Shri Vile Parle Kelavani Mandal's

SHRI BHAGUBHAI MAFATLAL POLYTECHNIC

Computer Engineering Department

Control Structure Part - I

Course: Programming in C

Course Code: PRC238912

Name of Staff: Mr. Pratik H. Shah

SEMESTER : II

DIVISION : A

Reference: https://www.geeksforgeeks.org/decision-making-c-cpp/

Control Structures in Programming Languages

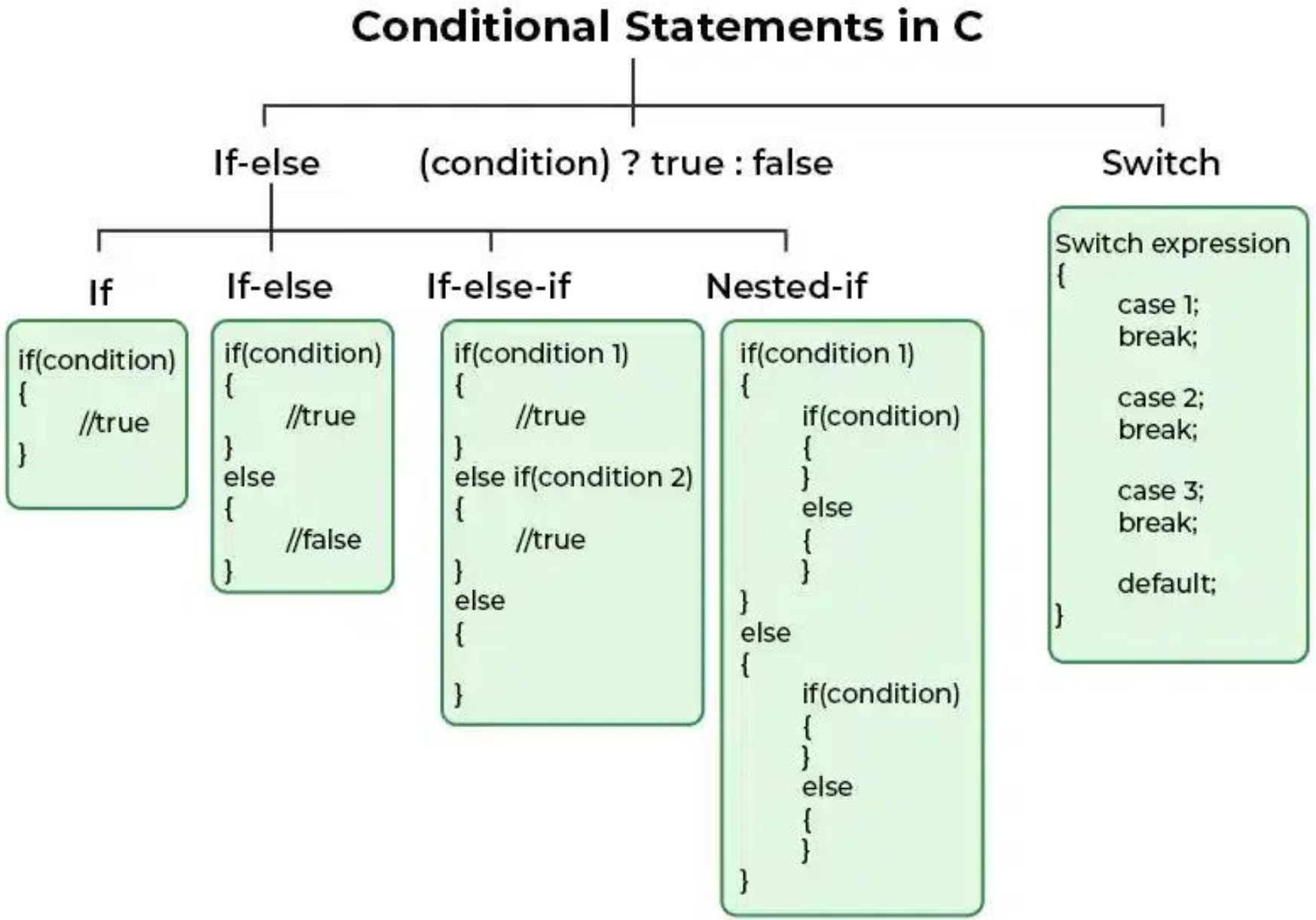
Control Structures are just a way to specify flow of control in programs. Any algorithm or program can be more clear and understood if they use self-contained modules called as logic or control structures. It basically analyzes and chooses in which direction a program flows based on certain parameters or conditions. There are three basic types of logic, or flow of control, known as:

1. Sequence logic, or sequential flow
2. Selection logic, or conditional flow
3. Iteration logic, or repetitive flow

Sequential Logic (Sequential Flow) Sequential logic as the name suggests follows a serial or sequential flow in which the flow depends on the series of instructions given to the computer. Unless new instructions are given, the modules are executed in the obvious sequence.

Selection Logic (Conditional Flow) Selection Logic simply involves a number of conditions or parameters which decides one out of several written modules. The structures which use these type of logic are known as **Conditional Structures**.

Iteration Logic (Repetitive Flow)
The Iteration logic employs a loop which involves a repeat statement followed by a module known as the body of a loop.


Conditional Statements in C
If-else
(condition) ? true : false
Switch
If
If-else
If-else-if
Nested-if
if(condition)
{
//true
}
if(condition)
{
//true
}
else
{
//false
}
if(condition 1)
{
//true
}
else if(condition 2)
{
//true
}
else
{
}
if(condition 1)
{
if(condition)
{
}
else
{
}
}
else
{
if(condition)
{
}
else
{
}
}
Switch expression
{
case 1;
break;
case 2;
break;
case 3;
break;
default;
}

1. if in C

The if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statements is executed otherwise not.

Syntax of if Statement

```
if(condition)
{
  // Statements to execute if
  // condition is true
}
```

Here, the **condition** after evaluation will be either true or false. C if statement accepts boolean values – if the value is true then it will execute the block of statements below it otherwise not. If we do not provide the curly braces '{' and '}' after if(condition) then by default if statement will consider the first immediately below statement to be inside its block.

```
// C program to illustrate If statement
#include <stdio.h>
void main()
{
    int i = 10;
    if (i > 15)
   {
        printf("10 is greater than 15");
    }
    printf("I am Not in if");
}
```

2. if-else in C

The *if* statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else when the condition is false? Here comes the C *else* statement. We can use the *else* statement with the *if* statement to execute a block of code when the condition is false. The if-else statement consists of two blocks, one for false expression and one for true expression.

Syntax of if else in C

```
if (condition)
{
   // Executes this block if
   // condition is true
}
else
{
   // Executes this block if
   // condition is false
}
```

```
// C program to illustrate If-else statement
#include <stdio.h>
void main()
{
    int i = 20;
    if (i < 15)
    {
        printf("i is smaller than 15");
    }
    else
   {
        printf("i is greater than 15");
    }
}
```

3. Nested if-else in C

A nested if in C is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, C allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

Syntax of Nested if-else

```
if (condition1)
{
  // Executes when condition1 is true
  if (condition_2)
  {
    // statement 1
  }
  else
  {
      // Statement 2
   }
}
else {
    if (condition_3)
  {
    // statement 3
  }
  else
  {
      // Statement 4
   }
}
```

```
// C program to illustrate nested-if statement
#include <stdio.h>
void main()
{
    int i = 10;
    if (i == 10) {
            if (i < 15)
                printf("i is smaller than 15\n");
            else
                printf("i is greater than 15");
    }
    else {
        if (i == 20) {
                if (i < 22)
                    printf("i is smaller than 22 too\n");
                else
                 printf("i is greater than 25");
        }
    }
}
```

4. if-else-if Ladder in C

The if else if statements are used when the user has to decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. if-else-if ladder is similar to the switch statement.

Syntax of if-else-if Ladder

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

```c
// C program to illustrate nested-if statement
#include <stdio.h>
void main()
{
    int i = 20;
    if (i == 10)
        printf("i is 10");
    else if (i == 15)
        printf("i is 15");
    else if (i == 20)
        printf("i is 20");
    else
        printf("i is not present");
}
```

Goto Statement

The goto statement in C also referred to as the unconditional jump statement can be used to jump from one point to another within a function.

Syntax of goto

```
Syntax1       |   Syntax2
--------------------------
goto label;   |    label:
.             |    .
.             |    .
.             |    .
label:        |    goto label;
```

In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here, a label is a user-defined identifier that indicates the target statement. The statement immediately followed after ‘label:’ is the destination statement. The ‘label:’ can also appear before the ‘goto label;’ statement in the above syntax.

```
// C program to print numbers from 1 to 10 using goto statement
#include <stdio.h>
void main()
{
    int n = 1;
label:
    printf("%d ", n);
    n++;
    if (n <= 10)
        goto label;
}
```

THANK YOU !!!!