Chapter 2: Program Control Constructs (Conditional and Iterative Statements)

Lesson 1: if-else

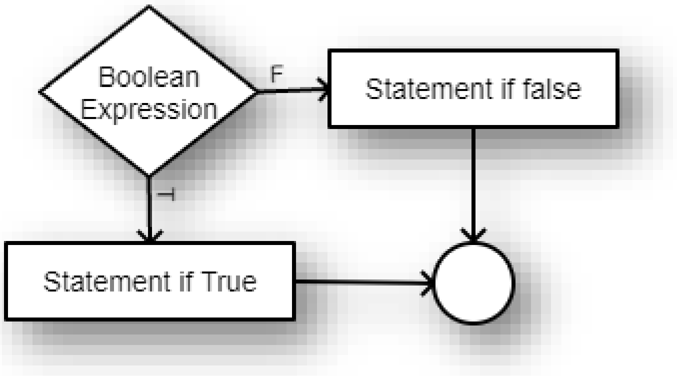
Learning Outcomes:

- Solve programming problems with conditional components using if-else statement
- Cascade if-else statement to form much complex decision path.

A computer program doesn't only comprise of a sequential piece of code; it is mostly composed of a logical condition to alter the sequential path. A conditional statement instructs the computer to execute a certain block of code or alter certain data only if a specific condition has been met. In this chapter, the logical analysis and implementation of conditional constructs would be covered.

Single path condition

The if-else statement in C can be visualized using a flowchart below.



The actual format of C code is as follows:

```
if (boolean_expression)
{
      //statement if true
}
else
{
      //statement if false
}
```

The decision making logic in C is treated as an arithmetic operation, the value 0 (zero) represents false and any other value as true. So, in an if-else statement the Boolean expression within the parenthesis is evaluated only as true or false. Typically the relational, equality and logical operators would be used to perform Boolean expressions. For example, The code below would print the “Your can vote” if age greater than or equal to 18 otherwise it would print you cannot vote.

```
int age = 0;
printf("Enter Age: ");
scanf("%d",&age);
if (age>=18)
{
    printf("You can vote!");
```

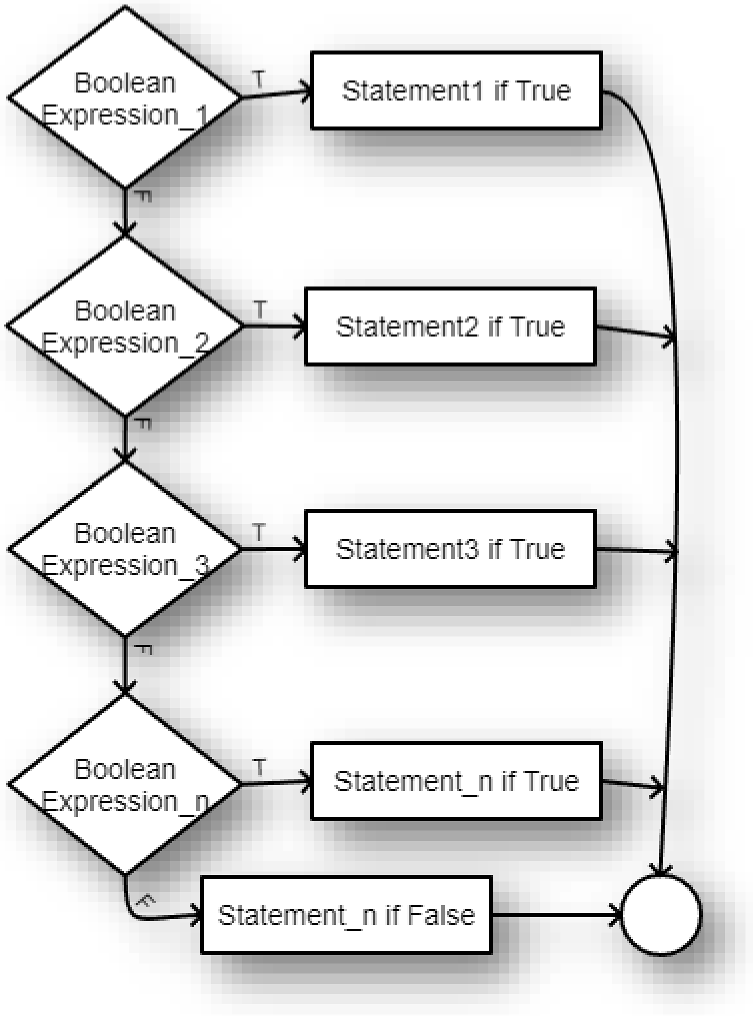
```
        }
        else
        {
            printf("You cannot vote!");
        }
```

Take note that the assignment operator is different from the equality operator. For example:

```
int a=10;
if (a==11) //evaluated as false since a is 10 thus no equal to 11
if (a=11) //evaluated as true since a is not zero.
```

Multiple path condition

By cascading several single path condition, it can be extended to implement a multiple path condition. The flowchart below illustrates this operation.



The C format of if-else-if is as follows:

```
if(boolean_expression_1)
{
//statements
}
else if(boolean_expression_2)
{
//statements
}
else if(boolean_expression_3)
{
//statements
}
else if(boolean_expression_n)
{
//statements
}
```

The previous example of the age can further be extended to include different age brackets in order to illustrate multiple condition.

```
int age = 0;
printf("Enter Age: ");
scanf("%d",&age);

if (age>=18 && age<=34)
{
    printf("You're a millennial");
}
else if (age>=35 && age<=50)
{
    printf("You're a Gen X");
}
else if (age>=51 && age<=69)
{
    printf("You're a Baby Boomer");
}
else if (age>=70 && age<=87)
{
    printf("You're a Silent Generation");
}
 else
 {
     printf ("No classification for your age");
 }
```

This code would display the appropriate bracket of a particular age. For instance if the inputted age is 39 the output would be “You’re a Gen X”.

Lesson 2: switch-case statement and Nesting Conditional

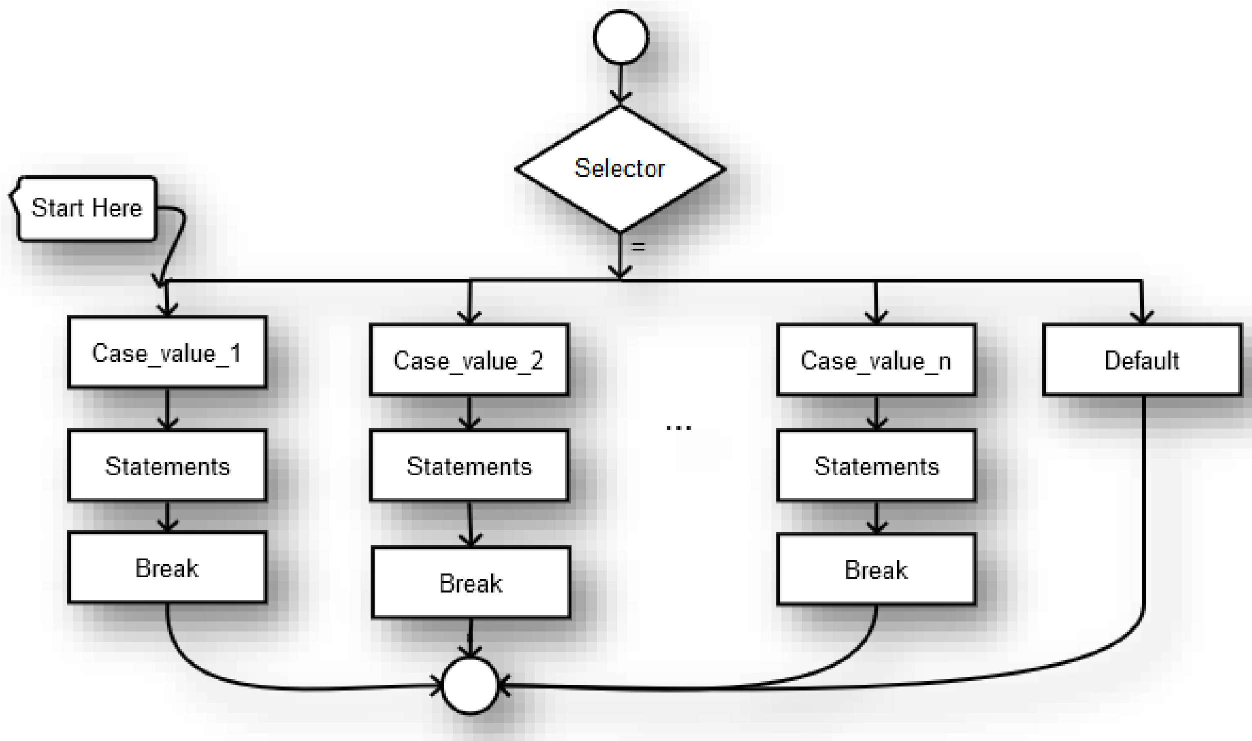
Learning Outcomes:

- Implement a switch-case on conditional related problems
- Nest if-else with switch-case statements

A switch-case statement performs conditional branching by equality matching an integral value (char or int) to a particular case constant. It can be used as an alternative to if-else especially when creating a selection menu. Moreover, by nesting it with if-else more complex decision path can be achieved.

Switch-case structure

By chaining multiple if-else-if-else-if-else can be a pain to read programmers. However, the switch-case construct simplify this by taking an integral value (int or char) then comparing it to the value specified by the case. When specified case would be match, it would “activate” that particular case, then exits when it would encounter break statement. Nevertheless, when there is no matching case exist it would go into default case. The flowchart below summarizes the switch-case structure.



The C construct of the switch case is as follows.

```
int/char selector = 1;
switch (selector)
{
Case 1:
      //statements;
      break;
Case 2:
      //statements;
      break;
Case n:
      //statements;
```

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

Suppose you would write a program that would accept an input from 1-5 that correspond to student grades and display its literal equivalent in which 1 correspond to A, 2 to B and so on.

```
int grade = 1;
    switch (grade)
    {
        case 1:
            printf("Grade: A");
            break;
        case 2:
            printf("Grade: B");
            break;
        case 3:
            printf("Grade: C");
            break;
        case 4:
            printf("Grade: D");
            break;
        case 5:
            printf("Grade: E");
            break;
        default:
             printf("No matching grade");

    }
```

The example above would display the output “Grade: A” since the selector (i.e. grade) is assigned with a value of 1.

Nesting conditional statements

By nesting together the if-else statement together with a case statement or other if-else statement you could create much complex decision path. For instance you would have a menu to convert cm to inches or the other way around. Moreover, the zero input would be filtered accordingly. To do this you have a case statement and within the case statement is an if statement.

```
char menu;
    int varToConvert;
    printf("[I] Inches to CM \n[C] CM to Inches \n");
    scanf("%c",&menu);
    switch(toupper(menu))
      //the toupper  - converts any character input to uppercase
    {
        case 'I':
            printf("Enter inches: ");
            scanf("%d",&varToConvert);
            if (varToConvert>0)
            {
                printf("%d Inches is Equal to: %f
CM",varToConvert,varToConvert*2.54);
            }
            break;
        case 'C':
            printf("Enter CM: ");
            scanf("%d",&varToConvert);
            if (varToConvert>0)
            {
                printf("%d CM is Equal to: %f Inches
",varToConvert,varToConvert/2.54);
            }
            break;
        default:
            printf("No such conversion exist!");
    }
```

The output would be as follows:

```
[I] Inches to CM
[C] CM to Inches
```

When a character input is place for instance I, it would direct you to the path of inches – CM conversion as follows.

```
[I] Inches to CM
[C] CM to Inches
I
Enter inches:
```

When a positive integer input for instance 10 is made then it would display the output below.

```
[I] Inches to CM
[C] CM to Inches
I
Enter inches: 10
10 Inches is Equal to: 25.400000 CM
```

Lesson 3: Pretest Iterations

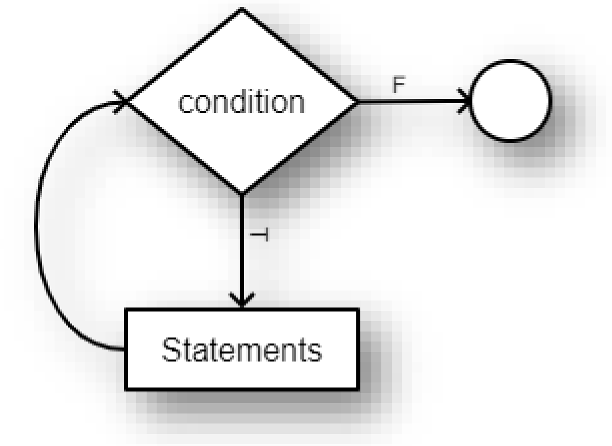
Learning Outcomes:

- Trace the step-by-step process(es) when executing a loop
- Solve iteration related programming problems using while loop and for loop

Numerous cases in which a certain statement would be repeated a certain number of times before a condition is met. It is tedious and practically unrealistic to perform repetition by only using conditional statements. For example, it is better to state “Walk until destination is reached” than to state “Walk then, walk then walk…. Until destination is reached”. The iterative construct would perform repetitive tasked also called as loops.

While loop

The while loop is the most basic kind of loop. It would perform a block of statements as long as the condition remains true. If the condition would be falsified, then and only then will the loop terminate. Refer to the flowchart below.



A while loop construct is shown by the format below.

```
while (condition)
{
      statements;
}
```

A while loop that would not exit (i.e. the loop condition is always true) as shown on the example below is also known as an infinite loop.

```
int x;
while (5)
{
      printf("%d,",x);
}
```

In order to make the loop terminate, you must find a way that a process within the loop body would falsify the loop condition. By modifying previous example, it would now terminate

```
int x = 0;
while (x<5)
{
      printf("%d,",x);
      x++;
}
```

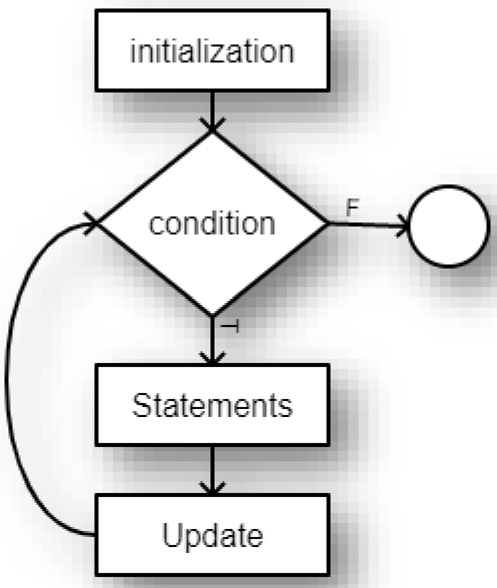
The output of the code is

```
0,1,2,3,4
```

Since you are displaying the value of variable x at each cycle of the loop.

For loop

A for loop is a condensed expression of a while loop. All the necessary items needed to initialize, terminate and update the loop is on one line. It can be illustrated by the flowchart below



The general form of for loop is:

```
for (initialization;condition;update)
{
      //statements
}
```

The while loop example can be expressed on a for loop construct using the example below.

```
int x;
for (x=0;x<5;x++)
{
      printf("%d,", x);
}
```

It is also common to use the comma operator within the loop initialization and update fields to indicate multiple variable initialization and multiple update statements respectively.

```
int i,j;
for (i=0,j=5;i<5;j++,i++)
```

```
{
      printf("%d, %d \n",i,j);
}
```

The output of the code is

```
0,5,1,6,2,7,3,8,4,9,
```

Since the i and j variables are updated and displayed on the same line

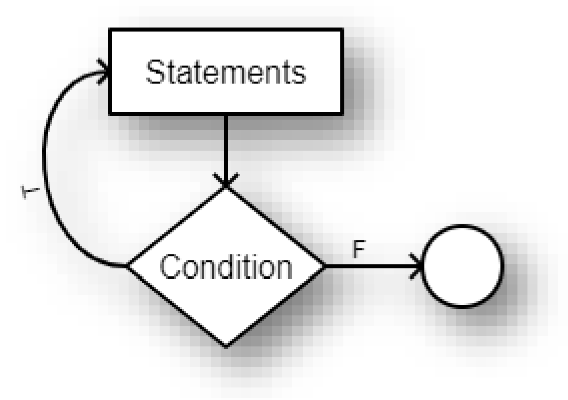
Lesson 4: Posttest Iteration, Break and Continue Statements

Learning Outcomes

- Solve programming problems using do-while loop
- Control loop flow and termination using continue and break statements respectively

Do-while loop

The do-while loop would check the loop condition after the loop body. As a result it would execute the loop body at least once even if the loop condition is false. The loop structure is illustrated by the flowchart below.



The framework of the loop structure in C statement is:

```
do
{
//statements
} while(condition);
```

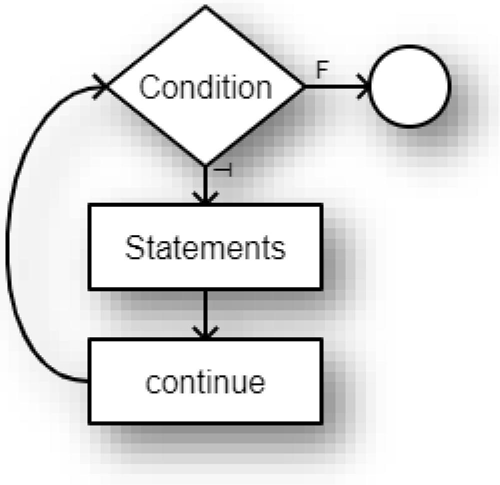
The example below illustrates that the do while loop will execute even if the loop condition is false.

```
int i = 0
do
{
    printf("%d,"x);
}while (i>0);
```

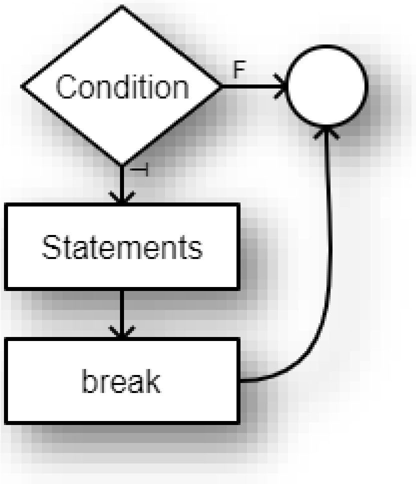
The output is 0 even if the loop condition is if i>0 which is false.

The break and continue

A break continue statement within a loop structure jumps to the loop condition and skips all the succeeding code from it. The flowchart below illustrates this operation.



On the other hand a break statement pre-emptively would exit a loop. Refer to the flowchart below.



The example below shows the usage of the break, continue and do-while.

```
int i=0;
   do
   {  i++;
      if (i<3)
      {
            printf("%d, ",i);
        }
        else if (i<4)
        {
            continue;
        }
        else if (i<5)
        {
            break;
        }
    }while(i<6);
```

The output of the code is only

```
1,2
```

Since, the control value i would skip the break statement if i is greater than or equal to 3 but lesser than 4 and will break the loop structure (exit the loop) if i will be greater than or equal to 4.

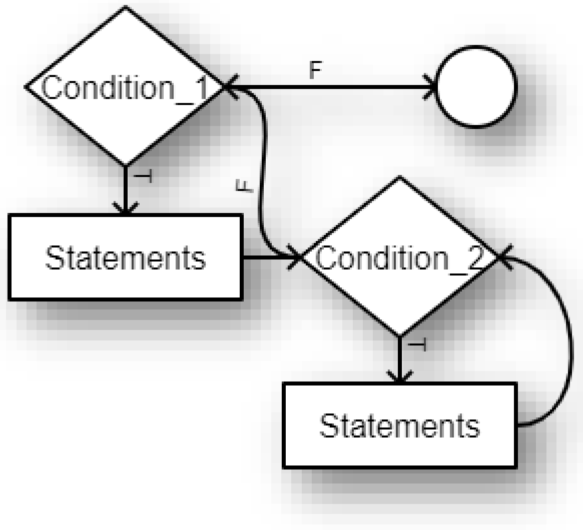
Lesson 5: Nested Loops

Learning Outcome

- Visualize nested loops and solve problems related to it

Nested Loop diagram and examples

Nested loop can be created by placing a loop structure inside another loop structure. Refer to the flowchart representation below.



d

Although there could be multiple levels (i.e. loop within a loop), most common implementations are limited to just 2 to 3 nested loops. For example, a multiplication table can be implemented using a nested loop.

```
int i=1, j=1;
for (i=1;i<=5;i++)
{
    for (j=1;j<=5;j++)
    {
        printf("%d,",i*j);
    }
    printf("\n");
}
```

The output of the code is:

```
1,2,3,4,5,
2,4,6,8,10,
3,6,9,12,15,
4,8,12,16,20,
5,10,15,20,25,
```

Since each inner loop it would display the product of i and j. In which j is the control value for the inner loop and i is the control value of the outer loop. So, each time the inner loop would execute the value of j would be reset to 1 and multiplied by the current value of i. When the inner loop would terminate a newline would be displayed, thus moving the cursor to the next line. Repeating the pattern until the last value of i would effectively implement the multiplication table.