CS110 Review 2

Midterm Review

Rand()

Figure 2.15.1: The rand() function.

```
#include <iostream>
#include <cstdlib> // Needed to use rand()
using namespace std;

int main() {

   cout << "5 random numbers...\n";
   cout << rand() << "\n";
   cout << rand() << "\n";
   cout << rand() << "\n";
   cout << rand() << "\n";
   cout << rand() << "\n";

   return 0;
}
```

```
5 random numbers...
16807
282475249
1622650073
984943658
1144108930
```

The returned number is between 0 and RAND_MAX, inclusive, which is defined in cstdlib and whose value is system dependent but is at least 32767

- The pseudorandom number sequence can be changed by first calling the function ***srand()*** with an integer argument whose value is known as the seed
- each unique seed causing a different sequence of numbers to be returned from rand()
- A common seeding approach uses the current time, as returned by the ***time()*** function available in the time library

Figure 2.15.2: Using rand() to generate a random sequence in the range 1..2.

Using the time() function to seed the rand function causes different sequences for the two program runs.

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

int main() {

   cout << "1 means heads, 2 tails.\n";
   cout << "5 coin flips...\n";

   srand(time(0)); // "random" seed

   // rand() % 2 yields 0 or 1
   // so + 1 makes that 1 or 2
   cout << ((rand() % 2) + 1) << " ";
   cout << ((rand() % 2) + 1) << " ";
   cout << ((rand() % 2) + 1) << " ";
   cout << ((rand() % 2) + 1) << " ";
   cout << ((rand() % 2) + 1) << "\n";

   return 0;
}
```

```
1 means heads, 2 tails.
5 coin flips...
1 1 2 2 1
...
1 means heads, 2 tails.
5 coin flips...
2 2 1 2 2
...
1 means heads, 2 tails.
5 coin flips...
2 2 1 2 1
```

type conversion

- conversion of one data type to another, such as an int to a double

?

- What does the term “implicit conversion” mean?
- What is the value of the expression **3.0 / 2**?

Type Casting

static_cast<type>(expression)

Figure 2.13.1: Example program casting floating-variable to int type.

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

int main() {
   double floatInput;
   int castedInt;

   cout << "Enter a floating-point number: "; cin >> floatInput;

   castedInt = static_cast<int>(floatInput);

   cout << "   " << floatInput << " casted as an int is: ";
   cout << castedInt << "\n\n";

   return 0;
}
```

```
Enter a floating-point number: 405.341
   405.341 casted as an int is: 405
...
Enter a floating-point number: -405.341
   -405.341 casted as an int is: -405
...
Enter a floating-point number: 0.1259
   0.1259 casted as an int is: 0
```

Numeric Data Types

Table 2.11.1: Integer numeric data types.

Definition	Size	Supported number range	Standard-defined minimum size
char myVar;	8 bits	-128 to 127	8 bits
short myVar;	16 bits	-32,768 to 32,767	16 bits
long myVar;	32 bits	-2,147,483,648 to 2,147,483,647	32 bits
long long myVar;	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	64 bits
int myVar;	32 bits	-2,147,483,648 to 2,147,483,647	*16 bits*

This table lists the sizes for numeric integer data types commonly used along with the minimum size for those data types defined by the language standard.

Numeric Data Types

Table 2.11.2: Floating-point numeric data types.

Definition	Size	Supported number range
float x;	32 bits	-3.4×10^{38} to $3.4 * 10^{38}$
double x;	64 bits	-1.7×10^{308} to $1.7 * 10^{308}$

Overflow

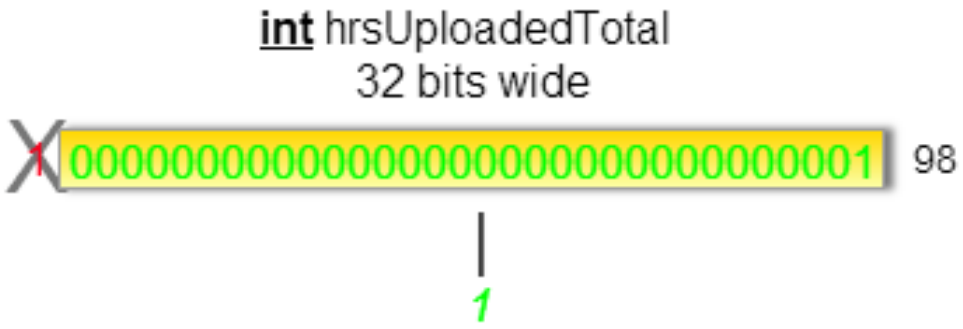
- What is **overflow**?

Overflow

- An integer variable cannot store a number larger than the range determined by the variable's data type

```
int hrsUploadedTotal;
...
hrsUploadedTotal = 4294967297;
```

100000000000000000000000000000001



```
int main() {
    int num = 1000;

    num = num * 100;
    cout << "num: " << num << endl;

    num = num * 100;
    cout << "num: " << num << endl;

    num = num * 100;
    cout << "num: " << num << endl;

    num = num * 100;
    cout << "num: " << num << endl;

    num = num * 100;
    cout << "num: " << num << endl;

    num = num * 100;
    cout << "num: " << num << endl;
```

Run

```
Compiling...done.
Running...done.
num: 100000
num: 10000000
num: 1000000000
num: 1215752192
num: 1316134912
num: -1530494976
```

Logical operators

- A ***logical operator*** treats operands as being true(1) or false(0), and evaluates to true or false
- Evaluates first condition then second

Logical operator	Description
&&	Logical AND, evaluating to true when both of its operands are true, else evaluating to false
||	Logical OR, evaluating to true if at least one of its two operands are true, else evaluating to false
!	Logical NOT, evaluating to true when its single operand is false, else evaluating to false

?

Evaluate the following logical expressions:

Assume `int age = 20; which are T/F?`

1) age > 30 && age < 50
2) age > 30 || age < 50
3) !(age < 50)
4) age / 10 == 2 && age <= 20

Assume int x = 5, y = 5, z = 6;

1) ! x == 5
2) ! x + y == 10

?

Express the following condition in a boolean expression

1) x is NOT an even number (must use ! operator)
2) x is a non-negative number less than 10
3) x is a 2-digit negative number
4) x is a uppercase letter (A – Z) or x is an underscore

Loops

While loop

```
while (expression) { // Loop expression
    /* Loop body: Sub-statements to execute if the
    expression evaluated to true */
}
/* Statements to execute after the expression evaluated to false */
```

while loop example:

Figure 4.1.1: While loop example: Face-printing program that ends when user enters 'q'.

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

/* Prints a face using user-entered character for eyes and mouth */

int main() {
  const char nose = '0'; // Looks a little like a nose
  char usr = '-';           // User-entered char
  while (usr != 'q') {
     // Print face
     cout << "\n " << usr << " " << usr << "\n";  // Eyes
     cout << "  " << nose << "\n";                // Nose
     cout << usr << usr << usr << usr << usr << "\n\n"; // Mouth

     // Get new character for eyes and mouth
     cout << "Enter a character ('q' to quit): ";
     cin >> usr;
  }

  cout << "\nGoodbye.\n\n";

  return 0;
}
```

```
 - -
  0
-----

Enter a character ('q' to quit): a

 a a
  0
aaaaa

Enter a character ('q' to quit): x

 x x
  0
xxxxx

Enter a character ('q' to quit): q

Goodbye.
```

For loop

```
for (initial_expression; condition_expression; update_expression) { // Loop expression
  /* Loop body: Sub-statements to execute if the
      expression evaluated to true */
}
/* Statements to execute after the expression evaluated to false */
```

```
for (int i = 1; i <= 10; ++i)
{
        cout << “the i has a value of” << i << endl;
}
```

Do-while loops

do-while loop

- A ***do-while loop*** is a loop construct that first executes the loop body's statements, then checks the loop condition.

```
do {
        /* Loop body statements */
} while ( /* Loop condition */ );
```

Compared to a while loop, a do-while loop is useful when the loop body should be executed *at least one time*.

```
int number = 1;
while (number < 0)
{
    cout << number << endl;
}
```

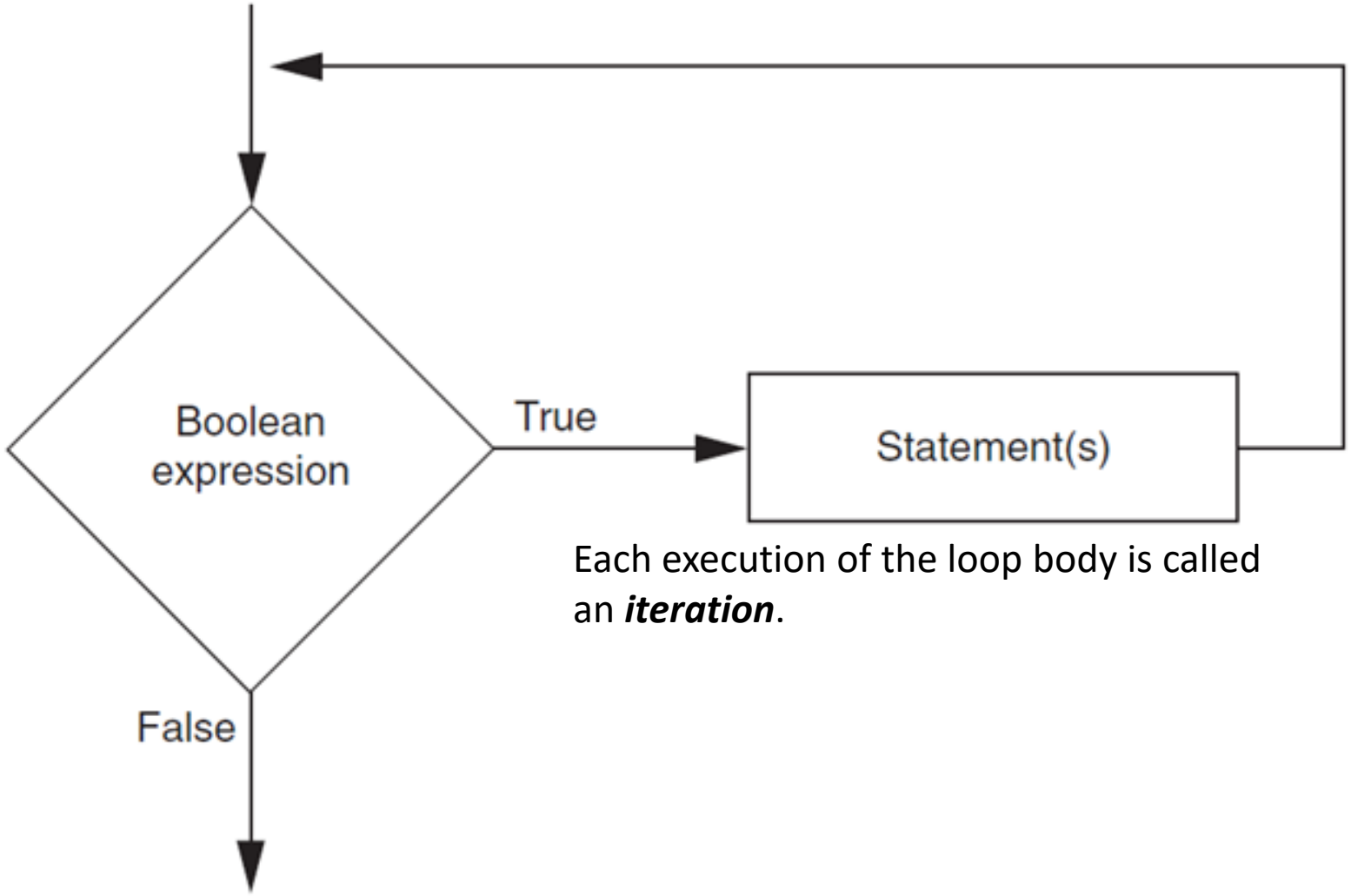
In the `while` loop, the `cout` statement will not execute

```
int number = 1;
do
{
    cout << number << endl;
} while (number < 0);
```

In the `do-while` loop, the `cout` statement will execute once

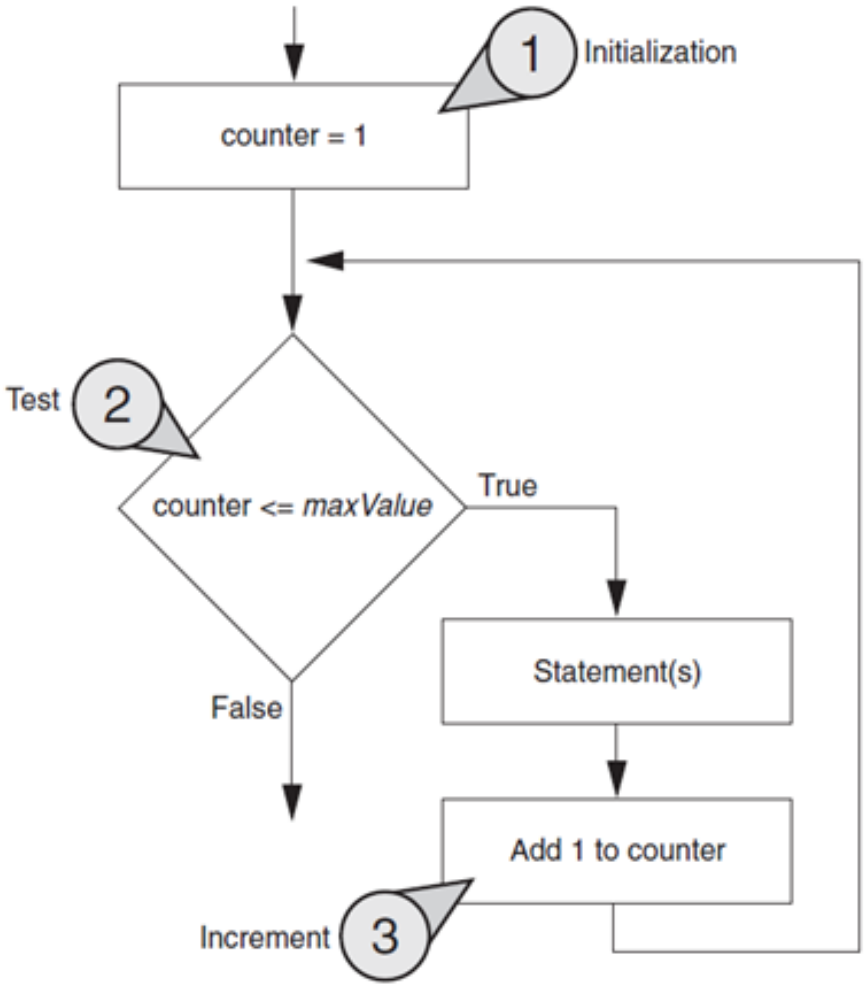
Loops, logic, evaluation and Decisions

While logic

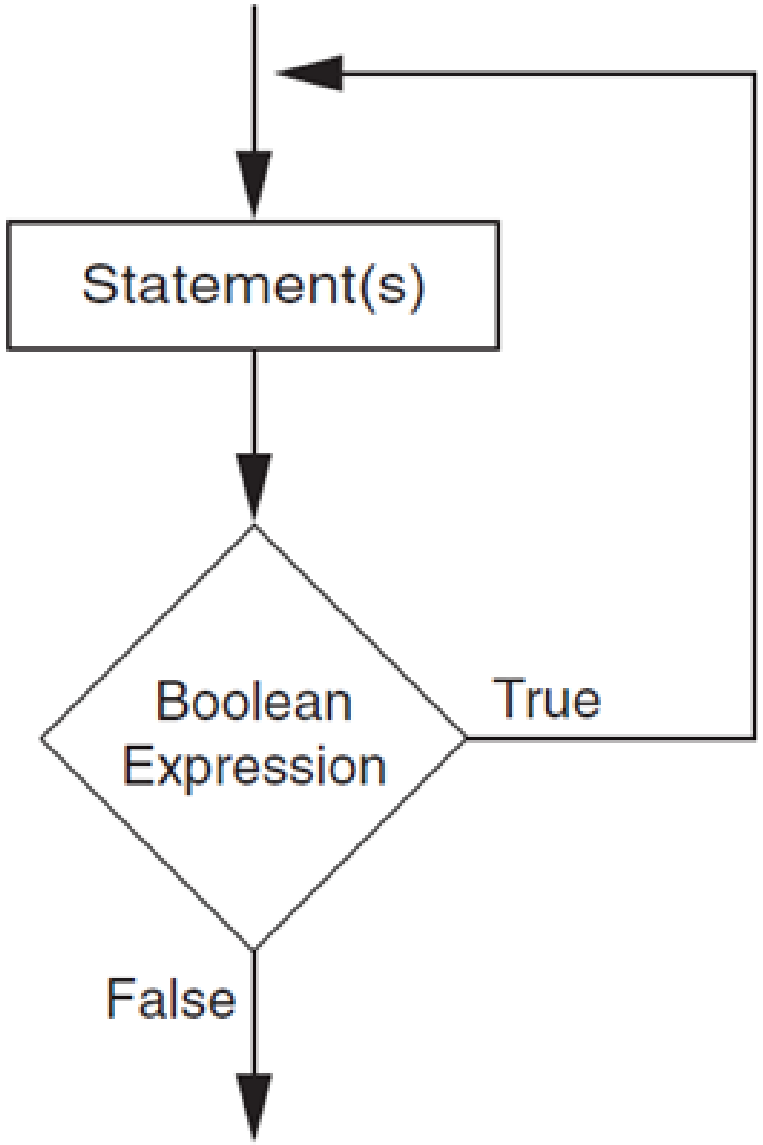


Each execution of the loop body is called an ***iteration***.

For loop logic



do-while loop



Nested loops

- What is the output of the following code?

```
int i1, i2;
i1 = 1;
while (i1 < 29) {
        i2 = 3;
        while (i2 < 7) {
                cout << i1 << i2 << " ";
                i2 = i2 + 3;
        }
        i1 = i1 + 10;
}
```

Switch statements

int or char
(not floating-point or string)

```
switch (expression) {
  case const_expr:
     // statements
     break;

  case const_expr:
     // statements
     break;
  ...
   default: // If no other case matches
     // statements
}
```

literals
(not variables)

Figure 3.5.1: Switch example: Number spelling.

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

int main() {
   int userNum;

   cout << "Enter a number between 0 and 5: ";
   cin >> userNum;

   switch (userNum) {
      case 0:
         cout << userNum << " is spelled zero.\n";
         break;
      case 1:
         cout << userNum << " is spelled one.\n";
         break;
      case 2:
         cout << userNum << " is spelled two.\n";
         break;
      case 3:
         cout << userNum << " is spelled three.\n";
         break;
      case 4:
         cout << userNum << " is spelled four.\n";
         break;
      case 5:
         cout << userNum << " is spelled five.\n";
         break;
      default:
         cout << "Uh-oh! Invalid number. Please try again.\n";
         break;
   }

   return 0;
}
```

```
Enter a number between 0 and 5: 4
4 is spelled four.
...
Enter a number between 0 and 5: 0
0 is spelled zero.
...
Enter a number between 0 and 5: 7
Uh-oh! Invalid number. Please try again.
```

Boolean data type

- ***Boolean*** refers to a quantity that has only two possible values, ***true*** and ***false***.
- The language has the built-in data type ***bool*** for representing Boolean quantities.

?

What is stored in variable isFamous after executing the following statements?

```
bool isTall = true;
bool isRich = true;
bool isFamous = false;
if (isTall && isRich)
{
        isFamous = true;
}
```

Other things to keep in mind

- Expressions
- Order of operations
- Data types
- Variable Naming Conventions
- If/else statements
- Binary