11.1 Grouping data: struct

Sometimes two data items are really aspects of the same data. For example, time might be recorded in hours and minutes, as in 4 hours and 23 minutes. Or a point on a plot might be recorded as x = 5, y = 7. Storing such data in separate variables, such as runTimeHours and runTimeMinutes, is not as clear as grouping that data into a single variable, like runTime, which might have subitems runTime.hourValue and runTime.minuteValue.

PARTICIPATION ACTIVITY 11.1.1: Naturally grouped data.

1) Select the pair forming part of a person's height (in U.S. units)
   - Feet and inches
   - Inches and salary
   - Pounds and ounces

2) Select the group of items indicating the change provided to a person who pays for a meal.
   - Ounce, gill, pint, quart, and gallon
   - Mile, furlong, yard, feet, and inches
   - Dollars, quarters, dimes, nickels, and pennies

The ***struct*** construct defines a new type, which can be used to declare a variable with subitems. The following animation illustrates.

PARTICIPATION ACTIVITY 11.1.2: A struct enables creating a variable with data members.


```
struct TimeHrMin {
   int hourValue;
   int minuteValue;
};

...

   TimeHrMin runTime1;
   TimeHrMin runTime2;
```

96	5	hourValue	runTime1
97	46	minuteValue	
98	?	hourValue	runTime2
99	?	minuteValue	
100	5	hourValue	runTime3
101	?	minuteValue	

```
    TimeHrMin runTime3;

    runTime1.hourValue = 5;
    runTime1.minuteValue = 46;
    runTime3.hourValue = runTime1.hourValue;
```

102

Animation content:

Code snippet is as follows:

```
struct TimeHrMin {
  int hourValue;
  int minuteValue;
};

...

  TimeHrMin runTime1;
  TimeHrMin runTime2;
  TimeHrMin runTime3;

  runTime1.hourValue = 5;
  runTime1.minuteValue = 46;
  runTime3.hourValue = runTime1.hourValue;
```

Final memory contents is as follows:
96 (runTime1's hourValue): 5
97 (runTime1's hourValue): 46
98 (runTime2's hourValue): ?
99 (runTime2's hourValue): ?
100 (runTime3's hourValue): 5
101 (runTime3's hourValue): ?
102: empty

Animation captions:

1. The struct construct just declares new type; no memory is allocated.
2. Variable definitions allocate memory for each object's member.
3. Accesses refer to an object member's memory location.

The programmer uses a struct to define and use a new type as follows.

Construct 11.1.1: Defining and using a new struct type.

```
struct StructTypeName
{
   type item1;
   type item2;
   ...
   type itemN;
};

...
StructTypeName myVar;

myVar.item1 = ...
```


Each `type` may be any type like int or char. Each struct subitem is called a ***data member***. For a declared variable, each struct data member can be accessed using ".", known as a ***member access*** operator, sometimes called ***dot notation***.

Assigning a variable of a struct type to another such variable automatically assigns each corresponding data member, as shown below.

PARTICIPATION ACTIVITY 11.1.3: Assigning a struct type.

```
struct TimeHrMin {
   int hourValue;
   int minuteValue;
};

...

   TimeHrMin runTime1;
   TimeHrMin runTime2;
   TimeHrMin runTime3;

   runTime1.hourValue = 5;
   runTime1.minuteValue = 46;
   runTime2 = runTime1;
```

96	5	hourValue	runTime1
97	46	minuteValue	
98	5	hourValue	runTime2
99	46	minuteValue	
100	?	hourValue	runTime3
101	?	minuteValue	
102			

Animation content:

Code snippet is as follows:
struct TimeHrMin {
  int hourValue;
  int minuteValue;
};

...

```
TimeHrMin runTime1;
TimeHrMin runTime2;
TimeHrMin runTime3;

runTime1.hourValue = 5;
runTime1.minuteValue = 46;
runTime2 = runTime1;
```


Final memory contents is as follows:
96 (runTime1's hourValue): 5
97 (runTime1's hourValue): 46
98 (runTime2's hourValue): ?
99 (runTime2's hourValue): ?
100 (runTime3's hourValue): 5
101 (runTime3's hourValue): ?
102: empty

Animation captions:

1. Assigning a variable of a struct type to another such variable automatically assigns each corresponding data member.

PARTICIPATION ACTIVITY 11.1.4: The struct construct.

1) A struct definition for CartesianPoint has subitems int x and int y. How many int locations in memory does the struct definition allocate?

Check Show answer

2) If struct definition CartesianPoint has subitems int x and int y, how many total int locations in memory are allocated for these variable declarations?

```
int myNum;
CartesianPoint myPoint1;
CartesianPoint myPoint2;
```

Check Show answer

3) Given time1 is of type TimeHrMin defined earlier. What is the value of variable min after the following statements?

```
time1.hourValue   = 5;
time1.minuteValue = 4;
min = (60 * time1.hourValue)
+ time1.minuteValue;
```

Check Show answer

4) Write a statement to assign 12 to the hourValue data member of TimeHrMin variable time1.

Check Show answer

5) Write a statement that assigns the value of the hourValue data member of time1 into the hourValue data member of time2.

Check Show answer

6) Write a single statement that assigns the values of all data members of time1 to the corresponding data members of time2.

Check Show answer

7) Declare a variable person1 of type Person, where Person is already defined as a struct type.

Check Show answer

CHALLENGE ACTIVITY 11.1.1: Enter the output using struct.

539740.3879454.qx3zqy7

Start

Type the program's output

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

struct Birthday {
   int month;
   int day;
};

int main() {
   Birthday annBirthday;

   annBirthday.month = 4;
   annBirthday.day = 8;

   cout << "Ann: " << annBirthday.month << "/" << annBirthday.day << endl;

   return 0;
}
```

Ann:

1 2 3 4

Check Next

CHALLENGE ACTIVITY 11.1.2: Declaring a struct.

Define a struct named PatientData that contains two integer data members named heightInches and weightPounds. Sample output for the given program with inputs 63 115:

```
Patient data: 63 in, 115 lbs
```


Learn how our autograder works

539740.3879454.qx3zqy7

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

/* Your solution goes here  */

int main() {
   PatientData lunaLovegood;

   cin >> lunaLovegood.heightInches;
   cin >> lunaLovegood.weightPounds;

   cout << "Patient data: "
        << lunaLovegood.heightInches << " in, "
        << lunaLovegood.weightPounds << " lbs" << endl;

```

Run

CHALLENGE ACTIVITY 11.1.3: Accessing a struct's data members.

Write a statement to print the data members of InventoryTag. End with newline. Ex: if itemID is 314 and quantityRemaining is 500, print:

```
Inventory ID: 314, Qty: 500
```


Learn how our autograder works

539740.3879454.qx3zqy7

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

```
4  struct InventoryTag {
5     int itemID;
6     int quantityRemaining;
7  };
8
9  int main() {
10    InventoryTag redSweater;
11
12    cin >> redSweater.itemID;
13    cin >> redSweater.quantityRemaining;
14
15    /* Your solution goes here  */
```

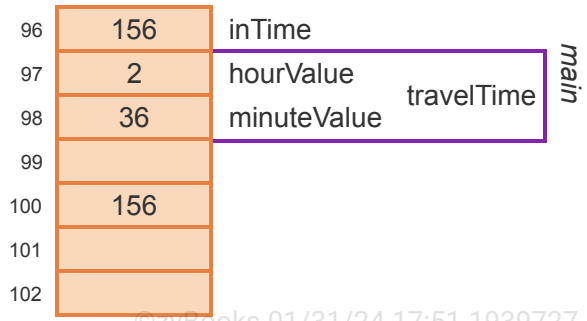

Run

11.2 Structs and functions

The struct construct's power is evident when used with functions. A struct can be used to return multiple values. Although ConvHrMin() has two output values, the struct type allows the function to return a single item, avoiding a less-clear approach using two pass by reference parameters.

PARTICIPATION ACTIVITY | 11.2.1: Using a struct that is returned from a function; the struct's data members are copied upon return.

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

struct TimeHrMin {
   int hourValue;
   int minuteValue;
};

TimeHrMin ConvHrMin(int totalTime) {
   TimeHrMin timeStruct;

   timeStruct.hourValue   = totalTime / 60;
   timeStruct.minuteValue = totalTime % 60;

   return timeStruct;
}

int main() {
   int inTime;
   TimeHrMin travelTime;

   cout << "Enter total minutes: ";
   cin >> inTime;
```




```
Enter total minutes: 156
Equals: 2 hrs 36 mins
```

```
   travelTime = ConvHrMin(inTime);

   cout << "Equals: ";
   cout << travelTime.hourValue << " hrs ";
   cout << travelTime.minuteValue << " mins";

   return 0;
}
```

Animation content:

Code snippet is as follows:

```
#include
using namespace std;

struct TimeHrMin {
  int hourValue;
  int minuteValue;
};

TimeHrMin ConvHrMin(int totalTime) {
  TimeHrMin timeStruct;

  timeStruct.hourValue  = totalTime / 60;
  timeStruct.minuteValue = totalTime % 60;

  return timeStruct;
}

int main() {
  int inTime;
  TimeHrMin travelTime;

  cout << "Enter total minutes: ";
  cin >> inTime;

  travelTime = ConvHrMin(inTime);

  cout << "Equals: ";
  cout << travelTime.hourValue << " hrs ";
  cout << travelTime.minuteValue << " mins";

  return 0;
}
```


Final memory contents is as follows:

96 (main's inTime): 156
97 (main's travelTime hourValue): 2
98 (main's's travelTime hourValue): 36
99: empty
100 (ConvHrMin's totalTime): 156
101 (ConvHrMin's timeStruct hourValue): 2
102 (ConvHrMin's timeStruct minuteValue): 36

Animation captions:

1. The program prompts a user to enter travel time in minutes, then calls the ConvHrMin function to convert travel time to hours and minutes.
2. Upon return, timeStruct's data members are copied to main's travelTime variable.
3. Returning a struct type allows the ConvHrMin function to return a single item, avoiding a less-clear approach of using two pass-by-reference parameters.

zyDE 11.2.1: Monetary change program.

Complete the program to compute monetary change, using the largest coins possible.

Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  struct MonetaryChange {
5     int quarters;
6     // FIXME: Finish dat
7  };
8
9  MonetaryChange ComputeCl
10    MonetaryChange chang
11
12    // FIXME: Finish fun
13    change.quarters = 0;
14
15    return change;
```

119

Run

PARTICIPATION ACTIVITY 11.2.2: Functions returning struct values.

1) Complete the function definition for a function ComputeLocation that returns a struct of type GPSPosition.

```
(double latitude, double
longitude) {

     ...

}
```

Check Show answer

2) Complete the function to return the calculated elapsed time, which gets stored in elapsedTime.

```
TimeEntry CalcElapsedTime(int
startSecs, int endSecs) {

     TimeEntry elapsedTime;

     ...

     elapsedTime.totalSecs =
endSecs - startSecs;

     elapsedTime.hours =
(endSecs - startSecs) / 3600;

     ...

;

}
```

Check Show answer

Likewise, a variable of a struct type can be a function parameter. And just like other types, a pass by value parameter would copy the item, while a pass by reference parameter would not.

PARTICIPATION ACTIVITY 11.2.3: Functions with struct parameters.

1) Complete the function definition for a function CalcSpeed that returns a double value and has two struct type parameters startLoc and endLoc (in that order) of type GPSPosition.

```
double CalcSpeed(
                 )
{
   ...
}
```

Check Show answer

2) Complete the following statement to calculate the speed between gpsPos1 and gpsPos2 by making a call to the CalcSpeed function.

```
double vehicleSpeed;
GPSPosition gpsPos1;
GPSPosition gpsPos2;
...
vehicleSpeed =
              ;
...
```

Check Show answer

CHALLENGE ACTIVITY 11.2.1: Enter the output of the struct and function.

539740.3879454.qx3zqy7

Start

Type the program's output

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

struct Home {
   int numBathrooms;
   int numFloors;
   int numPeople;
   int numRooms;
};

Home InitHome() {
   Home tempHome;

   tempHome.numBathrooms = 5;
   tempHome.numFloors = 3;
   tempHome.numPeople = 8;
   tempHome.numRooms = 7;

   return tempHome;
}

int main() {
   Home myHome = InitHome();

   cout << myHome.numBathrooms << " Bathrooms" << endl;
   cout << myHome.numPeople << " People" << endl;

   return 0;
}
```



```
5 Bathroom
8 People
```

1 2 3

Check Next

CHALLENGE ACTIVITY 11.2.2: Structs and functions.

539740.3879454.qx3zqy7

Start

Write a statement that calls the function AddToStock with parameters computerInfo and addQty. Assign computerInfo with the value returned.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  struct ProductInfo {
6     string itemName;
7     int itemQty;
8  };
9
10 ProductInfo AddToStock(ProductInfo productToStock, int increaseValue) {
11    productToStock.itemQty = productToStock.itemQty + increaseValue;
12
13    return productToStock;
```

```
}

int main() {
   ProductInfo computerInfo;
   int addQty;

   cin >> computerInfo.itemName >> computerInfo.itemQty;
   cin >> addQty;

   /* Your code goes here */

   cout << "Name: " << computerInfo.itemName << ", stock: " << computer

   return 0;
}
```


1 2 3

Check Next

11.3 Structs and vectors

The power of structs becomes even more evident when used in conjunction with vectors. Consider a TV watching-time program where a user can enter a country name, and the program outputs the average daily TV-watching hours for a person in that country. One approach is to use two same-sized vectors, one to hold names, and the other to hold numbers corresponding to each name. Instead of those two vectors, a struct allows for the declaration of just one vector that stores items that each have a name and number data member.

Figure 11.3.1: A vector of struct items rather than two vectors of more basic types.

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

struct CountryTvWatch {
   string countryName;
   int tvMinutes;
};

int main() {
   // Source: www.statista.com, 2010
   const int NUM_COUNTRIES = 4;

   vector<CountryTvWatch> countryList(NUM_COUNTRIES);
   string countryToFind;
   bool countryFound;
   int i;

   countryFound = false;

   countryList.at(0).countryName = "Brazil";
   countryList.at(0).tvMinutes = 222;
   countryList.at(1).countryName = "India";
   countryList.at(1).tvMinutes = 119;
   countryList.at(2).countryName = "U.K.";
   countryList.at(2).tvMinutes = 242;
   countryList.at(3).countryName = "U.S.A.";
   countryList.at(3).tvMinutes = 283;

   cout << "Enter country name: ";
   cin >> countryToFind;

   for (i = 0; i < NUM_COUNTRIES && !countryFound; ++i) {
// Find country's index
      if (countryList.at(i).countryName == countryToFind)
{
         countryFound = true;
         cout << "People in " << countryToFind << endl;
         cout << "watch " << countryList.at(i).tvMinutes;
         cout << " minutes of TV daily." << endl;
      }
   }
   if (!countryFound) {
      cout << "Country not found, try again." << endl;
   }

   return 0;
}
```



```
Enter country name:
U.S.A.
People in U.S.A.
watch 283 minutes
of TV daily.
...
Enter country name:
UK
Country not found,
try again.
...
Enter country name:
U.K.
People in U.K.
watch 242 minutes
of TV daily.
```

©zyBooks 01/31/24 17:51 1939727
Rob Daglio
MDCCOP2335Spring2024

The countryList variable is declared as `vector<CountryTvWatch> countryList(NUM_COUNTRIES)`, meaning a vector of items of

type CountryTvWatch. Thus, each vector element will have memory allocated for the struct's two data members, countryName and tvMinutes.

The notation countryList.at(i).countryName is equivalent to (countryList.at(i)).countryName, because the member access operator is evaluated left-to-right (as are any equal precedence operators). The left-to-right member access operator evaluation is well-known among programmers so parentheses are typically omitted.

PARTICIPATION ACTIVITY 11.3.1: Using structs with vectors.

Use .at() notation for vector element access.

1) Declare a vector countryList of 5 CountryTvWatch elements

Check Show answer

2) Given a vector countryList consisting of 5 CountryTvWatch struct elements, write a statement that assigns the value of the 0th element's tvMinutes data member to the variable countryMin.

Check Show answer

3) Given a vector countryList consisting of 5 CountryTvWatch struct elements, write one statement that copies element 4's struct values to element 0's.

Check Show answer

zyDE 11.3.1: Modify the TV watch program.

Finish the PrintCountryNames() function to print all country names in the list.

Load default template...

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  using namespace std;
5
6  struct CountryTvWatch {
7     string countryName;
8     int tvMinutes;
9  };
10
11 void PrintCountryNames(
12 {
13    cout << "FIXME: Fini
14 }
15
```

USA

Run

CHALLENGE ACTIVITY 11.3.1: Enter the output of the struct and vector.

539740.3879454.qx3zqy7

Start

Type the program's output

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

struct Car {
   string type;
   string color;
};

int main() {
   vector<Car> carList(3);

   carList.at(0).type = "sedan";
   carList.at(0).color = "red";
   carList.at(1).type = "SUV";
   carList.at(1).color = "white";
   carList.at(2).type = "truck";
   carList.at(2).color = "brown";

   cout << carList.at(0).color << " " << carList.at(0).type << endl;
   cout << carList.at(1).color << " " << carList.at(1).type << endl;

   return 0;
}
```


1 2 3

Check Next

CHALLENGE ACTIVITY 11.3.2: Structs and vectors.

539740.3879454.qx3zqy7

Start

Declare a vector named availablePizzas that stores 3 items of type PizzaIngredients.

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  using namespace std;
5
6  struct PizzaIngredients {
7     string pizzaName;
8     string ingredients;
9  };
10
11 int main() {
12
13    /* Your code goes here */
14
15    availablePizzas.at(0).pizzaName = "Barbecue";
16    availablePizzas.at(1).pizzaName = "Carbonara";
17    availablePizzas.at(2).pizzaName = "Ham and Cheese";
18    availablePizzas.at(0).ingredients = "Beef, chicken, bacon, barbecue
```

```
    availablePizzas.at(1).ingredients = "Mushrooms, onion, creamy sauce"
    availablePizzas.at(2).ingredients = "Ham, cheese, bacon";

    cout << availablePizzas.at(0).pizzaName << ": " << availablePizzas.a
    cout << availablePizzas.at(1).pizzaName << ": " << availablePizzas.a
    cout << availablePizzas.at(2).pizzaName << ": " << availablePizzas.a

    return 0;
}
```


1 2 3

Check Next

11.4 Enumerations

Some variables only need to store a small set of named values. For example, a variable representing a traffic light need only store values named GREEN, YELLOW, or RED. An ***enumeration type*** (enum) declares a name for a new type and possible values for that type.

Construct 11.4.1: Enumeration type.

```
enum identifier {enumerator1, enumerator2,
...};
```

The items within the braces ("enumerators") are integer constants automatically assigned an integer value, with the first item being 0, the second 1, and so on. An enumeration declares a new data type that can be used like the built-in types int, char, etc.

Figure 11.4.1: Enumeration example.

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

/* Manual controller for traffic light */
int main() {
   enum LightState {LS_RED, LS_GREEN, LS_YELLOW,
LS_DONE};
   LightState lightVal;
   char userCmd;

   lightVal = LS_RED;
   userCmd = '-';

   cout << "User commands: n (next), r (red), q
(quit)." << endl << endl;

   lightVal = LS_RED;
   while (lightVal != LS_DONE) {

      if (lightVal == LS_GREEN) {
         cout << "Green light  ";
         cin >> userCmd;
         if (userCmd == 'n') { // Next
            lightVal = LS_YELLOW;
         }
      }
      else if (lightVal == LS_YELLOW) {
         cout << "Yellow light  ";
         cin >> userCmd;
         if (userCmd == 'n') { // Next
            lightVal = LS_RED;
         }
      }
      else if (lightVal == LS_RED) {
         cout << "Red light  ";
         cin >> userCmd;
         if (userCmd == 'n') { // Next
            lightVal = LS_GREEN;
         }
      }

      if (userCmd == 'r') { // Force immediate
red
         lightVal = LS_RED;
      }
      else if (userCmd == 'q') { // Quit
         lightVal = LS_DONE;
      }
   }

   cout << "Quit program." << endl;

   return 0;
}
```

```
User commands: n (next), r
(red), q (quit).

Red light  n
Green light  n
Yellow light  n
Red light  n
Green light  r
Red light  n
Green light  n
Yellow light  n
Red light  q
Quit program.
```


©zyBooks 01/31/24 17:51 1939727
Rob Daglio
MDCCOP2335Spring2024

The program declares a new enumeration type named LightState. The program then declares a new variable lightVal of that type. The loop updates lightVal based on the user's input.

The example illustrates the idea of a ***state machine*** that is sometimes used in programs, especially programs that interact with physical objects, wherein the program moves among particular situations ("states") depending on input; see What is: State machine.

Because different enumerated types might use some of the same names, e.g., `enum Colors {RED, PURPLE, BLUE, GREEN};` might also appear in the same program, the program above follows the practice of prepending a distinguishing prefix, in this case "LS" (for Light State).

One might ask why the light variable wasn't simply declared as a string, and then compared with strings "GREEN", "RED", and "YELLOW". Enumerations are safer. If using a string, an assignment like `light = "ORANGE"` would not yield a compiler error, even though ORANGE is not a valid light color. Likewise, `light == "YELOW"` would not yield a compiler error, even though YELLOW is misspelled.

One could instead declare constant strings like `const string LS_GREEN = "GREEN";` or even integer values like `const int LS_GREEN = 0;` and then use those constants in the code, but an enumeration is clearer, requires less code, and is less prone to error.

Note: Each enumerator by default is assigned an integer value of 0, 1, 2, etc. However, a programmer can assign a specific value to any enumerator. Ex:
`enum TvChannels {TC_CBS = 2, TC_NBC = 5, TC_ABC = 7};`

PARTICIPATION ACTIVITY 11.4.1: Enumeration syntax.

1) Which of the following declares a new enumeration type named CarGear, with PARK, REVERSE, and DRIVE?

- `enum CarGear (PARK, REVERSE, DRIVE);`
- `enum CarGear {PARK, REVERSE, DRIVE}`
- `enum CarGear {PARK, REVERSE, DRIVE};`
- `CarGear {PARK, REVERSE, DRIVE};`

PARTICIPATION ACTIVITY 11.4.2: Enumerations.

1) Declare a new enumeration type named HvacStatus with three named values HVAC_OFF, AC_ON, FURNACE_ON, in that order.

Check Show answer

2) Declare a variable of the enumeration type HvacStatus named systemStatus.

Check Show answer

3) Assign AC_ON to the variable systemStatus.

Check Show answer

4) What is the integer value of systemStatus after the following?

```
systemStatus =
FURNACE_ON;
```

Check Show answer

5) Given `enum TvChannels {TC_CBS = 2, TC_NBC = 5, TC_ABC = 7};`, what does `cout << TC_ABC;` output?

Check Show answer

CHALLENGE ACTIVITY 11.4.1: Enumerations: Grocery items.

Output "Fruit" if the value of userItem is a type of fruit. Otherwise, if the value of userItem is a type of drink, output "Drink". Otherwise, output "Unknown". End each output with a newline.

Ex: If userItem is `GR_APPLES`, then the output is:

```
Fruit
```


Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   enum GroceryItem {GR_APPLES, GR_BANANAS, GR_JUICE, GR_WATER};
   GroceryItem userItem;

   userItem = GR_APPLES; /* Your code will be tested with GR_APPLES and othe

   /* Your solution goes here  */

   return 0;
}
```

Run

CHALLENGE ACTIVITY 11.4.2: Soda machine with enums.

The following program reads a number from input to indicate the coin inserted into a soda machine. Complete the code provided to add the appropriate amount to totalDeposit.

- Enumerator ADD_QUARTER has a value of 0 (add 25).
- Enumerator ADD_DIME has a value of 1 (add 10).
- Enumerator ADD_NICKEL has a value of 2 (add 5).

Learn how our autograder works

539740.3879454.qx3zqy7

```
#include <iostream>
```

```
using namespace std;

int main() {
   enum AcceptedCoins {ADD_QUARTER, ADD_DIME, ADD_NICKEL};
   int totalDeposit;
   int userInput;

   totalDeposit = 0;
   cin >> userInput;

   if (userInput == ADD_QUARTER) {
      totalDeposit = totalDeposit + 25;
   }

```


Run

11.5 Structs, vectors, and functions: A seat reservation example

A programmer commonly uses structs, vectors, and functions together. Consider a program that allows a reservations agent to reserve seats for people, useful for a theater, an airplane, etc. The below program defines a Seat struct whose data members are a person's first name, last name, and the amount paid for the seat. The program declares a vector of 5 seats to represent the theater, airplane, etc., initializes all seats to being empty (indicated by a first name of "empty"), and then allows a user to enter commands to print all seats, reserve a seat, or quit.

Figure 11.5.1: A seat reservation system involving a struct, vector, and functions.

```
Enter command (p/r/q):
p
0: empty empty, Paid:
0
1: empty empty, Paid:
0
2: empty empty, Paid:
0
3: empty empty, Paid:
0
4: empty empty, Paid:
0

Enter command (p/r/q):
r
Enter seat num: 2
Enter first name: John
Enter last name: Smith
Enter amount paid: 500
Completed.

Enter command (p/r/q):
p
0: empty empty, Paid:
0
1: empty empty, Paid:
0
2: John Smith, Paid:
500
3: empty empty, Paid:
0
4: empty empty, Paid:
0

Enter command (p/r/q):
r
Enter seat num: 2
Seat not empty.

Enter command (p/r/q):
r
Enter seat num: 3
Enter first name: Mary
Enter last name: Jones
Enter amount paid: 198
Completed.

Enter command (p/r/q):
p
0: empty empty, Paid:
0
1: empty empty, Paid:
0
2: John Smith, Paid:
500
3: Mary Jones, Paid:
198
4: empty empty, Paid:
0

Enter command (p/r/q):
q
Quitting.
```

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

struct Seat {
   string firstName;
   string lastName;
   int amountPaid;
};

/*** Functions for Seat ***/
void SeatMakeEmpty(Seat& seat) {
   seat.firstName   = "empty";
   seat.lastName    = "empty";
   seat.amountPaid  = 0;
}

bool SeatIsEmpty(Seat seat) {
   return(seat.firstName == "empty");
}

void SeatPrint(Seat seat) {
   cout << seat.firstName << " ";
   cout << seat.lastName << ", ";
   cout << "Paid: " << seat.amountPaid << endl;
}
/*** End functions for Seat ***/

/*** Functions for vector of Seat ***/
void SeatsMakeEmpty(vector<Seat>& seats) {
   unsigned int i;
   for (i = 0; i < seats.size(); ++i) {
      SeatMakeEmpty(seats.at(i));
   }
}

void SeatsPrint(vector<Seat> seats) {
   unsigned int i;
   for (i = 0; i < seats.size(); ++i) {
      cout << i << ": ";
      SeatPrint(seats.at(i));
   }
}
/*** End functions for vector of Seat ***/

int main() {
   char userKey;
   int  seatNum;
   vector<Seat> allSeats(5);
   Seat currSeat;

   userKey = '-';

   SeatsMakeEmpty(allSeats);

   while (userKey != 'q') {

      cout << endl << "Enter command (p/r/q): ";
```


©zyBooks 01/31/24 17:51 1939727
Rob Daglio
MDCCOP2335Spring2024

```
      cin >> userKey;

      if (userKey == 'p') { // Print seats
         SeatsPrint(allSeats);
      }
      else if (userKey == 'r') { // Reserve seat
         cout << "Enter seat num: ";
         cin  >> seatNum;

         if (!SeatIsEmpty(allSeats.at(seatNum)))
{
            cout << "Seat not empty." << endl;
         }
         else {
            cout << "Enter first name: ";
            cin >> currSeat.firstName;
            cout << "Enter last name: ";
            cin >> currSeat.lastName;
            cout << "Enter amount paid: ";
            cin >> currSeat.amountPaid;

            allSeats.at(seatNum) = currSeat;
            cout << "Completed." << endl;
         }
      }
      // FIXME: Add option to delete
reservations
      else if (userKey == 'q') {
         cout << "Quitting." << endl;
      }
      else {
         cout << "Invalid command." << endl;
      }
   }

   return 0;
}
```


The programmer first defined several functions related to the Seat struct, such as checking if a seat is empty or printing a seat. The programmer then defined some functions related to a *vector* of seat items. To distinguish, the programmer named the former starting with Seat and the latter starting with Seats.

The programmer left a "FIXME" comment indicating that the program also requires the ability to delete a reservation. That functionality is straightforward to introduce, just requiring the user to enter a seat number and then making use of the existing SeatMakeEmpty() function.

Notice how main() is relatively clean, dealing mostly with the user commands, and then using functions to carry out the appropriate work. Actually, the "reserve seat" command could be improved; main() currently fills the reservation information (e.g., "Enter first name..."), but main() would be cleaner if it just called a function such as `SeatFillReservationInfo(currSeat)`.

The seat reservation program loses all its information when exited. An improvement is to save all reservation information in a file. Commands 's' and 'g' would save and get information to/from a file, respectively.

PARTICIPATION ACTIVITY 11.5.1: Seat reservation example with struct, vector, and functions.

Refer to the above example.

1) The number of seats is 5.
   - True
   - False

2) SeatsMakeEmpty(seats) has a loop that sets each seat in the seats vector to have a first name of "empty".
   - True
   - False

3) SeatIsEmpty() checks if all the seats in the vector are empty.
   - True
   - False

4) Deleting a reservation would reduce the vector size from 5 down to 4.
   - True
   - False

zyDE 11.5.1: Introduce delete behavior to the reservation program.

Modify main() to allow the user to enter command 'd', followed by the user entering a seat number. Call SeatMakeEmpty() to delete the seat.

Load default template...

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  using namespace std;
5
6  struct Seat {
7     string firstName;
8     string lastName;
9     int amountPaid;
10 };
11
12 /*** Functions for Seat
13 void SeatMakeEmpty(Seat&
14    seat.firstName   = "
15    seat.lastName    = "
```

```
p
r 2 John Smith 500
p
```

Run

©zyBooks 01/31/24 17:51 1939727
Rob Daglio
MDCCOP2335Spring2024