Ternary/conditional operator

```
if (condition) {
    // execute if condition is true
}
else {
    // execute if condition is false
}
```

- if-else can be nested within each other.

```
condition ? expression1 : expression2;
```

- If condition is true, expression1 is executed; otherwise expression2 is executed.
- Ternary operators can also be nested.
- expression1 & expression2 must be expressions (not statement).
    - expression – evaluate to some value.
    - statement – C statement ends with ;

switch case

```
switch (expression) {
     case const-expr1:
        statement(s);
        break;
    case const-expr2:
        statement(s);
        break;
         …
    default:
        statement(s);
        break;
}
```

- Switch-case is used to select one of the several paths to execute depending on value of int expression.
- case constants cannot be duplicated.
- break statement skips remaining statements and continues execution at the end of switch closing brace.
- If break is missing, statements under sub-sequent case continue to execute.
- default case is optional and it is executed only if an expression is not matching with any of the case constant.
- Sequence of cases and default case doesn't matter.

Loops

- Control statements used for repeating a set of instructions number of times is called as “LOOP”.
- Every loop has
- Initialization statement
- Terminating condition
- Modification statement(Increment/Decrement)
- Body of loop
- The variable that is used for terminating condition is referred as ‘loop variable’.

while loop

- Used to repeat a statement (or block) while an expression is true (not zero).
- Syntax:

```
initialization;
while(condition) {
    statement1;
    statement2;
    modification;
}
```

for loop

- Used to repeat a statement (or block) while an expression is true (not zero).
- Syntax:

```
for(initialization; condition; modification) {
    statement1;
    statement2;
}
```

do while loop

- Used to repeat a statement (or block) while an expression is true (not zero).
- Syntax:

```
do {
    statement1;
    statement2;
} while(condition);
```

- do-while is exit control loop.
- while & for are entry control loops.
- do-while is executed at least once.

Infinite loop

- If loop condition is always true, program never terminates.

```
while(1) {
    …
}

for( ; ; ) {
    …
}

do {
    …
} while(1);
```

break and Continue

- break statement
  - Used to early exit from loop, or to exit an infinite loop
  - Takes control out of current loop and continues execution of statements after the loop.
  - Statements after break are skipped.
- continue statement
  - Used to continue next iteration of the loop.
  - Statements after continue are skipped (for current iteration).
- break is used with loop/switch case.
- continue used with only loop.
- In case of nested loops, break/continue affects current loop only (not outer).

goto statement

- Jumps to statement label, must be within same function as the goto.
- Statement label is an identifier followed by a colon (😃
- Unstructured control statement
- Used rarely (less readable)
- Advised to use only for forward jump
- Best use is to exit from deeply nested loops.
- Syntax:

```
goto label_name;
..
..
label_name: C-statements
```