CS 121 Lab 9

Call by reference in C++ *Paul Cao*

Call by reference method in C++ functions allows us to "return" multiple values from the function to the caller. In this lab, we will practice call by reference, in addition to the call by value approach we have practiced before. The code to this lab is due before midnight on 11/9/2010 for CS121A and 11/11/2010 for CS121B.

Pre-lab

To simply put the reason why we need to introduce call by reference: *a function cannot return more than one value using the return statement, but can use the call-by-reference to update many values.*

We have seen many examples of functions that returned one value or no value at all. Sometimes we may want to return more than one value from a function. The return statement can be used to return one value only. Instead, we will use the **call-by-reference** mechanism to update the arguments that are passed to a function. A call-by-reference parameter is marked by an `&` so that the compiler will distinguish it from other parameters.

Let's look at a program and see how this works. In the following example, the `get_input` function obtains two values from the user, then returns (brings) them to the main function. In a sense, `get_input` "returns" two values. This cannot be done with the return statement because the return statement returns exactly one value. If a function must produce more than one output value, then we must use call-by-reference parameters (one for each output value.) In the `process` function, since the second parameter is not done via call by reference but call by value instead, the corresponding variable in the main function won't be changed (i.e. the value of `j` is still what the user inputs in the main function after the `process` function call).

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

// This is the declaration for the function that reads the values for i and j
void get_input(int& i, int& j);
// This is the declaration for the function that adds 10 to i but it won't be able to add 20 to j
void process(int& i, int j);

int main() {
     int i, j;
     get_input(i, j);
     cout << "I am about to call function process, i = " << i << " j = " << j << endl;
     process(i,j);
     cout << "I just came back from function Process, i = " << i << " j = " << j << endl;
    return 0;
}
void get_input(int& i, int& j) {
     cout << "Please enter two values for i and j separated by a single space, then press <Enter>:";
     cin >> i >> j;
     cout << endl;
     return;   // a void function, returns nothing
}
void process(int& i, int j) {
     i = i+10;
     j = j+20;
     cout << "Inside function Process \n";
     cout << "I added 10 to i, and 20 to j, i = " << i << " and j = " << j << "\n";
     return;
}
```

One sample output of this code is:

```
Please enter two values for i and j separated by a single space, then press <Enter>:3 5

I am about to call function process, i = 3 j = 5
Inside function Process
I added 10 to i, and 20 to j, i = 13 and j = 25
I just came back from function Process, i = 13 j = 5
```

In-Lab

In the mathematics department at the University of Puzzleland, Tom, Gordon, and Darren had an argument over which one of them was the greatest puzzler of all time. To end the argument once and for all, they agreed on a duel to the death. Tom is a poor shooter and only hits his target with a probability of 1/3. Gordon is a bit better and hits his target with a probability of 1/2. Darren is an expert marksman and never misses. A hit means a kill and the person drops out of the duel.

To compensate for the inequalities in their marksmanship skills, it is decide that the contestants would fire in turns starting with Tom, followed by Gordon, and then by Darren. The cycle would repeat until there was one man standing. And that man would be remembered as the greatest puzzler of all time.

Write a code to simulate the game 1000 times using a loop, keeping track of how many times each contestant wins. Output the probability that each contestant will win in the end. It is assumed that everyone uses the strategy of shooting at the most accurate shooter left alive.

Implement your code with the following requirements

1. Use a function to simulate a single shot. It should use the following declaration:

```
void shoot(bool& targetAlive, double accuracy);
```

This would simulate someone shooting at targetAlive with the given accuracy by generating a random number between 0 and 1. If the random number is less than accuracy, then the target is hit and targetAlive should be set to false.

For example, if Gordon is shooting at Darren, this would be invoked as:

```
shoot(darrenAlive, .5);
```

Here, `darrenAlive` is a Boolean variable that indicates if Darren is alive.

To generate a number between 0 and 1, you can borrow the idea from the following code.

```
// this code generate a number between 0 and 1 inclusive.
#include <iostream>
#include <cstdlib>
#include <time.h>
using namespace std;
int main(){
     srand(time(0));// initialize the random number generator. You only need to do it once for the whole 
code!
     int x=rand()%100;// generate an integer between 0 and 99
     double y=x/99.0;// y has a value between 0 and 1 and it is a random number
     cout<<y;
}
```

2. Write a function named `startDuel` that uses the `shoot` function to simulate an entire duel using the strategy that each man will shoot at the most accurate shooter still alive on the grounds (because this

shooter is the deadliest and has the best chance of firing back). This function should loop until only one contestant is left, invoking the `shoot` function with the proper target and probability of hitting the target according to who is shooting. The function should return a value that indicates who won the duel.

From legaljuice.com