

CHAPTER THREE

CONTROL STRUCTURES

Introduction

A running program spends all of its time executing statements. The order in which statements are executed is called flow control (or control flow).

This term reflects the fact that the currently executing statement has the control of the CPU, which when completed will be handed over (flow) to another statement.

Flow control in a program is typically sequential, from one statement to the next, but may be diverted to other paths by branch statements.

Flow control is an important consideration because it determines what is executed during a run and what is not, therefore affecting the overall outcome of the program.

Cont...

Like many other procedural languages, C++ provides different forms of statements for different purposes. Declaration statements are used for defining variables. Assignment like statements are used for simple, algebraic computations.

Branching statements are used for specifying alternate paths of execution, depending on the outcome of a logical condition.

Loop statements are used for specifying computations, which need to be repeated until a certain logical condition is satisfied.

Flow control statements are used to divert the execution path to another part of the program.

(Decision) Conditional Statements

The if Statement

- It is sometimes desirable to make the execution of a statement dependent upon a condition being satisfied.
- The if statement provides a way of expressing this, the general form of which is:

```
if (expression)  
    statement;
```

Cont....

- First expression is evaluated.
- If the outcome is nonzero (true) then statement is executed.
- Otherwise, nothing happens.
- To make multiple statements dependent on the same condition, we can use a compound statement:

```
if (count != 0)
    average = sum / count;
```

```
if (balance > 0) {
    interest = balance * creditRate;
    balance += interest;
}
```

if-else

- A variant form of the if statement allows us to specify two alternative statements: one which is executed if a condition is satisfied and one which is executed if the condition is not satisfied.
- This is called the if-else statement and has the general form:

```
if (expression)  
    statement1;  
else  
    statement2;
```

Cont..

- First expression is evaluated. If the outcome is nonzero (true) then statement1 is executed. Otherwise, statement2 is executed.

```
if (balance > 0) {  
    interest = balance * creditRate;  
    balance += interest;  
}  
else {  
    interest = balance * debitRate;  
    balance += interest;  
}
```

if else if

- First “expression1” is evaluated and if the outcome is none zero (true), then “statements1” will be executed.
- Otherwise, which means the “expression1” then “expression2” will be evaluated: if the out put of expression2 is True then “statements2” will be executed otherwise the next “expression” in the else if statement will be executed and so on.
- If all the expressions in the “if” and “else if” statements are False then “statements” under else will be executed

its General Syntax is:

```
if (expression1)
    statements1; else
    if(expression2)
        statements2;
        :
    else if(expressionN)
        statementsN;
    else
        statements
```

Cont...

E.g.:

```
if(score >=90)
    cout<< "\n your grade is A";
else if(score >= 80)
    cout<< "\n your grade is B";
else if(score >=70)
    cout<< "\n your grade is C";
else if(score >= 60)
    cout<< "\n your grade is D";
else
    cout<< "\n your grade is F";
```

The switch Statement

- The switch statement provides a way of choosing between a set of alternatives, based on the value of an expression.
- The switch statement has four components: *f*
 - ❖ Switch *f*
 - ❖ Case *f*
 - ❖ Default *f*
 - ❖ Break
- The general form of the switch statement is:

```
switch(expression)
{
    case constant1:
        statements;
        .
        .
        .
    case constant n:
        statements;
    default:
        statements;
}
```

Cont...

```
switch ( choice) {  
int num1,num2,sum,mul;  
case 1:  
{  
cout<<"enter the numbers"<<endl;  
cin>>num1>>num2;  
sum=num1+num2;  
cout<<sum;  
break;  
}  
case 2:{  
cout<<"enter the numbers"<<endl;  
cin>>num1>>num2;  
mul=num1*num2;  
cout<<mul;  
break;}  
default:  
cout << "wrong choice"; }  
return 0;  
}
```

Looping Statements

- Loop statements are used for specifying computations, which need to be repeated until a certain logical condition is satisfied.
- There are three C++ repetition statements:
 - 1) The For Statement or loop
 - 2) The While statement or loop
 - 3) The do...while statement or loop

The 'while' Statement

- The while statement (also called while loop) provides a way of repeating a statement while a condition holds.
- The general form of the while statement is:

```
while (expression)  
    statement;
```

Cont...

- First expression (called the loop condition) is evaluated.
- If the outcome is nonzero then statement (called the loop body) is executed and the whole process is repeated.
- Otherwise, the loop is terminated.

The 'for' statement

- The 'for' statement (also called for loop) is similar to the while statement, but has two additional components:
 - an expression which is evaluated only once before everything else, and
 - an expression which is evaluated once at the end of each iteration.
- The general form of the 'for' statement is:

*for (expression₁; expression₂; expression₃)
statement;*

Cont....

- First expression1 is evaluated.
- Each time the loop, expression2 is evaluated.
 - If the outcome is nonzero then statement is executed and expression3 is evaluated.
- Otherwise, the loop is terminated. The general for loop is equivalent to the following while loop:

The 'do...while' Statement

- The do statement (also called do loop) is similar to the while statement, except that its body is executed first and then the loop condition is examined.
- The general form of the do statement is:

```
do  
    statement;  
while (expression);
```

Cont....

- First statement is executed and then expression is evaluated.
- If the outcome of the latter is nonzero then the whole process is repeated.
- Otherwise, the loop is terminated.
- The do loop is less frequently used than the while loop.
- It is useful for situations where we need the loop body to be executed at least once, regardless of the loop condition.

Cont...

- For example, suppose we wish to repeatedly read a value and print its square, and stop when the value is zero.
- This can be expressed as the following loop:

```
do {  
    cin >> n;  
    cout << n * n << '\n';  
} while (n != 0);
```

Other Statements

The 'continue' Statement

- The continue statement terminates the current iteration of a loop and instead jumps to the next iteration.
- It applies to the loop immediately enclosing the continue statement.
- It is an error to use the continue statement outside a loop.
- In while and do loops, the next iteration commences from the loop condition.
- In a for loop, the next iteration commences from the loop's third expression.

Cont....

- For example, a loop which repeatedly reads in a number, processes it but ignores negative numbers, and terminates when the number is zero, may be expressed as:

```
do {  
    cin >> num;  
    if (num < 0) continue;  
    // process num here...  
} while (num != 0);
```

This is equivalent to:

```
do {  
    cin >> num;  
    if (num >= 0) {  
        // process num here...  
    }  
} while (num != 0);
```

Cont...

- When the continue statement appears inside nested loops, it applies to the loop immediately enclosing it, and not to the outer loops. For example, in the following set of nested loops, the continue applies to the for loop, and not the while loop:

```
while (more) {  
    for (i = 0; i < n; ++i) {  
        cin >> num;  
        if (num < 0) continue;           // causes a jump to: ++i  
        // process num here...  
    }  
    //etc...  
}
```

The 'break' Statement

- A break statement may appear inside a loop (while, do, or for) or a switch statement.
- It causes a jump out of these constructs, and hence terminates them.
- Like the continue statement, a break statement only applies to the loop or switch immediately enclosing it.
- It is an error to use the break statement outside a loop or a switch.

Cont....

- For example, suppose we wish to read in a user password, but would like to allow the user a limited number of attempts:

```
for (i = 0; i < attempts; ++i) {  
    cout << "Please enter your password: ";  
    cin >> password;  
    if (Verify(password))    // check password for correctness  
        break;              // drop out of the loop  
    cout << "Incorrect!\n";  
}
```