PIC 10A

Lecture 10: `do while` and `for` loops

do - while

Syntax:

```
do
{
   /* code */
} while( boolean expression);
```

Main difference between while and do loop is that the do loop runs always at least once.

The condition is checked after the code is run.

Example

```
string response;

cout << "Do you want to play?";
cin >> response;

while (response == "Y" || response =="y")
{

   /* code for a game */

   cout << "Do you want to play?";
   cin >> response;


}
```

Example

```
string response;

do
{

   /* code for a game */

   cout << "Do you want to play?";
   cin >> response;


}  while (response == "Y" || response =="y");
```

`for` loop

We have seen that in many of our examples with loops we had a counter that counted how many times we have looped.

In a for loop the counter is central part of its syntax.

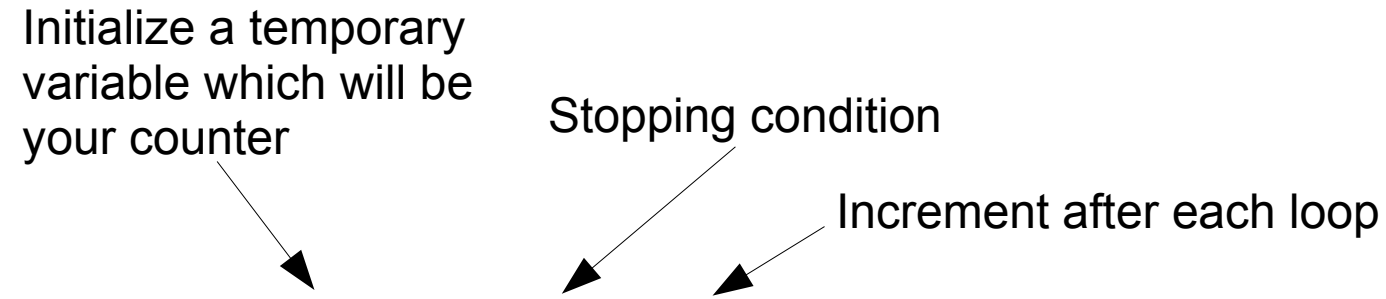
```
for (int i=0; i < 100; ++i)
{

/* code */

}
```

`for` loop

In the for loop we have a few possible options, but most common way of using the for loop is as in this example:




```
for (int i=0; i < 100; ++i)
{

/* code */

}
```

`for` loop example

```
for (int i = 1; i < 100;  ++i)
{
  cout << "Wow! This is the " << i <<
  "th line of output\n";
}
```

for loop basics

The syntax of a for loop is:

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

You should use a for loop when you know in advance that your loop should run a fixed number of times

If your stopping condition is more complicated than a number comparison (eg. i <100) you should use a while loop instead

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.

for loop basics

Usually initialization declares a new variable to be the counter, but you could use a variable that already exists as well.

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

The update could be any change to the counter.

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

Any of the components of the for loop (initialization, stopping condition or update) can be omitted but you still need to have 3 semi-colons.

```
for ( ; counter < 100 ; counter+=2)
{
 // code
}
```

Examples

Example

```
for (int i=1000; i > 0; i--)
    cout << i << " bottles of beer on the wall\n";
```

Example

```
for (int i = 100; i > 0 ; i--)
{
    cout << "i is now " << i << "\n";
    cout << "Weee! I am counting backwards!\n";
}
```

Examples

Example

```
for (int i=0; i < 100; i=i+2)
{
    cout << i << "\n";
}
```

Example

```
int number;
cout << "Gimme a starting number ";
cin >> number;

for ( ; number < 1000; number++)
{
    cout << number << "\n";
}
```

Starting and stopping

You have to be careful when picking the starting value and end condition.

How many times does each of these loops get executed?

```
for (int i=1; i<10 ; i++)
{
...
}

for (int i=0; i<10 ; i++)
{
...
}

for (int i=1; i<=10; i++)
{
...
}
```

How many times does it run?

Consider a more general kind of for loops:

```
for (int i=a; i < b; i+=c)
{
// code
}
```

This loop runs floor((b-a)/c) times.

```
for (int i=a; i <= b; i+=c)
{
// code
}
```

This code runs floor((b-a)/c + 1) times

Changing the counter inside a loop

It is considered very bad programming style to change the counter inside a for loop.

```
for (int i=1; i < 10; i++)
{
    if (i >5)
        i+=2;

    // code
}
```

How many times does the above code run?

You should be able to see how many times the loop runs by looking at the initialization, the stopping condition and the update. If this is not the case then you should consider changing your program.

Example

Example:

How many times does this loop run?

```
for (int i = 1; i <= 10; i++)
{
    cout << "Mmmm...Donut...\n";
    if (i >= 5)
        i--;
}
```

What is wrong with these loops?

```
for (double answer = .33333; answer != 1; answer +=.33333)
{
// Code
}

for (int i=0; lets_play==true; ++i)
{
// Code
}

//Assume s1 and s2 are strings
for (int i=17; i > s1.length(); i--)
{
s1.erase(0,1);
cin >> s2;
s1+=s2;
}
```

What loop to use?

- When you know ahead of time that your loop will run a fixed number of times, use the **for loop**.
- When your loop is not going to run a fixed number of times or the stopping condition is anything more complicated than a simple comparison, use a **while loop**.
- If your loop is always going to run at least one time, use a **do loop**.

What is the best loop?

1) Play Tic-Tac-Toe after the user says yes.

2) Play Tic-Tac-Toe once, then ask the user if she wants to play again.

3) Play Tic-Tac-Toe 100 times. (Do not ask for user response.)

Factorial

Recall that N! is 1 * 2 * 3 * ... * (N-1) * N.

For example, 5! = 1 * 2 * 3 * 4 * 5 = 120.

We will write a program to compute N! in three different ways.

Factorial example

1) while loop:

```
int n, number, factorial=1;
cout << "Please give me a number. ";
cin >> number;
n = number;

while (n > 1)
{
    factorial = factorial * n; // (or factorial *=n;)
    n--;
}

cout << number << " factorial is: " << factorial << "\n";
```

Factorial example

2) do-while loop:

```
int n, number, factorial=1;
cout << "Please give me a number. ";
cin >> number;
n = number;

do
{
    factorial = factorial * n; // (or factorial *=n;)
    n--;
} while (n > 1);

cout << number << " factorial is: " << factorial << "\n";
```

Factorial example

3) for loop:

```
int number, factorial=1;
cout << "Please give me a number. ";
cin >> number;

for (int i = number; i > 1; i--)
{
   factorial = factorial * i;
}


cout << number << " factorial is: " << factorial << "\n";
```

Nested loops

We have already seen an example of a nested loop:

```
int x=1;
int y=1;

while ( x <= 10)
{
    while (y <= 10)
    {
        cout << x << " X " << y << " = " << x*y << "\n";
        y++;
    }
    y=1;
    x++;
}
```

Nested for loop

```
for (int x = 1;  x <= 10; ++x)
{
    for (int y = 1; y <= 10; ++y)
    {
        cout << x << " X " << y << " = " << x*y << "\n";
    }
}
```

When we know how many times each loop runs, nested loops are particularly easy to do with a for loop.

Nested loop example

We want to output a triangle made of blocks [ ]. See the picture below. When the user enters the height of the triangle, for example 4, the computer will write to the console a triangle with 4 rows made of the blocks [ ].

[ ]
[ ][ ]
[ ][ ][ ]
[ ][ ][ ][ ]

Triangle example cont.

We can start thinking how to accomplish this with nested loops.

The outer loop can control how many rows are printed. If we want 4 rows the outer loop should run 4 times.

The outer loop will be responsible for writing a newline.

The inner loop can write the blocks. Lets say that each time the inner loop runs it draws a single block [ ]. It is clear then that the number of times the inner loop runs depends on what row we are printing!

Triangle example

```
int main()
{
   cout << "Enter number of rows: ";
   int n;
   cin >> n;

   for (int i = 1; i <= n; i++)
   {
      for (int j = 1; j <= i; j++)
      {
         cout << "[]";
      }
      cout << "\n";
   }

   return 0;
}
```

switch statement

```
int vehicle;
double toll;
cout << "Please give your vehicle type. Input 1 for car, 2 for bus, 3 for truck; "
cin >> vehicle;

switch(vehicle)
{
    case 1:
        cout << "Passenger car ";
        toll = 0.50;
        break;
    case 2:
        cout << "Bus ";
        toll = 1.50;
        break;
    case 3:
        cout << "Truck ";
        toll = 2.00;
        break;
    default:
        cout << "unknown vehicle class\n";
        toll = -1;
}
cout << " toll is: " << toll << "\n";
```

switch

Switch statement is most like an if-else if-else compound statement.

Limitation is that a single variable is always compared with == against the values in the case statements.

Another limitation is that the values must be integer or char values.

When the computer encounters a break; it leaves the switch block.

If a match is found but the break is omitted the computer will execute code under other case statements until it finds a break!!

Omitting the break statement is usually an error, but there are some situations where it makes sense.

Switch example

```
char letter;
cin >> letter;

switch(letter)
{
   case 'A':
   case 'a':
      cout << “Your letter is A”;
      break;
   case 'B':
   case 'b':
      cout << “Your letter is B”;

   ...

}
```