

# **(6-1) Iteration in C**

## **H&K Chapter 5**

Instructor – Beiyu Lin

CptS 121 (May 20<sup>th</sup>, 2019)

Washington State University

# Iterative Constructs (1)

- Recall that algorithms are composed of three different kinds of statements:
  - Sequence: the ability to execute a series of instructions, one after the other.
  - Conditional: the ability to execute an instruction contingent upon some condition.
  - Iteration: the ability to execute one or more instructions repeatedly.



# Iterative Constructs (2)

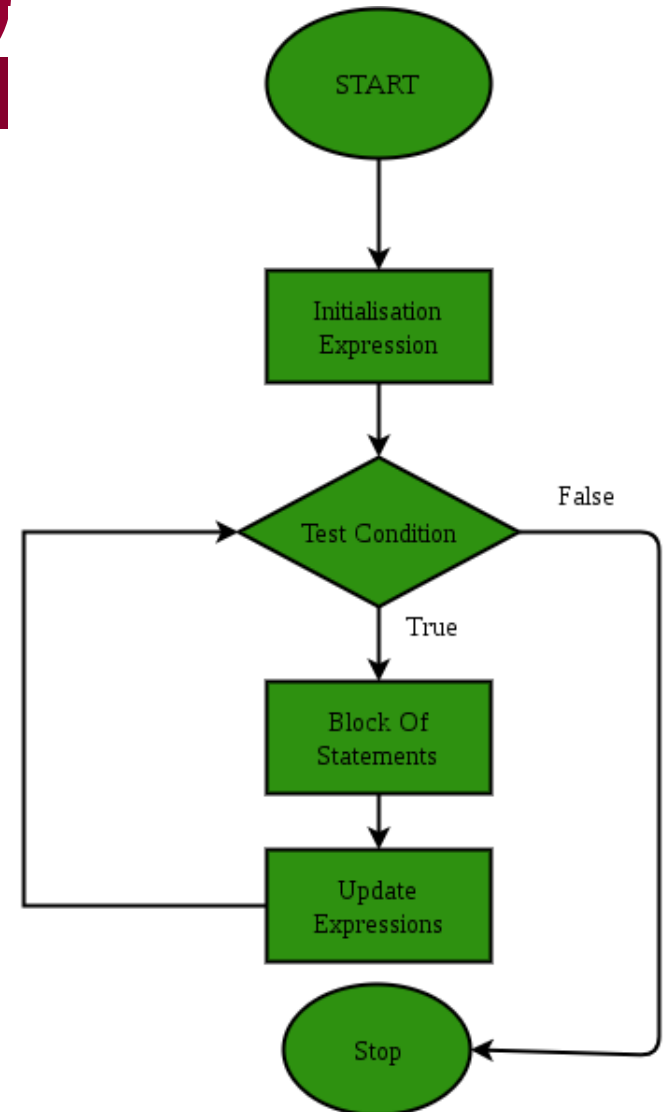
Count the number of people each day in the room Sloan 138 for the month of May:

- Only count on weekday
- For each weekday, only count at 11am



# Iterative Constructs (3)

- How to decide when a loop is needed  
Are any steps repeated?
  - No → No loop required
  - Yes → Do you know in advance how many steps are repeated?
    - No → Use a conditional loop
    - Yes → Use a counting loop



# Iteration Constructs (3)

- We'll discuss several loop patterns:
  - Counter loops
    - ( e.g. calculate a student's GPA based on 3 courses)
  - Conditional loops
    - (e.g. stop calculating GPA if the grade of a course < 80)
  - Sentinel-controlled loops
    - (e.g. calculate accumulated GPA as long as the tuition < \$12000)
    - Tuition = class1 \* credits1 + class2\*credits2 + .....
  - End-of-file controlled loops
    - (e.g. read to the end of the file)
  - Flag-controlled loops
    - (e.g. use a flag status)



# Iterative Constructs (3)

- Kinds of loops
  - *Counting loop* (`for` or `while`): executes a fixed number of times)
  - *Sentinel-controlled or Endfile-Controlled loop* (`for` or `while`): (process data until a special value is encountered, e.g., end-of-file)
  - *Input validation loop* (`do-while`): Repeatedly accept interactive input until a value within a specified range is entered
  - *General conditional loop* (`while`, `for`): Repeatedly process data until a desired condition is met



# Counter Loops

- Implementing Counter Loops: the `while` loop

```
while (<repetition-condition>)  
{  
    <body>  
}
```

=====

e.g. Calculate the monthly payment.

(pseudo code was written on the white board.)

(The living coding results in class was uploaded on the course website. )

Simple example:

```
int i = 0;  
while(i<3)  
{  
    printf("Hello");  
    i++;  
}
```



# Counter Loops

- Another alternative for implementing Counter Loops: the `for` loop

```
for (<initialization>; <repetition-condition>;<update-expression>)
{
    <body>
}
```

=====

( detailed examples/pseudocode were written on the whiteboard.)

Simple example:

```
for (i = 0; i < 3; i++)
{
    printf("Hello");
}
```





# Counter Loops (3)

- Notes on `while` loops:
  - `<repetition-condition>` is evaluated at beginning of loop. If it evaluates to true, the loop body is executed. If it evaluates to false, control shifts to first statement after loop body
  - `<body>` contains one or more C statements
  - After last statement in `<body>` is executed, control is shifted back to beginning of loop, and `<repetition-condition>` is re-evaluated.
  - “Progress” must be made within the loop. That is, something must be done so that `<repetition-condition>` eventually evaluates to false. Otherwise we have an “infinite loop”



# Counter Loops (6)

- Notes on `for` loops:
  - `<initialization>` statement initializes the loop control variables before loop is executed the first time
  - `<repetition-condition>` is tested at beginning of loop. If it is true, loop `<body>` is executed.
  - `<body>` contains one or more C statements
  - After last statement in `<body>` is executed, control is shifted back to beginning of loop. Then, `<update-expression>` is executed. Finally, `<repetition-condition>` is re-evaluated.
  - As with while loops, the `<update-expression>` must define “progress.” That is, something must be done so that `<repetition-condition>` eventually evaluates to false. Otherwise we have an “infinite loop”



# Aside: Compound Assignment Operators

- Notice that the `<update-expression>`s in loops are often of the form:

```
count = count + 1
```

- C defines special assignment operators to define statements of this form more compactly:

- `count += 1` is equivalent to `count = count + 1`
- `count -= increment` is equivalent to `count = count - increment`
- `product *= product` is equivalent to `product = product * product`
- `sum /= divisor` is equivalent to `sum = sum/divisor`
- `remainder %= 2` is equivalent to `remainder = remainder % 2`



# Aside: Increment and Decrement Operators (1)

- The `++` and `--` operators take a single variable as their operands. The *side effect* of the operator is to increment or decrement its operand by one:
  - `count++` has the effect of `count = count + 1`
  - `count--` has the effect of `count = count - 1`
- Note: `++` and `--` can be placed either *before* or *after* their variable operator:
  - Pre-increment or pre-decrement (e.g., `++count`, `--count`): value of expression is value of variable *after* the increment or decrement is applied
  - Post-increment or post-decrement (e.g., `count++`, `count--`): value of expression is value of variable *before* the increment or decrement is applied



## Aside: Increment and Decrement Operators (2)

- You try it: What are the values of `i`, `j`, and `k` after each of the following statements is executed?

```
int i, j, k;  
i = 2;  
j = 3 + i++;  
k = 3 + ++i;  
i *= ++k + j--;  
i /= k-- + ++j;
```

- Let's do the coding for GPA calculation using what we learned today!



# Next Lecture...

- We'll discuss several additional loop patterns:
  - Conditional loops
  - Sentinel-controlled loops
  - Endfile-controlled loops
  - Flag-controlled loops



# References

- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (8<sup>th</sup> Ed.)*, Addison-Wesley, 2016
- P.J. Deitel & H.M. Deitel, *C How to Program (7<sup>th</sup> Ed.)*, Pearson Education, Inc., 2013.



# Collaborators

- Chris Hundhausen
- Andrew O'Fallon

