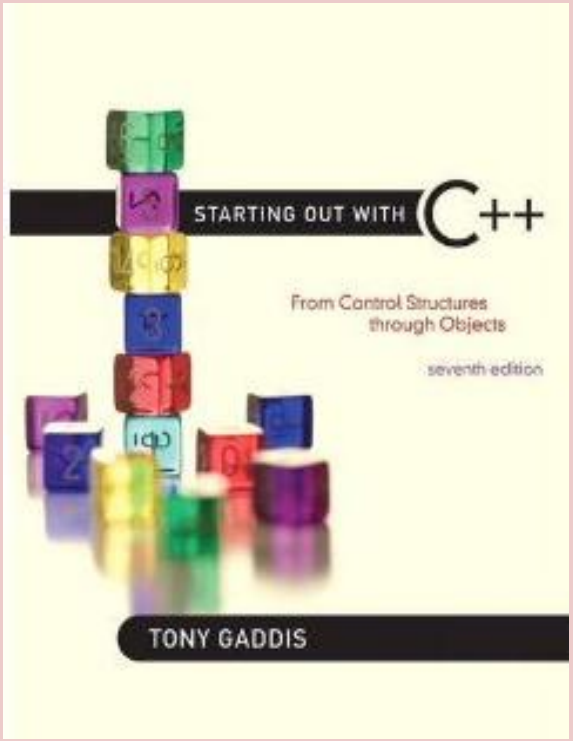
Software Design & Programming I



Starting Out with C++ (From Control Structures through Objects) 7th Edition
Written by: Tony Gaddis
Pearson - Addison Wesley
ISBN: 13-978-0-132-57625-3

Chapter 1

Introduction to Computers and Programming

Why Program?

Computers can do such a wide variety of things because they can be programmed. This means that computers are not designed to do just one job, but any job that their programs tell them to do. A program is a set of instructions that a computer follows to perform a task.

Programs are commonly referred to as **software**. Software is essential to a computer because without software, a computer can do nothing. All of the software that we use to make our computer useful is created by individuals known as programmers or software developers. A programmer, or software developer, is a person with the training and skills necessary to design, create, and test computer programs. Computer programming is an exciting and rewarding career.

Today, you will find programmers working in business, medicine, government, law enforcement, agriculture, academics, entertainment, and almost every other field.

Computer programming is both an art and a science. It is an art because every aspect of a program should be carefully designed.

Listed below are a few of the things that must be designed for any real-world computer program:

- The logical flow of the instructions
- The mathematical procedures
- The appearance of the screens
- The way information is presented to the user
- The program's "user-friendliness"
- Manuals and other forms of written documentation

There is also a scientific, or engineering side to programming. Because programs rarely work right the first time they are written, a lot of testing, correction, and redesigning is required. This demands patience and persistence from the programmer. Writing software demands discipline as well. Programmers must learn languages like C++ because computers do not understand English or other human languages. Languages such as C++ have strict rules that must be carefully followed.

Hardware and Software

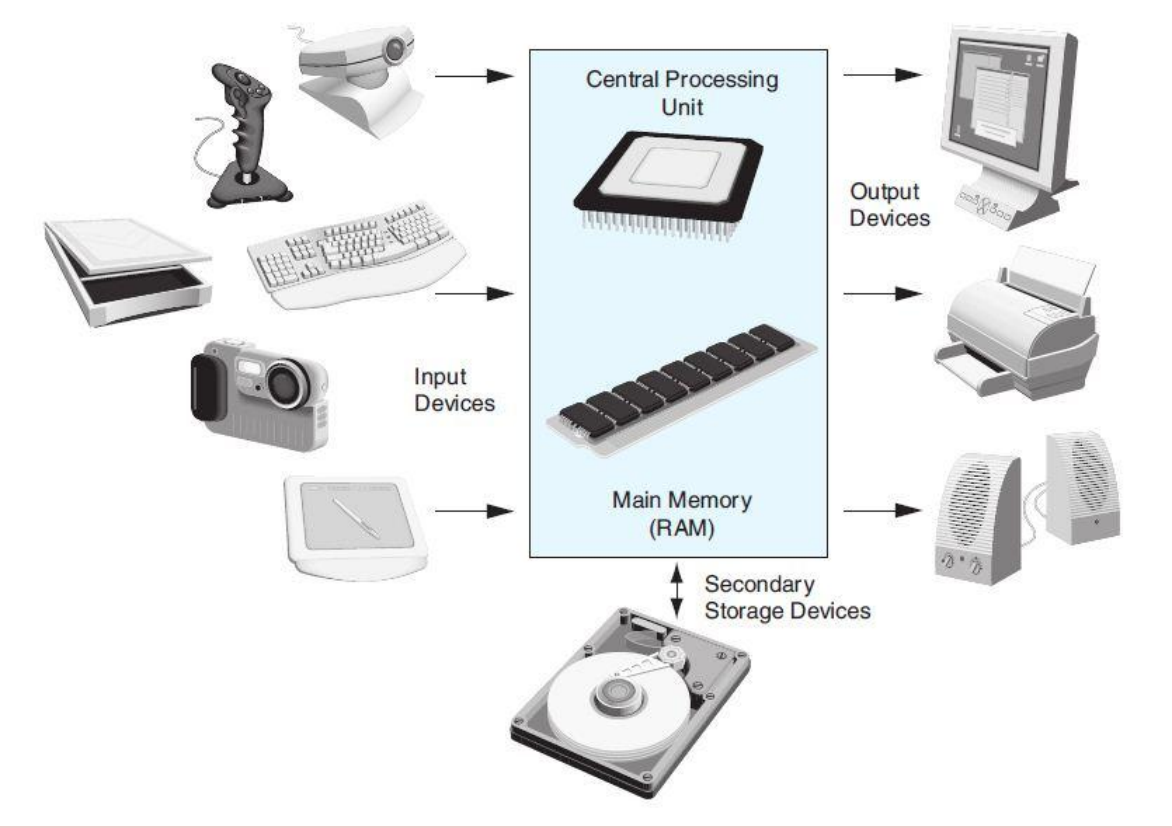
All computer systems consist of similar hardware devices and software components.

Hardware

Hardware refers to the physical components that a computer is made of. A computer, as we generally think of it, is not an individual device, but a system of devices.

A typical computer system consists of the following major components:

1. The central processing unit (CPU), 2. Main memory (RAM), 3. Secondary storage devices (hard drive, disks, etc.), 4. Input devices (keyboard, mouse), 5. Output devices (printer, speakers)

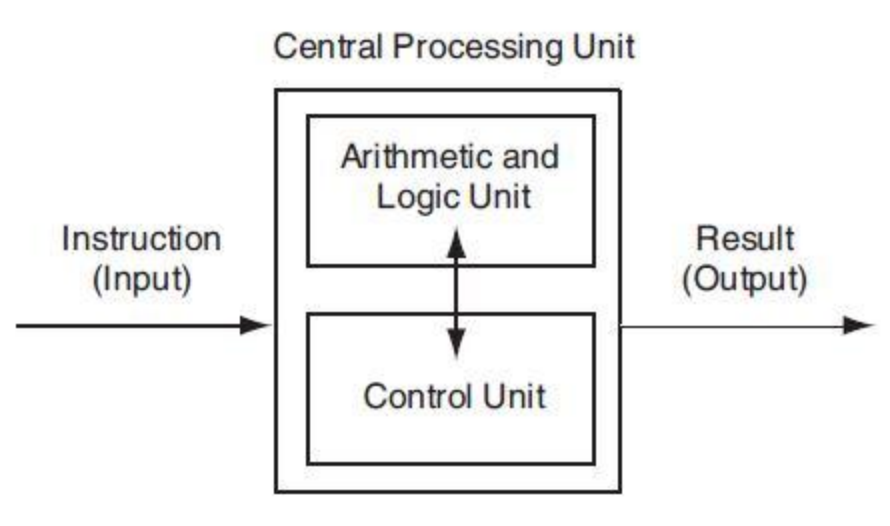


CPU

When a computer is performing the tasks that a program tells it to do, we say that the computer is running or executing the program. The central processing unit, or CPU, is the part of a computer that actually runs programs. The CPU is the most important component in a computer because without it, the computer could not run software.

The CPU's job is to fetch instructions, follow the instructions, and produce some result. Internally, the CPU consists of two parts: the control unit and the arithmetic and logic unit (ALU). The control unit coordinated all of the computer's operations. It is responsible for determining where to get the next instruction and regulating the other major components of the computer with control signals. The ALU is designed to perform mathematical operations.

CPU (Instruction Flow)



A program is sequence of instructions stored in the computer's memory. When a computer is running a program, the CPU is engaged in a process known formally as the **fetch/decode/execute cycle**.

The steps in the fetch/decode/execute cycle are as follows:

- Fetch — The CPU's control unit fetches, from main memory, the next instruction in the sequence of program instructions.
- Decode — The instruction is encoded in the form of a number. The control unit decodes the instruction and generates an electronic signal.
- Execute — The signal is routed to the appropriate component of the computer (ALU, disk drives, etc.) The signals cause the components to perform an operation.

These steps are repeated as long as there are instructions to perform.

Main Memory

You can think of main memory as the computer's work area. This is where the computer stores a program while the program is running, as well as the data that the program is working with.

Main memory is commonly known as random access memory (RAM). It is called this because the CPU is able to quickly access data stored at any random location in RAM. RAM is usually a volatile type of memory that is used only for temporary storage while the program is running. When the computer is turned off, the contents of RAM is erased.

A computer's memory is divided into tiny storage locations known as bytes. One byte is enough memory to store only a letter of the alphabet or a small number. Most computers today have millions, or even billions, of bytes of memory.

Each byte is divided into eight smaller storage locations known as bits. The term bit stands for binary digit. Computer scientists usually think of bits as tiny switches that can be either on or off (1's and 0's)

Secondary Storage

Secondary storage is a type of memory that can hold data for long periods of time-even when there is no power to the computer. Frequently used programs are stored in secondary memory and loaded into main memory as needed. Important information, such as word processing documents, payroll data, and inventory figures, is saved to secondary storage as well.

The most common type of secondary storage device is the disk drive. A disk drive stores data by magnetically encoding it onto a circular disk.

Input Devices

Input is any information the computer collects from the outside world. The device that collects the information and sends it to the computer is called an input device. Common input devices are the keyboard, mouse, scanner, digital camera, and microphone. Disk drive, CD/DVD drives, and USB drives can also be considered input devices because programs and information are retrieved from them and loaded into the computer's memory.

Output Devices

Output is any information the computer sends to the outside world. It might be a sales report, a list of names, or a graphic image. The information is sent to an output device, which formats and presents it. Common output devices are monitors, printers, and speakers. Output sent to a monitor is sometimes called “softcopy,” while output sent to a printer is called “hardcopy.” Disk drives, USB drives, and CD/DVD recorders can also be considered output devices because the CPU sends them information to be saved.

Hardware and Software

All computer systems consist of similar hardware devices and software components.

Software

If a computer is to function, software is not optional. Everything that a computer does from the time you turn the power switch on until you shut the system down, is under the control of software. There are two general categories of software: system software and application software. Most computer program clearly fit into one of these two categories.

System Software

The programs that control and manage the basic operations of a computer are generally referred to as system software. System software typically includes the following types of programs:

- **Operating systems**: an operating system is the most fundamental set of programs on a computer. The operating system controls the internal operations of the computer's hardware, manages all devices connected to the computer, allows data to be saved to and retrieved from storage devices, and allow other programs to run on the computer.

System Software (cont.)

- **Utility Programs**: a utility program performs a specialized task that enhances the computer's operation or safeguards data. Examples of utility programs are virus scanners, file-compression programs, and data-backup programs.
- **Software Development Tools**: the software tools that programmers use to create, modify, and test software. Compilers and integrated development environments are examples of programs that fall into this category.

Application Software

Programs that make a computer useful for everyday tasks are known as application software. These are the programs that people normally spend most of their time running on their computers.

Programs and Programming Languages

A program is a set of instructions a computer follows in order to perform a task. A programming language is a special language used to write computer programs.

What is a Program?

A computer program is a set of instructions that tells the computer how to solve a problem or perform a task.

A program that calculates someone's pay should do the following:

1. Display a message on the screen asking "How many hours did you work?"
2. Wait for the user to enter the number of hours worked. Once the user enters a number, store it in memory.
3. Display a message on the screen asking "How much do you get paid per hour?"
4. Wait for the user to enter an hourly pay rate. Once the user enters a number, store it in memory.
5. Multiply the number of hours by the amount paid per hour, and store the results in memory
6. Display a message on the screen that tells the amount of money earned. The message must include the result of the calculation performed in step 5.

Collectively, the previous instructions (steps) are called an algorithm. An algorithm is a set of well-defined steps for performing a task or solving a problem. Notice these steps are sequentially ordered. Step 1 should be performed before Step 2, and so forth. It is important that these instructions be performed in their proper sequence.

Although you and I may understand the instruction in the pay-calculating algorithm, it is not ready to be executed by a computer. A computer can only process instructions that are written in machine language. (0's and 1's)

Pay-calculating Algorithm

Program 1-1

```
1  // This program calculates the user's pay.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     double hours, rate, pay;
8
9     // Get the number of hours worked.
10    cout << "How many hours did you work? ";
11    cin >> hours;
12
13    // Get the hourly pay rate.
14    cout << "How much do you get paid per hour? ";
15    cin >> rate;
16
17    // Calculate the pay.
18    pay = hours * rate;
19
20    // Display the pay.
21    cout << "You have earned $" << pay << endl;
22    return 0;
23 }
```

Program Output with Example Input Shown in Bold

```
How many hours did you work? 10 [Enter]
How much do you get paid per hour? 15 [Enter]
You have earned $150
```

Inclass/Homework Assignment (Program #1)

Using Visual Studio 2012, input, successfully compile, and print source code and output. *Due date will be given once assigned.*

Also: Following the first commented line in the program add three additional commented lines that include you First and Last Name, Course Number and Section and Date Submitted.

Ex:

```
// *****  David Sylvester  *****
// *****    CSCI 193-02    *****
// ***** January 22, 2013 *****
```

Assignment should be turned in stapled with a Cover Sheet, Source Code, Output, and Data Dictionary.

All documents must be typed and printed through the use of a printer, and stapled in the order stated above. ***No exceptions!!***

Homework Assignment (Program #2)

Using Visual Studio 2012, write a program that will input three numbers and then calculate and print the sum, then calculate and print the average. *Due date will be given once assigned.*

Also: The first four lines should be commented with Program #, your First and Last Name, Course Number and Section and Date Submitted. All cin, cout and assignment statements should be commented.

Ex:

```
// *****     Program #2     *****
// *****  David Sylvester   *****
// *****     CSCI 193-02    *****
// ***** January 22, 2013 *****
```

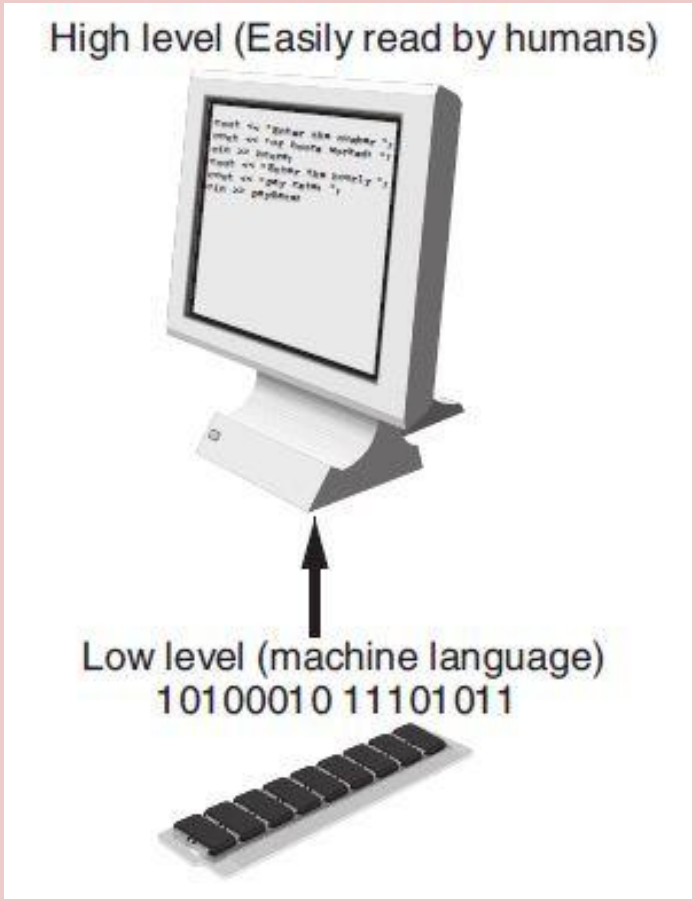
Assignment should be turned in stapled with a Cover Sheet, Statement of the Problem, Flowchart, Source Code, Output, and Data Dictionary.

All documents must be typed and printed through the use of a printer, and stapled in the order stated above. ***No exceptions!!***

Programming Languages

There are two categories of programming languages: low-level and high-level. A low-level language is close to the level of the computer, which means it resembles the numeric machine language of the computer more than the natural language of humans. The easiest languages for people to learn are high-level languages. They are called “high-level” because they are closer to the level of human-readability than computer-readability.

Language Levels



C++ is based on the C language, which was invented for purposes such as writing operating systems and compilers. Since C++ evolved from C, it carries all of C's low-level capabilities with it.

C++ is popular not only because of its mixture of low- and high-level features, but also because of its portability.

Well-known High-Level Languages

Language	Description
BASIC	Beginners All-purpose Symbolic Instruction Code. A general programming language originally designed to be simple enough for beginners to learn.
FORTRAN	Formula Translator. A language designed for programming complex mathematical algorithms.
COBOL	Common Business-Oriented Language. A language designed for business applications.
Pascal	A structured, general-purpose language designed primarily for teaching programming.
C	A structured, general-purpose language developed at Bell Laboratories. C offers both high-level and low-level features.
C++	Based on the C language, C++ offers object-oriented features not found in C. Also invented at Bell Laboratories.
C#	Pronounced "C sharp." A language invented by Microsoft for developing applications based on the Microsoft .NET platform.
Java	An object-oriented language invented at Sun Microsystems. Java may be used to develop programs that run over the Internet, in a Web browser.
JavaScript	JavaScript can be used to write small programs that run in Web pages. Despite its name, JavaScript is not related to Java.
Python	Python is a general purpose language created in the early 1990s. It has become popular in both business and academic applications.
Ruby	Ruby is a general purpose language that was created in the 1990s. It is increasingly becoming a popular language for programs that run on Web servers.
Visual Basic	A Microsoft programming language and software development environment that allows programmers to quickly create Windows-based applications.

Source Code, Object Code & Executable Code

When a C++ program is written, it must be typed into the computer and saved to a file. A text editor, which is similar to a word processing program, is used for this task. The statements written by the programmer are called source code, and the file they are saved in is called the source file.

After the source code is saved to a file, the process of translating it to machine language can begin.

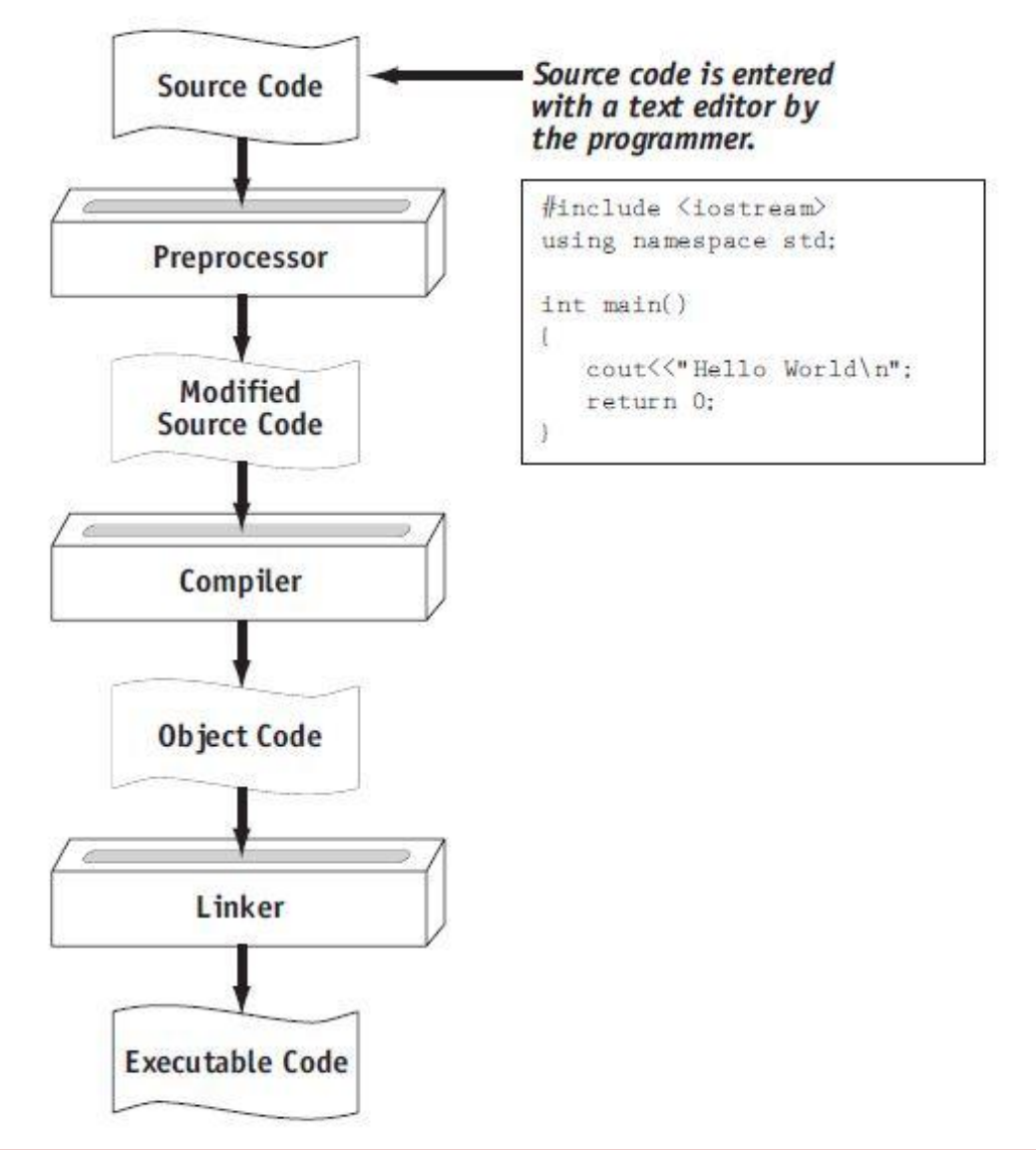
During the first phase of this process (translation), a program called the preprocessor reads the source code. The preprocessor searches for special lines that begin with the **#** symbol. These lines contain commands that cause the preprocessor to modify the source code in some way. During the next phase the compiler steps through the preprocessor source code, translating each source code instruction into the appropriate machine language instruction. This process will uncover any syntax errors that may be in the program. Syntax errors are illegal uses of key words, operators, punctuation and other language elements

If the program is free of syntax errors, the compiler stores the translated machine language instructions, which is called object code, in an object file.

Although an object file contains machine language instructions, it is not a complete program, because C++ is equipped with a library or prewritten code for performing common operations or sometimes-difficult tasks. Such as: libraries for IO, math function, etc. This code call the run-time library, is extensive. Programs almost always use part of it.

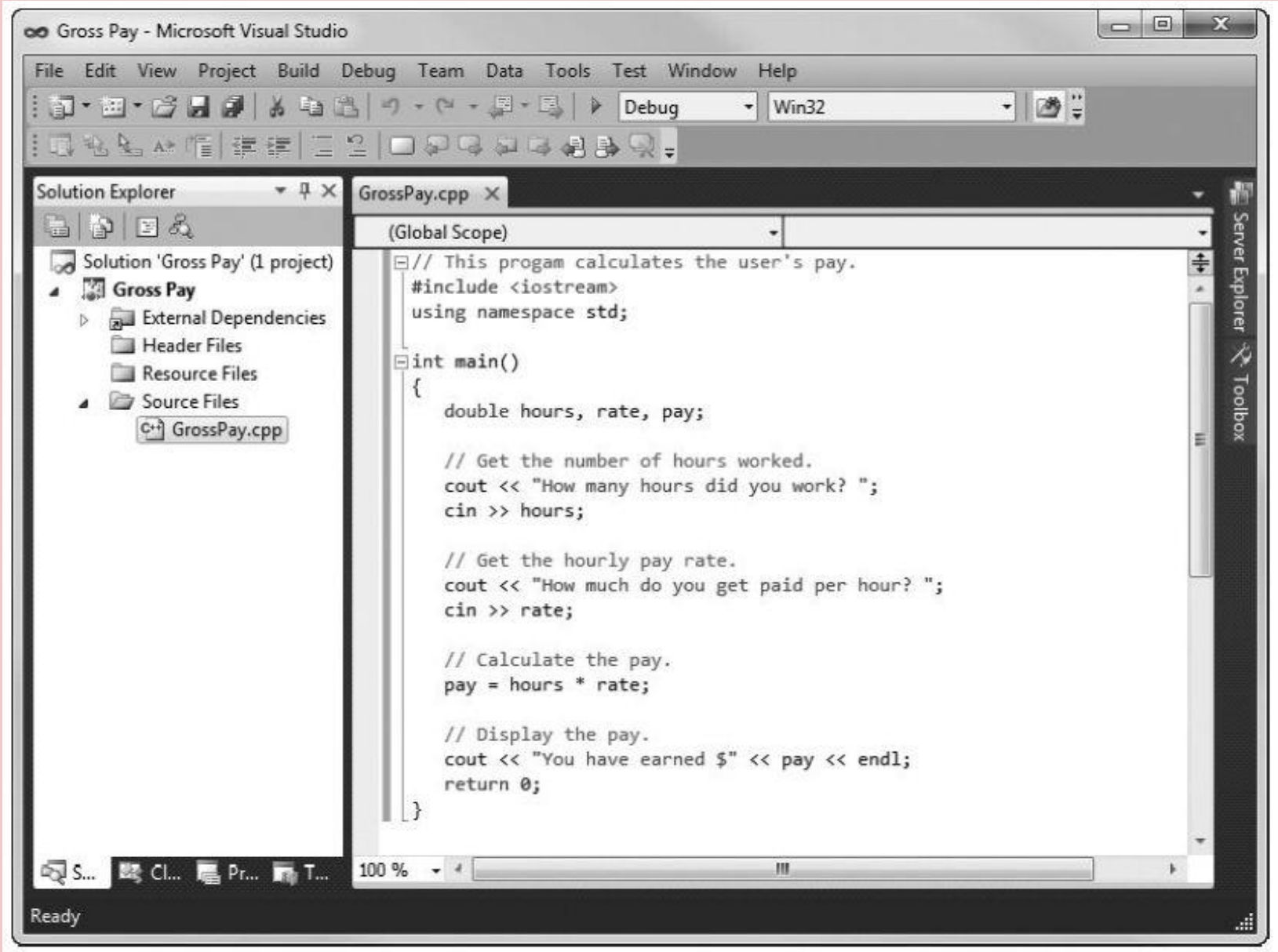
When the compiler generates an object file, however, it does not include machine code for any run-time library routines the programmer might have used. During the last phase of the translation process, another program called the linker combines the object file with the necessary library routines. Once the linker has finished with this step, an executable file is created. The executable file contains machine language instructions, or executable code, and is ready to run on the computer.

From Source Code to Executable File



Many development systems, particularly those on personal computers, have integrated development environments (IDE's). These environments consist of a text editor, compiler, debugger, and other utilities integrated into a package with a single set of menus. Preprocessing, compiling, linking, and even executing a program is done with a single click of a button, or by selecting a single item from a menu.

Microsoft Visual Studio (IDE)



What is a Program Made of?

Language Element	Description
Key Words	Words that have a special meaning. Key words may only be used for their intended purpose. Key words are also known as reserved words.
Programmer-Defined Identifiers	Words or names defined by the programmer. They are symbolic names that refer to variables or programming routines.
Operators	Operators perform operations on one or more operands. An operand is usually a piece of data, like a number.
Punctuation	Punctuation characters that mark the beginning or ending of a statement, or separate items in a list.
Syntax	Rules that must be followed when constructing a program. Syntax dictates how key words and operators may be used, and where punctuation symbols must appear.

Program Elements

Key Words (Reserved Words)

- using
- namespace
- double

These words, which are always written in lowercase, each have a special meaning in C++ and can only be used for their intended purposes. Part of learning a programming language is learning what the key words are, what they mean, and how to use them.

Programmer-Defined Identifiers

In Program 1-1, the words hour, rate and pay that appear in lines 7, 11, 15, 18 and 21 are programmer-defined identifiers. They are not part of the C++ language but rather are names made up by the programmer. In this particular program, these are the names of variables. Variables are the names of memory locations that may hold data.

Operators

On line 18 the following code appears:

pay = hours * rate;

The = and * symbols are both operators. They perform operations on pieces of data know as operands. The * operator multiplies its two operands, which in this example are the variables hours and rate. The = symbol is called the assignment operator. It takes the value of the expression on the right and stores it in the variable whose name appears on the left.

Punctuation

Notice that lines 3, 7, 10, 11, 14, 15, 18, 21 and 22 all end with a semicolon. A semicolon in C++ is similar to a period in English. It marks the end of a complete sentence (or statement, as it is called in programming jargon). Semicolons do not appear at the end of every line in a C++ program, however. There are rules that govern where semicolons are required and where they are not. Part of learning C++ is learning where to place semicolons and other punctuation symbols.

Lines and Statements

Often, the contents of a program are thought of in terms of lines and statements. A “line” is just that-a single line as it appears in the body of a program. Program 1-1 is shown with each of its lines numbered. Most of the lines contain something meaningful; however, some of the lines are empty. The blank lines are only there to make the program more readable.

A statement is a complete instruction that causes the computer to perform some action.

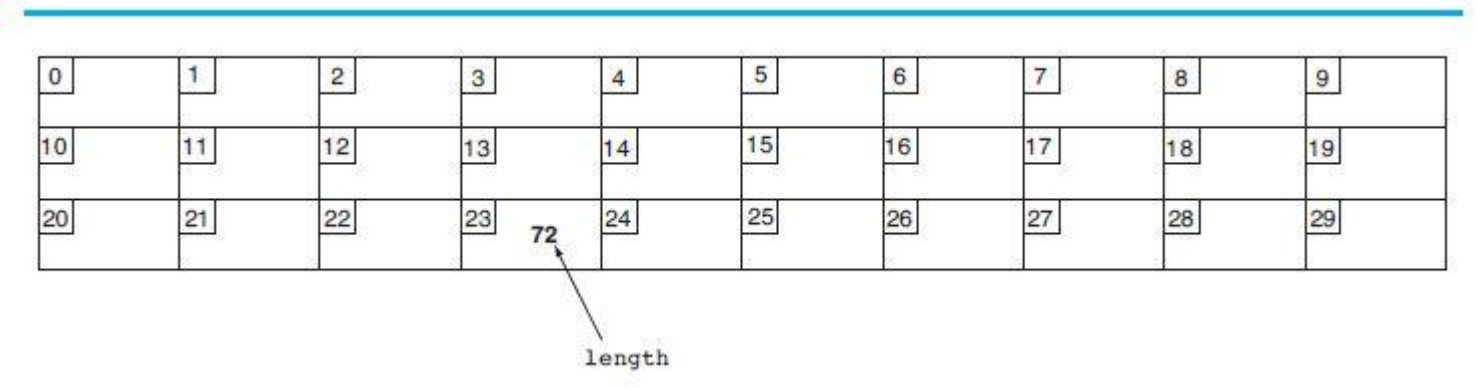
cout << “How many hours did you work? “;

Variables

A variable is a named storage location in the computer's memory for holding a piece of information. The information stored in variables may change while the program is running (hence the name "variable"). Notice that in Program 1-1 the words **hours**, **rate** and **pay** appears in several places.

Note: Notice that the variables have names that reflect their purpose.

Variables are symbolic names that represent locations in the computer's random-access memory (RAM). When information is stored in a variable, it is actually stored in RAM. Assume a program has a variable named **length**; this could be the way the variable name is represented in memory.



The variable length is holding the value 72. The number 72 is actually stored in RAM at address 23, but the name length symbolically represents this storage location. **72 is replaced when another value is stored in address 23.**

Variables Definitions

In programming, there are two general types of data: numbers and characters. Numbers are used to perform mathematical operations and characters are used to print data on the screen or on paper.

Numeric data can be categorized even further. (whole numbers and real numbers).

Whole numbers (integer)	Real number (floating point)
5 7 -129 32154	3.14159 6.7 1.002

When creating a variable in a C++ program, you must know what type of data the program will be storing in it. On line 7 of Program 1-1;

double hours, rate, pay;

The word double in this statement indicates that the variables hours, rate, and pay will be used to hold double precision floating-point numbers. This statement is called a variable definition. It is used to define one or more variables that will be used in the program, and to indicate the type of data they hold. The variable definition causes the variable to be created in memory, so all variables must be defined before they can be used. Note: Variable definitions must come before any other statement using those variables.

Input, Processing, and Output

These are the three activities of a program.

Computer programs typically perform a three-step process of gathering input, performing some process on the information gathered, and then producing output. Input is information a program collects from the outside world. It can be sent to the program from the user, who is entering data at the keyboard or using the mouse. It can also be read from disk files or hardware devices connected to the computer.

Program 1-1 allows the user to enter two pieces of information: The number of hours worked and the hourly pay rate. Lines 11 and 15 use the **cin**, (pronounced “see in”), object to perform these input operations.

cin >> hours;

cin >> rate;

Once information is gathered from the outside world, a program usually processes it in some manner. In the program, the hours worked and hourly pay are multiplied in line 18 and the result is assigned to pay rate.

pay = hours * rate;

Output is information that a program sends to the outside world. It can be words or graphics displayed on a screen, a report sent to the printer, data stored in a le, or information sent to any device connected to the computer. Lines 10, 14, and 21 in Program 1-1 all perform output:

```
cout << "How many hours did you work? ";
cout << "How much do you get paid per hour? ";
cout << "You have earned $" << pay << endl;
```

These lines use the **cout,** (pronounced “see out”), object to display messages on the computer’s screen.

The Programming Process

The programming process consists of several steps, which include design, creation, testing, and debugging activities.

Designing and Creating a Program

Quite often, when inexperienced students are given programming assignments, they have trouble getting started because they don't know what to do first. If you find yourself in this dilemma, follow these recommended steps.

Steps in Writing a Program

1. Clearly define what the program is to do.
2. Visualize the program running on the computer.
3. Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.
4. Check the model for logical errors.
5. Type the code, save it, and compile it.
6. Correct any errors found during compilation.

   Repeat Steps 5 and 6 as many times as necessary.
7. Run the program with test data for input.
8. Correct any errors found while running the program.

   Repeat Steps 5 through 8 as many times as necessary.
9. Validate the results of the program.

Step 1 – Clearly define what the problem is to do.

This step requires that you identify the purpose of the program, the information that is to be input, the processing that is to take place, and the desired output. Let s examine each of these requirements for the example program:

Purpose	To calculate the user's gross pay.
Input	Number of hours worked, hourly pay rate.
Process	Multiply number of hours worked by hourly pay rate. The result is the user's gross pay
Output	Display a message indicating the user's gross pay.

Step 2 – Visualize the program running on the computer.

Try to imagine what the computer screen looks like while the program is running. If it helps, draw pictures of the screen, with sample input and output, at various points in the program.

```
How many hours did you work? 10
How much do you get paid per hour? 15
You have earned $150
```

In this step, you must put yourself in the shoes of the user. What messages should the program display? What questions should it ask? By addressing these concerns, you will have already determined most of the program's output.

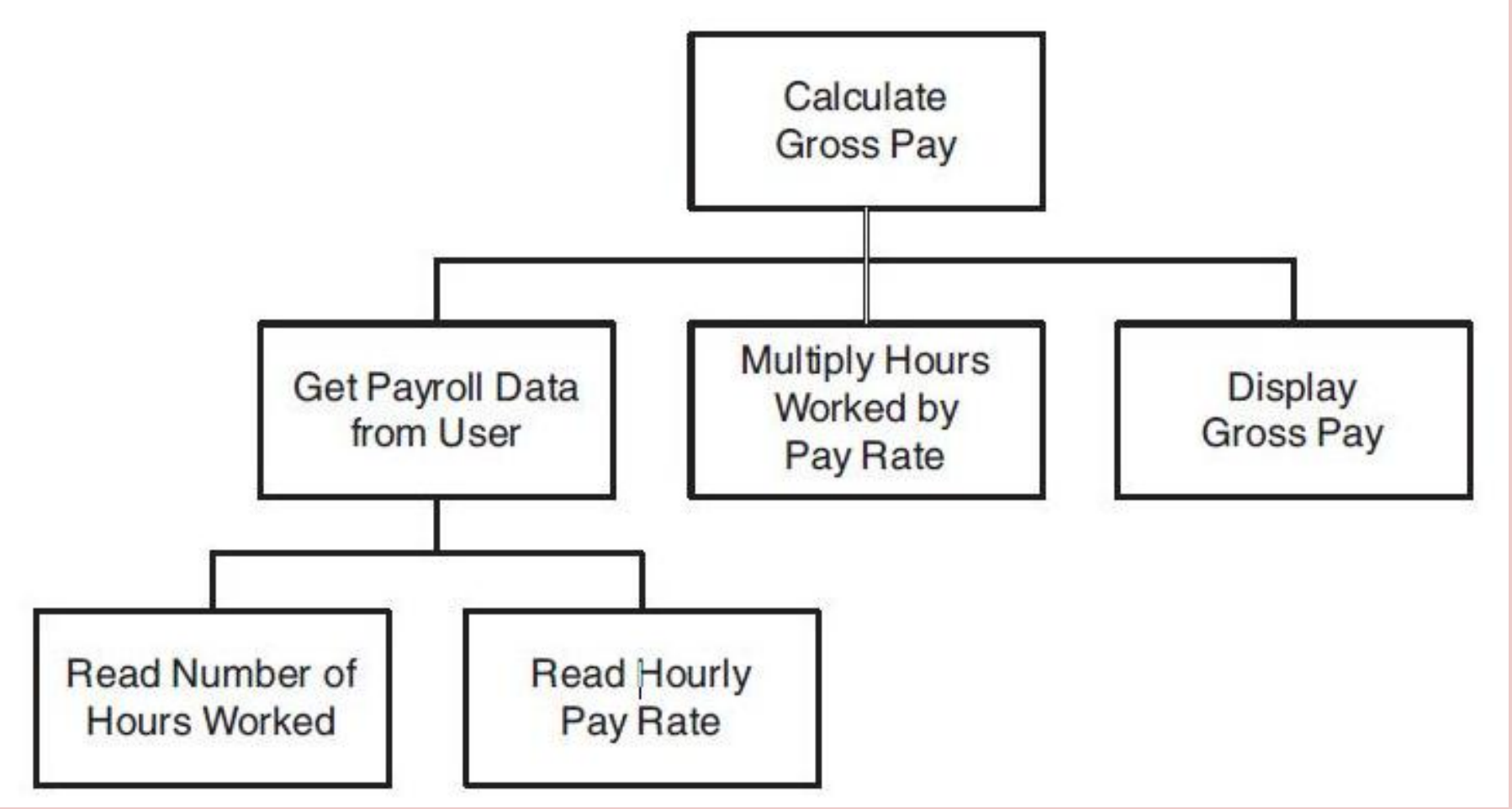
Step 3 – Use design tools such as a hierarchy chart, flowcharts, or pseudocode to create a model of the program.

Three common design tools are hierarchy charts, flowcharts, and pseudocode. A hierarchy chart is a diagram that graphically depicts the structure of a program. It has boxes that represent each step in the program. The boxes are connected in a way that illustrates their relationship to one another.

Hierarchy Chart

A hierarchy chart begins with the overall task, and then refines it into smaller subtasks. Each of the subtasks is then refined into even smaller sets of subtasks, until each is small enough to be easily performed. For instance, in Figure 1-10, the overall task Calculate Gross Pay is listed in the top-level box. That task is broken into three subtasks. The first subtask, Get Payroll Data from User, is broken further into two subtasks. This process of divide and conquer is known as top-down design.

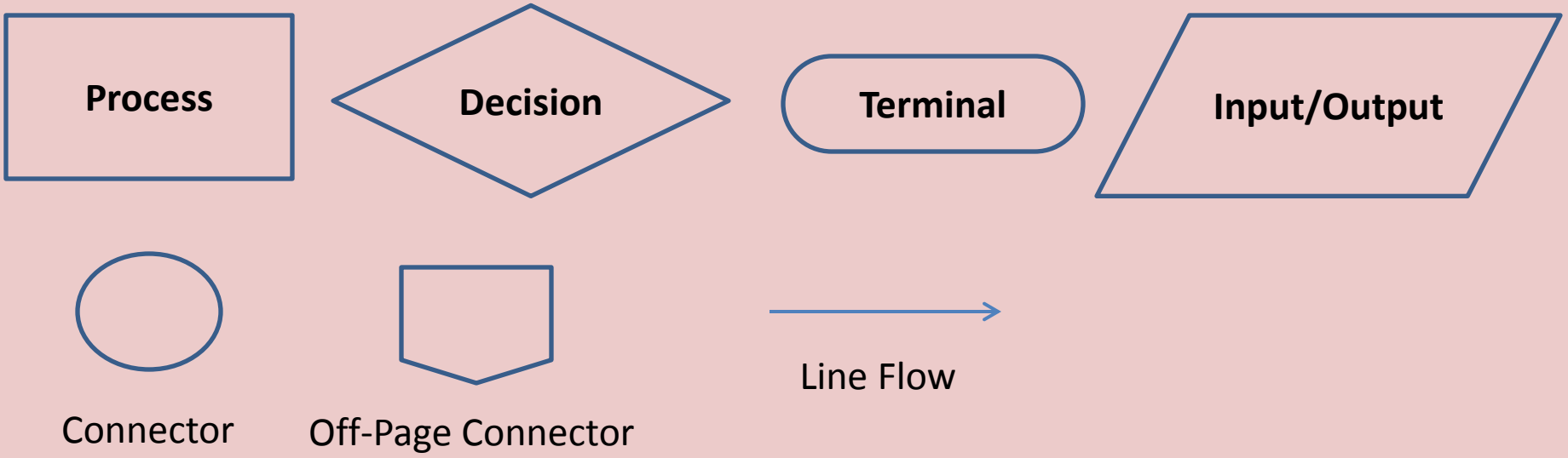
Hierarchy Chart Example



Flowchart

A flowchart is a diagram that shows the logical flow of a program. It is a useful tool for planning each operation a program performs, and the order in which the operations are to occur.

Common Flowchart Symbols



Pseudocode

Pseudocode is a cross between human language and a programming language. Although the computer can't understand pseudocode, programmers often find it helpful to write an algorithm in a language that s almost a programming language, but still very similar to natural language. For example, here is pseudocode that describes the pay-calculating program:

Get payroll data.

Calculate gross pay.

Display gross pay.

The pseudocode above gives a broad view of the program; it doesn't reveal all the program's details. A more detailed version of the pseudocode follows.

Display How many hours did you work? .
Input hours.
Display How much do you get paid per hour? .
Input rate.
Store the value of hours times rate in the pay variable.
Display the value in the pay variable.

Notice the pseudocode contains statements that look more like commands than the English statements that describe the algorithm.

Step 4 – Check the model for logical errors.

Logical errors are mistakes that cause the program to produce erroneous results. Once a hierarchy chart, flowchart, or pseudocode model of the program is assembled, it should be checked for these errors. The programmer should trace through the charts or pseudocode, checking the logic of each step. If an error is found, the model can be corrected before the next step is attempted.

Step 5 – Type the code, save it and compile it.

Once a model of the program (hierarchy chart, flowchart, or pseudocode) has been created, checked, and corrected, the programmer is ready to write source code on the computer. The programmer saves the source code to a file, and begins the process of translating it to machine language. During this step the compiler will find any syntax errors that may exist in the program.

Step 6 – Correct any errors found during compilation.

If the compiler reports any errors, they must be corrected. Steps 5 and 6 must be repeated until the program is free of compile-time errors.

Step 7 – Run the Program with test data for input.

Once an executable le is generated, the program is ready to be tested for run-time errors. A run-time error is an error that occurs while the program is running. These are usually logical errors, such as mathematical mistakes.

Testing for run-time errors requires that the program be executed with sample data or sample input. The sample data should be such that the correct output can be predicted. If the program does not produce the correct output, a logical error is present in the program.

Step 8 – Correct any run-time errors found while running the program.

When run-time errors are found in a program, they must be corrected. You must identify the step where the error occurred and determine the cause. Desk-checking is a process that can help locate run-time errors. The term desk-checking means the programmer starts reading the program, or a portion of the program, and steps through each statement. A sheet of paper is often used in this process to jot down the current contents of all variables and sketch what the screen looks like after each output operation. When a variable's contents change, or information is displayed on the screen, this is noted. By stepping through each statement, many errors can be located and corrected. If an error is a result of incorrect logic (such as an improperly stated math formula), you must correct the statement or statements involved in the logic.

Step 9 – Validate the results of the program.

When you believe you have corrected all the run-time errors, enter test data and determine whether the program solves the original problem.

What is Software Engineering?

It includes designing, writing, testing, debugging, documenting, modifying, and maintaining complex software development projects. Like traditional engineers, software engineers use a number of tools in their craft. Here are a few examples:

- Program specifications
- Charts and diagrams of screen output
- Hierarchy charts and flowcharts
- Pseudocode
- Examples of expected input and desired output
- Special software designed for testing programs

Most commercial software applications are very large. In many instances one or more teams of programmers, not a single individual, develop them. It is important that the program requirements be thoroughly analyzed and divided into subtasks that are handled by individual teams, or individuals within a team.

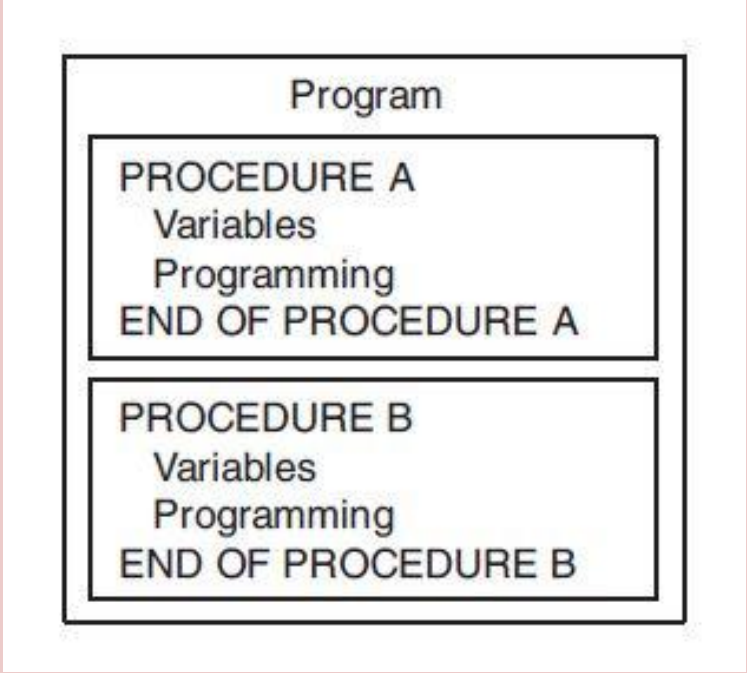
If the program is very large or complex, a team of software engineers can be assigned to work on the individual modules. As the project develops, the modules are coordinated to finally become a single software application.

Procedural and Object-Oriented Programming

C++ is a language that can be used for two methods of writing computer programs: procedural programming and object-oriented programming.

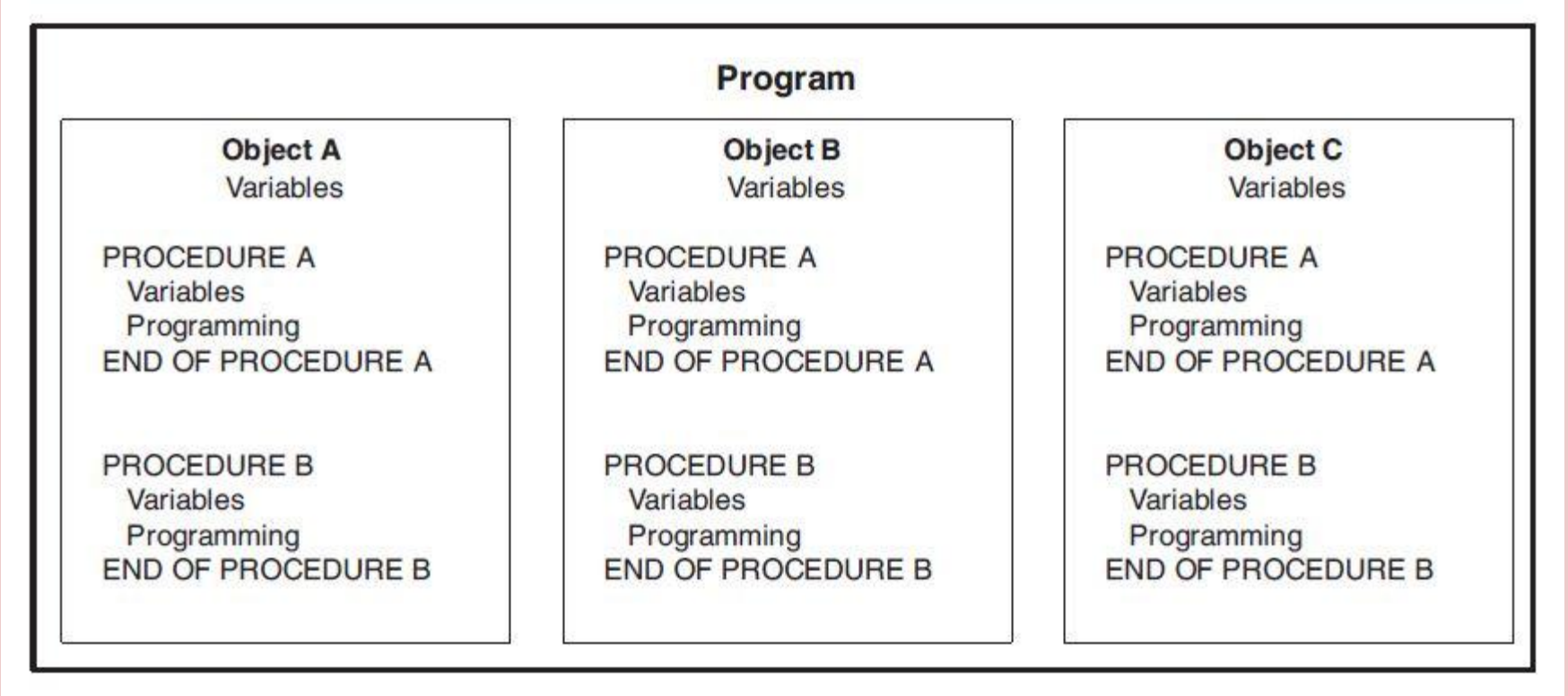
In procedural programming, the programmer constructs procedures (or functions, as they are called in C++). The procedures are collections of programming statements that perform a specific task. The procedures each contain their own variables and commonly share variables with other procedures.

Procedural Programming Example



Procedural programming is centered on the procedure, or function.

Object-Oriented Programming Example



Object-oriented programming (OOP), on the other hand, is centered on the object. An object is a programming element that contains data and the procedures that operate on the data. It is a self-contained unit.

The objects contain, within themselves, both information and the ability to manipulate the information. Operations are carried out on the information in an object by sending the object a message. When an object receives a message instructing it to perform some operation, it carries out the instruction.