# Control Flow

## Statements and Blocks

- A statement is a expression followed by a semicolon.
- Braces **{}** are used to group declarations and statements into a compound statement, or block, so that they are syntactically equivalent to a single statement.
  - braces that surround the statements of function
  - braces around multiple statements after **`if, else, while`** or **`for`**.

## `if - else`

```
if (expression)
    statement_1;
else
    statement_2;
```

- **`else`** part is optional.
- Firstly, the expression is evaluated.
- If it is true (non-zero value), statement_1 is evaluated.
- If it is false (expression is zero), statement_2 is evaluated.

## `else - if`

```
if (expression_1)
    statement_1;
else if (expression_2)
    statement_2;
else if (expression_3)
    statement_3;
else if (expression_4)
    statement_4;
else
    statement_5;
```

- The expressions are evaluated in order.
- If any expressions is true, the statement associated with it is executed, and this terminates the whole chain.
- **`else`** part is optional.
- The last **`else`** part handles the "none of the above" or default case where none of the other conditions is satisfied.

## switch

```
switch (expression) {
    case const_expr_1: statements_1
    case const_expr_2: statements_2
    default: statements_3
}
```

- Each case is labeled by one or more integer-valued constants or constant expressions.
- All cases expressions must be different.
- **`default`** statement is optional.
- The **`break`** statement causes an immediate exit from the switch.

```c
#include <stdio.h>

void foo(int expr) {
    switch(expr) {
        case 0: printf("0\n"); break;
        case 1:
        case 2: printf("1 or 2\n"); break;
        case 3: printf("3\n");
        case 4: printf("3 or 4\n"); break;
        default: printf("not 0 - 4\n"); break;
    }
}

int main() {
    for (int expr = 0; expr <= 5; ++expr) {
        printf("expr is %d\n", expr);
        printf("the result is ");
        foo(expr);
    }

    return 0;
}
```

```
expr is 0
the result is 0
expr is 1
the result is 1 or 2
expr is 2
the result is 1 or 2
expr is 3
the result is 3
3 or 4
expr is 4
the result is 3 or 4
expr is 5
the result is not 0 - 4
```

## Loops -- **`while`** and **`for`**

while loops:

```
while (expression)
    statement
```

for loops:

```
for (expr_1; expr_2; expr_3)
    statement
```

which is equivalent to

```
expr_1;
while (expr_2) {
    statement
    expr_3;
}
```

except for the behavior of **`continue`**.

## Loops -- **`do - while`**

```
do
    statement
while (expression);
```

## **`break`** and **`continue`**

- The **`break`** causes the **innermost** enclosing loop or **`switch`** to be exited immediately.
- The **`continue`** statement causes the next iteration of the enclosing loop.
  - In the **`while`** and **`do`**, this means that the test part is executed immediately.
  - In the **`for`**, control passes to the increment step.
  - only applied to loops, not to **`switch`**.

## `goto` and `label`

- **`goto`** is not necessary.
- In a few situations, it may find a place

```
for (...) {
    for (...) {
        for (...) {
            .........
            if (disaster)
                goto error;
        }
    }
}

error:
    // clean up the mess
```