cisc1110 fall 2010 lecture III.1

- *unit III: more data types*
- random numbers
- "for" loops
- arrays

random numbers

- computers can generate "random" numbers, which is like picking a number by rolling dice
- there are two steps necessary for generating random numbers:
  1. initializing the random number generator
  2. picking the random number
- the random numbers generated are integers (of type `int`)
- the `srand()` function (which is part of the C++ language) is used to initialize the random number generator
- the `rand()` function (which is also part of the C++ language) is used to return a random number
- `rand()` returns a random integer between 0 and `RAND_MAX`, where `RAND_MAX` is a constant (defined as part of the C++ language) that represents the biggest integer allowed in the system (in our case, $2^{31} - 1$)
- using the modulo operator (%) you can *scale* the result returned from `rand()` to get a number in the range you want (e.g., $0..10$)

- note that you may need to have the following two `#include` statements at the top of your program file, in order for the compiler to recognize `srand()` and `rand()`:

```
#include <time.h>
#include <stdlib.h>
```

these are IN ADDITION to the `#include <iostream>` that you normally put, so the beginning of your program file will look like this:

```
#include <iostream>
using namespace std;
#include <time.h>
#include <stdlib.h>
```

you can put them after the
`using namespace std;`
statement, because they are not part of the `std` namespace

random numbers: complete example

```
#include <iostream>
using namespace std;
#include <time.h>
#include <stdlib.h>

int main() {
  // declare variables
  int x, y, z;
 // initialize random number generator using the current time as the seed
  srand( time( NULL ));
  // pick a random number
  x = rand();
  // scale the value of x to an integer between 0 and 10
  y = x % 11;
  // or do the above two steps in one,
  // essentially picking a random number between 0 and 10
  z = rand() % 11;
} // end of main()
```

for loops

- *looping*, or *iteration*, means doing something more than once, perhaps doing something over and over and over and ... and over again
- there are times when you want your program to do something once, and there are other times when you want your program to do something more than once—without having to repeat the code again
- in C++ (and most programming languages), there is a category of constructs (or "control structures") called *loops* which tell a program to do something more than once
- today we will talk about one type of loop, called a **for** loop
- when you write a for loop, you need to decide two things:
  1. how many times do you want the program to loop (iterate)?
  2. will the behavior of the program each time the loop runs (iterates)?

- here's an example of what a for loop looks like:

```
int i;
for ( i=0; i<10; i++ ) {
  cout << "hello\n";
} // end of for loop
```

this example will print the word `hello` on the screen ten times, each word on its own line

- here's another example:

```
int i, n;
srand( time( NULL ));
n = rand() % 101;
for ( i=0; i<n; i++ ) {
  cout << "hello\n";
} // end of for loop
```

how many times will `hello` print out here??

components of a for loop

- a *for* loop contains three *clauses* and looks like this:

```
for ( <initialization> ; <termination> ; <continuation> ) {
  <body-of-for-loop>
} // end of for loop
```

- the `<initialization>` clause is something like `i=0;`
  this is done once, before the statements in the body of the loop are executed; often, it initializes a variable referred to as the *loop counter*; this variable keeps track of how many times the loop iterates.
- the `<termination>` clause is something like `i<10;`
  this is done after the initialization clause and before the statements in the body of the loop are executed; it checks to see if the loop should iterate (again); i.e., it asks "are we done yet?" typically, it evaluates the loop counter to make sure it has not exceeded its maximum (i.e., the number of times the loop should run).
- the `<continuation>` is something like `i++`
  this is done after the statements in the body of the loop are executed; typically, it increments (or decrements) the loop counter.

things to know about using for loops

- with a for loop, it is important that something happens in the continuation statement to change the value of the condition, eventually; otherwise you will have an *infinite loop*—and that is BAD because it can hang or crash your computer
- note that the condition can be false before the loop begins, in which case the loop will never execute!
  (there are reasons you might want to do this)

arrays

- arrays are used to hold sets of related types of data
- the data could be integers (`int`) or floating point numbers (`double`) or boolean values (`bool`)
- the data could also be characters;
  arrays of characters are special arrays called *strings*
  we'll talk about strings in the next class
- today, we'll focus on arrays that store numbers (e.g., `int` or `double`)
- common things to do with numeric data stored in arrays:
  - find the largest (or smallest) element
  - add up the elements
  - compute the average of the elements
  - count the number of elements with some feature

what is an array?

- you can think of an array as a set of variables of the same data type, which are grouped together and all use the same name
- whereas the example below declares one integer variable:

  program code:

  ```
  int x;
  ```

  computer's memory:

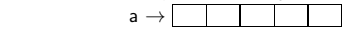
  x →

  this example declares an *array* of five variables:

  program code:

  ```
  int a[5];
  ```

  computer's memory:

  a → 

- the square brackets `[ ]` indicate to the compiler that it is dealing with an array
- elements of the array a are just integers, and we can do exactly the same things with them that we can do with integers
- the only difference is how we *address* (i.e., refer to) them

- we assign a value to `x` as follows:

  ```
  x = 12;
  ```

- to assign a value to one of the *elements* of `a`, we have to specify which element it is, using an *index* (i.e., a number in the brackets):

  ```
  a[0] = 12;
  ```

  which refers to the first element in array `a`
- all of the following are legal operations on array `a`:

```
a[0] += 2;
a[1] =  7 % 3;
a[2] =  a[1] - 5;
a[3] =  a[1] / a[2];
```


array indexes

- one thing to be careful of is the limits on the *index*, that is the number inside the square brackets `[ ]`
- the first element of an array always has index 0
- so the first element of `a` is:

  ```
  a[0]
  ```

  and, since `a` has 5 elements, the last element of `a` is:

  ```
  a[4]
  ```

- in other words, the last *index* is *the length of the array minus 1*
- this type of counting (from 0 to length-1) is standard in C++, C, Java and many other computer languages

why use arrays?

- arrays are useful when you want to store lots of data in memory
- if I want to use 3 integers in my program, then I would just declare 3 different integer variables
- however, if I wanted to use 30,000 integers in my program, it would be a lot easier to use an array than to declare 30,000 different integer variables!
- arrays also go very nicely with **for loop**s :-)

example: array and for loop

- here is some sample code that declares an integer array of 100 values and uses a for loop to *store* random numbers in the array:

```
int a[100]; // array
int i; // loop counter
for ( i=0; i<100; i++ ) {
  a[i] = rand();
} // end for
```

- once the data is stored in the array, we can do all kinds of stuff with it
- for example, let's print out the values in the array:

```
for ( i=0; i<100; i++ ) {
  cout << a[i] << endl;
} // end for
```


a complete example: computing the sum of the values in the array

```
#include <iostream>
using namespace std;
#include <time.h>
#include <stdlib.h>

int main() {
  int a[100];
  int i, sum;
  // store random values in array
  for ( i=0; i<100; i++ ) {
    a[i] = rand();
  } // end for
  // compute sum of values in array
  sum = 0;
  for ( i=0; i<100; i++ ) {
    sum += a[i];
  } // end for
  cout << "the sum of all the values in the array is: " << sum << endl;
} // end of main()
```