4.1. Control Statements

Introduction

The statements which alter flow of execution of the program are known as control statements. In the absence of control statements, instructions are executed in same order in the program. Such instructions are called sequential statements. Sometimes we have to perform certain task depending on outcome of logical test. Similarly sometimes it is necessary to perform repeated action or skip some statements. For these cases we need control statements. They control flow of program. So that it is not necessary to execute statements in the same order in which they appear in program.

Any program logic, no matter how complex, could be expressed by using only the following three simple control structures:

a. Sequence Logic
b. Selection Logic and
c. Iteration (or Looping) Logic

Sequence Logic: Sequence logic is used to perform instruction to be followed after another in sequential order. The logic of flow the instruction from top to bottom. In this type of instruction the instruction starts top most instruction execute first then sequential statements will execute in top down hierarchy.

4.2. Selection Logic/Decision Making Statements

Selection is a special kind of branching, in which one group of statements is selected from several alternatives groups, i.e., it allows alternative actions to be taken according to conditions that exist at particular stage in program execution. The conditions are normally in the form of expressions that are evaluated test results (true or false).
Selection logic are following types:

a. if …
b. if … … else
c. nested if
d. switch statement
e. goto

Since decision making statements control the flow of execution, they also fall under the category of control statements.

4.2.1. If… statement

The if statement is a powerful decision making statement is used to control flow of execution of the program. It is basically a two way decision statements and it is within expression. The if evaluates the expression first then if the value of expression is true, it executes the statements within its block. Otherwise it skips statements within its block and continues form first statement outside the if block.

```
if (test expression)
{
        statement block;
}
statement x;
```

The statement block can be a single statement or group of statements. If the test expression is true the statements block within if is executed; otherwise the statement block will be skipped. The execution will jump to the statement x are executed in sequence order. The brace {} can be omitted if there is only one statement following the true condition of if statement.

4.2.2. if .. else Statement

The if.. else statement is an extension of simple if statement. It is used when there are two possible actions are to be evaluated. When a condition is true it executes statements inside if condition. Else the statement inside else is executed.
If the test condition is true, then true block statement, immediately following if statements are executed, otherwise, false block statement are executed. In either case true or false block will be executed, not both at the same time. This is best suit in only two choices or alternative.

4.2.3. If... else if... else Statement

The if... else if... else statement is used when there are more than two possible actions depending upon outcome of test condition. When specific action is satisfied the specific action only is executed, no other action will be executed. In such situation if... elseif... elseif.. else... structure is used. If the elseif statement is satisfied then else if condition is executed else last else statement is executed.
The condition is evaluated from top to button. As soon as true condition is found, the statements associated within it are executed and the control is transferred to statement X, at last the others are skipping the structure. When all conditions become false then final else containing default statement will be executed.

4.2.4. Nested if... else Statement

If an entire if..else construct is written under either the body of an if statement and the body of an else statement, then such type of construction is called nested if.... else statement. In other words nested if else is such type of construction in which there is an if..else statement within another if.. else statement. One if...else is nested into another if... else statement in nested form. In nested form the condition of inner statement is evaluated only if the condition of outer if is satisfied. Otherwise it is skipped and the else part of outer if is evaluated.

Dangling else Problem: One of the most common problem of nested if... else statement is dangling else. This occurs when a matching else is not available of an if statement. The solution to this is to always match an else to the most recent unmatched if in the current block.

4.3. Loops

4.3.1. Loop Operation

Loop may be defined as block of statements which are repeatedly executed for a certain number of times or until a particular condition is satisfied. When an identical task is to be performed for a numbers of times, then loop is used. The loop is executed repeatedly until an expression is true. When expression become false, the loop terminates and then control is passed to the another statements followed by the loop. A loop consists of two segments, one is known as the control statement and other is body of loop. The control statement in loop decides whether the body is to be executed or not.

Types of loops:

- For Loop
- While loop
- Do... While loop

4.3.2. for Loop

The for loop is useful to execute a statement for a number of times. When number of repetition action is required we use for loop is more efficient and reliable. Thus, this loop is also known as a determinant of definite data.

Syntax:

```
for (counter initialization; test condition; increment or decrement)
{
        statements; or body of loop;
}
```

Counter Initialization: The counter variable is initialized using assignment statements such as i=0,i=10 and count=0. Here the variable i, j, count are known as counter control variable whose value controls the loop execution. This expression or statement is executed only once prior to the statements within the for loop.

Test condition: The value of counter variable is tested using test condition. If the condition is true, the body of the loop is executed, otherwise the loop is terminated and execution continues with the statements that immediately follows the loop structure.

Update Expression/ Increment or Decrement: When body of loop is executed, the control is transferred back to statement after evaluating last statement in the body of loop. The counter variable is updated either increment or decrement and then only it is to be tested with that conditions and the same process is repeated until the condition become false.

The different components in for loop are executed in direction sequence as above.

At first the counter is initialized to some value. Then counter variable is tested with test condition. If test is true, body of loop is executed. After finishing body, the counter variable is incremented or decremented and then again update counter variable is tested with the condition. The same process repeated as long as the condition is true. If the result with test condition becomes false the control passes outside the loop.

4.3.3. while Loop

Syntax:

```
while(test condition)
{
        body of loop ;
}
```

The test condition is evaluated and if the condition is true then body of the loop is executed. After execution of data once, test condition is again evaluated and if it is true, the body is executed one again. This process of repeated execution of body continues until test condition finally becomes false and control is transferred out of loop. On exit, the program continues with the statement immediately after the body of loop.

4.3.4. do while Loop

Syntax:

```
do
{
        statement or body of loop;
}
while (condition);
```

In do while loop body of loop is executed first without testing condition. At the end of loop, test condition in the while statement is evaluated. If condition is true, program continues to evaluate the body of t he loop once again. This process continues as long as condition is true. When condition becomes false, the loop is terminated and then control goes to the statements that appears immediately after the while statement. Since the test condition is evaluated at bottom of the loop do while loop construct provides an exit controlled, or bottom of the loop and therefore body of loop is always executed at least once.

While	**Do while**
While loop is entry control loop i.e. test condition is evaluated first and body of loop is executed only if test is true	Do while loop is exit control loop i.e. the body of the loop is executed first without checking condition and at the end of body of loop, the condition is evaluated.
The body of the loop may not be executed at all if the condition is not satisfied at the very first attempt.	The body of loop is always executed at least once

4.3.5. Nested Loop

When the body part of the loop contains another loop then the inner loop is said to be nested within the outer loop. Since a loop may contain other loops within its body, there is no limit of the number of loops that can be nested. In the case of nested loop, for each value of outer loop, inner loop is completely executed. Thus inner loop operates fast and outer loop operates slowly.

4.4. break, continue and goto Statement

4.4.1. break Statement

The brake statement terminates the execution of loop and controls are transferred to the statement immediately following the loop. Generally the loop is terminated when its test condition becomes false. But if we have to terminate the loop instantly without testing loop termination condition, the statement is useful. The syntax: break;
Break statement is also used in switch statement which causes a transfer of control out of the entire switch statement. To the first statement following the switch statement to the first following the stitch statement. The switch statement will be discussed later.

4.4.2. continue Statement

The continue statement is used to bypass expression of current process through a loop. The loop does not terminate when a continue statement is encountered. Instead of termination, it skipped some portion some part of program and the computation proceeds directly to the next stage through the loop. syntax is: continue;

4.4.3. goto Statement

The goto statement is used to alter normal sequence of program execution by unconditionally transferring control to some other part of program. The go to statement transfer control to the labeled statement somewhere in current function. The go to statement has the following syntax : goto label; Here label is an identifier used to the target statement to which control would be transferred. The target statement must be labeled using syntax: label statements:
Generally use of go to statement is avoided as it makes program illegible. This statement is used in unique statements like:

- Branching around statements or groups of statements under certain condition.
- Jumping to end of a loop under certain condition, thus passing remainder of loop during current pass.
- Jumping completely out of loop under certain conditions, Termination executions of a loop.

4.5. switch Statement

When there are number cases to be handled, switch statement is another way of representing this multi way selection. A switch statement allows user to choose a statement among several alternatives. The switch statement is useful when a variable is to be compared with different constants and in case it is equal to a constant a set of statements are to be executed. The constants in case statement may be either char or int type only.

Syntax:

```
switch (variable name)
{
        case statements:
                break;
        case statements:
                break;
        default:
                 statements;
}
```