_SALARY = "\nEnter annual salary (-1 to ex
   int annualSalary;
   double taxRate;
   int taxToPay;
   vector<int> salaryBase(5);
   vector<double> taxBase(5);

   salaryBase.at(0) = 0;
   salaryBase.at(1) = 30000;
   salaryBase.at(2) = 60000;
   salaryBase.at(3) = 90000;
   salaryBase.at(4) = numeric_limits<int>::max();
```



```
10000
50000
50001
```

Run

11.10 C++ example: Domain name availability with classes

zyDE 11.10.1: Domain name availability: Using classes.

The program below uses a class, DomainAvailabilityTools, which includes a table of registered domain names. The main function prompts for domain names until the user presses Enter at the prompt. The domain name is checked against a list of the registered domains in the DomainAvailabilityTools class. If the domain name is not available, the program displays similar domain names.

1. Run the program and observe the output for the given input.
2. Modify the DomainAvailabilityClass's function named GetSimilarDomainNames so that some unavailable domain names do not get a list of similar domain names. Run the program again and observe that unavailable domain names with TLDs of .org or .biz do not have similar names.

Current file: DomainAvailabilityMain.cpp Load default template...

```
#include <iostream>
#include <string>
#include <cctype>
#include "DomainAvailabilityTools.h"
using namespace std;

// ************************************************************

// prompts user string. Returns string.
string GetString(string prompt) {
   string userInput;

   cout << prompt << endl;
   cin >> userInput;

   return userInput;
}
// ************************************************************

int main() {
   const string PROMPT_DOMAIN_NAME =
```



```
programming.com
apple.com
oracle.com
```

Run

zyDE 11.10.2: Domain validation: Using classes (solution).

A solution to the above problem follows.

Current file: DomainAvailabilityMain.cpp Load default template...

```
#include <iostream>
#include <string>
#include <cctype>
#include "DomainAvailabilityTools.h"
using namespace std;

// ***************************************************************

// Prompts user for input string and returns the string
string GetString(string prompt) {
   string userInput;

   cout << prompt << endl;
   cin >> userInput;

   return userInput;
}
// ***************************************************************

int main() {
   const string PROMPT_DOMAIN_NAME =
```



```
programming.com
apple.com
oracle.com
```

Run