Lab Session 05

Decision Making with Switch structure and Comparison with If-else statement

Objectives:

1. Demonstration of Decision Making with Switch structure
2. Analyze switch structure with If-else statement
3. Design and Compilation of C++ programs using switch statement.

Switch Statements:

Switch statement falls in the category of selection statements like if and else. C++ provides three types of selection statements. The if selection statement either performs (selects) an action if a condition is true or skips the action if the condition is false. The if…else selection statement performs an action if a condition is true or performs a different action if the condition is false.

The switch selection statement performs one of many different actions, depending on the value of an integer expression. The switch selection statement is called a multiple-selection statement because it selects among many different actions (or groups of actions). Each action is associated with the value of a constant integral expression (i.e., any combination of character and integer constants that evaluates to a constant integer value).

If you have a large decision tree, and all the decisions depend on the value of the same variable, you will probably want to consider a switch statement instead of a ladder of if...else or else if constructions. Here's a simple example:

// demonstrates SWITCH statement

```
#include <iostream>
using namespace std;
int main()
{
int speed; // speed
cout << "\nEnter 80, 120, or 130 : ";
cin >> speed; //user enters speed
switch(speed) //selection based on speed
{
case 80: //user entered 80
cout << " Highway Normal Speedn";
break;
case 120: //user entered 120
```

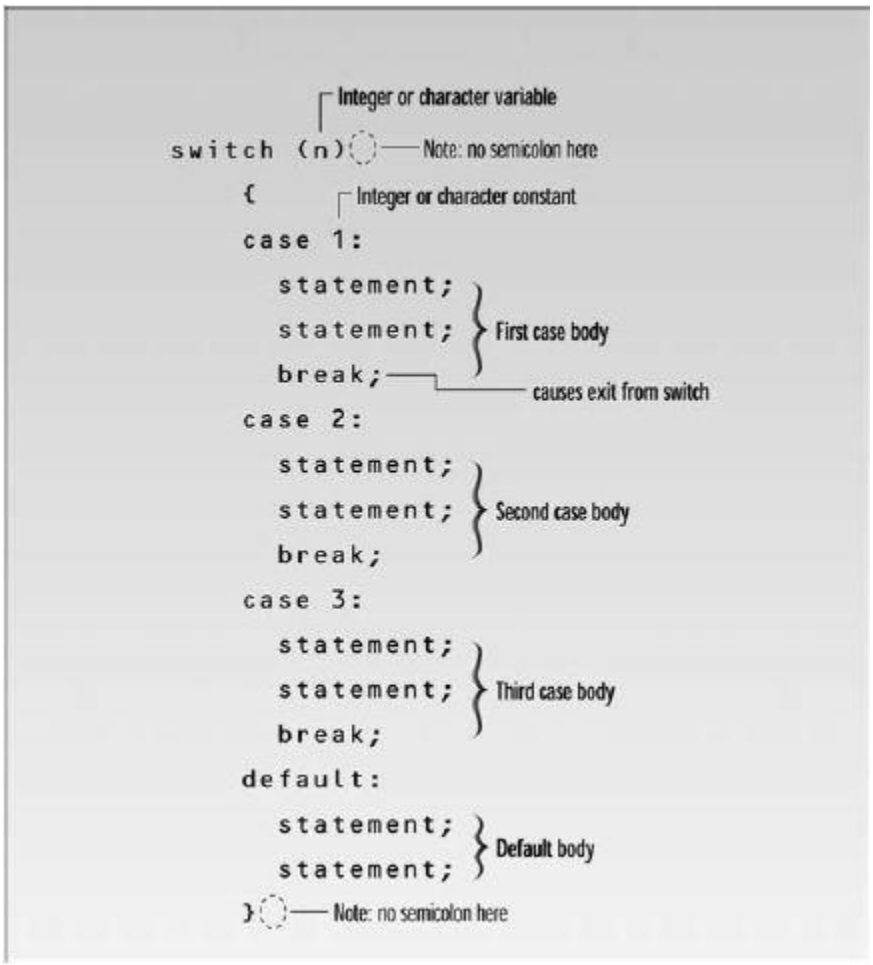
```
cout << "High Speed\n";
break;
case 130: //user entered 130
cout << "Over Speed\n";
break;
}
return 0;
}
```

This program prints one of three possible messages, depending on whether the user inputs the number 80, 120, or 130 then displays message associated w.r.t. speed.

The keyword switch is followed by a switch variable in parentheses.

switch(speed)

Syntax of the switch statement:



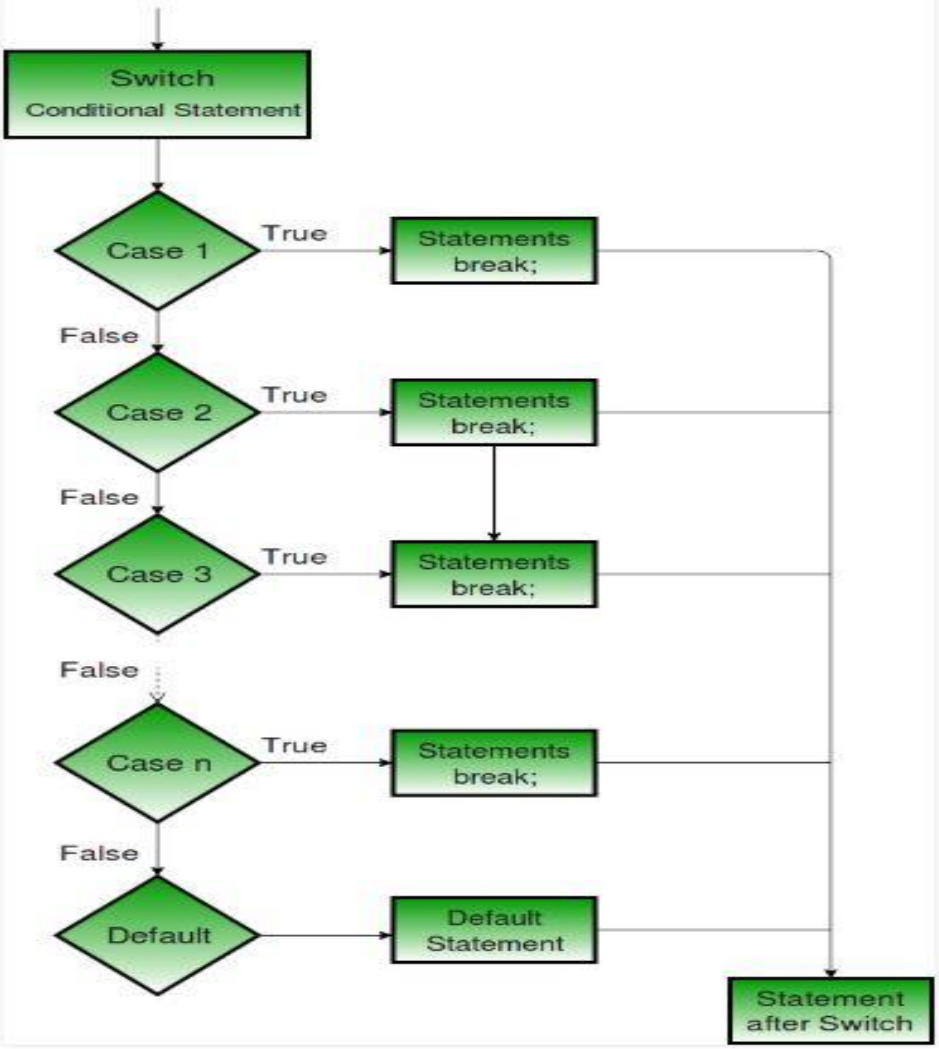
Syntax of the switch statement

Before entering the switch, the program should assign a value to the switch variable. This value will usually match a constant in one of the case statements. When this is the case (pun intended!), the statements immediately following the keyword case will be executed, until a break is reached.

The break Statement:

Above program has a break statement at the end of each case section. The break keyword causes the entire switch statement to exit. Control goes to the first statement following the end of the switch construction, which is the end of the program. Don't forget the break; without it, control passes down (or "falls through") to the statements for the next case, which is usually not what you want (although sometimes it's useful).

If the value of the switch variable doesn't match any of the case constants, control passes to the end of the switch without doing anything. The operation of the switch statement is shown in Figure below.



Operation of the switch statement

Important Points about Switch Case Statements:

1. The expression provided in the switch should result in a **constant value** otherwise it would not be valid.
   Valid expressions for switch:

```
// Constant expressions allowed

  switch(1+2+23)
```

```
switch(1*2+3%4)
```

Invalid switch expressions for switch:

```
// Variable expression not allowed

switch(ab+cd)

switch(a+b+c)
```

2. Duplicate case values are not allowed.
3. The default statement is optional.Even if the switch case statement do not have a default statement, it would run without any problem.
4. The break statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
5. The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.
6. Nesting of switch statements are allowed, which means you can have switch statements inside another switch. However nested switch statements should be avoided as it makes program more complex and less readable.

The default Keyword:

In the ADSWITCH program, where you expect to see the last case at the bottom of the switch construction, you instead see the keyword default. This keyword gives the switch construction a way to take an action if the value of the loop variable doesn't match any of the case constants. Here we use it to print Try again if the user types an unknown character. No break is necessary after default, since we're at the end of the switch anyway.

switch statement is a common approach to analyzing input entered by the user. Each of the possible characters is represented by a case. It's a good idea to use a default statement in all switch statements, even if you don't think you need it. A construction such as default:

cout << "Error: incorrect input to switch"; break;

It alerts the programmer (or the user) that something has gone wrong in the operation of the program. In the interest of brevity we don't always include such a default statement, but you should, especially in serious programs.

Example: Program to built a simple calculator using switch Statement

```
// Program to built a simple calculator using switch Statement
#include <iostream>
using namespace std;

int main()
{
    char o;
    float num1, num2;

    cout << "Enter an operator (+, -, *, /): ";
    cin >> o;

    cout << "Enter two operands: ";
    cin >> num1 >> num2;

    switch (o)
    {
        case '+':
            cout << num1 << " + " << num2 << " = " << num1+num2;
            break;
        case '-':
            cout << num1 << " - " << num2 << " = " << num1-num2;
            break;
        case '*':
            cout << num1 << " * " << num2 << " = " << num1*num2;
            break;
```

```
        case '/':
            cout << num1 << " / " << num2 << " = " << num1/num2;
            break;
        default:
            // operator is doesn't match any case constant (+, -, *, /)
            cout << "Error! operator is not correct";
            break;
    }

    return 0;
}
```

Output:

```
Enter an operator (+, -, *, /): +

-

Enter two operands: 2.3

4.5

2.3 - 4.5 = -2.2
```

The - operator entered by the user is stored in `o` variable. And, two operands 2.3 and 4.5 are stored in variables `num1` and `num2` respectively.Then, the control of the program jumps to

`cout << num1 << " - " << num2 << " = " << num1-num2;`Finally, the break statement ends the switch statement.If break statement is not used, all cases after the correct case is executed.

Example: Program that displays grade

```
#include <iostream>

using namespace std;


int main () {

   // local variable declaration:

   char grade = 'D';


   switch(grade) {

      case 'A' :

         cout << "Excellent!" << endl;

         break;

      case 'B' :

      case 'C' :

         cout << "Well done" << endl;

         break;

      case 'D' :

         cout << "You passed" << endl;

         break;

      case 'F' :

         cout << "Better try again" << endl;

         break;

      default :

         cout << "Invalid grade" << endl;

   }

   cout << "Your grade is " << grade << endl;
```

```
   return 0;

}
```

This would produce the following result −

```
You passed
Your grade is D
```

switch Versus if...else

When do you use a series of if...else (or else if) statements, and when do you use a switch statement? In an else if construction you can use a series of expressions that involve unrelated variables and are as complex as you like.

Both of the following code fragments have the same behavior, demonstrating the if-else equivalent of a switch statement:

For example:

switch example	if-else equivalent

`switch (x) {`
`  case 1:`
`  cout << "x is 1";`
`  break;`
`  case 2:`
`  cout << "x is 2";`
`  break;`
`  default:`
`  cout << "value of x unknown";`
`  }`
	`if (x == 1) {`
	`  cout << "x is 1";`
	`}`
	`else if (x == 2) {`
	`  cout << "x is 2";`
	`}`
	`else {`
	`  cout << "value of x unknown";`
	`}`

You are having trouble following the logic of the switch statement, it is essentially the same as writing an if statement for each case statement:
For example:

```
if ( 1 == input )
{
playgame();
}
else if ( 2 == input )
{
loadgame();
}
else if ( 3 == input )
{
playmultiplayer();
}
else if ( 4 == input )
{
cout << "Thank you for playing!\n";
```

```
}
else
{
cout << "Error, bad input, quitting\n";
}
```

If we can do the same thing with an if/else, why do we need a switch at all? The main advantage of the switch is that it's quite clear how the program flow works: a single variable controls the code path. With a series of if/else conditions, each condition needs to be carefully read.

Assignment:

1. **What is The getche() Library Function.**
2. **Elaborate 'continue' statement used in C++.**
3. **Write C++ statements to find vowels.**