Control Structures

in C Language

C Statements

- Describe the algorithm in C language
- Flow of control
- Three groups of statements:
  - sequence
  - selection
  - repetition

Sequence

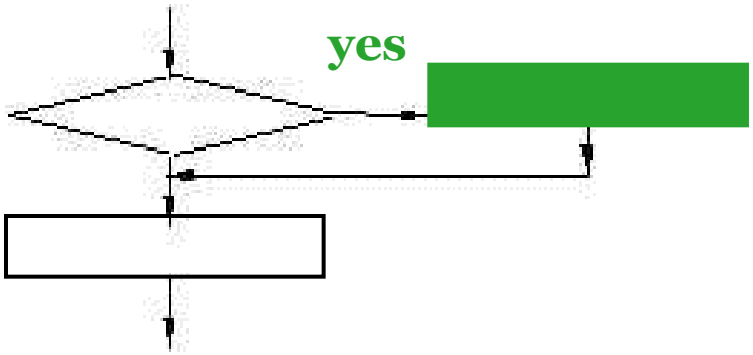
- Assignment =
- Function call
- Comma operator ,
- Empty Statement
- Compound Statement { }
- Transfer of Flow Control

break
continue
return

Selection

- Choice between two alternatives
  if
  if else
  ? :
- Choice between more alternatives
  switch case

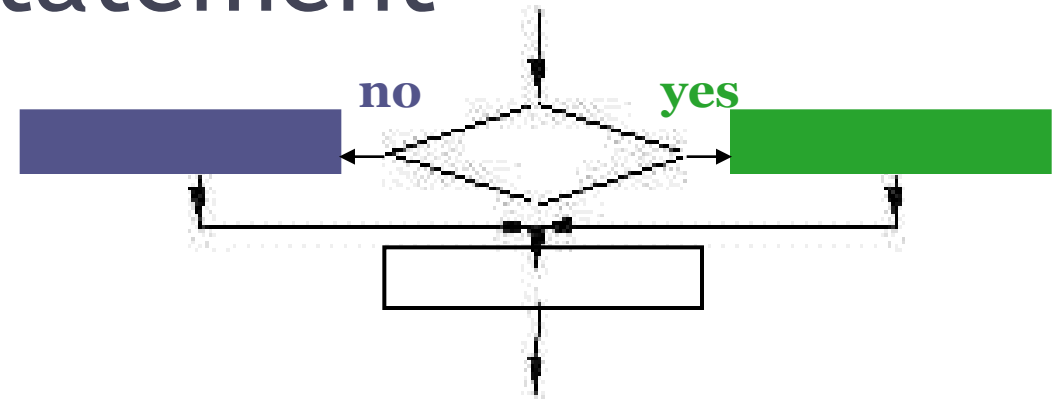
if Statement



if (expression) statement

- execute the statement if the expression is *true*

if else Statement



if (expression) statement_1
else statement_2

- execute statement_1 if expression is true, otherwise execute statement_2

switch case Statement

```
switch (expression)
{
    case value1:      statements1;  break;
    case value2:      statements2;  break;
   case value 3:     statements 3;  break;
    default:          default statements;
}
```

Expression is an integer expression, and is matched against case values which also must be integers!

Conditional Operator

expression 1 ? expression 2 : expression 3

- The *expression 1* is evaluated first.
  - If its value is nonzero (*true*), then the *expression 2* is evaluated and its value is a value of the whole operation.
  - If the value of the *expression 1* is zero (*false*), then the *expression 3* is evaluated and its value is taken as a result of the whole operation.
- Example

```
z = (x < y) ? x : y;
```

This is equivalent to:

```
if (x < y) z = x; else z = y;
```

- The type of the result is determined by the type of the *expression 2* and the *expression 3*. If they are of different types, the usual conversion rules are applied.
- The *conditional operator* has a precedence just above the assignment operators and it associates from right to left.

switch Example

```
switch (num)
{
   case 1:    printf("one\n"); break;
   case 2:
   case 3:
   case 4:    printf("some\n"); break;
   default :  printf("many\n");
}
```

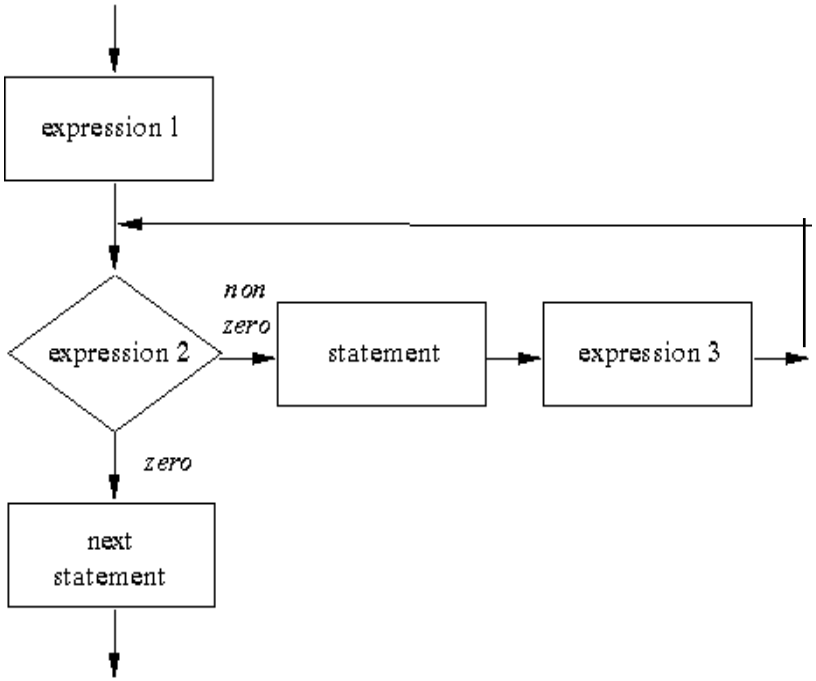
Repetition

- Counted number of times
  - for
- Uncounted number of times, conditional
  - pre-condition
    while
  - post-condition
    do while

for Statement

- The syntax is:

```
for (expression 1; expression 2; expression 3)
    statement;
next statement;
```



for Statement

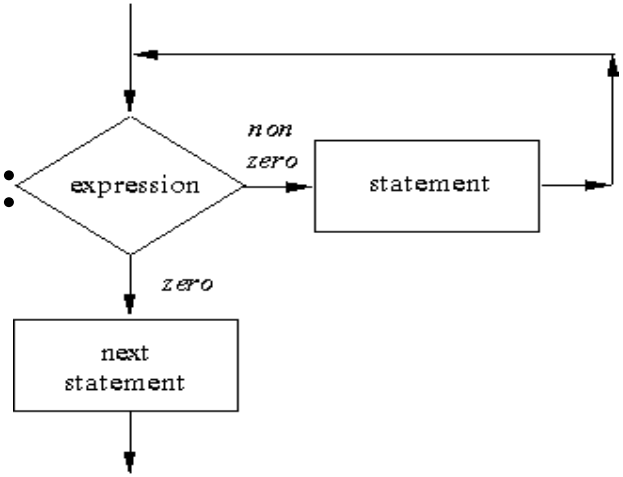
- The syntax is:

```
for (expression 1; expression 2; expression 3)
    statement;
next statement;
```

- First **expression 1** is executed. Typically it initializes the loop.
  - then **expression 2** is evaluated
  - if it is non-zero (*true*), the **statement** is executed
  - **expression 3** is performed. Typically **expression 3** changes the conditions of the loop.
  - after it the repetition of the **expression 2** evaluation and **statement** execution start
- The process continues while the **expression 2** stays *true*. When it changes to *false* (*zero*) the **next statement** is to be performed.

while Statement

- The syntax of while statement is:

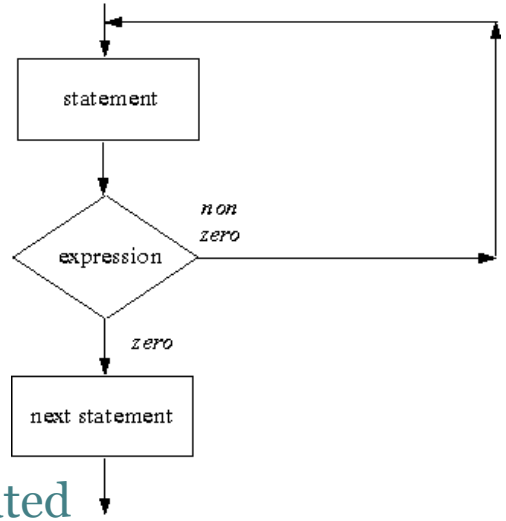


while (expression) statement;

- The **expression** is first evaluated.
  - if it is different from zero (*true*), the **statement** is executed and the evaluation of the **expression** is repeated.
  - if the expression is still true, the **statement** is executed again and the **expression** is evaluated again.
  - the repetition takes place until the **expression** becomes equal to zero (*false*).
  - then the **next statement** located after the while statement is executed

do while Statement

- The do_while statement is a variant of while statement. The difference is in the exchanged place of the statement and the expression
- The general form of the statement is:



do statement while (expression);

- First statement is executed,
  - then expression is evaluated
  - if the value of it is non-zero (true) the process is repeated
  - the repetition stops when the value of expression becomes zero (false).
- Unlike to while statement, do_while always executes statement at least once