PIC 10A

Lecture 14: More About Scope, Random numbers

Scope review

Variables scope is the section of code where it lives.

If a variable is declared inside a function it only exists inside that function.

```
void foo (int b)
{
    cout << "Value of b is: " << b << "\n";
    cout << "Value of a is: " << a;
}

int main()
{
    int a;
    foo();
    cout << "Value of b is: " << b << "\n";
    cout << "Value of a is: " << a;
}
```

What lines above are an error?

Variables inside { }

In C++ we can use the braces to separate blocks of code. We have done this a lot already when using if statements and while and for loops.

Rarely we have done this just by themselves. It is legal. But it doesn't do much.

```
//Code
{
   // Code
}
// Code
```

What it does do is change scope.

Warning about these examples

Do not start using braces in your code for no reason!

There may be some very specific circumstances where they might be beneficial, but not for us.

I am telling you about this for the purpose that you can solve bizarre bugs that may come up in your programming and to illustrate a point.

In following examples you can imagine that the code inside { } is under an if statement or a loop.

Code block and scope examples

Example:

```
{
    int a=5;
}
cout << "Value of a is: " << a; // Error
```

Example:

```
int a=3;
{
    int a=5;
    cout << "Value of a is: " << a;
}
cout << "Value of a is: " << a;
```

What can we infer from these examples?

We should be careful when declaring variables inside if, while or for loops.

When we declare a variable inside { } that variable exists only inside the { }.

When a variable is declared inside { } and has a same name as a variable declared earlier, the new variable supersedes the old variable.

Common mistake

Example:

```
int i=0;
int j=0;

while(i<=10)
{
   j++;
   int i = j;
}
```

The `i` is local to this code block. In fact every iteration of the loop a new `i` is created. This is not the same `i` that is in the while statement!!

This is an infinite loop!

Note how `j` is available inside this block even though it was created outside of it.

Global variables

Sometimes we want a variable that is available to all functions and whose value can be changed by any function.

We could keep passing this variable to all functions by reference. (Sounds painful).

A better solution is to use a ***global variable***.

To declare a variable as global, declare it outside all functions.

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

int x=0; // x is now a global variable
```

Warning!

Using global variables is considered bad programming style.

There are very few cases where it makes sense to have a global variable.

Best to use global variable for constants that are needed in multiple functions.

```
const double PI=3.14159;
```

Ultimate scope example

```
using namespace std;
void f();
void g(int &x);
int x=5;
void f()
{
	int x=0;
	cout << ++x << endl;
}
void g(int &x)
{
	cout << x++ << endl;
	{
		cout << x << endl;
		int x=9;
		cout << x << endl;
		{
			cout << x << endl;
			int x=3;
			cout << x << endl;
		}
	}
}
int main()
{
	f();
	cout << x << endl;
	int x;
	x=2;
	cout << x << endl;
	g(x);
	cout << x << endl;
	return 0;
}
```

Random numbers

C++ has a random number generator that can generate a random integer between 0 and RAND_MAX.

RAND_MAX depends on the compiler you are using.

Typically RAND_MAX is either 32767 or 2147483647.

To get a random number we write:

```
int x=rand();
```

To use `rand()` `#include <cstdlib>`

Random numbers

As programmers we want to control the range of the random number bit better.

Often we want a random number in some specific range. Say from 0 to 3.

But rand() gives us a random number in a huge range. Usually between 0 and 2147483647.

Solution: use the % operator. If we have a random number then when we apply the % operator on it we still have a random number but now it is in the range we want.

Random numbers

```
int x = rand() % 4;
```

Why does this give us a random number from 0 to 3?

We can assume that rand() is some large number like 235638.

For any number the remainder by division with 4 is going to be either 0,1,2,3.

Since rand() was random number my remainder will be some random number also , but in the range 0-3

Controlling the range

So this suggest the following:

To get a random number between 0 and n we do

```
rand() % (n+1)
```

Because all the possible remainders when I divide by `n+1` are

```
0,1,2, ... ,n-1,n
```

What if I want a random number between `1` and `n`? Lets just add 1 to `rand % (n+1)`. Now possible values are

```
1,2,3, ... ,n,n+1
```

This is not quite the range I wanted...So lets do

```
(rand() % n ) + 1
```

Random number range cont.

Our goal is to understand how to get a random number in the range [a,b].

There are b-a+1 values in this range.

As an example consider the range [5,10]. There are 10-5+1=6 numbers in this range.

if we write `rand()% (b-a+1)` we get b-a+1 values but these values are

0,1,2, ... , b-a

If we add a to each number we get

a, a+1, a+2, ... ,b

So

```
a + rand() % (b-a+1)
```

gives us what we want.

Examples

Example:
Write a command to get a random number between 100 and 200.

```
int x = rand()% (200 - 100 + 1)  + 100;
```

Example:
Write a command to get a random decimal number between 10 and 20 with two decimal places.

```
double x = (double)((rand() % (2000 – 1000 + 1) + 1000))/ 100;
```

To think at home:
Write a command to get a random even number between 50 and 100.

Pseudo random numbers

Recall our number guessing program.

If you stop the program and then run it again the magic number will be the same!

What is going on?!

Problem is in the way rand() gets random numbers.

Essentially C++ has a book of random numbers and rand() fetches the next number in the book. Every time the program starts over, rand() starts reading numbers from the beginning of the book.

srand

We need to somehow tell the computer not to start reading the random numbers from the first page of the book. We need to start reading from a random page of the book.

There is a way to do this and it is called seeding the random number generator.

Syntax is:

```
srand(int);
```

Example:

```
srand(55);
```

Problem:
If you set the seed to say 55 every time then you just start reading from "page 55" every time and still get the same set of "random" numbers.

srand cont.

We need to randomly set the seed...

Seems like we are in the same predicament as we were before.

Getting random numbers is hard!!

Computer scientists have spent years thinking about this problem.

srand cont.

One way is to use the number of seconds elapsed since Jan 1 1970.

`time(0)` returns the number of seconds elapsed since Jan 1 1970.

We need to include `<ctime>` library to use time.

Now we can set the seed

```
srand( (int)time(0) );
```

We need to cast `time(0)` as an int since time returns number of type time_t.

Use srand only once!

Lets fix our game program

Add to the libraries

```
#include <ctime>
```

at top of main we put

```
srand((int)time(0));
```

Example

```
#include<iostream>
#include<ctime>

using namespace std;

int main()
{
    srand( (int) time(0) );
    int k = 1;
    while (k <= 20)
    {
        int die1 = 1+rand()%6;
        int die2 = 1+rand()%6;
        cout << die1 << " " << die2 <<"\n";
        k++;
    }
    return 0;
}
```

Professor Wittman's example

Playing Craps

In the casino game of craps, a shooter rolls until a sum of 7 is rolled. Then the dice are passed to the next player.

On average, how many rolls does a player’s turn last? This is a very difficult mathematical question, but we can try to answer it by simulating it numerically.

Solution

Roll until the sum is 7.

```
srand( (int) time(0) );
int rolls = 0;
int sum = 0;
while (sum != 7)
{
   int die1 = 1+rand()%6;
   int die2 = 1+rand()%6;
   cout << die1 <<" "<< die2 <<"\n";
   rolls++;
   sum = die1 + die2;
}
cout << "# Rolls = " << rolls << "\n";
```

But this doesn’t answer our question!

Solution

What we need to do is run the previous loop many many times and then take

```
srand( (int) time(0) );
int total_rolls = 0;
int run = 1;
while (run <= 10000)
{
    int sum = 0;
    while (sum != 7)
    {
        int die1 = 1+rand()%6;
        int die2 = 1+rand()%6;
        total_rolls++;
        sum = die1 + die2;
    }
    run++;
}
cout << "Average # Rolls = " << (double) total_rolls/10000 << "\n";
return 0;
```

Biased dice example

normally dice has 1/6 chance (or ~17%) of landing on any given number between 1 and 6.

People have tried to cheat casinos by using their own dice that have a higher chance to land on some number.

Lets write a program that simulates a dice that has a 30% chance of landing on 1 and 14% chance of landing on the other numbers.

Solution

```
int seed = (int)(time(0));
srand(seed);
int dice=rand() % 100 + 1;

if (dice <=30)
    cout << "You rolled a 1" << endl;
else if ( (dice >30) && dice <= 44)
    cout << "You rolled a 2" << endl;
else if ( (dice >44) && dice <= 58)
    cout << "You rolled a 3" << endl;
else if ( (dice >58) && dice <= 72)
    cout << "You rolled a 4" << endl;
else if ( (dice >72) && dice <= 86)
    cout << "You rolled a 5" << endl;
else
    cout << "You rolled a 6" << endl;
```