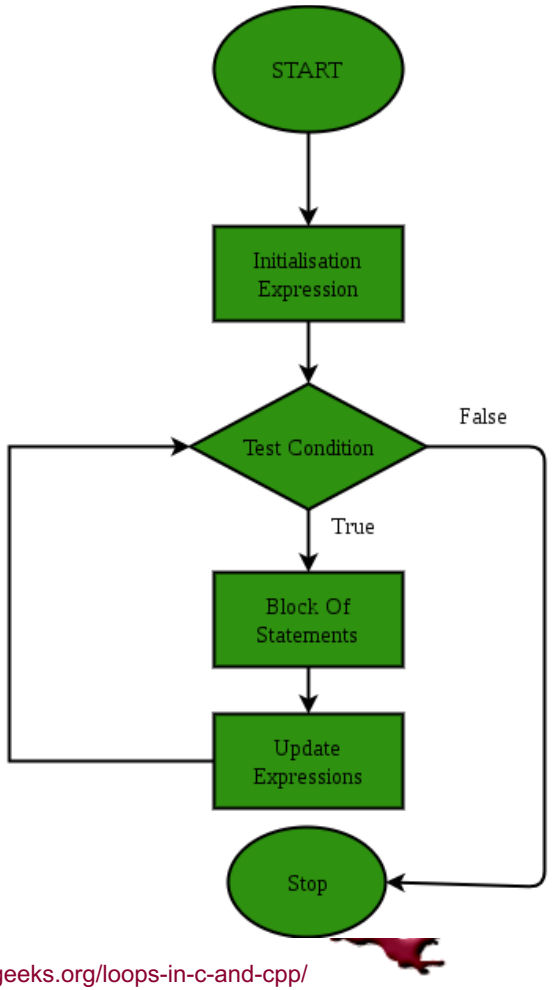
(6-2) Iteration in C II
H&K Chapter 5

Instructor – Beiyu Lin
CptS 121 (May 21st, 2019)
Washington State University

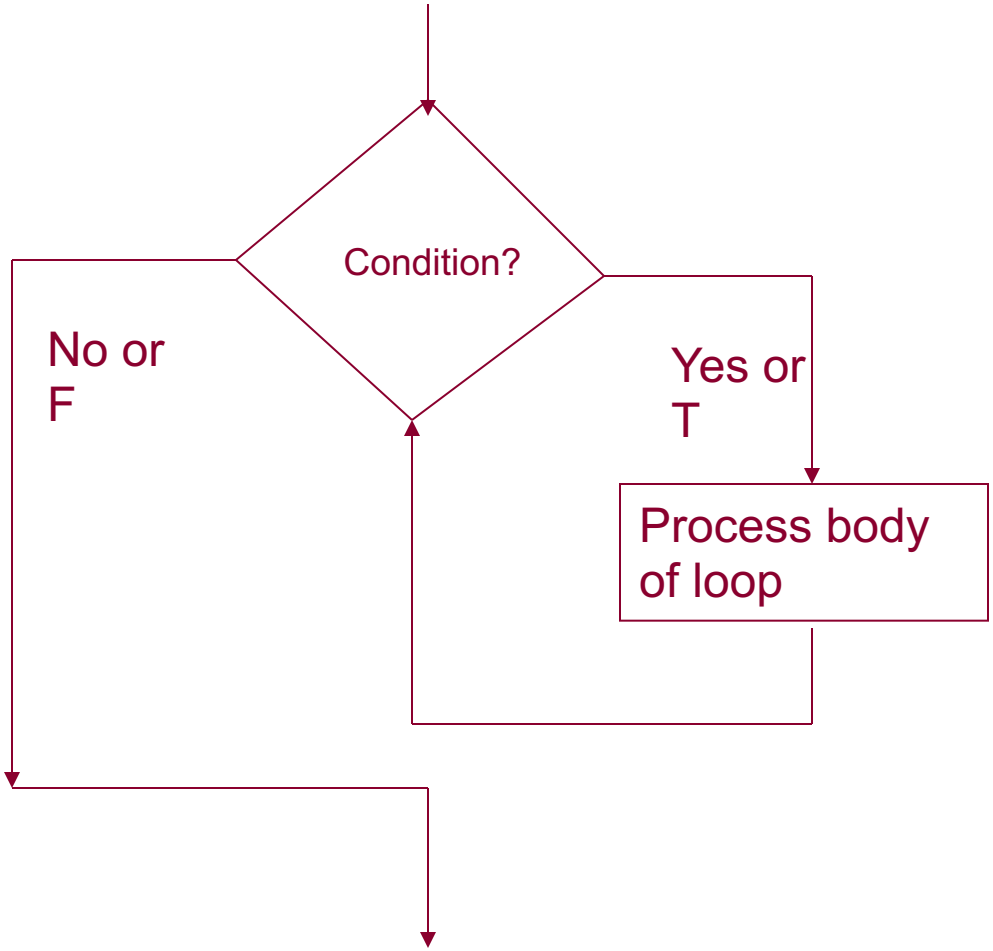
Don’t Forget About Flowcharts!

- Recall: flowcharts provide visual representations of algorithms and/or processes
- Excellent tool for verifying logical flow



Graph is from: https://www.geeksforgeeks.org/loops-in-c-and-cpp/

General Structure of Flowchart for Loops



Iteration Constructs

- We'll discuss several loop patterns:
  - Counter loops
    - (e.g. calculate a student's GPA based on 3 courses)
  - Conditional loops
    - (e.g. calculate accumulated GPA as long as the tuition < $12000)
    - Tuition = class1 * credits1 + class2*credits2 + …..
  - Sentinel-controlled loops
    - (e.g. user tells to stop the loop, such as enter "n" to stop the loop).
  - End-of-file controlled loops
    - (e.g. read to the end of the file)
  - Flag-controlled loops
    - (e.g. make sure the data enter in a certain format.)

Conditional Loops

- In the previous lecture, we considered loops whose number of iterations was known at the time the loop started
- In practice, we don't always know in advance how many times a loop will execute!
  - Often, the loop body itself determines whether another execution is necessary

Conditional Loops (2)

- Consider, for example, the following extension to the Tollbooth application:
  - Suppose that, there is a restriction that a student can not take more courses when the total credits in that semester reach 18.
  - Suppose that you want to read in grades and course information (credit, course ID) from a file to calculate the accumulated GPA.
  - When the maximum credits has been reached, not more data are read from file.
  - The program prints out a message reporting
    - the tuition for that semester
    - the number and total credits of all classes for a student in that semester.

  (Pseudo code was written on the white board.)

  (Live coding results are uploaded to the course website.)

Conditional Loops (4)

- A possible sequence of questions that can guide loop design, applied to previous example

Question	Answer	Implications for design
1. What are the inputs?	Class ID, credit, grade	Input vars: `class_id, credit, grade`
2. What are the outputs?	Total credits; number of classes; Tuition for that semester	Output vars: `total_credits, number_classes; tuitioins;`

Conditional Loops (5)

- A possible sequence of questions that can guide loop design, applied to previous example (cont.)

Question	Answer	Implications for design
3. Is there repetition?	Yes! We repeatedly	Program variable needed: `MAX_CREDITS;`

Conditional Loops (6)

- A possible sequence of questions that can guide loop design, applied to previous example (cont.)

Question	Answer	Implications for design
4. Do I know in advance how many steps will be repeated?	No.	Loop will NOT be controlled by counter
5. How do I know how long to keep repeating steps?	As long as the total credits of the semester is below the maximum	The loop repetition condition is `total_credits <MAX_CREDITS;`

Sentinel-Controlled Loops (1)

- Often we want to continue looping until a certain value, called a “sentinel,” is encountered
- For example, suppose we change the requirements of the Tollbooth application slightly:
  - There is no maximum on the total credits of the semester that a student can take.
  - We will read in the data of classes interactively
  - The user will tell us that there are no more classes for the semester by entering ‘n’ when asked whether there is another class that needs to cross (‘n’ = “No” the sentinel value)

(Pseudo code was written on the white board.)

(Live coding results are uploaded to the course website.)

Endfile-Controlled Loops (1)

- Often, as in the original GPA application, we read input data in from a file
- We want to continue processing data until there is no more data to process
- In other words, we want to continue processing data until the end of the file is encountered
- We can use the end-of-file-controlled loop pattern to do this

Endfile-Controlled Loops (2)

- For example, suppose that we change the requirements of the GPA calculation again
  - We will read the input values from a text file
  - We will continue reading class information and credits until we reach the end of the file
  - Let's look at the implementation…

(Pseudo code was written on the white board.)

(Live coding results are uploaded to the course website.)

Endfile-Controlled Loops (3)

- `fscanf` actually returns a value indicating the number of items it successfully reads in
- If it encounters the end of the file, it returns as its result the value of the standard constant `EOF` (which is a negative integer)
- We can thus redesign read_num_axles to return EOF if it encounters the end of the file:

```
int read_num_axles(FILE *infile) {
   int num_axles, input_status;
   input_status = fscanf(infile,"%d",&num_axles);
   if (input_status != EOF)
     return (num_axles);
   else
     return input_status;
}
```


Flag-Controlled Loops (1)

- In the previous examples, we have assumed that input data are always in the proper format:
  - When we ask for the number of axles, we will obtain an integer (either interactively from the user, or from the input file)
  - When we ask for the weight, we will obtain a double (either interactively from the user, or from the input file)
- In the real world, this assumption is faulty
  - People enter invalid data all the time
  - Files contain invalid data all the time
- Flag-controlled loops ensure that valid data are read in

Flag-Controlled Loops (2)

- Recall that the `fscanf` function returns `EOF` when the end of the file is encountered
- Likewise `fscanf` and `scanf` return the value 0 when at least one the data values it reads in could not be converted to the specified type
  - For example, assume the following scanf statement
  -
```
int my_int, input_status;
printf("Please enter an integer: ");
input_status = scanf("%d",&my_int);
```

    If the user were to type in "wow" here, `input_status` would be assigned the value `0`, since "wow" cannot be converted to an `int`

(Example in class: read in the date type "dd/mm/yyyy". pseudo code was on the white board, including usig do while loop. Live coding results is on the course website.)

Flag-Controlled Loops (3)

- The final C interative construct, the `do-while` loop, can be used to trap this situation and re-prompt the user:

```
 int my_int, input_status;
char skip_ch;
do {
    printf("Please enter an integer: ");
    input_status = scanf("%d",&my_int);
    do { /* nested do-while skips rest of data line */
      scanf("%c", skip_ch);
    } while (skip_ch != '\n');
} while (input_status == 0);
```

Notice that, unlike the `while` and `for` loop constructs, the `do-while` loop construct is guaranteed to execute at least once.

References

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

Collaborators

- Chris Hundhausen
- Andrew O'Fallon