Loops also called "repetition statements"

- **The While Statement**

  ```c
  while (condition)
  {
    statement;
  }
  ```

  **action:**
  1) condition is tested
  2) if the condition is true the statement is performed; if the condition is false, continue after the loop
  3) after completing the loop's body, go back to number 1

- **iteration** = one execution of the subsidiary statement
Trace

```c
int var = 1;
while (var < 5) {
    printf("%d\n", var);
    var = var + 1;
}
```

Infinite Loops

- The loop will **never** terminate on its own.
- In UNIX, to stop a program with an infinite loop
  - Control-c
  - there can be a delay

```c
int var = 1;
while (var < 5) {
    printf("%d\n", var);
}
```
The do/while loop

- Format:
  ```
  do {
    statements
  } while (condition);
  ```
- the curly braces are not required, but good style otherwise the line with "while" can be easily confused with the beginning of a while loop.
- Action:
  1) execute the body
  2) test the condition
  3) if the condition is true, go back to #1; if the condition is false, continue with the line after the loop

Same or Different?

```java
sum = 0;
do {
    j--;
    sum+=j;
} while (j > 0);
```

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

- Same or Different?
- Trace both with different starting values for j
**Types of repetition:**

- counter-controlled repetition
- indefinite repetition
- examples

**for loop**

- Useful for repeating loop body a fixed number of times
- syntax:
  
  ```
  for (expr1; expr2; expr3) 
  statement;
  ```

- Each of the three expressions is optional.
- Semicolons and parentheses are required.
- Typically:
  - `expr1` initializes
  - `expr2` is condition
  - `expr3` updates loop control variable
- Action:
  1) if present, perform `expr1`
  2) if present, `expr2` (condition) tested
     a) if false, continue to line after the loop
     b) if true (or omitted), continue with step (3)
  3) the subsidiary statement, or loop body is executed
  4) if present, `expr3` executed
  5) go back to step (2)
trace examples

- for (v= 1; v < 5; v++)
  printf("%d %d\n", v, v * v);

- for (v= 5; v > 0; --v)
  printf("%d %d\n", v, v * v);

More details about the three Expressions

- can initialize to any value
- can do loop control updates other than by one
- can do loop control updates that are negative
- expression 1 and 3 can have multiple expressions
  - connected by the comma operator
- expression 2 can have multiple boolean expressions
  - connected by logical operators therefore building a single boolean expression
**The Comma Operator**

- To put several expressions in a place where one expression can appear
- Makes a single expression out of any number of individual ones
  - Value returned (and its type) is last expression’s
  - Evaluated left to right
- \[ x = y \times z, \ 4.5, \ 6; \]

- More useful example:
  ```c
  for (a= 0, b= 10; a != b; a++, b--)
      printf("%d %d\n", a, b);
  ```

---

**One of the for loop expressions missing**

- **Expression 1 missing**
  ```c
  for ( ; a != b ; a++, b--)
      printf("%d %d\n", a, b);
  ```

- **Expression 2 missing**
  ```c
  for (a=1,b=2;; a++,b--)
      printf("%d %d\n", a, b);
  ```

- **Expression 3 missing**
  ```c
  for (a=1,b=2; a != b ;)
      printf("%d %d\n", a++, b--);
  ```
**Nested Loops**

- Follow the same procedure - just view each loop as its own statement following the action rules for that type of loop.

```c
for (a= 3; a > 1; a--) {
    b= 4;
    while (b > 1) {
        printf("%d %d\n", b, a);
        b--;
    }
}
```

Not always completely independent

- **Inner Loop Dependant On Outer Loop**

```c
a= 1;
while (a < 4) {
    b= a;
    while (b <= 4) {
        printf("X");
        b++;
    }
    printf("\n");
a++;
}
```

- Inner loop’s termination depends on the outer loop control variable

```c
a= 1;
while (a < 4) {
    b= 1;
    while (b <= a) {
        printf("%d\n", b);
        b++;
    }
    a++;
}
```
break and continue

- break causes loop to immediately quit
  - Exits only from innermost nested loop (in which it appears)

- continue skips rest of a loop body & begins next iteration
  - while, do-while
    - jumps immediately to testing loop termination condition
  - for loops
    - jumps to third expression in for loop header

- VERY IMPORTANT: break and continue should ONLY be used in loops when they improve a program's clarity

Most Common Errors

- Forgetting to modify the variable tested by the condition – result = infinite loop

- Fencepost error – result = one too many or one too few iterations

- The null statement – result = infinite loop
  ```c
  int j = 3;
  while (j < 3);
  printf("%d\n", j++);
  ```