Topic 9

Reference Parameters: Motivation

- Suppose you would like a function to get the user’s last name and ID number.
- The variables for this data are in your scope.
- But you want the function to change them for you.

- If you want to write a function that changes the value of a parameter, you must use a *reference parameter.*

Reference Parameter Example

Consider a function that simulates withdrawing a given amount of money from a bank account, provided that sufficient funds are available.

If the amount of money is insufficient, a $10 penalty is deducted instead.

The function would be used as follows:

```
double harrys_account = 1000;
withdraw(harrys_account, 100);
     // Now harrys_account is 900
withdraw(harrys_account, 1000);
     // Insufficient funds.
     // Now harrys_account is 890
```

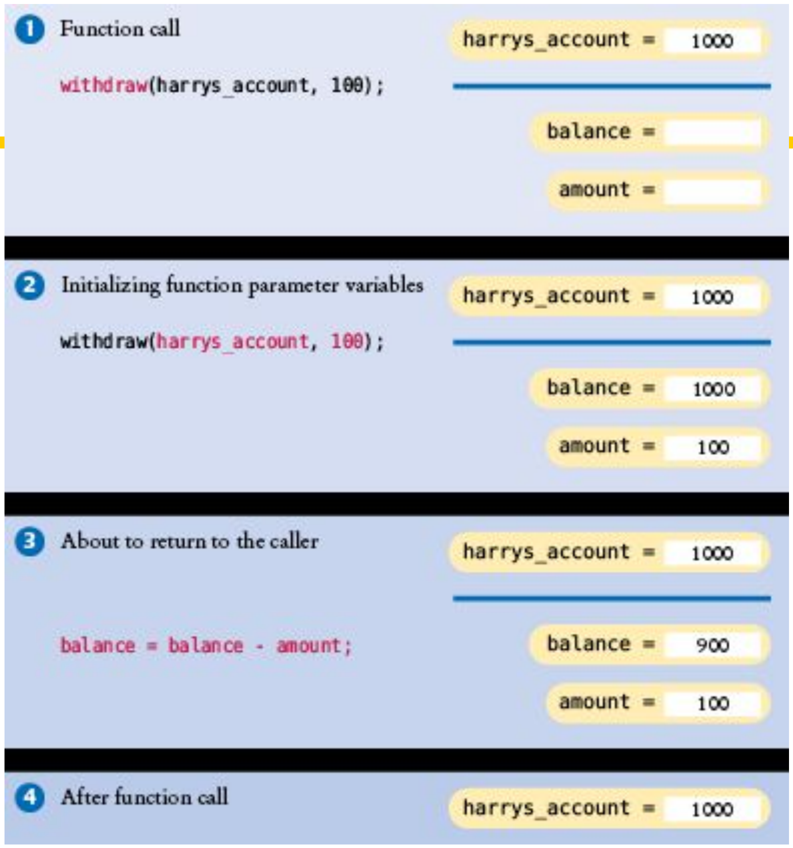
The Function without Reference Parameters Can’t Do It

Here is a first attempt:

```
void withdraw(double balance, double amount)
{
   const double PENALTY = 10;
   if (balance >= amount)
   {
      balance = balance - amount;
   }
   else
   {
      balance = balance - PENALTY;
   }
}
```

But this doesn’t work, *because a function cannot modify its input parameter value in the calling program*.

Withdraw Function Parameter



- `balance` is a "value" parameter
  - as all C++ function parameters are by default.
  - A COPY of the value of the main program's variable is provided to the function, not the location of the variable itself. Thus `withdraw` changes its local `balance` but that does not effect the value of the variable `harrys_account` in the scope of `main`
  - *Even if the variables had the same name, still the copy in `main` would not be changed by the call.*

Reference Parameters Provide the Solution

A reference parameter, indicated by &, "refers" to a variable that is supplied in a function call.

"refers" means that during the execution of the function, the reference parameter name is another name for the caller's variable.

This is how a function can change non-local variables:

changes to its reference parameters actually are changes to the variable in the calling function.

A reference parameter is actually the memory address of the caller's variable.

Reference Parameter Function Header: &

To indicate a ***reference parameter***,
you place an `&` after the type name.

```
void withdraw(double& balance, double amount)
```

To indicate a ***value parameter***,
you do *not* place an `&` after the type name.

Reference Parameter: Function Code

Here is correct code, using reference parameters:

```
void withdraw(double& balance, double amount)
{
   const int PENALTY = 10;
   if (balance >= amount)
   {
      balance = balance - amount;
   }
   else
   {
      balance = balance - PENALTY;
   }
}
```

Let’s see this in action.

Reference Parameter: Testbench Code

```
int main()
{
    double harrys_account = 1000;
    double sallys_account = 500;
    withdraw(harrys_account, 100);
        // Now harrys_account is 900
    withdraw(harrys_account, 1000); // Insufficient funds
        // Now harrys_account is 890
    withdraw(sallys_account, 150);
    cout << "Harry's account: " << harrys_account << endl;
    cout << "Sally's account: " << sallys_account << endl;

    return 0;
}
```

Diagram of Variable Values with Reference Parameter

`void withdraw(double& balance, double amount)`

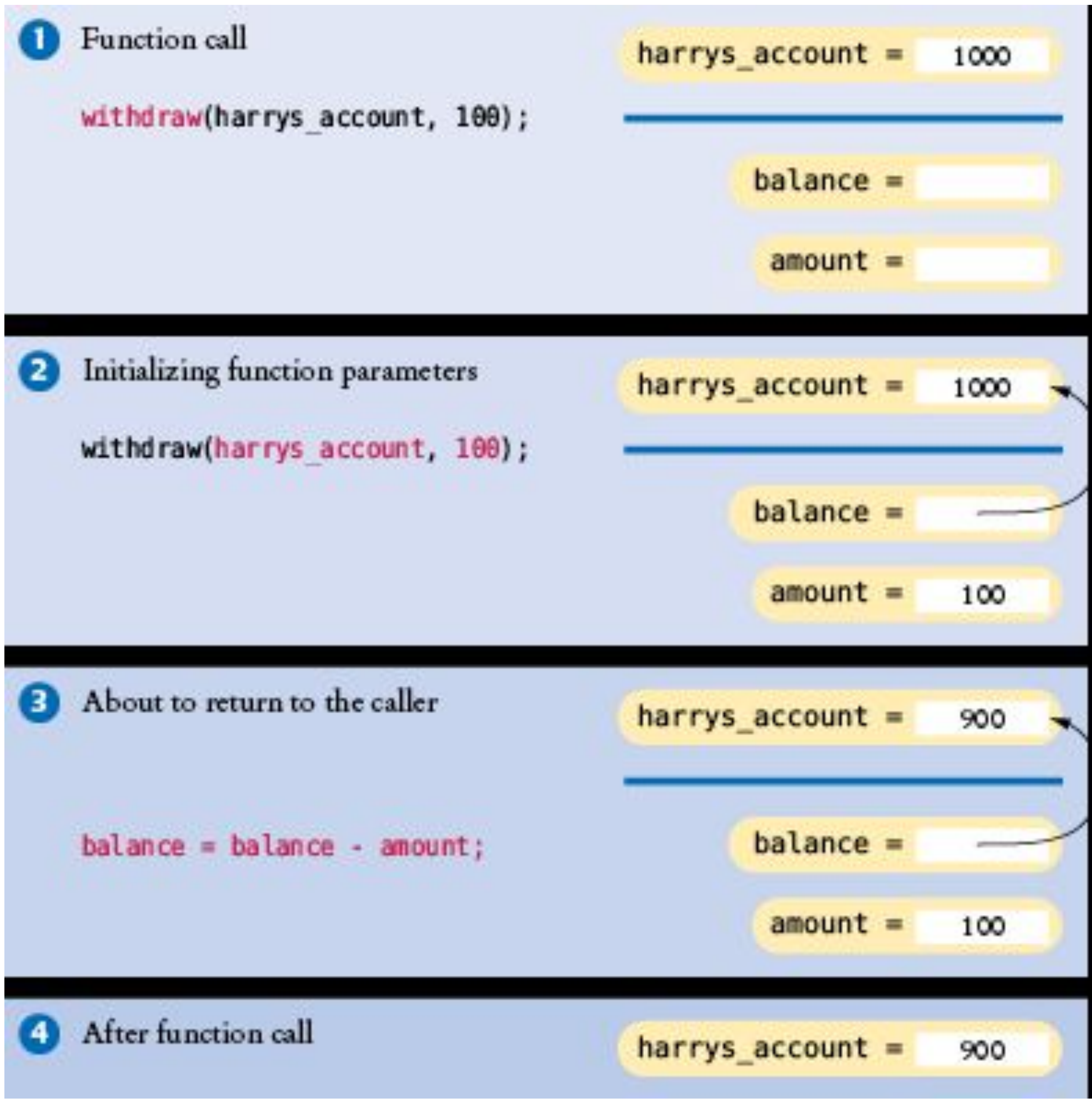
- With `balance` as a reference parameter, the `withdraw` function changes the value of `harrys_account` in the `main` program.

The type **`double&`** is pronounced:

reference to double

or

double ref



Reference Parameter Arguments Must be Variables

A reference parameter must always be called with a variable.

It would be an error to supply a number:

```
withdraw(1000, 500);
      // Error: reference parameter must be a variable
```

The reason is clear—the function modifies the reference parameter, but it is impossible to change the value of a number.

For the same reason, you cannot supply an expression:

```
withdraw(harrys_account + 150, 500); //Error
```

Prefer Return Values to Reference Parameters

Some programmers use reference parameters as a mechanism for setting the result of a function.

For example:

```
void cube_volume(double side_length, double& volume)
{
   volume = side_length * side_length * side_length;
}
```

However, this function is less convenient than our previous **`cube_volume`** function.

Why We Prefer Return Values to Reference Parameters

```
void cube_volume(double side_length, double& volume)
{
   volume = side_length * side_length * side_length;
}
```

This function cannot be used in expressions such as:

```
cout << cube_volume(2)
```

But You Can “Return” Multiple Values via References

The `return` statement can return only one value.

If caller wants more than two values, then the only way to do this is with reference parameters (one for each wanted value).

For example:

```
void powers(double x, double& square, double& cube)
{
   square = x * x;
   cube = square * x;
}
```

Constant References

It is not very efficient to have a value parameter that is a large object (such as a string value).

Copying the object into a parameter variable is less efficient than using a reference parameter.

With a reference parameter, only the location of the variable, not its value, needs to be transmitted to the function.

Constant Reference Example

You can instruct the compiler to give you the efficiency of a reference parameter and the meaning of a value parameter, by using a *constant reference*:

```
void shout(const string& str)
{
   cout << str << "!!!" << endl;
}
```

This is more efficient than having `str` be a value parameter.