Program Development
and
Integrated Development Environments

In order to write, compile and run a computer program, you need a text editor, a compiler, and a linker. Originally, these were separate programs. Now, typically all three functions are combined into a single software package called an Integrated Development Environment (IDE).

For Windows platforms, the best-known IDE is Microsoft Visual Studio, which supports development in C, C++, C#, Visual BASIC, and other languages. A version that supports only C++, called Microsoft Visual C++, is available as a free download. Apple's general-purpose IDE for the MAC is called XCode and is included with OS-X (but not automatically installed). For the Java language, the best knows IDEs are Netbeans and Eclipse. Dev-C++ is a simple freeware IDE for C++.

The principal components of an IDE are a text editor, a compiler, a library, a linker, and a debugger. This note explains what the text editor, compiler, and linker do, and how the library is used. Use of the debugger is beyond the scope of this course.

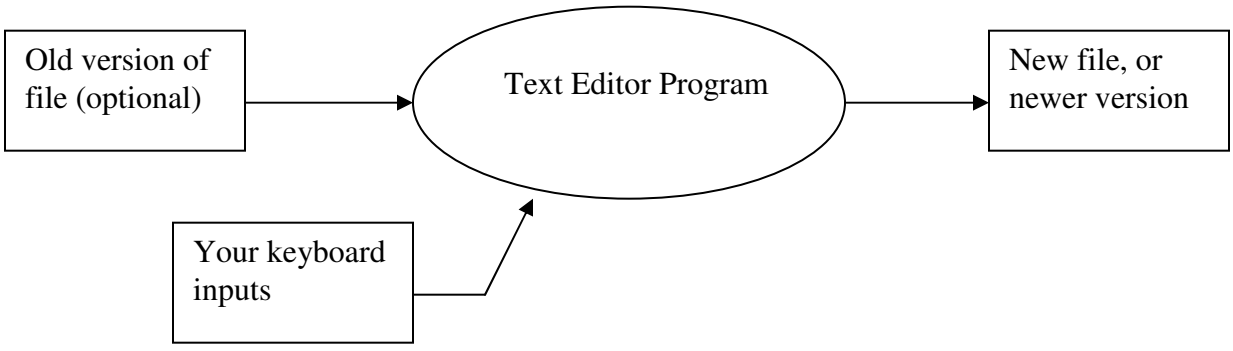
Text Editors

A text editor is a program that you can use to construct and edit a text file. A text file is a data file containing only printable characters (letters, digits, punctuation marks, and a few special symbols) and *newline*. A text file does *not* contain any formatting information like bold, italic, margins, or indentation.

The input to a text editor is an (optional) existing text file, and keystrokes that you type. The output is a new or modified text file.

The best known text editor is Microsoft Notepad. I'm sure most of you have had occasion to use it for something. If not, try it out – open it (look in Start/Programs/Accessories), type anything you like, save it with a name like `mystuff.xyz`, close the program, open it again, and edit your file. You'll see that it is like a word processor stripped down to the bare bones.

What a text editor does is shown in the following *flow diagram*:



Text files have many uses. Some contain information on parameters and options used by application programs; in Windows, such files have names ending in `.ini`. A text file may contain a C++ program; it is conventional to give such a file a name ending in `.cpp`. Or a text file may contain a Java program; it is conventional to give such a file a name ending in `.java`.

Text files that just contain nondescript data intended as input to some application often have names ending in `.txt`.

Compilers

You are probably aware that a personal computer has lots of electronic devices in it. (Have you ever looked inside a desktop computer? If not, you should.) The biggest and hottest of these (I mean that literally) is the *processor*. In many PC's, the processor is an Intel Pentium. There are various competitors. In some large or old computers, the processor is made up of several electronic devices instead of just one.

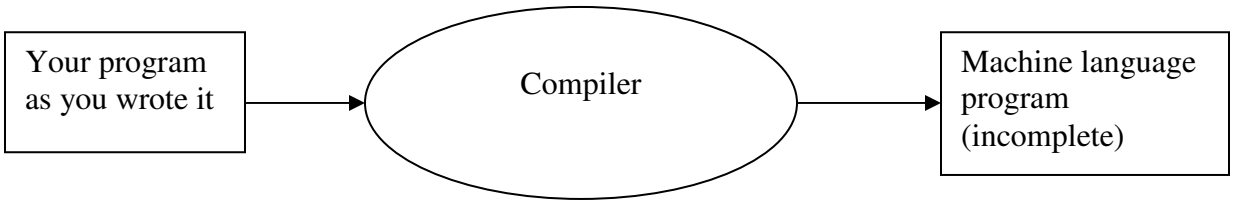
The processor does the hard work of a computer. It reads instructions from memory, and executes the instructions. Groups of instructions are, of course, programs.

The processor in a computer does not under stand C++ or Java or Visual Basic. It does not understand COBOL or FORTRAN either. It only understands an arcane language called *machine language*. Different makes/models of processors understand different machine languages.

So in order for your C++ program to run on your laptop PC with a Pentium processor, your C++ program (which is intelligible to you, but not the processor) must be *translated* into Pentium machine language (which is intelligible to the processor but not to you).

A program that does such a translation is called a *compiler*.

The following diagram shows what a compiler does:

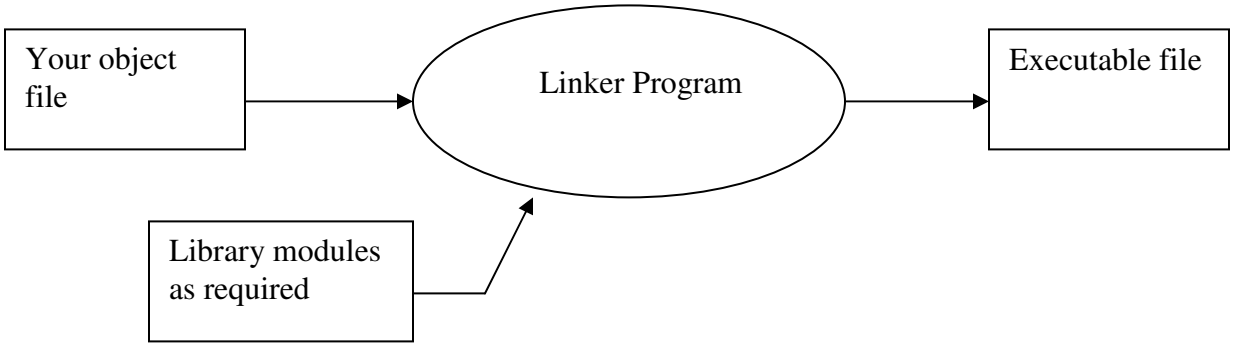


The output of a compiler is often called an *object file*, and is often given a name ending in `.o`.

The Linker

The object file made by the compiler is not a complete machine language program. In order to run on the processor, the object file must be combined with a variety of programs (actually pieces of programs called *modules*) that have been written by others. The *library* provided with

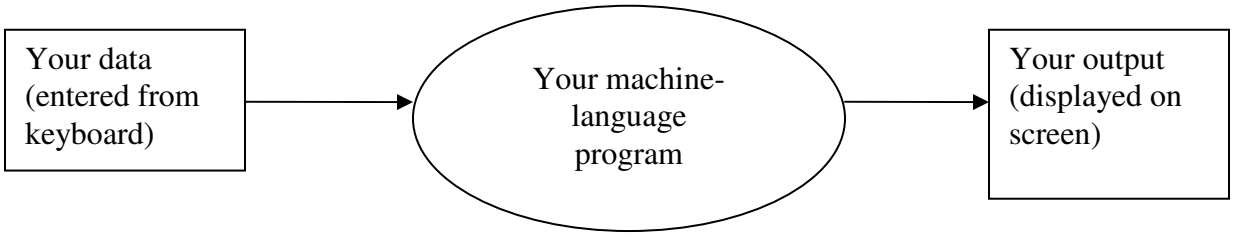
the IDE contains a wide variety of modules. The program that combines your object file with whichever library modules are needed as called the *linker*. The following diagram shows what the linker does:



The output of the linker is called an executable file, and is often given a name ending in `.exe`. It can be loaded into the computer's memory and run.

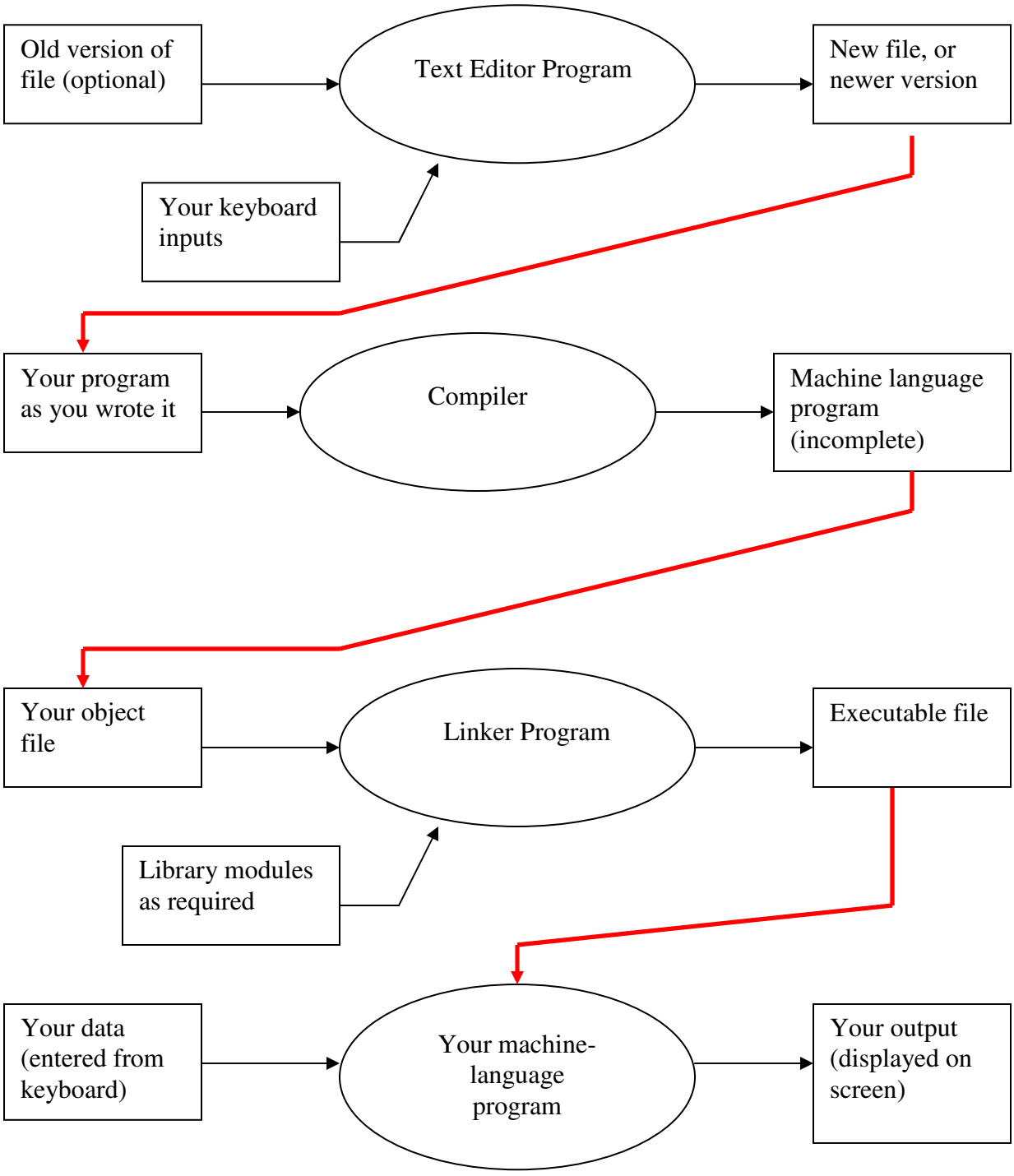
Your Program

We can make a similar flow diagram for your program:



The Big Picture

Now lets combine these four diagrams. The result is on the next page.


Old version of file (optional)
Text Editor Program
New file, or newer version
Your keyboard inputs
Your program as you wrote it
Compiler
Machine language program (incomplete)
Your object file
Linker Program
Executable file
Library modules as required
Your data (entered from keyboard)
Your machine-language program
Your output (displayed on screen)

Notice the three red arrows that have been added. The first shows that the output of the text editor *is the input to* the compiler. The second red arrow shows that the output of the compiler *is one of the inputs of* the linker; the other is the library. The third red arrow shows that the output of the linker *is* your application program.

In earlier days, the process of preparing a computer program for execution consisted of three separate steps:

- Use a text editing program to prepare the source program.
- Use a compiler to make an object file.
- Use a linker to make an executable file.

Programmers carried out each step separately, using separate programs.

Integrated Development Environments

In an IDE, the process we studied above takes place exactly as shown in the preceeding diagram. But the administration of all this is partially hidden from the user. In an IDE,

- The text editor is built into the IDE program. When you type a program, a text file is created (or modified) and saved.
- When you click Execute/Compile, the IDE requests the operating system to run the compiler program, handing it your C++ program as input. The compiler generates an object file. Then the IDE requests the operating system to run the linker, handing it your object file, and the library, as inputs. An executable file is generated.
- When you click Execute/Run, the IDE requests the operating system to run your executable program.
- When you click Execute/Compile and Run, the IDE does both of the above two bullets.

(Of course, the exact items you click on depend on which IDE you are using.)

So when you develop a program in an IDE, all the same steps occur as in a "classic" environment, but it is far more efficient for the programmer.

A Conundrum

You now understand that for one of us mere humans to write a computer program, we have to have a compiler, linker, and library available. (Before text editors were invented, we used punched cards or worse yet, punched paper tape.)

But the compiler and linker are programs, and the library is a collections of program modules. Who wrote these? Whoever did so needed to have a compiler and linker, right? Is this a chicken-and-egg issue? Do post your thoughts on this issue.