C Programming

Statements

Statements

- A statement is a command which makes a computer to take one or more specific actions
  - E.g. assigning a value to a variable, calling functions, jumping to another statement and etc.
- A computer program is made up of a series of statements.

- Statements in C
  - Labeled Statements
  - Compound Statements
  - Expression Statements
  - Selection Statements
  - Iteration Statements
  - Unconditional Jump Statements

Statements

- A simple identifier followed by a colon (:) is a label. E.g. ***exit:***
- A **labeled statement** is preceded by a label.
  - A simple label which is the target of **goto** statement. E.g. goto **exit**;
  - case labeled statements in a switch statements
  - default labeled statements in a switch statements
- A **compound statement** consists of multiple statements and declarations within curly brackets (**{}**).

  { [list of declarations and statements] }
  - Usually the declarations are placed at the beginning
  - E.g. body of the **main** function in a program
  - It is also called as block statement

```
switch (expression)
{
case CASE_1:
    /* Statements */
    break;

case CASE_2:
    /* Statements */
    break;

default:
    /* The default statements */
    break;
}
```

Statements

- An **expression statement** consists of an optional expression followed by a semicolon

  [expression] ; For examples: int a = 2; printf(“Hello World”); and etc.

  - If no expression exists, the statement is often called a **null** statement.
- **Selection Statements** in C
  - Are used to direct the flow of execution along different branches depending on a condition
  - Are also called **branching** and there are two types of selection statement; ***if*** and ***switch***
- An **if** statement is in the form of ***if (expression) statement_1 [else statement_2]***
  - The expression shall be a **boolean** and varying. The expression is evaluated first
    - If its value is **1**(true) then **statement_1** is executed**;** otherwise **statement_2** (if exists)
  - The statements shall be **compound statements**; i.e. enclosed in **{}**
  - The **else** clause is optional and it can be **immediately** followed by an **if** statement

Statements

- It is possible to have nested and cascaded if statements
  - if … else if constructs shall be terminated with an else statement
- A **switch statement** compares the value of an expression with multiple cases. Once the case match is found, the statement associated with the case is executed. If there is no match, a default statement will be executed.
  - Type of the expression can only be **integer** or **character**
  - Every switch statement shall have a default label
    - The default label shall appear as the first or the last label
  - The expression shall not have essentially **boolean** type
  - It is possible to have nested switch statements

```
switch (expression)
{
case CASE_1:
    /* Statement */
    break;

case CASE_2:
    /* Statement */
    break;

default:
    /* Statement */
    break;
}
```

```
if (expression)
{
    if (expression)
    {
    }
    else
    {
    }
}
else if (expression)
{
    if (expression)
    {
    }
}
else
{
}
```

Statements

- The case labels shall be integer or character constants; i.e. *case* ***constant****: statement*
- A switch statement shall have at least two switch-clause statements
- An unconditional break statement shall terminate every switch-clause statement
- In a multiple choice situation where only one branch is chosen, instead of multiple if..else statements it is better to use a switch statement
  - ➢ Your code will be more readable and maintainable
  - ➢ Generally a switch-statement has a better performance than an if-statement
- Note that a break ends a switch-clause statements
- Temporary local variables can only be declared within compound switch-clause statements.

```c
// This is NOT OK
switch (expression)
{
default:
    /* Statement */
    break;
}

// This is NOT OK
switch (expression)
{
case CASE_1:
default:
    /* Statement */
    break;
}
```

```c
// This is OK
switch (expression)
{
case CASE_1:
case CASE_2:
    /* Statement */
    break;

default:
    /* Statement */
    break;
}
```

Statements

- An **iteration statement** is used to execute a group of statements repeatedly in its body
  - The body of an iteration-statement shall be a compound-statement; i.e. enclosed in **{}**
- There are three kinds of loops; while, do… while, and for
  - The number of iterations is controlled by a **condition** which is called **controlling expression**
    - The controlling expression shall have essentially **boolean** type
    - The body repeatedly is executed as long as the controlling expression is true (1)
  - It is possible to jump out or back to the top of loops using
    - Unconditional jump statements; i.e. break, continue, return and goto
    - We should not have more than one break or goto statement to terminate an iteration
  - It is possible to have nested iteration statements
  -

Statements

- In a while statement first the controlling expression is evaluated.
  - If its value is true then the body will be executed
  - If the expression is always **true**, we have a forever loop

```
while (1)
{
    /* body */
}
```

```
while (expression)
{
    /* body */
}
```

- A do…while statement executes the body statement once before evaluation of the controlling expression (at least one iteration of the body is performed)

```
do
{
    /* body */
} while (expression);
```

- In a for statement there are three optional expressions
  - Generally for loops are used when we have a counter to control the loop

```
for ([initialization expression]; [controlling expression]; [update expression])
{
    /* the body of the for statement */
}
```

Statements

- The expressions in the head of a for loop are optional.
  - If none of them exists, we have a forever loop

```
for (;;)
{
    /* body */
}
```

- The **initialization expression** is used to perform any necessary initialization.
  - It is evaluated only once, before the first evaluation of the **controlling expression**.
  - Shall assign a value to the loop counter, or define and initialize the loop counter
  - Multiple loop variables can be declared and initialized using comma operator
- The **controlling expression** is tested before each iteration.
  - Loop execution ends when this expression evaluates to false.
  - Shall use the loop counter and optionally boolean loop control flags
- The **update expression** is performed after each iteration and before the controlling expression is tested again. It is used to update the loop counters (incrementation and decrementation)

Statements

- Variables declared in the **initialization expression** of a for loop are local
- In the **update expression** it is possible to use the comma operator to update multiple variables.

```
for (int i = 0, j = 100; i < 10 && j > 0; i++, j--)
{
    /* body */
}
```

  In this expression we shall not use objects that are modified in the for loop body
- A for-loop counter shall not have essentially floating type
- A for-loop counter shall not be be modified in the loop body

- Unconditional jump statements: break; continue; goto ***label***; return ***[expression]***;
- A break statement is used to jump to the first statement after a loop or a switch.
- A continue statement can only be used in the body of a loop to jump to the head of the loop
- A goto statement is used to jump unconditionally to a labeled statement in the same function

Statements

- Generally we should avoid using goto statements
- The return statement ends a function and jumps back to where the function was called
  - If there is an expression, its value will be returned to the caller
- A function should have no more than one return statement
- No more than one break or goto statement shall be used to terminate an iteration

```c
for (int i = 0; i < 10; i++)
{
    printf("%d ", i);
    if (i == 5)
    {
        break;
    }
}
```

```c
for (int i = 0; i < 100; i++)
{
    if (i % 2 == 0)
    {
        continue;
    }
    else
    {
        printf("%d\n", i);
    }
}
```