Lecture 10: for, do, and switch

PIC 10A
Todd Wittman

Recall the while Loop

- The while loop has the general form

```
while ( boolean condition )  {
        **STATEMENTS**
}
```

- The while loop is like a repeated if statement.
- It will repeat the statements within the braces { } as long as the boolean condition is true.
- But to run the loop the first time, you have to make sure that boolean condition starts out true.
- Sometimes we want to always execute the loop at least once.

Sec 3.8: The do loop

- The do loop (also called do-while loop) checks the boolean condition at the end of the statements.

```
do {
        **STATEMENTS**
} while ( boolean condition );
```

- The do loop will always run at least once.
- Note the semi-colon after the while. (In the standard while loop, that semi-colon would have caused big problems.)

The Tic-Tac-Toe Game

- We use a do loop instead of a while loop if we want to run the code at least once.
- For example, in tic-tac-toe we want to play a first game and then ask the user to play again.
- For the while loop, we had to make sure the boolean condition was initially true by initializing response = "y";
- The do loop doesn't need to initialize the response string.

```
string response = "y";
while (response == "y") {
   **Play Tic-Tac-Toe**
   cout << "Again? (y/n)";
   cin >> response;
}
```

```
string response;
do {
        **Play Tic-Tac-Toe**
        cout << "Again? (y/n)";
        cin >> response;
} while (response == "y");
```

Sec 3.7: The for loop

- Often we want to run a loop a fixed number of times.
- In a while loop, we create a *counter*. Usually we use a single letter, like i.

```
i = start;
while ( i <= end ) {
        **STATEMENTS**
        i++;
}
```

- This same loop can be written as a for loop.


```
for (int i = start;  i <= end;  i++)  {
        **STATEMENTS**
}
```

The for Loop

- The general form of a for loop is:

```
for (initalization; stopping condition; update) {
    **STATEMENTS**
}
```

- Note the semi-colons in between the 3 pieces.
- The initialization and update should be able to stand on their own as lines of C++ code.
- The initialization could create the counter, or use an existing one.

```
int i = 0;    j = 0;    double starting_point = 100.0;
```

- The stopping condition is any boolean statement, but usually a < or > condition with the counter.

```
i <= 100;     i > 0;     isInCircle(i,r,center);
```

- If the stopping condition is more exotic than a number comparison (i<=100), better to use a while loop.
- The update could be any change to the counter.

```
i++      i --      i=i+2      i=i+0.1
```

Some Simple for loops

- Ex Print out the odd number 1 through 99.

```
for (int i = 1; i <= 99; i = i+2)  {
        cout << i << " ";
}
```

- Ex Count down from 100 to 1.

```
for (int i = 100; i >= 1; i --)  {
        cout << i << " ";
}
```

- Ex Count down from an integer specified by the user.

```
int x;
cin >> x;
for (int i = x; i >= 1; i--) {
        cout << i << " ";
}
```

When to Start and Stop

- You have to be careful setting your upper and lower bounds for your counter.
- A very common error is to run your loop one too many or one too few times.
- Ex1: How many hellos are printed?

```
for (int i = 1; i < 5; i ++)  {
        cout << "Hello Frodo!\n";
}
```

- Ex2: How many hellos are printed?

```
for (int i = 0; i <= 5; i ++)  {
        cout << "Hello Frodo!\n";
}
```

Don't Mess With the Counter

- It's *very bad* style to change the counter's value in the body of the for loop.
- Ex1 How many times do we say hello?

```
for (int i = 1; i <= 10; i++) {
        cout << "Hello Gollum!\n";
        if (i >= 5)
                i = i+3;
}
```

- Ex2 How many times do we say hello?

```
for (int i = 1; i <= 10; i++) {
        cout << "Hello Gollum!\n";
        if (i >= 5)
                i --;
}
```

When to use what...

- We use a for loop when we know *ahead of time* exactly how many times we want to run our loop.
- The for loop is for fixed # of iterations.

- Ex Play Tic-Tac-Toe after the user says yes.
  Use a while loop.
- Ex Play Tic-Tac-Toe once, then ask the user if she wants to play again.
  Use a do loop.
- Ex Play Tic-Tac-Toe 100 times. *(Do not ask for user response.)*
  Use a for loop.

The Factorial

- Recall that the factorial of a integer N≥1 is
  N! = N*(N-1)*(N-2)*...*3*2*1
- e.g. 5! = 5*4*3*2*1 = 120
- Ex Write code that compute the N! in 3 different ways:
  a.) Use a while loop.
  b.) Use a do loop.
  c.) Use a for loop.

The while Factorial

```
//Compute N! with a while loop.
    int fact = 1;
    while (N >= 1) {
          fact *= N;
          N--;
    }
```

The do Factorial

```
//Compute N! with a do loop.
    int fact = 1;
    do {
          fact *= N;
          N--;
    } while (N>=1);
```

The for Factorial

```
//Compute N! with a for loop.

    int fact = 1;
    for (int i = N; i >= 1; i--) {
          fact *= i;
    }
```

Note we could omit the braces { } here, because the for loop body is just one line.
Good style to include them anyways.

Sec 3.8: Nested Loops

- Suppose we have two we want to iterate over two dimensions.
- For example, print out a rectangular table of (row,column) coordinates.

m=3 rows

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)

n=4 columns

- We can accomplish this with a nested for loop.

Creating a Table of Values

- We want a table with m rows and n columns.

```
int m, n;
cout << "Enter number of rows: ";
cin >> m;
cout << "Enter number of columns: ";
cin >> n;
for (int row=1; row <= m; row++)  {
   for (int col=1; col <= n; col++)
       cout << "(" << row << ", " << col<< ")    ";
   cout << "\n";
}
```

Creating a Triangle of Values

- OK hotshot, can you modify the last program to create a triangle of values?

(1,1)
(2,1) (2,2)
(3,1) (3,2) (3,3)

- Note the # rows must equal the # columns.
- Suppose we want to create a NxN triangle.

Creating a Triangle of Values

- We want a NxN triangle of coordinates.

```
int N;
cout << "Enter size of triangle: ";
cin >> N;
for (int row=1; row <= N; row++)  {
   for (int col=1; col <= row; col++) {
         cout << "(" << row << ", " << col<< ")    ";
   }
   cout << "\n";
}
```

Sec 3.3: Multiple Alternatives

- We can check several cases with an if/else sequence.
- But you have to be careful.

```
if (first_name == "Frodo")
        if (last_name == "Baggins")
                cout << "Your name is Frodo Baggins.\n";
else
        cout << "Your first name is not Frodo."
```

- The indentation is misleading. What it actually does is this:

```
if (first_name == "Frodo")
        if (last_name == "Baggins")
                cout << "Your name is Frodo Baggins.\n";
        else
                cout << "Your first name is not Frodo."
```

- Should have added braces { } around the second if.

The switch Statement

- We can replace a sequence of if/else statements with a switch statement.

```
switch (integer_variable)  {
        case value1:
                **STATEMENTS**
                break;
        case value2:
                **STATEMENTS**
                break;
        case ...
        default:
                **STATEMENTS**
                break;
}
```

This is equivalent to:

```
if (integer_variable == value1)  {
        **STATEMENTS**
}
else if (integer_variable == value_2)  {
        **STATEMENTS**
}
else if ...

else {
        **STATEMENTS**
}
```

- Don't forget the break. Compiler won't catch it. Gives a strange "fall-through" run-time error.

The switch Statement

- You can check multiple values by stacking the cases:

```
if (x == 1 || x==3) {
   cout << x;
   x = x+2;
}
else
   cout << 2*x;
```

```
switch (x) {
        case 1:
        case 3:
                cout << x;
                x = x+2;
                break;
        default:
                cout << 2*x;
                break;
}
```

Example: A Simple Calculator

- The switch statement is particularly good for setting up menus.
- Suppose we want a basic calculator.

```
Enter two numbers: 2 3
1.) Add      2.) Multiply     3.) Divide
Choice:  3
0.6667
```

Example: A Simple Calculator

After you set the I/O to get integers x & y, the switch statement can implement your decision nicely.

```
int choice;
cin >> choice;
switch (choice)  {
      case 1:   //Add
                cout << x + y;
                break;
      case 2:   //Multiply
                cout << x * y;
                break;
      case 3:   //Divide
                cout << (double) x / y;
                break;
      default:  //User entered invalid number.
                cout << "Invalid choice";
                break;
}
```

```
int choice;
cin >> choice;
if (choice == 1)
      cout << x + y;
else if (choice == 2)
      cout << x * y;
else if (choice == 3)
      cout << (double)x/y;
else
      cout << "Invalid choice";
```

What would this program do if we forget the breaks?

The switch Statement

- Unfortunately, the switch statement only works on int and char data types.

```
string response;
cin >> response;
switch (response)   {    // GIVES A COMPILE ERROR!
          case "yes":
```

- But we could handle a single letter response with a char.

```
char response;
cin >> response;
switch (response)  {
          case 'y':
          case 'Y':
                    cout << "You said yes.\n";
                    break;
          default:
                    cout << "You said something else.\n";
                    cout << "You jerk.\n";
                    break;
}
```

Note we can stack the cases to pick up more than one value for response.