Constructing Expressions

Introduction

There are techniques you can use to combine conditional statements when one of them cannot fully implement the desired behavior.

We will continue with our traffic light analogy.

Combining Statements

Nesting Conditions

A condition can be created inside of another to write a more effective statement. This is referred to as nesting conditions. Almost any condition can be part of another and multiple conditions can be included inside of others.

As we have learned, different conditional statements are applied in specific circumstances. In some situations, they are interchangeable or one can be applied just like another, which becomes a matter of choice. Statements can be combined to render a better result with each playing an appropriate role.

To continue with our ergonomic program, imagine that you would really like the user to sit down and your program would continue only once she answers that she is sitting down, you can use the do…while statement to wait for the user to sit down; but as the do…while is checking the condition, you can insert an if statement to enforce your request. Here is an example of how you can do it:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
char SittingDown;

    do {
        cout << "Are you sitting down now(y/n)? ";
        cin >> SittingDown;

        if( SittingDown != 'y' )
```



```
                cout << "Could you please sit down for the next exercise?";
        cout << "\n\n";
    } while( !(SittingDown == 'y') );

    cout << "Wonderful!!!";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Here is an example of running the program:

```
Are you sitting down now(y/n)? n
Could you please sit down for the next exercise?

Are you sitting down now(y/n)? n
Could you please sit down for the next exercise?

Are you sitting down now(y/n)? y

Wonderful!!!

Press any key to continue...
```

One of the reasons you would need to nest conditions is because one would lead to another. Sometimes, before checking one condition, another primary condition would have to be met. The ergonomic program we have been simulating so far is asking the user whether she is sitting down. Once the user is sitting down, you would write an exercise she would perform. Depending on her strength, at a certain time, one user will be tired and want to stop while for the same amount of previous exercises, another user would like to continue. Before continuing with a subsequent exercise, you may want to check whether the user would like to continue. Of course, this would be easily done with:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char SittingDown;

    do {
        cout << "Are you sitting down now(y/n)? ";
        cin >> SittingDown;

        if( SittingDown != 'y' )
                cout << "Could you please sit down for the next exercise?";
        cout << "\n\n";
    }while( SittingDown != 'y' );

    cout << "Wonderful. Now we will continue today's exercise...";
    cout << "\n...\nEnd of exercise";

    char WantToContinue;

    cout << "Do you want to continue(y=Yes/n=No)? ";
    cin >> WantToContinue;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
```

```
}
//---------------------------------------------------------------------------
```

If the user answers No, you can stop the program. If she answers Yes, you would need to continue the program with another exercise. Because the user answered Yes, the subsequent exercise would be included in the previous condition because it does not apply for a user who wants to stop. In this case, one “if” could be inserted inside of another. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char SittingDown;

    do {
        cout << "Are you sitting down now(y/n)? ";
        cin >> SittingDown;

        if( SittingDown != 'y' )
            cout << "Could you please sit down for the next exercise?";
        cout << "\n\n";
    }while( SittingDown != 'y' );

    cout << "Wonderful. Now we will continue today's exercise...\n";
    cout << "\n...\n\nEnd of exercise\n";

    char WantToContinue;

    cout << "\nDo you want to continue(1=Yes/0=No)? ";
    cin >> WantToContinue;

    if(WantToContinue == '1')
    {
        char LayOnBack;

        cout << "Good. For the next exercise, you should lay on your back";
        cout << "\nAre you laying on your back(1=Yes/0=No)? ";
        cin >> LayOnBack;

        if(LayOnBack == '1')
            cout << "\nGreat.\nNow we will start the next exercise.";
        else
            cout << "\nWell, it looks like you are getting tired...";
    }
    else
        cout << "\nWe had enough today";

    cout << "\nWe will stop the session now\nThanks.\n";
    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

In the same way, you can nest statements as you see fit. The goal is to provide an efficient and friendly application. You can insert and nest statements that provide valuable feedback to the user while minimizing boredom. The above version of the program can be improved as followed:

```
//---------------------------------------------------------------------------
```



```
#include <iostream.h>
#pragma hdrstop


//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char SittingDown;

    do {
        cout << "Are you sitting down now(y/n)? ";
        cin >> SittingDown;

        if( SittingDown != 'y' )
            cout << "Could you please sit down for the next exercise?";
        cout << "\n\n";
    }while( SittingDown != 'y' );

    cout << "Wonderful. Now we will continue today's exercise...\n";
    cout << "\n...\n\nEnd of exercise\n";

    char WantToContinue;

    cout << "\nDo you want to continue(1=Yes/0=No)? ";
    cin >> WantToContinue;

    if(WantToContinue == '1')
    {
        char LayOnBack;

        cout << "Good. For the next exercise, you should lay on your back";
        cout << "\nAre you laying on your back(1=Yes/0=No)? ";
        cin >> LayOnBack;

        if(LayOnBack == '0')
        {
            char Ready;

            do {
                cout << "Please lay on your back";
                cout << "\nAre you ready(1=Yes/0=No)? ";
                cin >> Ready;
            }while(Ready == '0');
        }
        else if(LayOnBack == '1')
            cout << "\nGreat.\nNow we will start the next exercise.";
        else
        cout << "\nWell, it looks like you are getting tired...";
    }
    else
        cout << "\nWe had enough today";

    cout << "\nWe will stop the session now\nThanks.\n";
    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Practical Learning: Nesting Conditions

1. To nest an if...else condition in a do...while statement, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char Light;

    do {
        cout << "What is the current light "
             << "color(g=Green/y=Yellow/r=Red)? ";
        cin >> Light;

        if( Light == 'g' )
        {
            cout << "\nThe light is green";
            cout << "\nYou can proceed and drive through.\n";
        }
        else if( Light == 'y' )
        {
            cout << "\nYellow Light";
            cout << "\nBe careful!\n";
        }
        else if( Light == 'r' )
        {
            cout << "\nShow respect for the red light";
            cout << "\nPlease Stop!!!\n";
        }
        else
            cout << endl << Light << " is not a valid color.\n";
    } while( Light == 'r' );

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

2. Test the program and return to your development environment
3. To add a while and a for loops, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char Light;
    int Timer;

    do {
        cout << "What is the current light "
             << "color(g=Green/y=Yellow/r=Red)? ";
        cin >> Light;

        if( Light == 'g' )
        {
            cout << "\nThe light is green";
            cout << "\nYou can proceed and drive through.\n";
        }
```



```
        else if( Light == 'y' )
        {
            Timer = 0;

            while(Timer < 5)
            {
                cout << Timer << ". Yellow Light - Be Careful!\n";
                Timer++;
            }

            cout << "\nYellow light ended. Please Stop!!!\n";
        }
        else if( Light == 'r' )
        {
            cout << "\nShow respect for the red light";
            cout << "\nPlease Stop!!!\n";

            for( Timer = 1; Timer < 60; ++Timer)
            {
                if( Timer < 10 )
                    cout << " " << Timer << ".Red ";
                else
                    cout << Timer << ".Red ";

                if( Timer % 10 == 0 )
                    cout << endl;
            }
            cout << "\n - Red light ended -\n\n";
        }
        else
            cout << endl << Light << " is not a valid color.\n";

    } while( Light == 'r' );

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

4. Test the program and return to development environment.
5. Save your project

The break Statement

The break statement is used to stop a loop for any reason or condition the programmer sees considers fit. The break statement can be used in a while condition to stop an ongoing action. The syntax of the break statement is simply:

```
break;
```

Although made of only one word, the break statement is a complete statement; therefore, it can (and should always) stay on its own line (this makes the program easy to read).

The break statement applies to the most previous conditional statement to it; provided that previous statement is applicable.

The following program would display the letter d continuously unless something or somebody stops it. A break statement is inserted to stop this ever looping process:

```
//---------------------------------------------------------------------------
#include <iostream.h>
```

```
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char Letter = 'd';

    while( Letter <= 'n' )
    {
        cout << "Letter " << Letter << endl;
        break;
        }

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

The break statement can also be used in a do…while or a for loop the same way.

The break statement is typically used to handle the cases in a switch statement. We saw earlier that all cases in a switch would execute starting where a valid statement is found.

Consider the program we used earlier to request a number from 1 to 3, a better version that involves a break in each case would allow the switch to stop once the right case is found. Here is a new version of that program:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    int Number;

    cout << "Type a number between 1 and 3: ";
    cin >> Number;

    switch (Number)
    {
    case 1:
        cout << "\nYou typed 1.";
        break;
    case 2:
        cout << "\nYou typed 2.";
        break;
    case 3:
        cout << "\nYou typed 3.";
        break;
    default:
        cout << endl << Number << " is out of the requested range.";
    }

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Even when using the break statement, the switch allows many case to execute as

one. To do this, as we saw when not using the break, type two cases together. This technique is useful when validating letters because the letters could be in uppercase or lowercase. This illustrated in the following program:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    char Letter;

    cout << "Type a letter: ";
    cin >> Letter;

    switch( Letter )
    {
    case 'a':
    case 'A':
    case 'e':
    case 'E':
    case 'i':
    case 'I':
    case 'o':
    case 'O':
    case 'u':
    case 'U':
        cout << "The letter you typed, " << Letter << ", is a vowel\n";
        break;

    case 'b':case 'c':case 'd':case 'f':case 'g':case 'h':case 'j':
    case 'k':case 'l':case 'm':case 'n':case 'p':case 'q':case 'r':
    case 's':case 't':case 'v':case 'w':case 'x':case 'y':case 'z':
        cout << Letter << " is a lowercase consonant\n";
        break;

    case 'B':case 'C':case 'D':case 'F':case 'G':case 'H':case 'J':
    case 'K':case 'L':case 'M':case 'N':case 'P':case 'Q':case 'R':
    case 'S':case 'T':case 'V':case 'W':case 'X':case 'Y':case 'Z':
        cout << Letter << " is a consonant in uppercase\n";
        break;

    default:
        cout << "The symbol " << Letter
             << " is not an alphabetical letter\n";
    }

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

The switch statement is also used with an enumerator that controls cases. This is also a good place to use the break statement to decide which case applies. An advantage of using an enumerator is its ability to be more explicit than a regular integer.

To use an enumerator, define it and list each one of its members for the case that applies. Remember that, by default, the members of an enumerator are counted with the first member having a value of 0, the second is 1, etc. Here is an example of a switch statement that uses an enumerator.

```
//---------------------------------------------------------------------------
```

```
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
enum TEmploymentStatus { esFullTime, esPartTime, esContractor, esNS };

int main(int argc, char* argv[])
{
    int EmplStatus;

    cout << "Employee's Contract Status: ";
    cout << "\n0 - Full Time | 1 - Part Time"
         << "\n2 - Contractor | 3 - Other"
         << "\nStatus: ";
    cin >> EmplStatus;
    cout << endl;

    switch( EmplStatus )
    {
    case esFullTime:
        cout << "Employment Status: Full Time\n";
        cout << "Employee's Benefits: Medical Insurance\n"
             << " Sick Leave\n"
             << " Maternal Leave\n"
             << " Vacation Time\n"
             << " 401K\n";
        break;

    case esPartTime:
        cout << "Employment Status: Part Time\n";
        cout << "Employee's Benefits: Sick Leave\n"
             << " Maternal Leave\n";
        break;

    case esContractor:
        cout << "Employment Status: Contractor\n";
        cout << "Employee's Benefits: None\n";
        break;

    case esNS:
        cout << "Employment Status: Other\n";
        cout << "Status Not Specified\n";
        break;

    default:
        cout << "Unknown Status\n";
    }

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Practical Learning: Using the break Statement

1. Create a new project called Statements
2. Create a new C++ source file called Main
3. To apply the break statement, change the program as follows:

```
//---------------------------------------------------------------------------
```



```
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    char Light;

    cout << "What is the current light color(g=Green/y=Yellow/r=Red)? ";
    cin >> Light;

    switch( Light )
    {
    case 'g':
        cout << "\nThe light is green";
        cout << "\nYou can proceed and drive through.\n";
        break;

    case 'y':
        cout << "\nBe careful!\n";
        break;

    case 'r':
        cout << "\nPlease Stop!!!\n";
        break;

    default:
        cout << endl << Light << " is not a valid color.\n";
    }

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

4. Test the program and return to development environment.
5. Assuming the user might enter a color in uppercase, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    char Light;

    cout << "What is the current light color(g=Green/y=Yellow/r=Red)? ";
    cin >> Light;

    switch( Light )
    {
    case 'g':
    case 'G':
        cout << "\nThe light is green";
        cout << "\nYou can proceed and drive through.\n";
        break;

    case 'y':
```

```
    case 'Y':
        cout << "\nBe careful!\n";
        break;

    case 'r':
    case 'R':
        cout << "\nPlease Stop!!!\n";
        break;

    default:
        cout << endl << Light << " is not a valid color.\n";
    }

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

6. Test the program and return to your development environment.
7. To use an enumerator in a switch statement, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

enum TTrafficLight { tlGreen = 1, tlYellow, tlRed };

int main(int argc, char* argv[])
{
    int Light;

    cout << "What is the current light color(1=Green/2=Yellow/3=Red)? ";
    cin >> Light;

    switch( Light )
    {
    case tlGreen:
        cout << "\nThe light is green";
        cout << "\nYou can proceed and drive through.\n";
        break;

    case tlYellow:
        cout << "\nYellow Light";
        cout << "\nBe careful!\n";
        break;

    case tlRed:
        cout << "\nRespect the red light";
        cout << "\nPlease Stop!!!\n";
        break;

    default:
        cout << endl << Light << " is not a valid color.\n";
    }

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

8. Test the program and return to your programming environment.

The continue Statement

The continue statement uses the following syntax:

```
continue;
```

When processing a loop, if the statement finds a false value, you can use the continue statement inside of a while, do…while or a for conditional statements to ignore the subsequent statement or to jump from a false Boolean value to the subsequent valid value, unlike the break statement that would exit the loop. Like the break statement, the continue keyword applies to the most previous conditional statement and should stay on its own line.

The following programs asks the user to type 4 positive numbers and calculates the sum of the numbers by considering only the positive ones. If the user types a negative number, the program manages to ignore the numbers that do not fit in the specified category:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    // Declare necessary variables
    int posNumber, Sum = 0;

    // Request 4 positive numbers from the user
    cout << "Type 4 positive numbers.\n";
    // Make sure the user types 4 positive numbers
    for( int Count = 1; Count <= 4; Count++ )
    {
        cout << "Number: ";
        cin >> posNumber;

        // If the number typed is not positive, ignore it
        if( posNumber < 0 )
            continue;

        // Add each number to the sum
        Sum += posNumber;
    }

    // Display the sum
    cout << "\nSum of the numbers you entered = " << Sum << "\n\n";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

The goto Statement

The goto statement allows a program execution to jump to another section of the

function in which it is being used.

In order to use the goto statement, insert a name on a particular section of your function so you can refer to that name. The name, also called a label, is made of one word and follows the rules we have learned about C++ names (the name can be anything), then followed by a colon. The following program uses a for loop to count from 0 to 12, but when it encounters 5, it jumps to a designated section of the program:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    cout << "We need to count from 0 to 12\n";
    for(int Count = 0; Count <= 12; ++Count)
    {
        cout << "Count " << Count << endl;

        if( Count == 5 )
            goto MamaMia;
    }

MamaMia:
    cout << "Stopped at 5";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Logical Operators

The conditional Statements we have used so far were applied to single situations. You can combine statements using techniques of logical thinking to create more complex and complete expressions. One way to do this is by making sure that two conditions are met for the whole expression to be true. On the other hand, one or the other of two conditions can produce a true condition, as long as one of them is true. This is done with logical conjunction or disjunction.

Using the Logical Not

When a driver comes to a light that he expects to be green, we saw that he would use a statement such as, "The light is green". If in fact the light is green, we saw that the statement would lead to a true result. If the light is not green, the "The light is green" statement produces a false result. This is shown in the following table:

Color	Statement	Boolean Value
Green Light	The light is green	true
Non-Green Light	The light is green	false

As you may realize already, in Boolean algebra, the result of performing a comparison depends on how the *Condition* is formulated. If the driver is approaching a light that he is expecting to display any color other than green, he would start from a statement such as "The light is not green". If the light IS NOT green, the expression "The light is not green" is true (very important). This is illustrated in the following table:

Color	Statement	Boolean Value
Green Light	The light is green	true
Non-Green Light	The light is not green	true

The "The light is not green" statement is expressed in Boolean algebra as “Not the light is green”. Instead of writing “Not the light is green", in C++, using the logical Not operator , you would formulate the statement as, !"The light is green". Therefore, if P means “The light is green”, you can express the negativity of P as !P. The Boolean table produced is:

Color	Statement	Boolean Value	Symbol
Green Light	The light is green	true	P
Green Light	The light is not green	false	!P

When a statement is true, its Boolean value is equivalent to a non-zero integer such as 1. Otherwise, if a statement produces a false result, it is given a 0 value. Therefore, our table would be:

Color	Statement	Boolean Value	Integer Value
Green Light	The light is green	true	1
Green Light	The light is not green	false	0

Even though a program usually asks a straightforward question, the compiler would only consider the expression that needs to be evaluated; that is, the expression included between the parentheses of the if *Condition*. Suppose you are writing an ergonomic program that would guide the user on when and how to exercise. One of the questions your program would ask might be: "Are you sitting down?" There are three classic variances to this issue: the user might be sitting down, standing up, or laying down. Your program might look like this:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    int Position;
```

```
    cout << "Specify your position:\n"
         << "1 - Sitting Down\n"
        << "2 - Standing Up\n"
        << "3 - Laying Down\n";
    cin >> Position;

    if( Position == 1 )
        cout << "\nNow, position your back as vertically as you can.\n";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

That program allows the user to give one of three answers; and you might do something depending on the user's answer. Now, suppose you only want to know whether the user is sitting down; in fact, the program might expect the user to be sitting down for the subsequent assignment. Such a program could be:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    int SittingDown;

    cout << "Are you sitting down (1=Yes/0=No)? ";
    cin >> SittingDown;

    if( SittingDown == 1 )
        cout << "\nGood, now we will continue with the next exercise.";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

If the user is standing up, you would like her to sit down. If she is laying down, you still would like her to sit down. Based on this requirement, you might want to check whether the user is sitting down and you would not be interested in another position. The question could then be, "Aren't you sitting down?". In Boolean algebra, the question would be asked as, " Are you NOT sitting down?". A better C++ question would be, "Not " "Are you sitting down?". In other words, the statement (in this case the question) would be the negation of the regular question. If P represents the "Are you sitting down?" question, the negativity of P is expressed as !P. The new version of our program would be more concerned with the position the user has. Since the user is expected to type 1 for Yes, the program would display a concern for any other answer; in short, it would check the negativity of the *Condition*:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
```



```
    int SittingDown;

    cout << "Are you sitting down (1=Yes/0=No)? ";
    cin >> SittingDown;

    if( !(SittingDown == 1) )
        cout << "\nCould you please sit down for the next exercise?";

    cout << "\nWonderful!!!\n\n";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Practical Learning: Using the Logical not Operator

1. Create a new console application named Not
2. Create a C++ source file named Exercise
3. To apply the logical not operator, change the content of the file as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    char Light;

    cout << "What is the current light color(g=Green/y=Yellow/r=Red)? ";
    cin >> Light;

    if ( !(Light == 'g') )
    {
        cout << "\nThe light is not green";
        cout << "\nPlease stop and wait.";
    }
    else
        cout << "\nThe road is cleared. You can drive through";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

4. Test your program.

Logical Conjunction: AND

The law of the traffic light states that if a driver drives through a red light, he or

she has broken the law. Three things happen here:

1. The traffic light is red
2. The driver is driving
3. The law is broken

Let's segment these expressions and give each a name. The first statement will be called L. Therefore,

L <=> The traffic light is red

The second statement will be called D. This means

D <=> The driver is driving through the light

The last statement will be called B, which means

B <=> The law is broken

Whenever the traffic light is red, the "The traffic light is red" statement is true. Whenever a driver is driving, the "The driver is driving" statement is true, which means D is true. Whenever the law is broken, the "The law is broken" statement is true. When a statement is true, it receives a Boolean value of true:

L
true

D
true

B
true

These three statements are completely independent when each is stated in its own sentence. The third bears any consideration only when the first two are combined. Therefore, the third statement is a consequence or a result. The fact that a driver is driving and/or a light is red or displays any color, does not make a law broken. The law is broken only when or IF a driver drives through a red light. This means L and D have to be combined to produce B.

A combination of the first two statements means you need Statement1 AND Statement2. Combining Statement1 AND Statement2 means L AND D that produces

"The traffic light is red" AND "The driver is driving through the light"

In C++, the AND keyword is called an operator because it applies for one or more variable. The AND operator is specifically called a binary operator because it is used on two variables. The AND operator is used to concatenate or add two statements or expressions. It is represented by &&. Therefore, a concatenation of L and D would be written as L && D. Logically, what does the combination mean?

When the traffic light is red, L is true. If a driver is driving through the light, D is true. If the driver is driving through the light that is red, this means L && D. Then the law is broken:

L	D	L && D	B
true	true	true	TRUE

When the traffic light is not red, regardless of the light's color, L is false. If a driver drives through it, no law is broken. Remember, not only should you drive through a green light, but also you are allowed to drive through a yellow light. Therefore, B is false:

L	D	L && D	B
false	true	false	FALSE

If the traffic light is red, L is true. If no driver drives through it, D is false, and no law is broken. When no law is broken, B, which is the result of L && D, is false:

L	D	L && D	B
true	false	false	FALSE

If the light is not red, L is false. If no driver drives through it, D is false. Consequently, no law is broken. B, which is the result of L && D, is still false:

L	D	L && D	B
false	false	false	FALSE

From our tables, the law is broken only when the light is red AND a driver drives through it. This produces:

L	D	L && D	B
true	true	true	TRUE
false	true	false	FALSE
true	false	false	FALSE
false	false	false	FALSE

The logical conjunction operator && is used to check that the combination of two statements results in a true condition. This is used when one condition cannot satisfy the intended result. Consider a pizza application whose valid sizes are 1 for small, 2 for medium, 3 for large, and 4 for jumbo. When a clerk uses this application, you would usually want to make sure that only a valid size is selected to process an order. After the clerk has selected a size, you can use a logical conjunction to validate the range of the item's size. Such a program could be written (or started) as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    int PizzaSize;

    cout << "Select your pizza size";
    cout << "\n1=Small | 2=Medium";
    cout << "\n3=Medium | 4=Jumbo";
    cout << "\nYour Choice: ";
    cin >> PizzaSize;

    if(PizzaSize >= 0 && PizzaSize <= 4)
        cout << "\nGood Choice. Now we will proceed with the toppings";
    else
        cout << "\nInvalid Choice";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

When a program asks a question to the user who must answer by typing a letter, there is a chance that the user would type the answer in uppercase or lowercase. Since we know that C++ is case-sensitive, you can use a combined conditional statement to find out what answer or letter the user would have typed.

We saw that the truthfulness of a statement depends on how the statement is structured. In some and various cases, instead of checking that a statement is true, you can validate only negative values. This can be done on single or combined

statements. For example, if a program is asking a question that requires a Yes or No answer, you can make sure the program gets a valid answer before continuing. Once again, you can use a logical conjunction to test the valild answers. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    char SittingDown;

    do {
        cout << "Are you sitting down now(y/n)? ";
        cin >> SittingDown;

        if( SittingDown != 'y' && SittingDown != 'Y' )
            cout << "\nCould you please sit down for the next exercise?\n";
    } while( SittingDown != 'y' && SittingDown != 'Y' );

    cout << "\nWonderful!!!";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Logical Disjunction: OR

Let's assume that a driver has broken the law by driving through a red traffic light but there was no accident (to make our discussion simpler). There are two ways he can get a ticket: a police officer saw him, a special camera took a picture. This time again, we have three statements to make:

S <=> A police officer saw the driver

H <=> A camera took a picture of the action

T <=> The driver got a ticket

If a police officer saw the driver breaking the law, the "A police officer saw the driver" statement is true. Consequently, S is true.
If a (specially installed) camera took the picture (of the scene), the "A camera took the picture of the action" statement is true. This means H is true.
If the driver gets a ticket, the "The driver gets a ticket" statement is true, which means T is true.

S	H	T
true	true	true

Once again, the third statement has no bearing unless you consider the first two. Last time, we saw that if the first two statements were combined, only then the result would produce the third statement. Let's consider in which case the driver would get a ticket.

If a police officer saw the driver, would he get a ticket? Yes, because on many traffic lights there is no camera but a police officer has authority to hand an

infraction. This means if S is true, then T also is true. This produces:

S	H	T
true	Don't Care	true

Imagine a traffic light is equipped with a camera. If the driver breaks the law, the camera would take a picture, which means the driver would get a ticket. Therefore, if a camera takes a picture (H is true), the driver gets a ticket (T is true):

S	H	T
Don't Care	true	true

What if a police officer catches the action and a camera takes a picture. This means the driver will still get a ticket, even if one of both the police officer and the camera does not act but the other does. If both the police officer and the camera catch the action and act accordingly, the driver would get only one ticket (even if the driver receives two tickets, only one would be considered). Therefore, whether the first statement OR the second statement is true, the resulting third statement T is still true:

S	H	T
true	true	true

The only time the driver would not get a ticket is when no police officer catches him and no camera takes a picture. In other words, only when both of the first two statements are false can the third statement be false.

Since T is the result of S and H combined, we have seen that T is true whenever either S is true OR H is true. The OR logical disjunction is expressed in C++ with the || operator. Here is the resulting table:

S	H	S || H	T
true	true	true	TRUE
false	true	true	TRUE
true	false	true	TRUE
false	false	false	FALSE

Consider a program that asks a question and expects a yes or no answer in the form of y or n. Besides y for yes, you can also allow the user to type Y as a valid yes. To do this, you would let the compiler check that either y or Y was typed. In the same way, either n or N would be valid negations. Any of the other characters would fall outside the valid characters. Our hot-tempered program can be restructured as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    char Answer;

    cout << "Do you consider yourself a hot-tempered individual(y=Yes/n=No)? ";
    cin >> Answer;

    if( Answer == 'y' || Answer == 'Y' ) // Unique Condition
    {
        cout << "\nThis job involves a high level of self-control.";
        cout << "\nWe will get back to you.\n";
    }
    else if( Answer == 'n' || Answer == 'N' ) // Alternative
```

```
        cout << "\nYou are hired!\n";
    else
        cout << "\nThat was not a valid answer!\n";

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Practical Learning: Combining Conditions

1. Create a new project named ComboConditions.
2. Create a new C++ source file named Exercise
3. Change the file with the following:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

int main(int argc, char* argv[])
{
    char Light, L;
    int Timer;

    // Make sure the user enters a valid letter for a traffic light
    do {
        cout << "What is the current light "
             << "color(g=Green/y=Yellow/r=Red)? ";
        cin >> Light;
    } while( Light != 'r' && Light != 'R' &&
             Light != 'y' && Light != 'Y' &&
             Light != 'g' && Light != 'G' );

    // Process a message depending on the current traffic light
    if( Light == 'g' || Light == 'G' )
    {
        cout << "\nThe light is green";
        cout << "\nYou can proceed and drive through.\n";
    }
    else if( Light == 'y' || Light == 'Y' )
    {
        Timer = 0;

        while(Timer < 5)
        {
            cout << Timer << ". Yellow Light - Be Careful!\n";
            Timer++;
        }

        cout << "\nYellow light ended. Please Stop!!!\n";
    }
    else if( Light == 'r' || Light == 'R' )
    {
        cout << "\nShow respect for the red light";
        cout << "\nPlease Stop!!!\n";

        for( Timer = 1; Timer < 60; ++Timer)
        {
            if( Timer < 10 )
```



```
                cout << " " << Timer << ".Red ";
            else
                cout << Timer << ".Red ";

            if( Timer % 10 == 0 )
                cout << endl;
        }

        cout << "\n - Red light ended -\n\n";
    }
    else
        cout << endl << Light << " is not a valid color.\n";

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

4. Test the program and return to your programming environment
5. To further refine your application, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

enum TDrivingAnswer { daNo, daYes };

int main(int argc, char* argv[])
{
    char Light;
    int Timer;
    int Answer;

    do {
        // Make sure the user enters a valid letter for a traffic light
        do {
            cout << "What is the current light "
                 << "color(g=Green/y=Yellow/r=Red)? ";
            cin >> Light;
            // The user typed an invalid color for the traffic light
            if( Light != 'r' && Light != 'R' &&
                Light != 'y' && Light != 'Y' &&
                Light != 'g' && Light != 'G' )
                cout << "Invalid color\n";
        } while( Light != 'r' && Light != 'R' &&
                 Light != 'y' && Light != 'Y' &&
                 Light != 'g' && Light != 'G' );

        // Process a message depending on the current traffic light
        if( Light == 'g' || Light == 'G' )
        {
            cout << "\nThe light is green";
            cout << "\nYou can proceed and drive through.\n";
        }
        else if( Light == 'y' || Light == 'Y' )
        {
            Timer = 0;
            while(Timer < 5)
            {
                cout << Timer << ". Yellow Light - Be Careful!\n";
                Timer++;
            }
            cout << "\nYellow light ended. Please Stop!!!\n";
```

```
        }
        else if( Light == 'r' || Light == 'R' )
        {
            cout << "\nShow respect for the red light";
            cout << "\nPlease Stop!!!\n";

            for( Timer = 1; Timer < 60; ++Timer)
            {
                if( Timer < 10 )
                    cout << " " << Timer << ".Red ";
                else
                    cout << Timer << ".Red ";
                if( Timer % 10 == 0 )
                    cout << endl;
            }

            cout << "\n - Red light ended -\n\n";
        }
        else
            cout << endl << Light << " is not a valid color.\n";

        cout << "\nAre you still on the road(0=No/1=Yes)? ";
        cin >> Answer;
        cout << endl;
    } while( Answer == daYes );

    cout << "\nNice serving you";

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

6. Test the program and return to your development environment.
7. Save your program.

Conditional Statements and Functions

Using Conditions in Functions

The use of functions in a program allows you to isolate assignments and confine them to appropriate entities. While the functions take care of specific requests, you should provide them with conditional statements to validate what these functions are supposed to do. There are no set rules to the techniques involved; everything depends on the tasks at hand. Once again, you will have to choose the right tools for the right job. To make effective use of functions, you should be very familiar with different data types because you will need to return the right value.

The ergonomic program we have been writing so far needs to check different things including answers from the user in order to proceed. These various assignments can be given to functions that would simply hand the results to the main() function that can, in turn, send these results to other functions for further processing. Here is an example:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

//---------------------------------------------------------------------------
#define Or ||
#define And &&
//---------------------------------------------------------------------------
char __fastcall GetPosition()
{
        char Position;

        do {
                cout << "Are you sitting down now(y/n)? ";
                cin >> Position;

                if( Position != 'y' And Position != 'Y' And
                        Position != 'n' And Position != 'N' )
                        cout << "Invalid Answer\n";
        } while( Position != 'y' And Position != 'Y' And
                         Position != 'n' And Position != 'N' );

        return Position;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    char Position;

    Position = GetPosition();
    if( Position == 'n' Or Position == 'N' )
        cout << "\nCould you please sit down for the next exercise?\n";
    else
        cout << "\nWonderful!!!\n\n";

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

Functions do not have to return a value in order to be involved with conditional statements. In fact, both issues are fairly independent. This means, void and non-void functions can manipulate values based on conditions internal to the functions. This is illustrated in the following program that is an enhancement to an earlier ergonomic program:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused

//---------------------------------------------------------------------------
#define Or ||
#define And &&
//---------------------------------------------------------------------------
void __fastcall Exit()
{
    cout << "\nPress any key to continue...";
    getchar();
}
//---------------------------------------------------------------------------
char __fastcall GetPosition()
```

```
{
    char Position;

    do {
        cout << "Are you sitting down now(y/n)? ";
        cin >> Position;

        if( Position != 'y' And
            Position != 'Y' And
            Position != 'n' And
            Position != 'N' )
            cout << "Invalid Answer\n";
    } while( Position != 'y' And
             Position != 'Y' And
             Position != 'n' And
             Position != 'N' );

        return Position;
}
//---------------------------------------------------------------------------
void __fastcall NextExercise()
{
    char LayOnBack;

    cout << "Good. For the next exercise, you should lay on your back";
    cout << "\nAre you laying on your back(1=Yes/0=No)? ";
    cin >> LayOnBack;

    if(LayOnBack == '0')
    {
        char Ready;

        do {
            cout << "Please lay on your back";
            cout << "\nAre you ready(1=Yes/0=No)? ";
            cin >> Ready;
        }while(Ready == '0');
    }
    else if(LayOnBack == '1')
        cout << "\nGreat.\nNow we will start the next exercise.";
    else
        cout << "\nWell, it looks like you are getting tired...";
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    char Position, WantToContinue;

    Position = GetPosition();

    if( Position == 'n' Or Position == 'N' )
        cout << "\nCould you please sit down for the next exercise?";
    else
    {
        cout << "\nWonderful!\nNow we will continue today's exercise...\n";
        cout << "\n...\n\nEnd of exercise\n";
    }

    cout << "\nDo you want to continue(1=Yes/0=No)? ";
    cin >> WantToContinue;

    if( WantToContinue == '1' )
        NextExercise();
    else if( WantToContinue == '0' )
        cout << "\nWell, it looks like you are getting tired...";
    else
    {
        cout << "\nConsidering your invalid answer...";
        cout << "\nWe had enough today";
    }
    cout << "\nWe will stop the session now\nThanks.\n";
```



```
    Exit();
    return 0;
}
//---------------------------------------------------------------------------
```

Practical Learning: Conditions in Functions

1. Create a new project named Conditioner
2. Create a C++ source file named Exercise
3. To use conditions in functions, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
//---------------------------------------------------------------------------
#define OR ||
#define AND &&
//---------------------------------------------------------------------------
enum TDrivingAnswer { daNo, daYes };
//---------------------------------------------------------------------------
void __fastcall Exit()
{
    cout << "\nPress any key to continue...";
    getchar();
}
//---------------------------------------------------------------------------
char __fastcall CurrentLight()
{
    char Light;

    // Make sure the user enters a valid letter for a traffic light
    do {
        cout << "What is the current light "
             << "color(g=Green/y=Yellow/r=Red)? ";
        cin >> Light;

        // The user typed an invalid color for the traffic light
        if( Light != 'r' AND Light != 'R' AND
            Light != 'y' AND Light != 'Y' AND
            Light != 'g' AND Light != 'G' )
            cout << "Invalid color\n";
    } while( Light != 'r' AND Light != 'R' AND
             Light != 'y' AND Light != 'Y' AND
             Light != 'g' AND Light != 'G' );

    return Light;
}
//---------------------------------------------------------------------------
void __fastcall GreenLight()
{
    cout << "\nThe light is green";
    cout << "\nYou can proceed and drive through.\n";
}
//---------------------------------------------------------------------------
void __fastcall YellowLight()
{
    int Timer = 0;
```

```
    while(Timer < 5)
    {
        cout << Timer << ". Yellow Light - Be Careful!\n";
        Timer++;
    }

    cout << "\nYellow light ended. Please Stop!!!\n";
}
//---------------------------------------------------------------------------
void __fastcall RedLight()
{
    int Timer;

    cout << "\nShow respect for the red light";
    cout << "\nPlease Stop!!!\n";

    for( Timer = 1; Timer < 60; ++Timer)
    {
        if( Timer < 10 )
            cout << " " << Timer << ".Red ";
        else
            cout << Timer << ".Red ";

        if( Timer % 10 == 0 )
            cout << endl;
    }

    cout << "\n - Red light ended -\n\n";
}
//---------------------------------------------------------------------------
char __fastcall AreYouStillOnTheRoad()
{
    char Ans;

    cout << "\nAre you still on the road(y=Yes/n=No)? ";
    cin >> Ans;

    return Ans;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    char Light;
    int Timer;
    char Answer;

    do {
        Light = CurrentLight();

        // Process a message depending on the current traffic light
        if( Light == 'g' OR Light == 'G' )
            GreenLight();
        else if( Light == 'y' OR Light == 'Y' )
            YellowLight();
        else if( Light == 'r' OR Light == 'R' )
            RedLight();
        else
            cout << endl << Light << " is not a valid color.\n";

        Answer = AreYouStillOnTheRoad();
        cout << endl;
    } while((Answer == '1') OR (Answer == 'y') OR (Answer == 'Y') );

    cout << "\nNice serving you\n\n";
    Exit();
    return 0;
}
//---------------------------------------------------------------------------
```

4. Test the program and return to development environment

5. Save your project.

Conditional Returns

A function defined other than void must always return a value. Sometimes, a function will perform some tasks whose results would lead to different consequences. A function can return only one value (this is true for this context, but we know that there are ways to pass arguments so that a function can return more than one value) but you can make it render a result depending on a particular behavior. Image that a function is requesting an answer from the user. Since the user can provide different answers, you can treat each result differently.

In the previous section, we saw an example of returning a value from a function. Following our employment application, here is an example of a program that performs a conditional return:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
void __fastcall Exit()
{
    cout << "\nPress any key to continue...";
    getchar();
}
//---------------------------------------------------------------------------
bool __fastcall GetAnswer()
{
    char Ans; string Response;

    cout << "Do you consider yourself a hot-tempered individual(y=Yes/n=No)? ";
    cin >> Ans;

    if( Ans == 'y' )
        return true;
    else
        return false;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    bool Answer;

    Answer = GetAnswer();

    if( Answer == true )
    {
        cout << "\nThis job involves a high level of self-control.";
        cout << "\nWe will get back to you.\n";
    }
    else
        cout << "\nYou are hired!\n";

    Exit();
    return 0;
}
//---------------------------------------------------------------------------
```

Imagine you write the following function:

```
#include <iostream.h>

#define Or ||
#define And &&

string GetPosition()
{
        char Position;

        cout << "Are you sitting down now(y/n)? ";
        cin >> Position;

        if( Position == 'y' Or Position == 'Y' )
                return "Yes";
        else if( Position == 'n' Or Position == 'N' )
                return "No";
}

int main()
{
        string Answer;

        Answer = GetPosition();
        cout << "\nAnswer = " << Answer;

        return 0;
}
```

On paper, the function looks fine. If the user answers with y or Y, the function returns the string Yes. If the user answer with n or N, the function returns the string No. Unfortunately, this function has a problem: what if there is an answer that does not fit those we are expecting? In reality the values that we have returned in the function conform only to the conditional statements and not to the function. Remember that in if(*Condidion*)*Statement*;, the *Statement* executes only if the *Condition* is true. Here is what will happen. If the user answers y or Y, the function returns Yes and stops; fine, it has returned something, we are happy. If the user answers n or N, the function returns No, which also is a valid value: wonderful. If the user enters another value (other than y, Y, n, or N), the execution of the function will not execute any of the return statements and will not exit. This means that the execution will reach the closing curly bracket without encountering a return value. Therefore, the compiler will issue a warning. Although the warning looks like not a big deal, you should take care of it: never neglect warnings. The solution is to provide a return value so that, if the execution reaches the end of the function, it would still return something. Here is a solution to the problem:

```
//---------------------------------------------------------------------------
string __fastcall GetPosition()
{
    char Position;

    cout << "Are you sitting down now(y/n)? ";
    cin >> Position;

    if( Position == 'y' Or Position == 'Y' )
        return "Yes";
    else if( Position == 'n' Or Position == 'N' )
        return "No";

    // If you reach here, it means no valid answer was provided.
    // Therefore
    return "Invalid Answer";
}
//---------------------------------------------------------------------------
```


Here is an example from running the program:

```
Are you sitting down now(y/n)? w

Answer = Invalid Answer
Press any key to continue...
```

This is illustrated in the following program that has two functions with conditional returns:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
//---------------------------------------------------------------------------
#define Or ||
#define And &&
//---------------------------------------------------------------------------
void __fastcall Exit()
{
    cout << "\nPress any key to continue...";
    getchar();
    getchar();
}
//---------------------------------------------------------------------------
char __fastcall GetPosition()
{
    char Position;

    do {
        cout << "Are you sitting down now(y/n)? ";
        cin >> Position;

        if( Position != 'y' And
            Position != 'Y' And
            Position != 'n' And
            Position != 'N' )
            cout << "Invalid Answer\n";
        } while(Position != 'y' And
            Position != 'Y' And
            Position != 'n' And
            Position != 'N' );

    if( Position == 'y' Or Position == 'Y' )
        return 'y';
    else if( Position == 'n' Or Position == 'N' )
        return 'n';

    // If you reach this point, none of the answers was valid
    return Position;
}
//---------------------------------------------------------------------------
void __fastcall NextExercise()
{
    char LayOnBack;

    cout << "Good. For the next exercise, you should lay on your back";
    cout << "\nAre you laying on your back(1=Yes/0=No)? ";
    cin >> LayOnBack;

    if(LayOnBack == '0')
    {
        char Ready;
```

```
        do {
            cout << "Please lay on your back";
            cout << "\nAre you ready(1=Yes/0=No)? ";
            cin >> Ready;
        } while(Ready == '0');
    }
    else if(LayOnBack == '1')
        cout << "\nGreat.\nNow we will start the next exercise.";
    else
        cout << "\nWell, it looks like you are getting tired...";
}
//---------------------------------------------------------------------------
bool __fastcall ValidatePosition(char Pos)
{
    if( Pos == 'y' Or Pos == 'Y' )
        return true;
    else if( Pos == 'n' Or Pos == 'N' )
        return false;

    // If you reached this point, something was not valid
    return false;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    char Position, WantToContinue;
    bool SittingDown;

    Position = GetPosition();
    SittingDown = ValidatePosition(Position);

    if( SittingDown == false )
        cout << "\nCould you please sit down for the next exercise?";
    else
    {
        cout << "\nWonderful!\nNow we will continue today's exercise...\n";
        cout << "\n...\n\nEnd of exercise\n";
    }

    cout << "\nDo you want to continue(1=Yes/0=No)? ";
    cin >> WantToContinue;

    if( WantToContinue == '1' )
        NextExercise();
    else if( WantToContinue == '0' )
        cout << "\nWell, it looks like you are getting tired...";
    else
    {
        cout << "\nConsidering your invalid answer...";
        cout << "\nWe had enough today";
    }

    cout << "\nWe will stop the session now\nThanks.\n";

    Exit();
    return 0;
}
//---------------------------------------------------------------------------
```

Practical Learning: Functions and Conditional Returns

1. To implement a function that conditionally returns a value, modify the program as follows:


```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
//---------------------------------------------------------------------------
#define OR ||
#define AND &&
//---------------------------------------------------------------------------
enum TDrivingAnswer { daNo, daYes };
//---------------------------------------------------------------------------
void __fastcall Exit()
{
    cout << "\nPress any key to continue...";
    getchar();
}
//---------------------------------------------------------------------------
char __fastcall CurrentLight()
{
    ...
}
//---------------------------------------------------------------------------
void __fastcall GreenLight()
{
    ...
}
//---------------------------------------------------------------------------
void __fastcall YellowLight()
{
    ...
}
//---------------------------------------------------------------------------
void __fastcall RedLight()
{
    ...
}
//---------------------------------------------------------------------------
bool __fastcall AreYouStillOnTheRoad()
{
    int Ans;

    cout << "\nAre you still on the road(0=No/1=Yes)? ";
    cin >> Ans;

    if( Ans == 0 )
        return false;
    else if( Ans == 1 )
        return true;

    // If you got so far, something went wrong, therefore, return false
    return false;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    char Light;
    int Timer;
    int Answer;

    do {
        Light = CurrentLight();

        // Process a message depending on the current traffic light
        if( Light == 'g' || Light == 'G' )
            GreenLight();
        else if( Light == 'y' || Light == 'Y' )
            YellowLight();
        else if( Light == 'r' || Light == 'R' )
```

```
            RedLight();
        else
            cout << endl << Light << " is not a valid color.\n";

        Answer = AreYouStillOnTheRoad();
        cout << endl;
    } while(Answer == 1);

    cout << "\nNice serving you";
    Exit();
    return 0;
}
//---------------------------------------------------------------------------
```

2. Test the program and return to development environment
3. To conditionally return an enumerator from a function, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
//---------------------------------------------------------------------------
#define OR ||
#define AND &&
//---------------------------------------------------------------------------
enum TDrivingAnswer { daNo, daYes };
enum TTrafficColor { tcRed, tcYellow, tcGreen, tcNoColor };
//---------------------------------------------------------------------------
void __fastcall Exit()
{
    cout << "\nPress any key to continue...";
    getchar();
}
//---------------------------------------------------------------------------
TTrafficColor __fastcall CurrentLight()
{
    char Light;

    cout << "What is the current light "
         << "color(g=Green/y=Yellow/r=Red)? ";
    cin >> Light;

    if( Light == 'r' || Light == 'R' )
        return tcRed;
    else if( Light == 'y' || Light == 'Y' )
        return tcYellow;
    else if( Light == 'g' || Light == 'G' )
        return tcGreen;

    return tcNoColor;
}
//---------------------------------------------------------------------------
void __fastcall GreenLight()
{
    ...
}
//---------------------------------------------------------------------------
void __fastcall YellowLight()
{
    ...
}
//---------------------------------------------------------------------------
void __fastcall RedLight()
```



```
{
    ...
}
//---------------------------------------------------------------------------
bool __fastcall AreYouStillOnTheRoad()
{
    ...
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TTrafficColor Light;
    int Timer;
    int Answer;

    do {
        Light = CurrentLight();

        switch( Light )
        {
        case tcRed:
            RedLight();
            break;

        case tcYellow:
            YellowLight();
            break;

        case tcGreen:
            GreenLight();
            break;
        }

        Answer = AreYouStillOnTheRoad();
        cout << endl;
    } while(Answer == 1);

    cout << "\nNice serving you\n\n";
    Exit();
    return 0;
}
//---------------------------------------------------------------------------
```

4. Test the program and return to development environment
5. Save your project

Previous

Next