Programming in C

Repetition/Looping

Repetition Repetition

Repetition Repetition

Repetition Repetition

Repetition Repetition …

Example 1

```
// Read two integers and print sum
int num1, num2, sum;

scanf("%d %d", &num1, &num2);
sum = num1 + num2;
printf("%d + %d = %d\n", num1, num2, sum);
```

- What if we want to process three different pairs of integers?

Example 2

- One solution is to copy and paste the necessary lines of code. Consider the following modification:

```
scanf("%d %d", &num1, &num2);
sum = num1 + num2;
printf("%d + %d = %d\n", num1, num2, sum);

scanf("%d %d", &num1, &num2);
sum = num1 + num2;
printf("%d + %d = %d\n", num1, num2, sum);

scanf("%d %d", &num1, &num2);
sum = num1 + num2;
printf("%d + %d = %d\n", num1, num2, sum);
```

- What if you wanted to process four sets? Five? Six? ….

Processing an arbitrary number of pairs

- We might be willing to copy and paste to process a small number of pairs of integers but
- How about 1,000,000 pairs of integers?
- The solution lies in mechanisms used to control the flow of execution
- In particular, the solution lies in the constructs that allow us to instruct the computer to perform a task repetitively

Repetition (Looping)

- Use looping when you want to execute a block of code several times
  - Block of code = Body of loop
- C provides three types of loops
  - *while* statement
    - *Most flexible*
    - *No 'restrictions'*
  - *for* statement
    - *Natural* 'counting' loop
  - *do-while* statement
    - *Always executes body at least once*

The while Repetition Structure

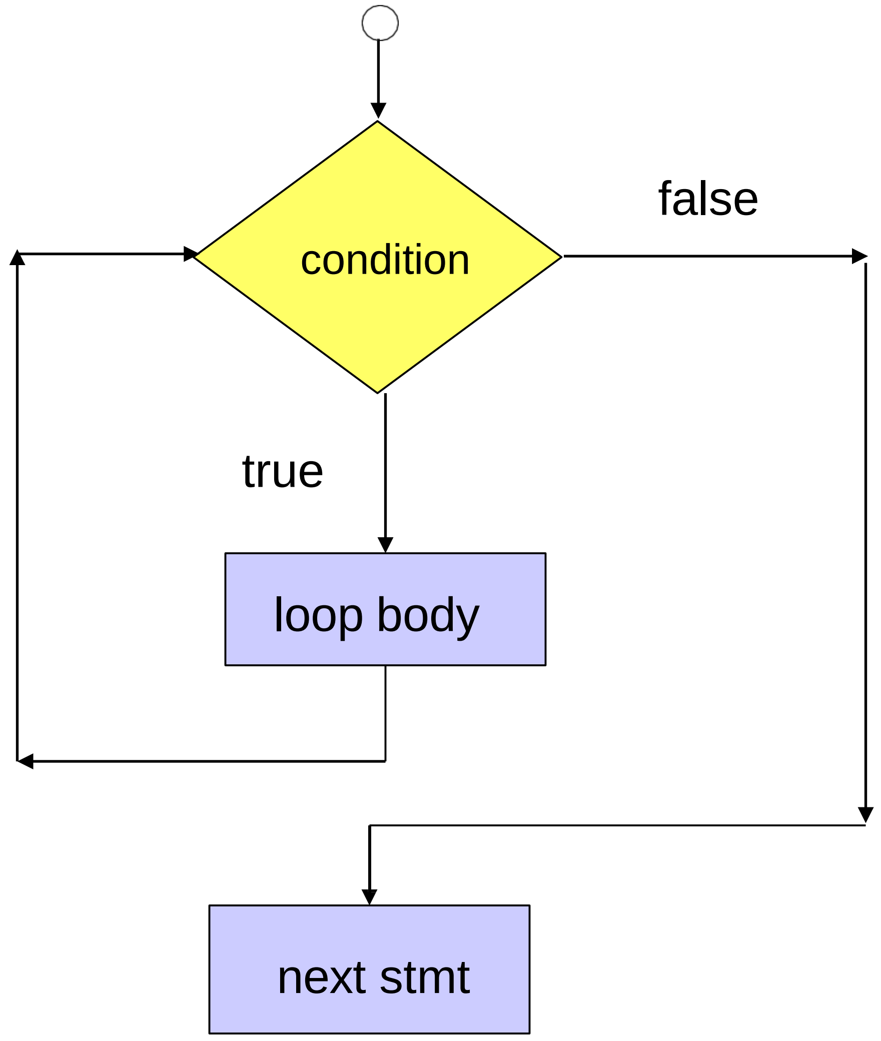
- Repetition structure
  - Programmer specifies
    - Condition under which actions will be executed
    - Actions to be repeated
  - Psuedocode

```
While there are more items on my shopping list
    Purchase next item and cross it off my list
```

- **`while`** loop repeated
  - As long as condition is true
  - Until condition becomes false

The while Repetition Structure

- The condition is tested
- If the condition is true, the loop body is executed and the condition is retested.
- When the condition is false, the loop is exited.



The while Repetition Structure

- Syntax:

```
while (expression)
      basic block
```

- Expression = Condition to be tested
  - Resolves to true or false
- Basic Block = Loop Body
  - Reminder – Basic Block:
    - Single statement or
    - Multiple statements enclosed in braces

Loop Control Variable (LCV)

- The loop control variable is the variable whose value controls loop repetition.
- For a while loop to execute properly, the loop control variable must be
  - declared
  - initialized
  - tested
  - updated in the body of the loop in such a way that the expression/condition will become false
    - If not we will have an endless or infinite loop

Counter-Controlled Repetition

- Requires:
  1. Counter variable , LCV, initialized to beginning value
  2. Condition that tests for the final value of the counter (i.e., whether looping should continue)
  3. Constant increment (or decrement) by which the control variable is modified each time through the loop
- Definite repetition
  - Loop executes a specified number of times
  - Number of repetitions is known

Example 3

```
int count;                    // LCV: Loop Control Variable
int num1, num2, sum;

count = 0;                    // 1. Initialize LCV
while (count < 5) {           // 2. Test LCV
   scanf("%d %d", &num1, &num2);
   sum = num1 + num2;
   printf("%d + %d = %d\n", num1, num2, sum);
   count = count + 1;         // 3. Increment LCV
}
```

EXECUTION CHART

count	count<5	repetition
1	true	1
2	true	2
3	true	3
4	true	4
5	true	5
6	false	

Loop Pitfalls

```c
// Echo numbers entered back to user
printf("Enter number or zero to end: ");
scanf("%d", &num);
while (num != 0);
{
    printf("Number is %d\n\n", num);
    printf("Enter another number or zero to end: ");
    scanf("%d", &num);
}
```

```
Enter value or zero to end: 2
```

What is wrong with my program? It just sits there!

Loop Pitfalls: Misplaced semicolon

```
// Echo numbers entered back to user
printf("Enter number or zero to end: ");
scanf("%d", &num);
while (num != 0);
{
    printf("Number is %d\n\n", num);
    printf("Enter another number or zero to end: ");
    scanf("%d", &num);
}
```

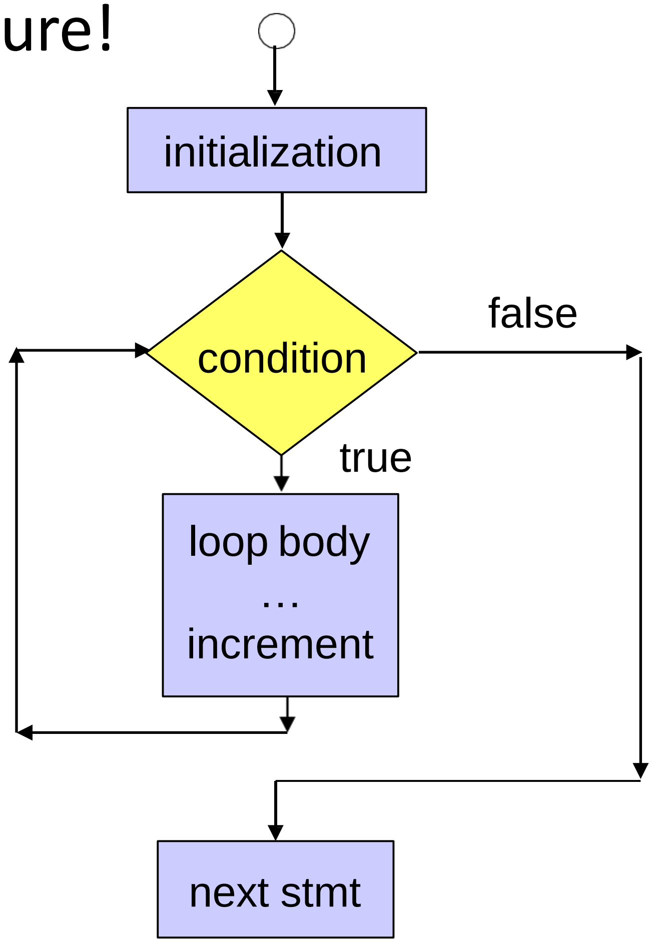
Do not place semi-colon here!

- Notice the ';' after the while condition!
  - Body of loop is between) and ;
- Result here: **INFINITE LOOP!**
  Ctrl-c = Kill foreground process

The for Repetition Structure

- A natural 'counting' loop
- Steps are built into for structure!

1. Initialization
2. Loop condition test
3. Increment or decrement



Review: Assignment Operators

- Statements of the form

  variable = variable *operator* expression;

  can be rewritten as

  variable *operator=* expression;

- Examples of assignment operators:

```
a += 5        (a = a + 5)
a -= 4        (a = a - 4)
b *= 5        (b = b * 5)
c /= 3        (c = c / 3)
d %= 9        (d = d % 9)
```

Review: Pre-increment operator

Pre-increment operator: ++n

i) Stand alone: add 1 to n

If n equals 1, then after execution of the statement

```
++n;
```

the value of n will be 2.

ii) In an expression:

Add 1 to n and then use the new value of n in the expression.

```
printf("%d", ++n);
```

If n is initially 1, the above statement will print the value 2.

After execution of printf, n will have the value 2.

Review: Post-increment operator

Pre-increment operator: n++

i) Stand alone: add 1 to n

If n equals 1, then after execution of the statement

```
n++;
```

the value of n will be 2.

ii) In an expression:

Use the value of n in the expression and then add 1 to n.

```
printf("%d", n++);
```

If n is initially 1, the above statement will print the value 1 and then add 1 to n. After execution, n will have the value 2.

Pre- and Post-decrement operator

- Pre- and post-decrement operators, --n, n-- , behave in a similar manner
- **Use caution when using in an expression**
  - Do not use unless you know what you are doing!

The *for* Repetition Structure

- Syntax:

```
for (initialization; test; increment)
       basic block
```

for loop example

- Prints the integers from one to ten

```
int counter;
for (counter = 1; counter <= 10; counter++)
{
   printf("%d\n", counter);
}
```

```
int counter;
counter = 1;
while (counter <= 10)
{
   printf("%d\n", counter);
   counter++;
}
```

for Loop Example

How many times does loop body execute?

```
int count;
for (count = 0; count < 3; count++) {
   printf("Bite %d  --   ", count+1);
   printf("Yum!\n");
}
```

for Loop Example

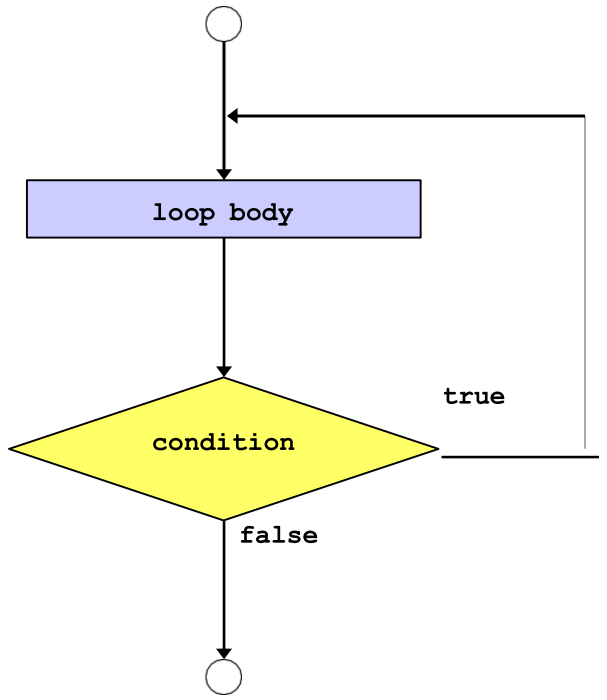
How many times does loop body execute?

```
int count;
for (count = 0; count < 3; count++) {
   printf("Bite %d  --  ", count+1);
   printf("Yum!\n");
}
```

```
Bite 1  --  Yum!
Bite 2  --  Yum!
Bite 3  --  Yum!
```

The do-while Repetition Structure

- The **`do-while`** repetition structure is similar to the **`while`** structure
  - Condition for repetition tested after the body of the loop is executed



All actions are performed at least once!

The do-while Repetition Structure

- Syntax:

```
do {
    statements
} while ( condition );
```

The do-while Repetition Structure

- Example

```
int counter = 1;
do {
   printf("%d\n", counter);
   counter ++;
} while (counter <= 10);
```

- Prints the integers from **1** to **10**

The do-while Repetition Structure

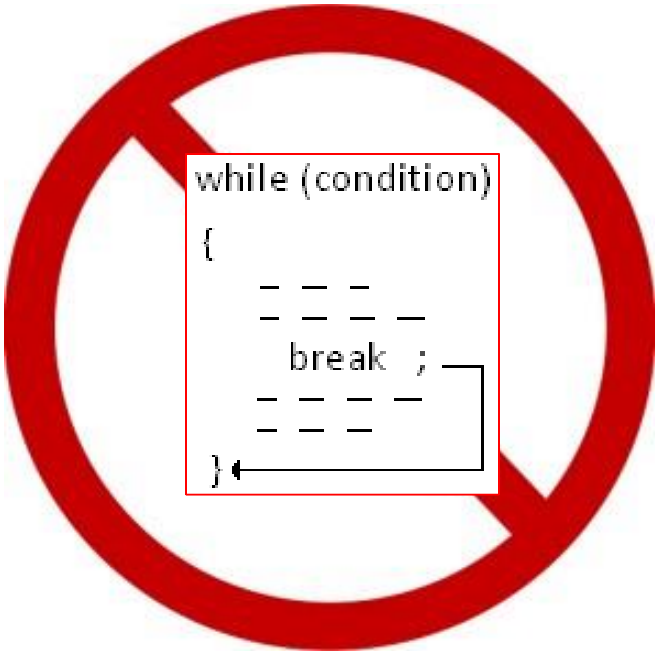
- Example

```
do {
    printf("Enter a positive weight: ");
    scanf("%d", &weight);
} while (weight <= 0);
```

● Makes sure that the user enters a valid weight

The break Statement

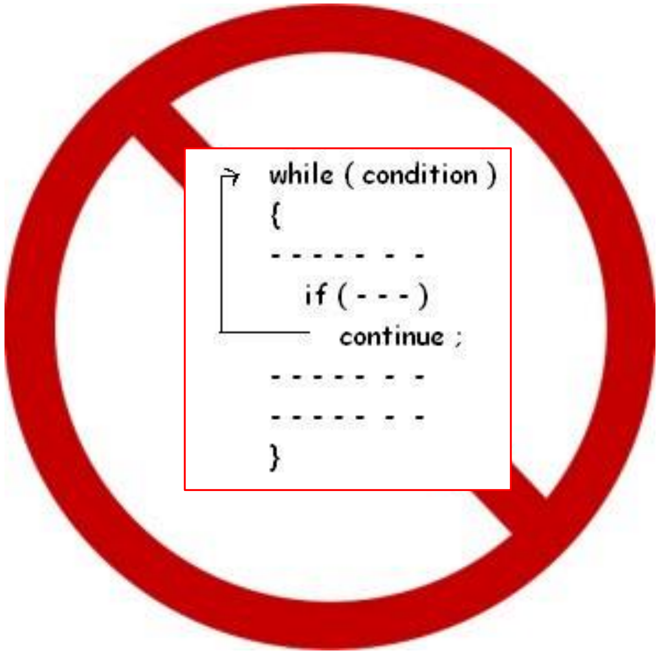
- **`break`**
  - Causes immediate exit from a **`while`**, **`for`**, **`do/while`** or **`switch`** structure
  - **We will use the break statement only to exit the switch structure!**



```
while (condition)
{
    _ _ _
    _ _ _ _
      break  ;
    _ _ _ _
    _ _ _
}
```

The continue Statement

- `continue`
  - Control passes to the next iteration
  - **We will not use the continue statement!**



```
while ( condition )
{
------ - -
   if ( - - - )
      continue ;
------ - -
------ - -
}
```

Programming in C

Repetition/Looping

_T H E   E N D_