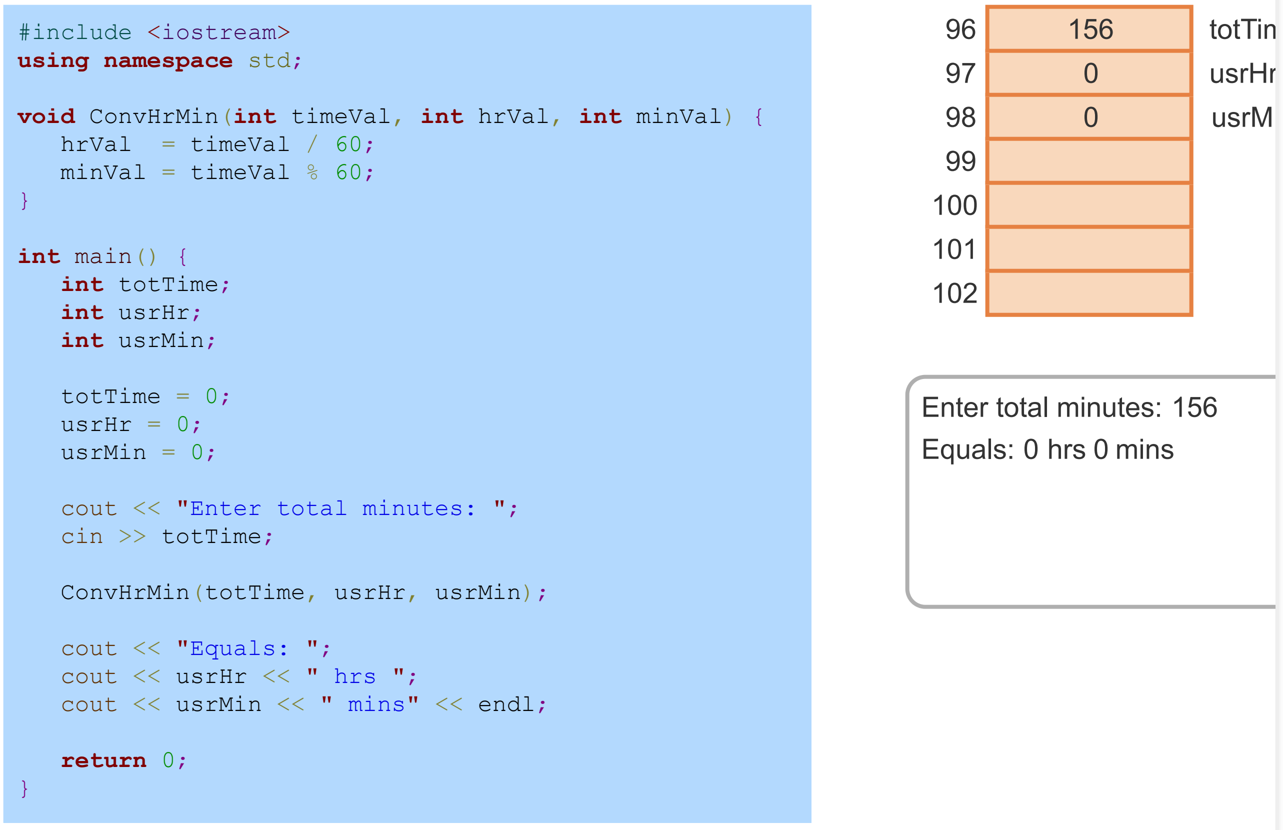
6.8 Pass by reference

Pass by reference

New programmers sometimes assign a value to a parameter, believing the assignment updates the corresponding argument variable. An example situation is when a function should return two values, whereas a function's *return* construct can only return one value. Assigning a normal parameter fails to update the argument's variable, because normal parameters are ***pass by value***, meaning the argument's value is copied into a local variable for the parameter.

PARTICIPATION ACTIVITY 6.8.1: Assigning a normal pass by value parameter has no impact on the corresponding argument.



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

void ConvHrMin(int timeVal, int hrVal, int minVal) {
   hrVal  = timeVal / 60;
   minVal = timeVal % 60;
}

int main() {
   int totTime;
   int usrHr;
   int usrMin;

   totTime = 0;
   usrHr = 0;
   usrMin = 0;

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

   ConvHrMin(totTime, usrHr, usrMin);

   cout << "Equals: ";
   cout << usrHr << " hrs ";
   cout << usrMin << " mins" << endl;

   return 0;
}
```

96	156	totTin
97	0	usrHr
98	0	usrM
99		
100		
101		
102		

```
Enter total minutes: 156
Equals: 0 hrs 0 mins
```

Upon return, ConvHrMin's local variables are discarded.
hrVal and minVal are local copies that do not impact usrHr and usrMin.

Feedback?

C++ supports another kind of parameter that enables updating of an argument variable. A ***pass by reference*** parameter does *not* create a local copy of the argument, but rather the parameter refers directly to the argument variable's memory location. Appending & to a parameter's data type makes the parameter pass by reference type.

PARTICIPATION ACTIVITY 6.8.2: A pass by reference parameter allows a function to update an argument variable.

1 2 3 4 2x speed

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

void ConvHrMin(int timeVal, int& hrVal, int& minVal) {
   hrVal = timeVal / 60;
   minVal = timeVal % 60;
}

int main() {
   int totTime;
   int usrHr;
   int usrMin;

   totTime = 0;
   usrHr = 0;
   usrMin = 0;

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

   ConvHrMin(totTime, usrHr, usrMin);

   cout << "Equals: ";
   cout << usrHr << " hrs ";
   cout << usrMin << " min" << endl;

   return 0;
}
```

96	156	totTime
97	2	usrHr
98	36	usrMin
99		
100		
101		
102		

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

Upon return from ConvHrMin, usrHr and usrMin retain the updated values.

Feedback?

Pass by reference parameters should be used sparingly. For the case of two return values, commonly a programmer should instead create two functions. For example, defining two separate functions `int StepsToFeet(int baseSteps)` and `int StepsToCalories(int totCalories)` is better than a single function `void StepsToFeetAndCalories(int baseSteps, int& baseFeet, int& totCalories)`. The separate functions support modular development, and enables use of the functions in an expression as in `if (StepsToFeet(mySteps) < 100)`.

Using multiple pass by reference parameters makes sense when the output values are intertwined, such as computing monetary change, whose function might be `void ComputeChange(int totCents, int& numQuarters, int& numDimes, int& numNick` or converting from polar to Cartesian coordinates, whose function might be `void PolarToCartesian(int radialPol, int anglePol, int& xCar, int& yCar)`.

zyDE 6.8.1: Calculating monetary change.

Complete the monetary change program. Use the fewest coins (i.e., using maximu coins first).

Load default

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

// FIXME: Add parameters for dimes, nickels, and pennies.
void ComputeChange(int totCents, int& numQuarters ) {

   cout << "FIXME: Finish writing ComputeChange" << endl;

   numQuarters = totCents / 25;
}

int main() {
   int userCents;
   int numQuarters;
   // FIXME add variables for dimes, nickels, pennies

   cout << "Enter total cents: " << endl;
   cin >> userCents;

   cout << "FIXME: Finish writing main()" << endl;
```

```
83
```

Run

Feedback?

PARTICIPATION ACTIVITY 6.8.3: Function definition returns and arguments.

Choose the most appropriate function definition.

1) Convert inches into centimeters.

- ○ `void InchToCM(double inches, double centimeters) ...`
- ○ `double InchToCM(double inches) ...`
- ○ More than one function should be written.

2) Get a user's full name by prompting "Enter full name" and then automatically separating into first and last names.

- ○ `void GetUserFullName(string& firstName, string& lastName) ...`
- ○ `string GetUserFullName() ...`
- ○ `string, string GetUserFullName() ...`
- ○ More than one function should be written.

3) Compute the area and diameter of a circle given the radius.

- ○ `void GetCircleAreaDiam(double radius, double& area, double& diameter) ...`
- ○ `double GetCircleAreaDiam (double radius, double& area) ...`
- ○ `double, double GetCircleAreaDiam(double radius) ...`
- ○ More than one function should be written.

Feedback?

PARTICIPATION ACTIVITY 6.8.4: Function definitions with pass by value and pass by reference.

Complete the function definition, creating pass by value or pass by reference parameters as appropriate.

1) Convert gallons to liters. Parameter is userGallons, type is double.

```
double GallonsToLiters(
                          ) {
```

Check Show answer

2) Convert userMeters into userFeet and userInches (three parameters, in that order), types are doubles.

```
void MetersToFeetInches(
                          ) {
```

Check Show answer

Feedback?

Avoid assigning pass by value parameters

Although a pass by value parameter creates a local copy, good practice is to avoid assigning such a parameter. The following code is correct but bad practice.

Figure 6.8.1: Programs should not assign pass by value parameters.

```
int IntMax(int numVal1, int numVal2) {
   if (numVal1 > numVal2) {
      numVal2 = numVal1; // numVal2 holds max
   }

   return numVal2;
}
```

Feedback?

Assigning a parameter can reduce code slightly, but is widely considered a lazy programming style. Assigning a parameter can mislead a reader into believing the argument variable is supposed to be updated. Assigning a parameter also increases likelihood of a bug caused by a statement reading the parameter later in the code but assuming the parameter's value is the original passed value.

PARTICIPATION ACTIVITY 6.8.5: Assigning a pass by value parameter.

1) Assigning a pass by value parameter in a function is discouraged due to potentially confusing a program reader into believing the argument is being updated.

○ True

○ False

2) Assigning a pass by value parameter in a function is discouraged due to potentially leading to a bug where a later line of code reads the parameter assuming the parameter still contains the original value.

○ True

○ False

3) Assigning a pass by value parameter can avoid having to declare an additional local variable.

○ True

○ False

Feedback?

Reference variables

A programmer can also declare a reference variable. A ***reference*** is a variable type that refers to another variable. Ex: `int& maxValRef` declares a reference to a variable of type int. The programmer must initialize each reference with an existing variable, which can be done by initializing the reference variable when the reference is declared. Ex:
`int& maxValRef = usrInput3;`

In the example below, usrValRef is a reference that refers to usrValInt. The user-entered number is assigned to the variable usrValInt. Because usrValRef refers to usrValInt, printing usrValInt or usrValRef will print the number.

Figure 6.8.2: Reference variable example.

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

int main() {
   int usrValInt;
   int& usrValRef = usrValInt;  // Refers to usrValInt

   cout << "Enter an integer: ";
   cin  >> usrValInt;

   cout << "We wrote your integer to usrValInt." << endl;
   cout << "usrValInt is: " << usrValInt << "." << endl;
   cout << "usrValRef refers to usrValInt, and is: " << usrValRef << "." << endl;

   usrValInt = 99;
   cout << endl << "We assigned usrValInt with 99." << endl;
   cout << "usrValInt is now: " << usrValInt << "." << endl;
   cout << "usrValRef is now: " << usrValRef << "." << endl;
   cout << "Note that usrValRef refers to usrValInt, so changed too." << endl;
   return 0;
}
```

```
Enter an integer: 42
We wrote your integer to usrValInt.
usrValInt is: 42.
usrValRef refers to usrValInt, and is: 42.

We assigned usrValInt with 99.
usrValInt is now: 99.
usrValRef is now: 99.
Note that usrValRef refers to usrValInt, so changed too.
```

Feedback?

PARTICIPATION ACTIVITY 6.8.6: Reference variables.

1) What does the following output?

```
int numAStudents = 12;
int numBStudents = 5;
int& studentsRef = numAStudents;

cout << studentsRef;
```

Check Show answer

2) What does the following output?

```
int examGrade = 95;
int& gradeRef = examGrade;

examGrade = examGrade + 1;
cout << gradeRef;
```

Check Show answer

3) What does the following output?

```
double treeHeightFt = 7.1;
double& heightRef = treeHeightFt;

heightRef = 12.2;
cout << treeHeightFt;
```

Check Show answer

4) Declare a reference named myScore and initialize the reference to the int variable teamScore.

Check Show answer

Feedback?

Exploring further:

- Passing arguments by value and by reference from msdn.microsoft.com

CHALLENGE

CHALLENGE ACTIVITY

6.8.1: Function pass by reference: Transforming coordinates.

Define a function CoordTransform() that transforms the function's first two input parameters xVal and yVal into two output parameters xValNew and yValNew. The function returns void. The transformation is new = (old + 1) * 2. Ex: If xVal = 3 and yVal = 4, then xValNew is 8 and yValNew is 10.

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

/* Your solution goes here  */
void CoordTransform(int xValUser, int yValUser, int& xValNew, int& yValNew){
   xValNew = (xValUser + 1) * 2;
   yValNew = (yValUser + 1) * 2;
}

int main() {
   int xValNew;
   int yValNew;
   int xValUser;
   int yValUser;

   cin >> xValUser;
   cin >> yValUser;

   CoordTransform(xValUser, yValUser, xValNew, yValNew);
   cout << "(" << xValUser << ", " << yValUser << ") becomes (" << xValNew << ", " << yV
```

Run ✓ All tests passed

✓ Testing with inputs: 3 4

Your output
```
(3, 4) becomes (8, 10)
```

✓ Testing with inputs: 0 0

Your output
```
(0, 0) becomes (2, 2)
```

Feedback?