System-Level Programming

8 Control Structures

Peter Wägemann

Lehrstuhl für Informatik 4
Systemsoftware

Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)

Summer Term 2025

http://sys.cs.fau.de/lehre/ss25

`goto` Instruction

```
    ...
    goto Label
    ...
Label:
    ...

    ...
Label:
    ...
    goto Label
    ...
```

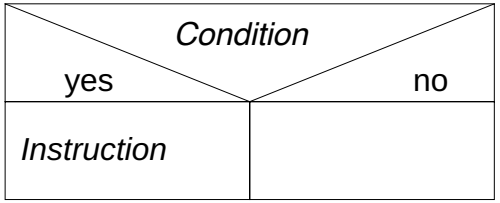
- `goto` statement rarely used in clean code
- Edgar Dijkstra: „Go To Statement Considered Harmful”
- `goto` leads to hard-to-read code
- Label must not be the function's last statement

- `goto` and `if (...) goto` statements are the only control structures that are hardware can directly execute.
- This aspect is essential for understanding interrupts!

Conditional Statements [≈Python]

- Minor differences in syntax between C and Java/Python
- `if` statement (conditional statement)

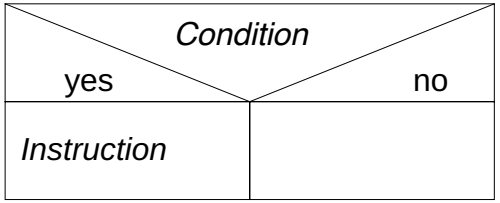
```
if (condition)
    instruction;
```



Conditional Statements [≈Python]

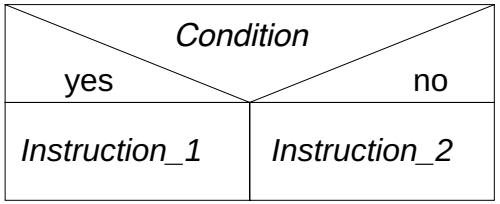
- Minor differences in syntax between C and Java/Python
- `if` statement (conditional statement)

```
if (condition)
    instruction;
```



- `if-else` statement (two branches)

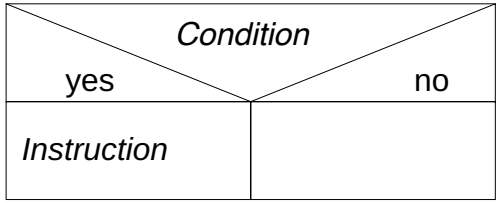
```
if (condition)
    instruction1;
else
    instruction2;
```



Conditional Statements [≈Python]

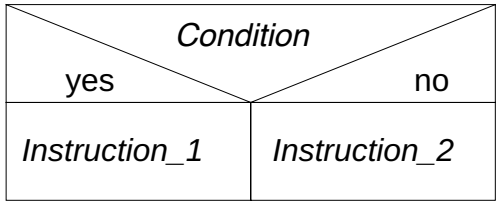
- Minor differences in syntax between C and Java/Python
- `if` statement (conditional statement)

```
if (condition)
    instruction;
```



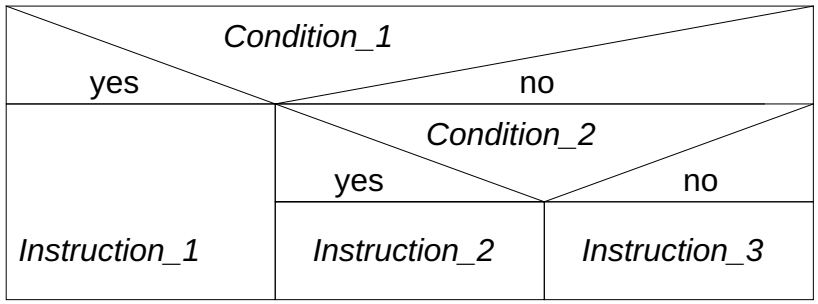
- `if-else` statement (two branches)

```
if (condition)
    instruction1;
else
    instruction2;
```



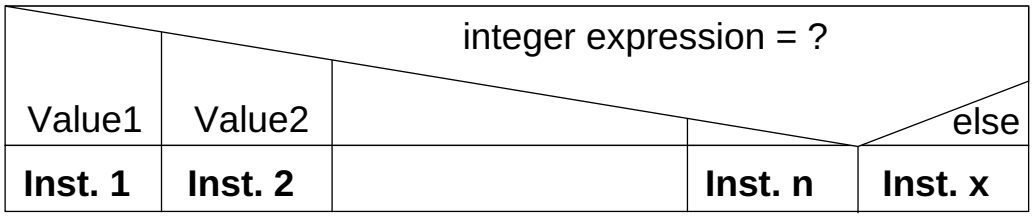
- `if-else-if` cascade (multiple branches)

```
if (condition1)
    instruction1;
else if (condition2)
    instruction2;
else
    instruction3;
```



Switch Cases [≈Python]

- `switch` statement (case selection)
  - alternative to `if` cascade when testing for integer values
  - Python: `match case`

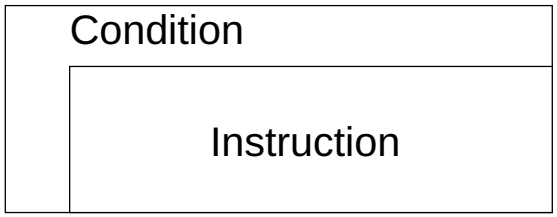



```
switch (expression) {
case value1:
     instruction1;
     break;
case value2:
     instruction2;
     break;
...
case valuen:
     instructionn;
     break;
default:
     instructionx;
}
```

Pre-Condition and Post-Condition Loops

Pre-condition loop

- `while`-loop
- executed zero or more times



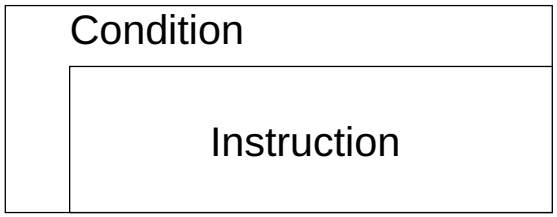

```
while(condition)
    instruction;
```

```
while (
  sb_button_getState(BUTTON0)
      == RELEASED
) {
  ... // do unless button press.
}
```

Pre-Condition and Post-Condition Loops

Pre-condition loop

- `while`-loop
- executed zero or more times

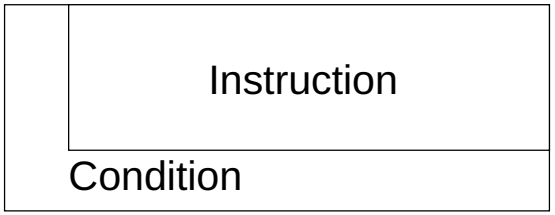



```
while(condition)
    instruction;
```

```
while (
  sb_button_getState(BUTTON0)
      == RELEASED
) {
  ... // do unless button press.
}
```

Post-condition loops

- `do-while` loops
- executed once or more



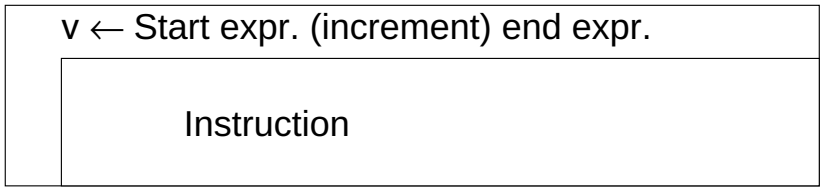

```
do
    instruction;
while(condition);
```

```
do {
  ... // do at least once
} while (
  sb_button_getState(BUTTON0)
      == RELEASED
);
```

Iterating Loops [≠Python]

- C: `for` loop has an explicitly managed counter
- Python: for item in iterable

```
for (starting_expression;
     terminating_expression;
     incrementing_expression)
  instruction;
```



- Example (usually: n executions with counter variable)

```
uint8_t sum = 0; // calc sum 1+...+10
for (uint8_t n = 1; n < 11; n++) {
  sum += n;
}
sb_7seg_showNumber( sum );
```

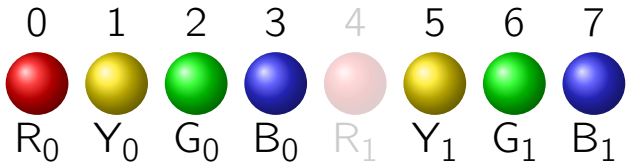
- Remarks
  - Declaring a variable (`n`) in the *starting_expression* is only possible from C99 onwards.
  - The loop is repeated as long as *terminating_expression* ≠ 0 (*true*)
    ⇝ the `for` loop is a more explicit `while` loop

Controlling Loops [=Python]

- The current iteration of the loop can be terminated with the `continue` instruction.

  ⇝ The loop continues with the next iteration

```
for (uint8_t led = 0; led < 8; led++) {
  if (led == RED1) {
    continue;          // skip RED1
  }
  sb_led_on(led);
}
```

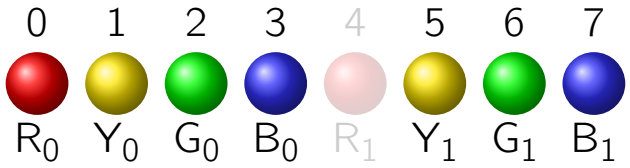


Controlling Loops [=Python]

- The current iteration of the loop can be terminated with the `continue` instruction.

  ⇝ The loop continues with the next iteration

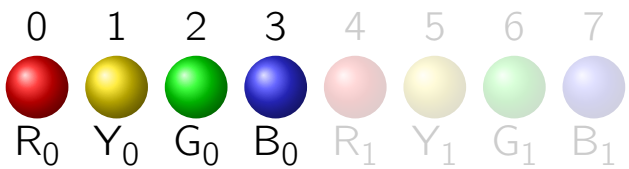
```
for (uint8_t led = 0; led < 8; led++) {
  if (led == RED1) {
    continue;          // skip RED1
  }
  sb_led_on(led);
}
```



- The execution of the *innermost loop* is terminate with the `break` instruction.

  ⇝ The program resumes execution *after* the loop

```
for (uint8_t led = 0; led < 8; led++) {
  if (led == RED1) {
    break;             // break at RED1
  }
  sb_led_on(led);
}
```



Loops & `goto` Instructions

- All loop types have semantically-equivalent sequences with `goto` statements
- Example:

```
for (uint8_t led = 0; led < 8; led++) {
  if (led == RED1) {
    continue; /* skip RED1 */
  }
  sb_led_on(led);
}
```

```
  uint8_t led = 0;
  goto test;
loop:
  if (led == RED1)
    goto next;
  sb_led_on(led);
next:
  led++;
test:
  if (led < 8)
    goto loop;
end:
```