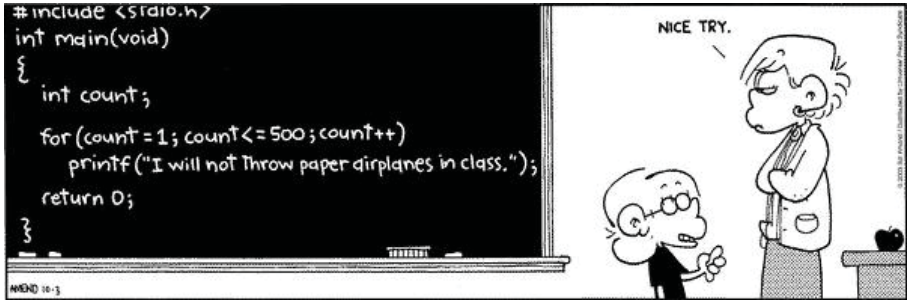
Slides #7

Loops

Chapter 5



CMPT 125/128

Topics

1) Is there a faster way to code x = x + 1; than x += 1?
2) While
3) Do While
4) For

Increment / Decrement Operators

Increment and Decrement

- Add 1 to x:
  Subtract 1 from x:
- Example: Equivalent:
  - `int index = 0;`
    `index++;`
  - `double sum = 100.0;`
    `sum--;`
- Prefix vs Postfix:
  - Prefix has the ++ the variable:
  - Postfix has the ++ the variable:

Prefix vs Postfix:

- 2 things happen with x++ or ++x (no particular order)
  -
  -
- Prefix (++x): increments, then gives you the value.
- Postfix (x++): gives you the value, then increments.
- Example:

```
int x = 1;
cout << x++;        // Same as: cout << x;
                    //          x = x + 1;
```

```
int y = 1;
cout << ++y;        // Same as: y = y + 1;
                    //          cout << y;
```

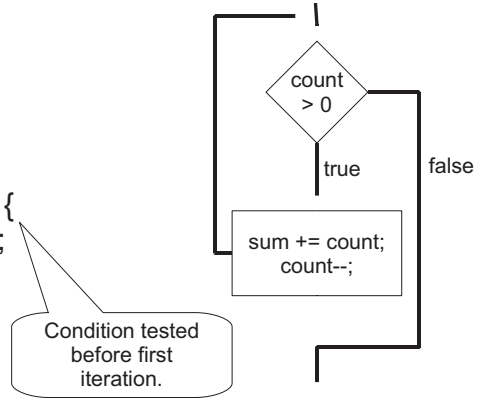

While Loops

while

- A while loop executes the body...

```
long sum = 0;
long count = 10;

while (count > 0) {
    sum += count;
    count --;
}
```



Infinite Loops

- Infinite Loop:
  - Example: a while loop with its condition always true.

```
while (true) {
    cout << “Still going.....”;
}
```

- or:

```
double index = 0;
while (index < 10) {
    cout << “Not done.”;
    index --;
}
```

More Infinite Loops

- Mystery Infinite Loop #1:

```
long i = 0;
while (i<10);
{
    cout << “Not done.”;
    i ++;
}
```

Loop runs the empty statement an infinite number of times.

- Mystery Infinite Loop #2:

```
long j = 0;
while ( j < 10 )
    cout << “Not done.”;
    j ++;
```

j++ is the first statement after the loop (not part of it).

Nested Loops

- You can nest all types of loops:

```
int i = 0;
while (i < 3) {
    int j = 0;
    while (j <= i) {
        cout << j;
        j++;
    }
    cout << endl;
    i++;
}
```


break and continue

- 2 special commands can be used inside a loop's body to control execution:
  - break:
  - continue:
    - Will re-evaluate the condition, and keep looping.
- Avoid these if possible:
  - They complicate how the loops execute.
  - Can be useful for handling error conditions.

Review

- What is the value of each variable, and what is printed to the screen?

```
int a = 10, b=10, c=10;
cout << a++ << endl;


cout << ++b << endl;


cout << --c << “ “
     << c-- << endl;
```

- What is printed to the screen?

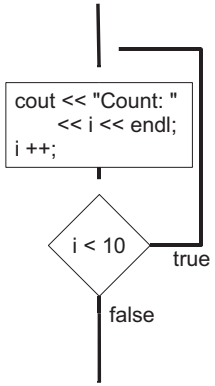
```
int i = 5;
while (i > 0) {
    cout << i;
    i -= 2;
}
```

Do While Loops

do Loops

- do loops are similar to while loops, but...

```
int i = 0;
do {
    cout << "Loop: " << i << endl;
    i ++;
} while (i < 10);
```



do vs while

- A do loop must be executed...
  whereas a while loop can execute...
- do Loop:

```
do {
    cout << "In DO loop." << endl;
} while (false)
```

- while loop:

```
while (false) {
    cout << "In WHILE loop." << endl;
}
```

- Total Output...

For Loops

for

- Definite Loop:
  - A loop where we know ...

    "Count from 1 to 10.."
- Indefinite Loop:
  - A loop where we...
    tell how many times we will execute the loop:
    "Count up from 1 to find first multiple of 3, 4 and 18"
- for loops are often useful to neatly organizing definite loops.

Simple Example

```
for (int i = 0; i < 5; i++)
{
    cout << "i: " << i << endl;
}
```

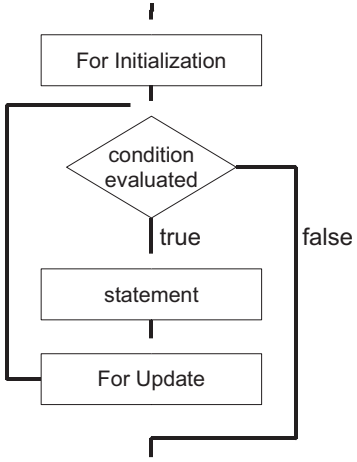
```
i: 0
i: 1
i: 2
i: 3
i: 4
```


Work Through Examples

```
for (int i = 0; i < 5; i++) {
    cout << i << endl;
}

for (int i = 100; i > 0; i -= 10) {
    cout << i << endl;
}

int i = 0;
for ( ; i < 100; ) {
    i *= 2;      // Not a good idea to
    i ++;        // change loop counter
}                // inside for loop.
```



Notes on for

- Variables declared in the for loop's initialization...

```
for (int i = 0; i < 10; i++) {
    cout << i << endl;
}
cout << i << endl;              // COMPILE ERROR
```

Summary

- Increment & Decrement:
  - Prefix: ++x, --x.
  - Postfix: x++, x--
- Loops done with:
  - while: Condition up front (pretest)
  - do: Condition at the end (post-test)
  - for: Best for a definite number of iterations (pretest)