-DECISION MAKING AND LOOPING-

CO-101: Programming Fundamentals

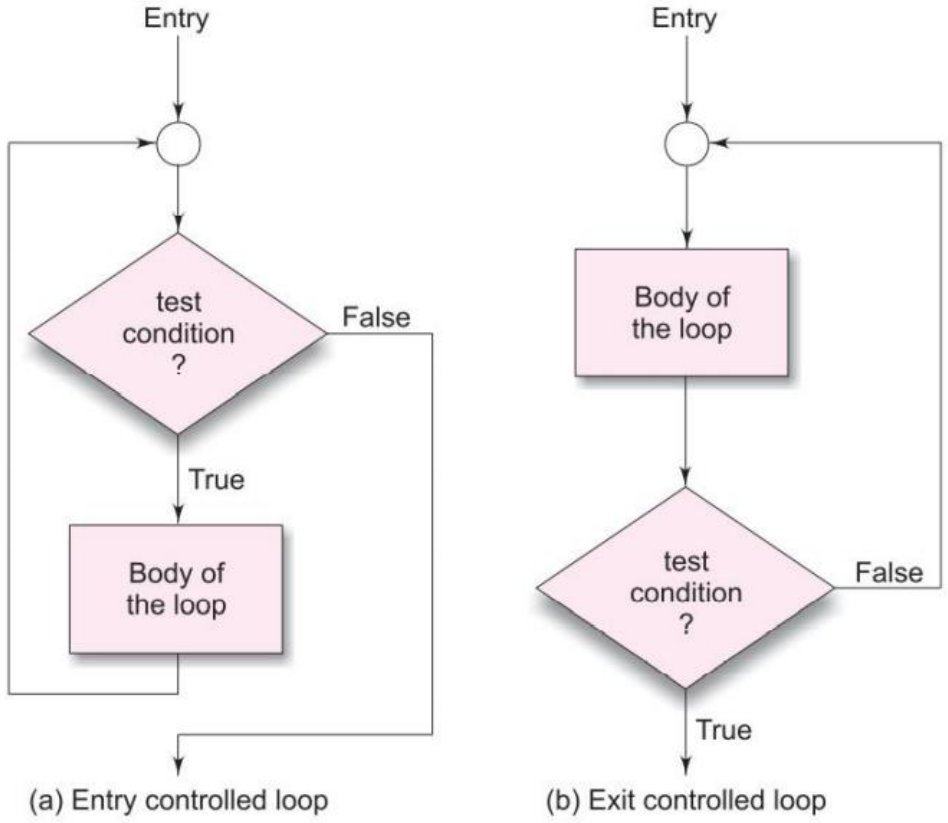
Background

- In looping, a **sequence of statements (to be repeated) are executed** until some conditions for termination of the loop is encountered therefore it consists of two segments, one known as the **body of the loop** and the other known as the **control statement**.
- The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

- **Loop Control Structures:**

Based on the position of the control statement in loop, control structure can be of two types:

1. entry-controlled loop
2. exit-controlled loop



- The test conditions should be carefully stated in order to perform the desired number of loop executions.
- Test condition transfers the control out of the loop.
- If not, the control sets up **infinite loop** an and the body is executed over and over again.

The C language provides for three constructs for performing loop operations.

1. **The while statement.**
2. **The do while statement.**
3. **The for statement**

Process involved in iterations:

1. Setting and initialization of a condition/loop invariant/ loop variable
2. Test for a specified value of the condition variable for loop execution.
3. Execution of the statements in the loop.
4. Update condition variable (increment/decrement)

Based on the **nature of control variable** and the kind of value assigned to it for testing the control expression, loops can be of following types:

1. Counter-controlled loop

When we know in advance exactly how many times the loop will be executed. We use a control variable known as counter. The counter must be initialized, tested and updated properly for the desired loop operations.

2. Sentinel-controlled loops/Indefinite repetition loop

A special control variable, called sentinel variable is used to change loop control expression (from true to false). Here, the number of repetitions is not known before the loop begins executing.

E.g.: When reading data we may indicate the “end of data” by a special value, like –1 and 999.

A. THE WHILE STATEMENT

- The while is an **entry-controlled** loop statement.
- Syntax:

```
while (test condition)
{
      body of the loop
}
```

- The test-condition is checked and if the condition is true, body of loop is evaluated.
- After body of loop, test condition is evaluated again and if it is true, the body is executed once again.
- This process is repeated until condition is false and the control is transferred out of the loop.
- On exit, the program continues with the statement immediately after the body of the loop /fall-through statement/ next sequential statement.
- It can be nested (one while loop inside other).

- The body of the loop may have one or more statements.
- It is counter-controlled loops.
- The braces are needed only if the body contains two or more statements.

B. THE DO WHILE STATEMENT

- Executes the body of the loop before testing condition.
- It provides **exit-controlled loop.**
- Body of the loop always executed at least once.
- do.... while loops can be in nested form
- Syntax:

```
do
{
      body of the loop
}
while (test-condition);
```

- On reaching the do, body of the loop is executed first.
- At the end of the loop, the test-condition in the while statement is evaluated.
- If the condition is true, program continues to evaluate the body of the loop once again.
- This process continues as long as the condition is true.
- When the condition becomes false, the loop will be terminated and the control goes to the statement that appears immediately after the while statement.

C. THE FOR STATEMENT

1. Simple ‘for’ Loops

- It is an entry-controlled loop that provides a more concise loop control structure.
- All the three actions, namely initialization, testing, and incrementing/decrementing, are placed in the for statement making them visible to the programmers and users, in one place.
- Syntax:

```
for ( initialization ; test-condition ; increment)
{
     body of the loop
}
```

- **Execution process of the for statement :**

1. Initialize the control variable. (e.g. count = 0). The variable count is known as **loop-control variables**.
2. The value of the control variable is tested using the test-condition. The test-condition is a relational expression, such as i < 10 that determines when the loop will exit.
3. If the condition is true, the body of the loop is executed; otherwise the loop is terminated and the execution continues with the statement that immediately follows the loop.
4. When the body of the loop is executed, the control is transferred back to the for statement. Now, the control variable is incremented/decremented using an assignment statement such as i = i+1 and the new value of the control variable is again tested again executed.
5. This process continues till the value of the control variable fails to satisfy the test-condition.

- Comparison of three loop/iteration structures:

  Illustration:

  Iteration (10 times) starting from 1 till 10.

for	*while*	*do*

for (n=1; n<=10; ++n)
{
_____
_____
{
	n = 1;
	while (n<=10)
	{
	_____
	_____
	n = n+1;
	}
		n = 1;
		do
		{
		_____
		_____
		n = n+1;
		}
		while(n<=10);

Additional Features of for Loops:

1. More than one variable can be initialized (comma separated) at a time in the for statement.
2. Increment/decrement section may also have more than one part (comma separated).

3. Test-condition may have any compound relation and the testing need not be limited only to the loop control variable.

4. One or more sections can be omitted, if necessary. In such cases, the sections are left ‘blank’. However, the semicolons separating the sections must remain.

E.g.:

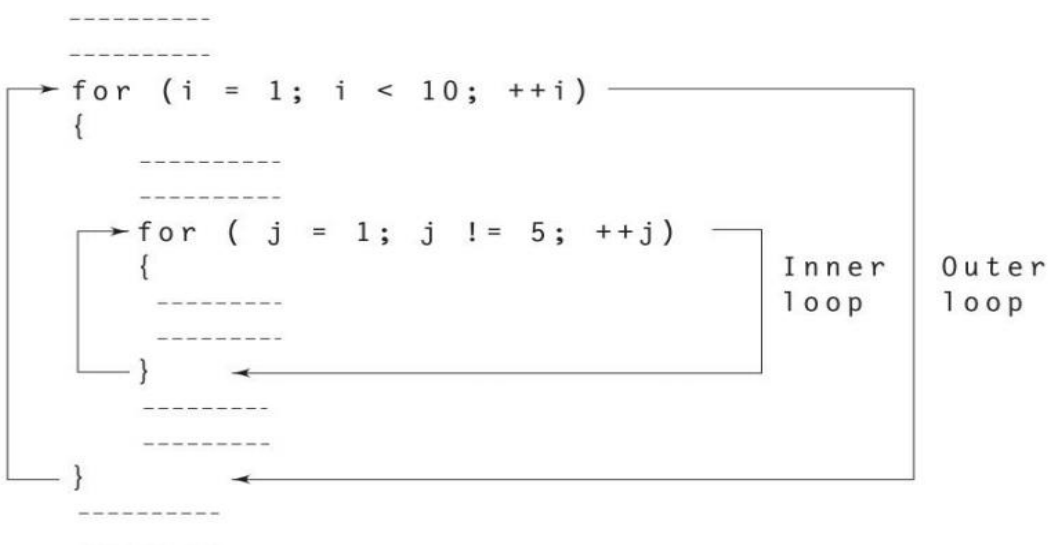
```
-------
m = 5;
for ( ; m != 100 ; )
{
     printf("%d\n", m);
     m = m+5;
}
-------
```

Here, both the initialization and increment sections are omitted in the for statement

If the test-condition is not present, the for statement sets up an **infinite loop**. Such loops can be broken using **break or goto** statements in the loop.

2. Nested for Loops:

- Nesting of loops, that is, one for statement within another for statement, is allowed in C.
- E.g.:



- Nesting may continue up to any desired level.
- Properly indented loops so for better clarity on which statements are contained within each for loops.

Selecting suitable loop construct

- Analyse the problem and see whether it required a pre-test or post-test loop.
- If it requires a post-test loop, then we can use only one loop, **do while**.
- If it requires a pre-test loop, then we have two choices: **for** and **while**.
- Decide whether the loop termination requires counter-based control or sentinel-based control.
- Use **for** loop if the counter-based control is necessary.
- Use **while** loop if the sentinel-based control is required.
- Note that both the counter-controlled and sentinel-controlled loops can be implemented by all the three control structures.

JUMPS IN LOOPS

- When executing loop, sometimes it is desired to skip a part of the loop or exit a loop as soon as a condition is met.
- C allows jump from one statement to other within a loop as well as outside the loop.
- E.g.: Finding prime number, searching a name in a file, finding a value in array.

1. Jumping Out of a Loop

It can be achieved by:

- break statement
- goto statement

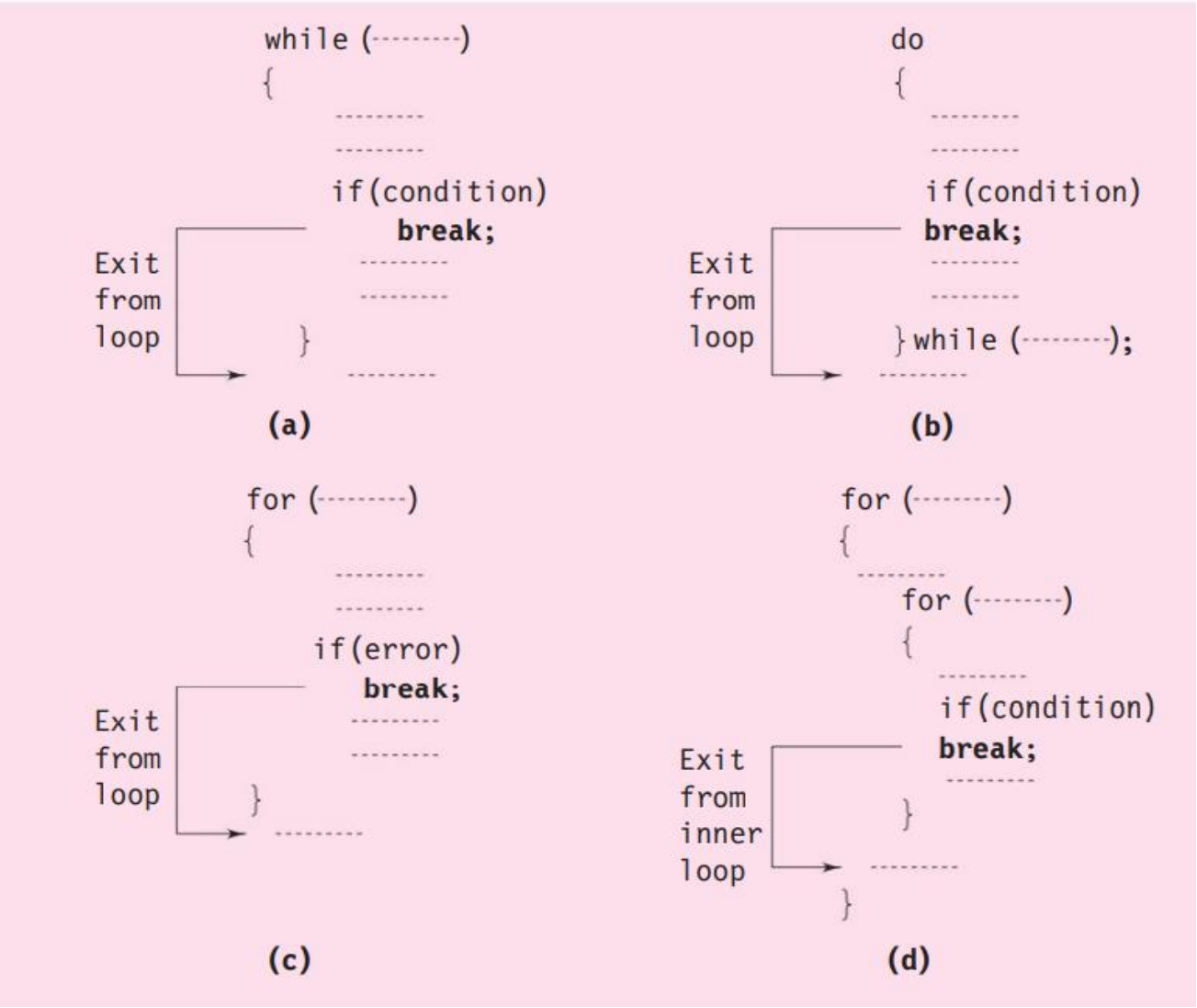
break: When break is encountered in a loop, it comes out of the loop.

If case of nested for-loops, break will take the control back to its immediate outer for-loop (next sequential statement).

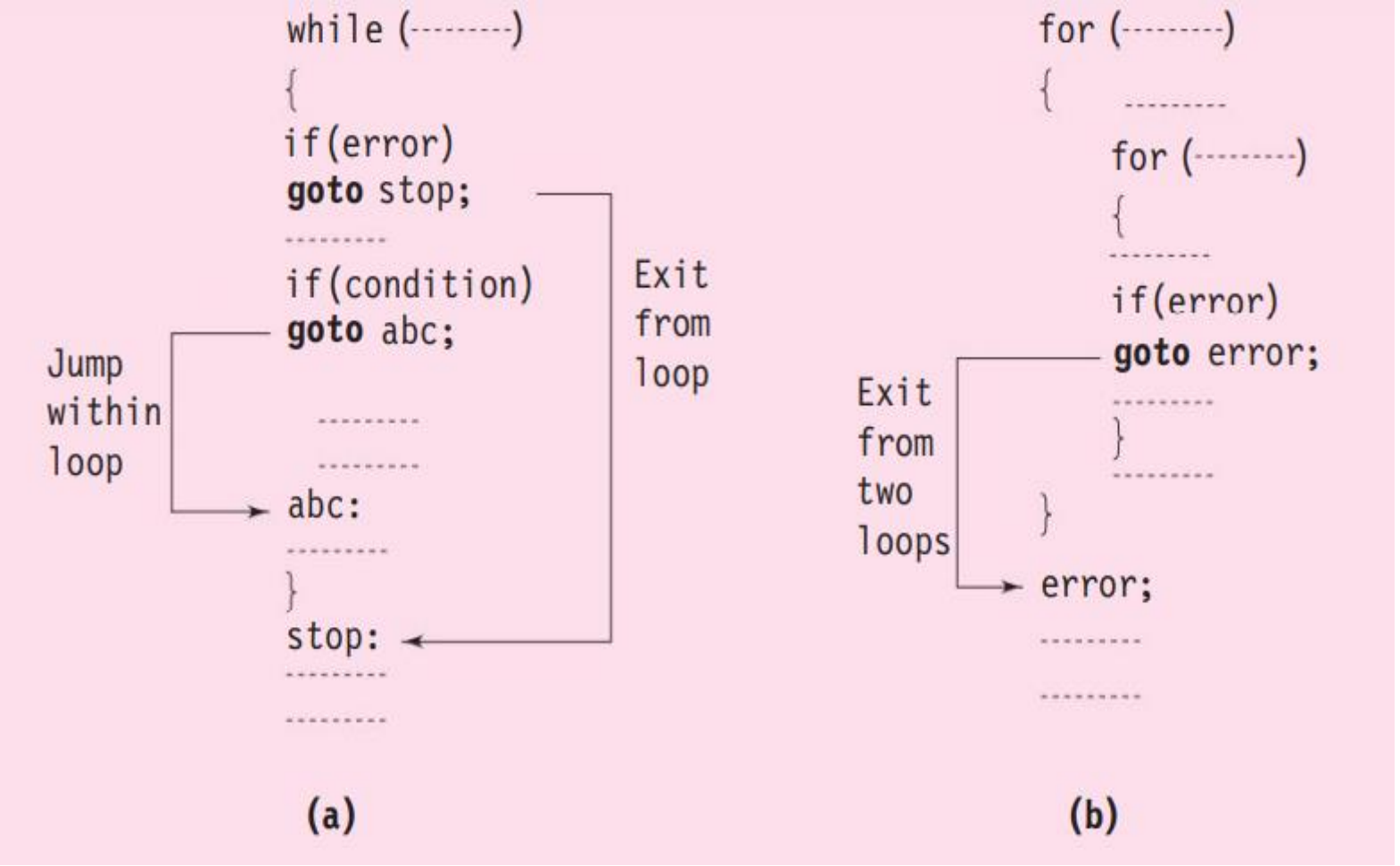
goto: transfers control to any place in a program;

Usage: useful for branching within a loop and to exit from deeply nested loops when an error break statement would not work here.

- Examples of using ‘break’ in loops:



- Examples of ‘goto’ in loops:



Structured Programming

- Structured programming is an approach to the design and development of programs.
- It is a discipline of making a program's logic easy to understand by using only the basic three control structures:

1. Sequence (straight line) structure
2. Selection (branching) structure
3. Repetition (looping) structure

Helps ensure well-designed programs that are easier to write, read, debug and maintain.

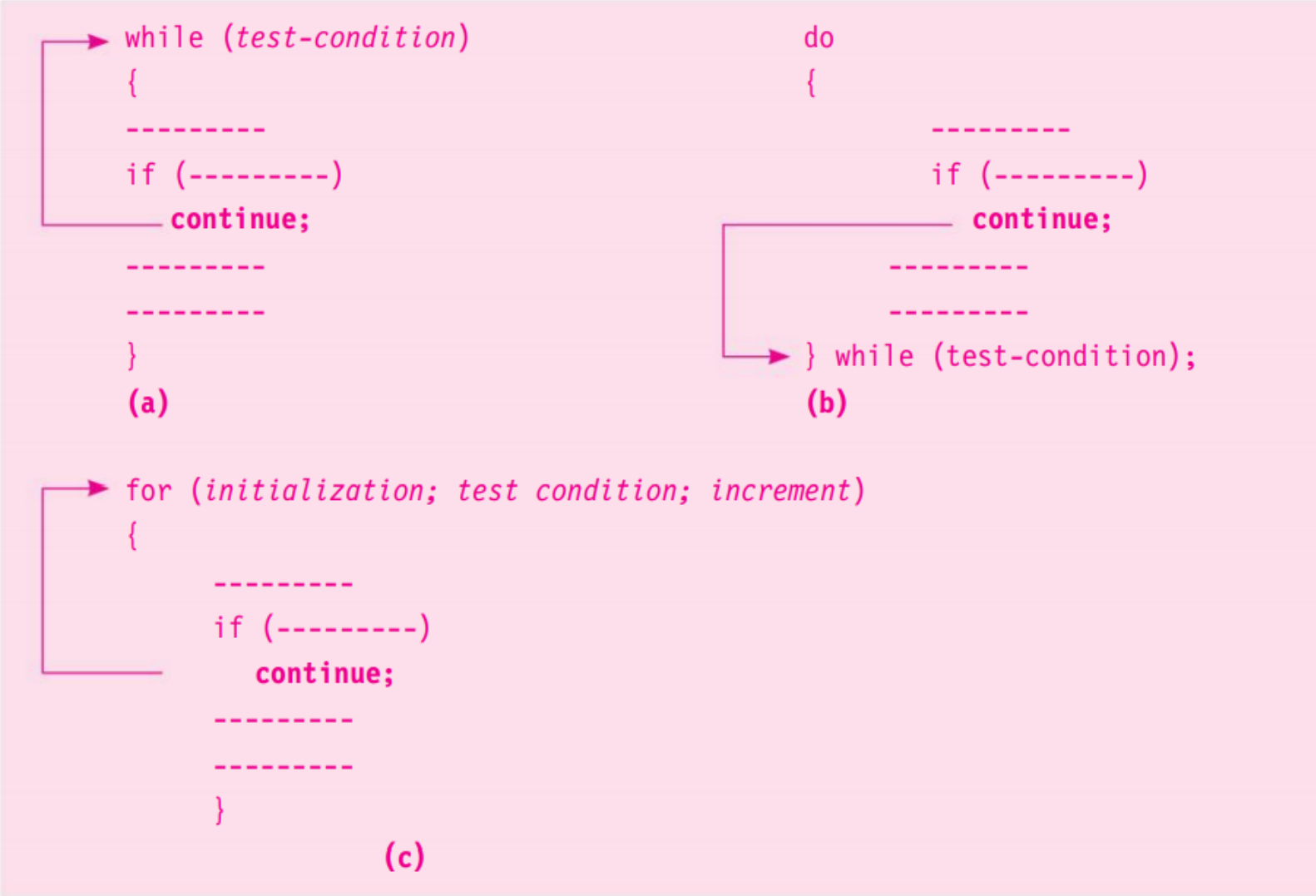
Discourages the implementation of unconditional branching using jump statements (e.g. goto, break and continue).

In its purest form, structured programming is synonymous with "goto less programming". Do not go to goto statement!

Skipping a Part of a Loop:

- Causes the loop to be continued with the next iteration after skipping any statements in between.
- In while and do loops, continue causes the program to go to the test-condition so as to continue the iteration process.
- In the case of for loop, continue causes the program to go to the increment section followed by test-condition.

- Using continue in different loops:



2. **Jumping out of the Program**

- To break out of a program and return to the operating system, we can use the exit( ) function.
- To use it, **#include<stdlib.h>** is required.
- Illustration:

```
........
........
if (test-condition) exit(0) ;
........
........
```

- The exit( ) function takes an integer value as its argument.
- Normally zero is used to indicate normal termination and a nonzero value to indicate termination due to some error or abnormal condition.

CONCISE TEST EXPRESSIONS

- Test expressions are evaluated and compared with zero for making branching decisions.
- Since every integer expression has a true/false value, we need not make explicit comparisons with zero.

if (expression ==0)

is equivalent to

if(!expression)

Similarly,

if (expression! = 0)

is equivalent to

if (expression)

For example,

if (m%5==0 && n%5==0) is same as **if (!(m%5)&&!(n%5))**