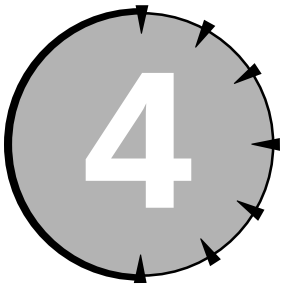
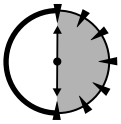
SESSION



C++ Instructions

Session Checklist

- ✔ Reviewing the Conversion program from Sessions 2 and 3
- ✔ Understanding the parts of a C++ program
- ✔ Introducing common C++ commands



30 Min. To Go

In Sessions 2 and 3, you were asked to enter a C++ program by rote. The idea was to learn the C++ environment (whichever environment you chose) rather than learn how to program. This session analyzes the Conversion.cpp program. You will see exactly what each part of the program does and how each part contributes to the overall solution.

The Program

Listing 4-1 is the Conversion.cpp program (again) except that it is annotated by commenting features which I describe in the remainder of the lesson.

There are several aspects to this program that you have to take on faith, at least for now. Be patient. Every structure found in this program is explained in time.

This version has extra comments....

Listing 4-1
Conversion.cpp

```
//
//  Conversion - convert temperature from Celsius
//               degree units into Fahrenheit degree
//               units:
//      Fahrenheit = Celsius  * (212 - 32)/100 + 32
//
#include <stdio.h>            // framework
#include <iostream.h>
int main(int nNumberofArgs, char* pszArgs[])
{
    // here is our first statement
    // it's a declaration
    int nCelsius;

    // our first input/output statements
    cout << "Enter the temperature in Celsius:";
    cin  > nCelsius;

    // the assignment marks a calculation expression
    int nFactor;
    nFactor = 212 - 32;

    // an assignment using an expression containing
    // variables
    int nFahrenheit;
    nFahrenheit = nFactor * nCelsius/100 + 32;

    // output the results
```

```
    cout << "Fahrenheit value is:";
cout << nFahrenheit;

    return 0;
}
```

The C++ Program Explained

Our Human Program back in Session 1 consisted of a sequence of commands. Similarly, a C++ program consists of a sequence of C++ statements that the computer processes in order. These statements fall into a series of broad types. Each of these is described here.

The basic program framework

Every program written in C++ begins with the same basic framework:

```
#include <stdio.h>
#include <iostream.h>
int main(int nNumberofArgs, char* pzArgs[])
{
...your code goes here...
    return 0;
}
```

You don't need to worry too much about the details of this framework — these details will come later — but you should have some idea of what they're about. The first two lines are called **include statements** because they cause the contents of the named file to be included at that point in the program. We'll just consider them magic at this point.

The next statement in every framework is the `int main(...)` statement. This is followed by an open and closed brace. Your programs are written within these braces. Execution of the program begins at the open brace and ends at the `return` statement, which immediately precedes the closed brace.

Unfortunately, a more detailed explanation of this framework must be left to future chapters. Don't worry . . . we get to it before the weekend is out.

Comments

The first few lines of the program appear to be free form text. Either this "code" was meant for human eyes or the computer is a lot smarter than anyone's ever given it credit for being. These first six lines are known as comments. A **comment** is a line or portion of a line that is ignored by the C++ compiler. Comments enable the programmer to explain what he or she was doing or thinking while writing a particular segment of code.

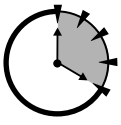
A C++ comment begins with a double slash ("//") and ends with a newline. You can put any character you want in a comment. A comment may be as long as you want, but it is customary to keep comments to 80 characters or so, because that's all that will fit on the computer screen.

A **newline** would have been known as a "carriage return" back in the days of typewriters, when the act of entering characters into a machine was called "typing" and not "keyboarding." A newline is the character that terminates a command line.

C++ allows a second form of comment in which everything appearing after a /* and before a */ are ignored; however, this form of comment is not normally used in C++ anymore.

It may seem odd to have a command in C++, or any other programming language, which is ignored by the computer. However, all computer languages have some version of the comment. It is critical that the programmer explain what was going through her or his mind at the time that the code was written. It may not be obvious to the next person who picks up the program and uses it or modifies it. In fact, after only a few months it may not be obvious to the programmer what he or she meant.

Use comments early and often.



20 Min. To Go

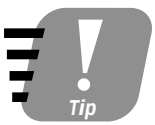
There's that framework again

The next four lines represent that framework I mentioned earlier. Remember that the program begins executing with the first statement after the open brace.

Statements

The first noncomment line after the brace is a C++ statement. A **statement** is a single set of commands. All statements other than comments end with a semicolon (;). (There is a reason that comments don't, but it's obscure. To my mind, comments should end in a semicolon as well, for consistency's sake if for no other reason.)

As you look through the program, you can see that spaces, tabs, and newlines appear throughout the program. In fact, I have placed a newline after every statement in this program. These characters are collectively known as white space because you can't see any of them on the monitor. A **white space** is a space, a tab, a vertical tab, or a newline. C++ ignores white space.



You may add white space anywhere you like in your program to enhance readability, except in the middle of a word.

While C++ may ignore white space, it does not ignore case. The variable `fullspeed` and the variable `FullSpeed` have nothing to do with each other. While the command `int` may be understood completely, C++ has no idea what `INT` means.

Declarations

The line `int nCelcius;` is a declaration statement. A **declaration** is a statement that defines a variable. A **variable** is a "holding tank" for a value of some type. A variable contains a value, such as a number or a character.

The term *variable* stems from algebra formulae of the following type:

```
x = 10
y = 3 * x
```

In the second expression, *y* is set equal to 3 times *x*, but what is *x*? The variable *x* acts as a holding tank for a value. In this case, the value of *x* is 10, but we could have just as well set the value of *x* to 20 or 30 or –1. The second formula makes sense no matter what the value of *x*.

In algebra, it's allowed to begin with a statement such as *x = 10*. In C++, the programmer must first define the variable *x* before it can be used.

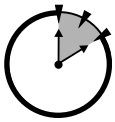
In C++, a variable has a type and a name. The line `int nCelcius;` declares an variable `nCelcius` designed to hold an integer. (Why they couldn't have just said `integer` instead of `int`, I'll never know. It's just one of those things that you learn to live with.)

The name of a variable has no particular significance to C++. A variable must begin with a letter ('A' through 'Z' or 'a' through 'z'). All subsequent characters must be a letter, a digit ('0' to '9'), or an underscore ('_'). Variable names can be as long as you want to make them within reason.

There actually is a limitation but it's much larger than the reader's. Don't exceed what the reader can comfortably remember — 20 characters or so.

By convention, variable names begin with a lowercase letter. Each new word in a variable begins with a capital as in `myVariable`. I explain the significance of the *n* in `nCelsius` in Session 5.

Try to make variable names short but descriptive. Avoid names like `x` because `x` has no meaning. A variable name such as `lengthOfLineSegment` is much more descriptive.



10 Min. To Go

Input/output

The lines beginning with `cout` and `cin` are known as input/output statements, often contracted to I/O statements. (Like all engineers, programmers love contractions and acronyms.)

The first I/O statement says output the phrase "Enter the temperature in Celsius:" to `cout` (pronounced "see-out"). `cout` is the name of the standard C++ output device. In this case, the standard C++ output device is your monitor.

The next line is exactly the opposite. This line says to extract a value from the C++ input device and store it in the integer variable `nCelsius`. The C++ input device is normally the keyboard. This is the C++ analogue to the algebra formula *x = 10* mentioned above. For the remainder of the program, the value of `nCelsius` is whatever the user enters here.

Expressions

In the next two lines, which are marked as a "calculation expression," the program declares a variable `nFactor` and assigns it the value resulting from a calculation. This command calculates the difference of 212 and 32. In C++, such a formula is called an expression.

An **operator** is a command that generates a value. The operator in this calculation is "–".

An **expression** is a command that has a value. The expression here is "212 – 32".

Assignment

The spoken language can be very ambiguous. The term *equals* is one of those ambiguities.

The word *equals* can mean that two things have the same value as in 5 cents equals a nickel. Equals can also imply assignment as in math when you say that y equals 3 times x.

To avoid ambiguity, C++ programmers call = the **assignment operator**.

The assignment operator says store the results of the expression on the right of the = in the variable to the left. Programmers say that `nFactor` is assigned the value 212 – 32.

Expressions (continued)

The second expression in Conversion.cpp presents a slightly more complex expression than the first. This expression uses the same mathematical symbols: * for multiplication, / for division, and + for addition. In this case, however, the calculation is performed on variables and not simply constants.

Done!

The value contained in the variable `nFactor` (calculated immediately prior, by the way) is multiplied by the value contained in `nCelcius` (which was input from the keyboard). The result is divided by 100 and summed with 32. The result of the total expression is assigned to the integer variable `nFahrenheit`.

The final two commands output the string "Fahrenheit value is:" to the display followed by the value of `nFahrenheit`.

REVIEW

You have finally seen an explanation of the Conversion program entered in Sessions 2 and 3. Of necessity, this explanation has been at a high level. Don't worry, however; the details are forthcoming.

- All programs begin with the same framework.
- C++ allows you to include comments that are explanations to yourself and others as to what different parts of a program do.

- C++ expressions look a lot like algebraic expressions, except that C++ variables have to be declared before they can be used.
- = is called assignment.
- C++ input and output statements default to the keyboard and the screen or MS-DOS window.

QUIZ YOURSELF

1. What does the following C++ statement do? (See Comments.)

```
// I'm lost
```

2. What does the following C++ statement do? (See Declarations.)

```
int nQuizYourself; // help me out here
```

3. What does the following C++ statement do? (See Input/Output.)

```
cout << "Help me out here";
```

4. What does the following C++ statement do? (See Expressions.)

```
nHelpMeOutHere = 32;
```