Programming Capabilities

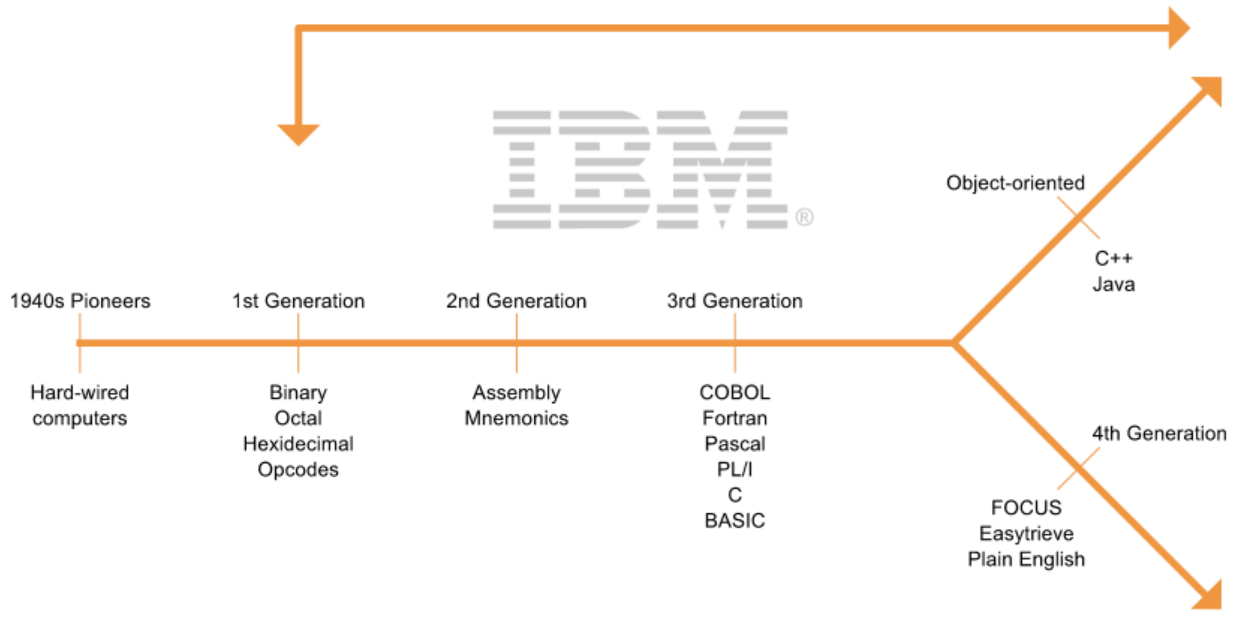
Objectives

Programming Capabilities

In this module, you will look at the most commonly used programming languages in the IBM enterprise environment. You will also discover where they fit in past and future strategies of IBM, and their strong points, weak points, and idiosyncrasies.

After completing this module, you will be able to:

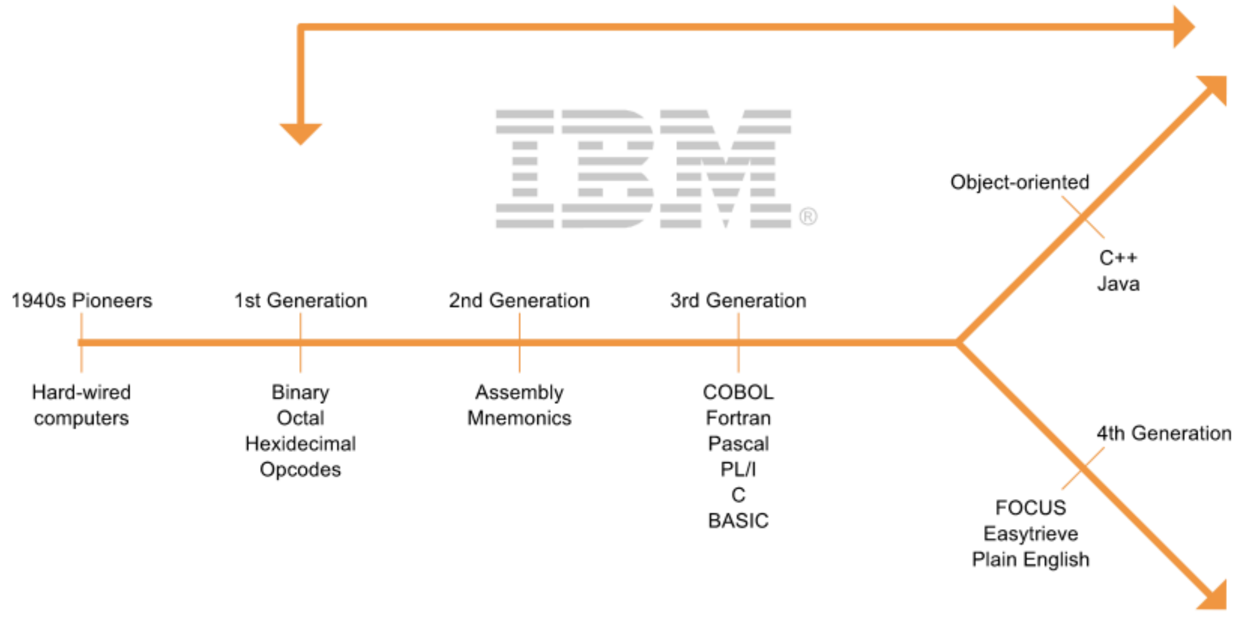
- Recognize Highlights in the Timeline of Programming Language Development
- Identify Commonly Used Programming Languages in the IBM Enterprise Environment
- Recognize Fourth-Generation (4GL) Languages
- Define the Integrated Language Environment
- Identify Stored Procedures



Computer programming languages, which are languages that are used to control the behavior of computers, have evolved along with computer hardware. As more storage and memory becam
available, these initial basic instructions, evolved into coding languages.

The earliest coding systems used machine code, entered as binary numbers. These codes were represented by short mnemonics that mapped directly back to an opcode, but were easier to remember and read.

These mnemonics evolved into the assembler that you see today.

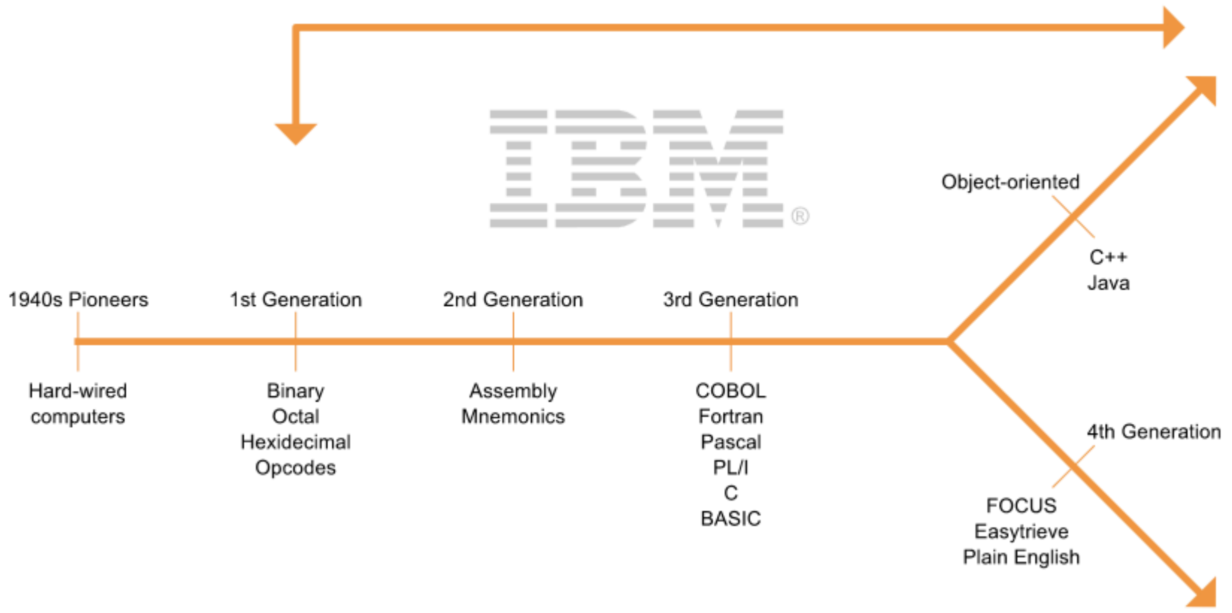


By the late 1950s, the initial third-generation languages appeared, where each command could represent multiple opcodes and structured programming was possible.

In the late 1970s and through to the 1980s, fourth-generation languages were designed to be more natural languages.

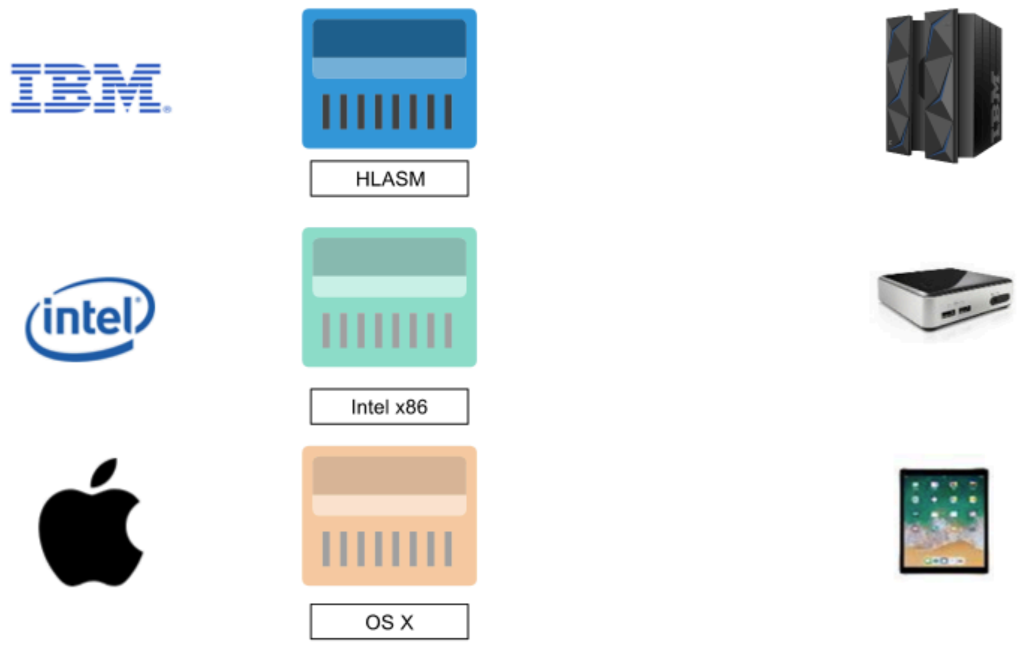
In the 1990s, object-oriented languages that built on an object-based design and structure gained in popularity.

In the 2000's many programming languages where extended to provide Web interfaces.



Due to the long history of the IBM enterprise environment, many of these programming languages are still being used and programs written in these languages continue to require maintenance development on today's mainframes.

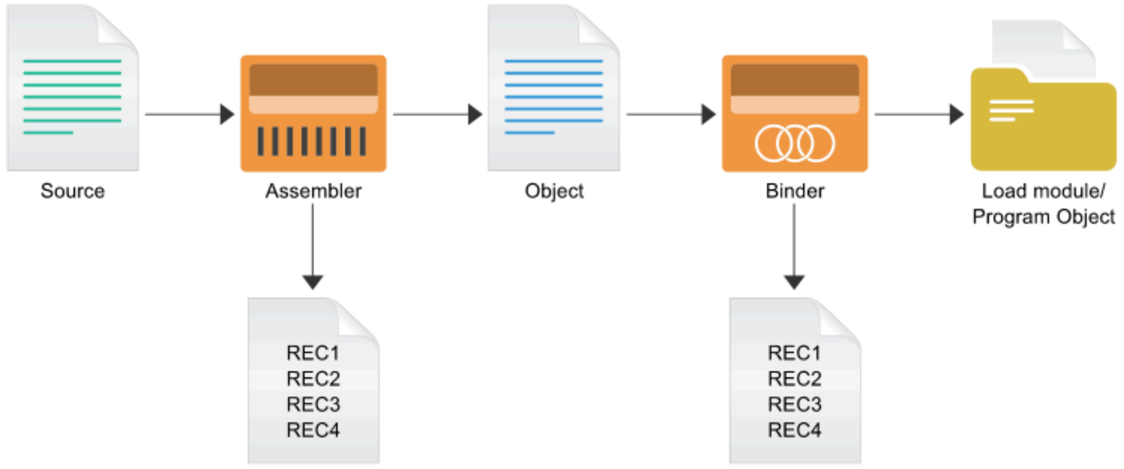
This includes areas where opcodes need to be coded, but these are no longer thought of as a programming language.



First, you will explore IBM Z Assembler, more correctly called High Level Assembler (HLASM). Other processing platforms such as Intel have their own version of Assembler; however HLASM i unique to IBM Z mainframes.

Assembler languages are low-level languages that have a close relationship to the CPU or processor where they execute.

The assembly process



Assembly language is a low-level or symbolic programming language containing two types of instructions:

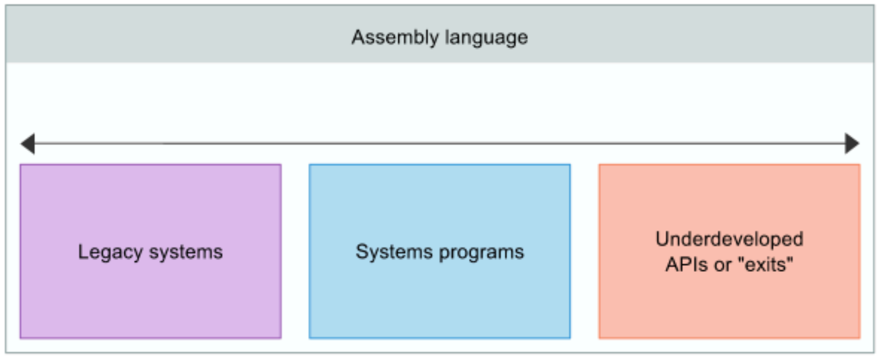
- Instructions that you intend for the hardware, known as mnemonic or machine instructions
- Instructions to tell the assembler what to do or how to generate the machine instructions

The assembler is the tool that converts source code written in assembly language to machine instructions.

1	10	16		72
(name)	operation	[operand(s)]	[comments]	C
MyLoop	LA	R2,1(R2)	Redo the loop	
	BCT	R3,MyLoop	n times.	

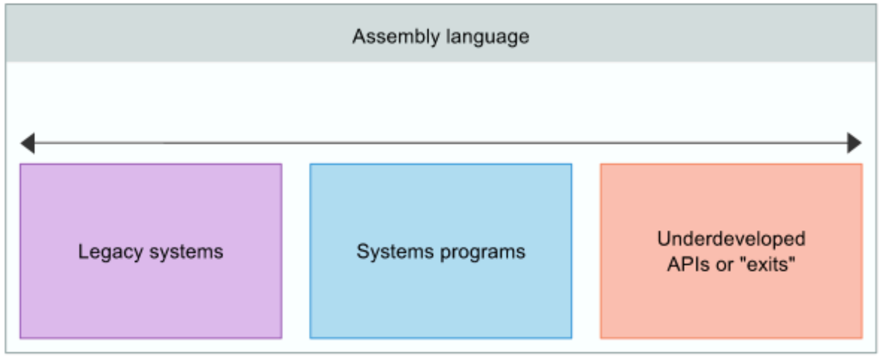
Assembly language is made up of name, symbolic operation codes or mnemonics, operands, and comments, all dependent on column positions.

There are many hundreds of symbolic operation codes, including sets for 24, 31, and 64 bit operations, each with their own set of allowed operands.



Although assembly language appears to be very cryptic and difficult to maintain, it is still used in legacy systems that were constructed when assembly language was all that was available.

Because it enables direct access to memory and machine resources, assembly language is used in many systems programs to perform tasks that would otherwise not be accessible.



Assembly language is also used to access underdeveloped APIs and interfaces where no suitable high-level language interface has been provided, and to provide generic extensions or exits to systems.

This is possible because whatever high-level language a system is written in, it runs as machine code, which is only one step down from assembly language.

COmmon Business Oriented Language

The most widely used programming language in the IBM enterprise environment is COmmon Business Oriented Language (COBOL). This name incorporates the two significant aspects of the language.

COBOL is common because it does not belong to any single computer manufacturer and it has been implemented on most computer architectures.

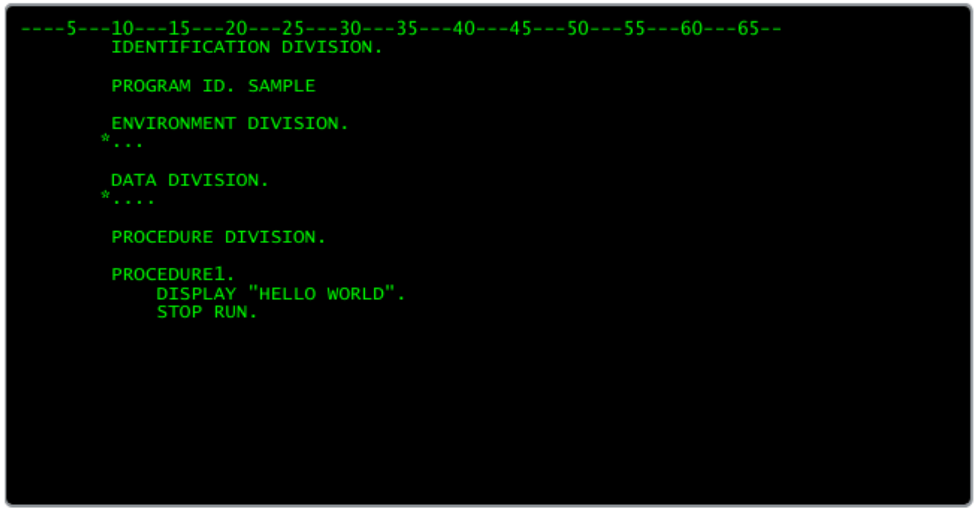
COBOL is business oriented because it is designed to handle files and records related to business transactions, and is not designed for complex mathematical computations.

COBOL

The original COBOL, which was designed in 1959, has evolved through a number of revisions to incorporate new ideas and advances in data storage, communications, and presentations.

Because of its long history and its universal nature, COBOL accounts for the largest proportion of code currently in existence in the IBM enterprise environment.

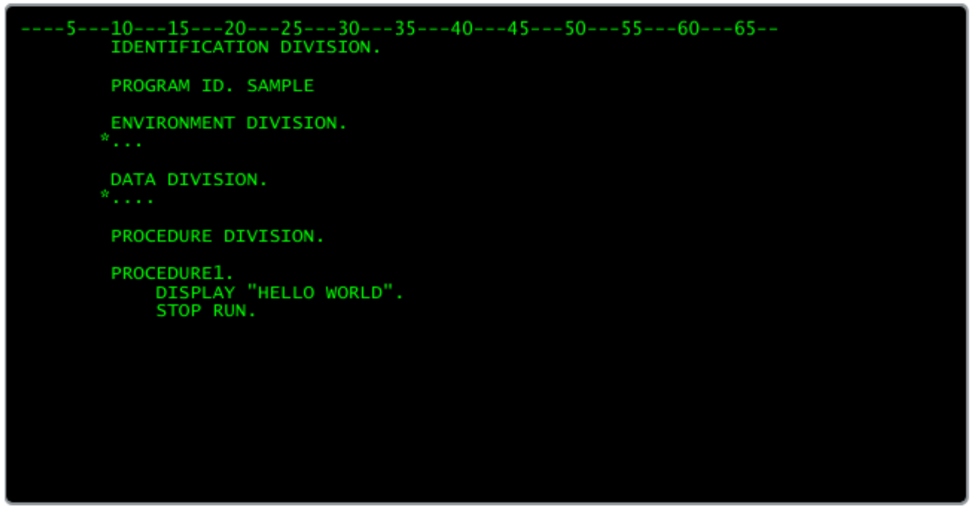
COBOL continues to be a viable language for ongoing development.



```
----5---10---15---20---25---30---35---40---45---50---55---60---65--
        IDENTIFICATION DIVISION.

        PROGRAM ID. SAMPLE

        ENVIRONMENT DIVISION.
       *...

        DATA DIVISION.
       *....

        PROCEDURE DIVISION.

        PROCEDURE1.
            DISPLAY "HELLO WORLD".
            STOP RUN.
```

COBOL is a highly structured language with four defined program divisions, coding in paragraphs and sections, and column formatting requirements in older and mainframe versions.

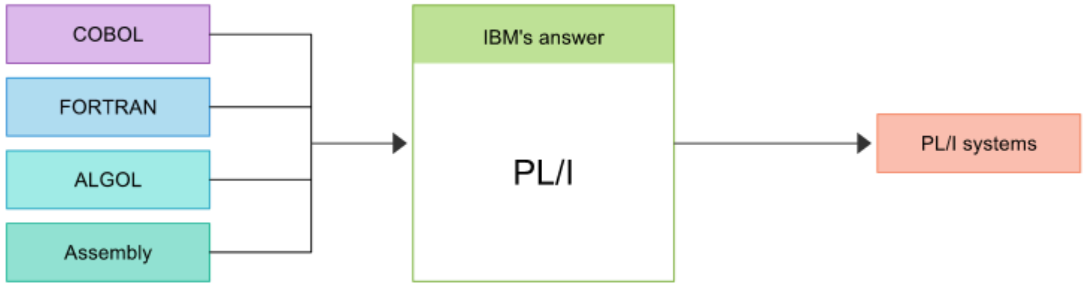
Compared with other third-generation languages, COBOL's use of English-like verbs and encouragement of descriptive variable names can, with discipline by developers, lead to code that is essentially self-documenting and thus easier to maintain.



```
----5---10---15---20---25---30---35---40---45---50---55---60---65--
       IDENTIFICATION DIVISION.

       PROGRAM ID. SAMPLE

       ENVIRONMENT DIVISION.
      *...

       DATA DIVISION.
      *....

       PROCEDURE DIVISION.

       PROCEDURE1.
           DISPLAY "HELLO WORLD".
           STOP RUN.
```

COBOL is a simple language with a limited scope of function. It is most suited to applications dealing with data records and data handling.

It is not suitable for complex mathematical formulae and calculations. When faced with such a requirement, other options should be considered.



PL/I is a language developed by IBM, originally as an attempt to create a definitive programming language that would provide all the facilities of COBOL, Fortran, and ALGOL, as well as a lot of system control that is available with Assembly.

For a period in the 1970s and 1980s, PL/I was heavily supported and promoted by IBM as a superior development environment, and it is found in many systems dating from that period.

PL/I is also referred to as PL1 and both names are in common use.

Enterprise PL/I for z/OS

PL/I for AIX

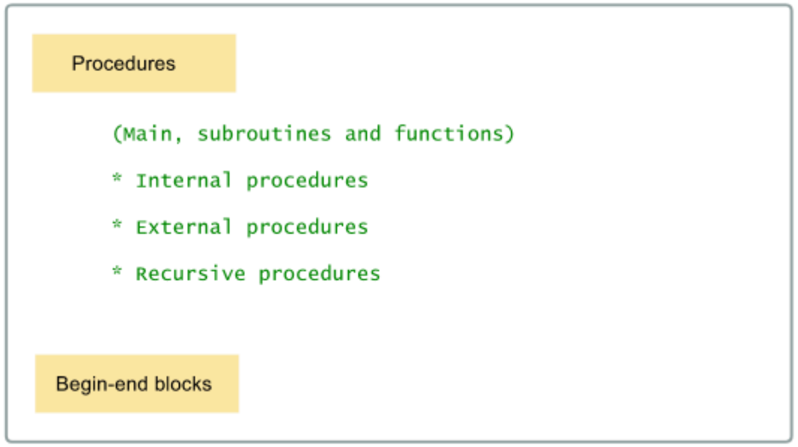
PL/I for MVS and VM

PL/I for z/VSE

Enterprise PL/I for z/OS is able to generate code that takes advantage of the latest IBM Z mainframe architecture, while supporting the latest versions of IBM mainframe software such as CICS, IMS, and Db2.

IBM continues to support PL/I, providing ongoing enhancements to PL/I cross platform products shown here.

Mouse-over the PL/I versions for more information.



Procedures

```
(Main, subroutines and functions)
* Internal procedures
* External procedures
* Recursive procedures
```

Begin-end blocks

In comparison to COBOL, PL/I delivers a great deal of flexibility in programming structure. This flexibility can provide a powerful programming platform, but it also requires extra care to ensure t
can be maintained.

PL/I is equally adept at file handling and complex mathematical functions.

```
PLIPROG1:  PROCEDURE;
     DCL ABC ENTRY EXTERNAL;
     FETCH ABC;
     CALL ABC;
     RELEASE ABC;
END PLIPROG1;

PLIPROG2: PROCEDURE;
     DCL ABC ENTRY EXTERNAL;
     DCL A FIXED BIN INIT(1);
     CALL ABC,
     IF A = 2 THEN RELEASE ABC;
END PLIPROG2;
```

In comparison to other third-generation languages, PL/I provides a large amount of low-level control of the execution environment, including the loading and unloading of external modules, FET
and RELEASE, and pointer access to internal and external storage.

Question 1 of 2

The assembler tool performs which action on assembly language source code?

Select the correct option.

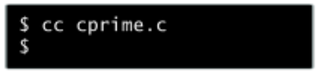
Click **Check My Answer** when you have finished.

- [ ] It is used to debug errors in the source code.
- [ ] It converts the source code to machine instructions.
- [ ] It is used to precompile code and check for syntax errors.
- [ ] It produces the executable load module.

Check My Answer

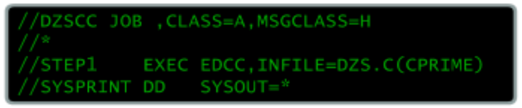
Executing C compiler on z/OS

UNIX command



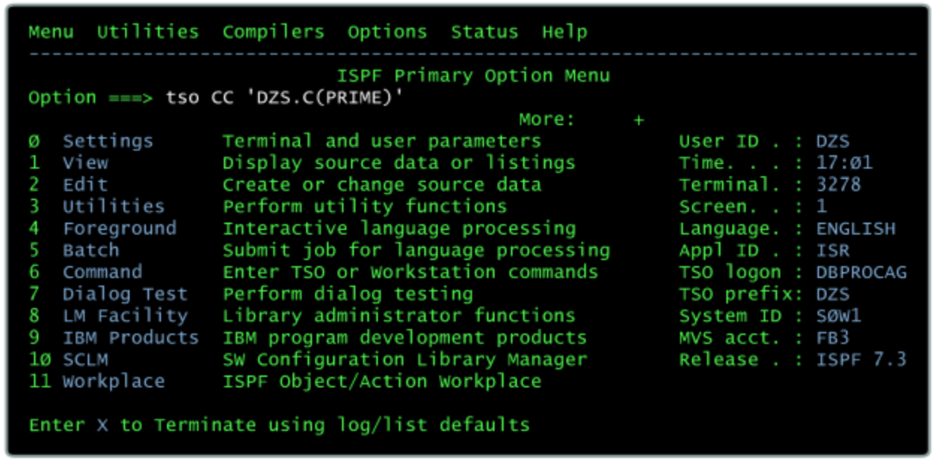
```
$ cc cprime.c
$
```

Batch



```
//DZSCC JOB ,CLASS=A,MSGCLASS=H
//*
//STEP1    EXEC EDCC,INFILE=DZS.C(CPRIME)
//SYSPRINT DD   SYSOUT=*
```

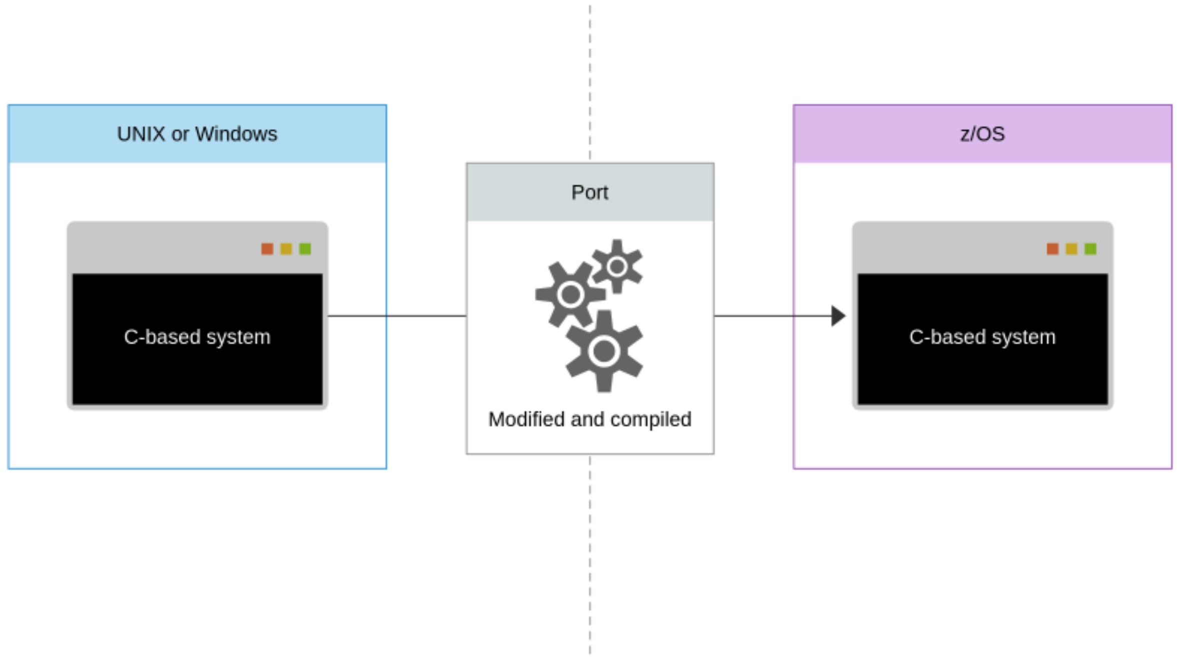
TSO command



```
Menu  Utilities  Compilers  Options  Status  Help
------------------------------------------------------------------------------
                           ISPF Primary Option Menu
Option ===> tso CC 'DZS.C(PRIME)'
                                          More:     +
0  Settings      Terminal and user parameters            User ID . : DZS
1  View          Display source data or listings         Time. . . : 17:01
2  Edit          Create or change source data            Terminal. : 3278
3  Utilities     Perform utility functions               Screen. . : 1
4  Foreground    Interactive language processing         Language. : ENGLISH
5  Batch         Submit job for language processing      Appl ID . : ISR
6  Command       Enter TSO or Workstation commands       TSO logon : DBPROCAG
7  Dialog Test   Perform dialog testing                  TSO prefix: DZS
8  LM Facility   Library administrator functions         System ID : S0W1
9  IBM Products  IBM program development products        MVS acct. : FB3
10 SCLM          SW Configuration Library Manager        Release . : ISPF 7.3
11 Workplace     ISPF Object/Action Workplace

Enter X to Terminate using log/list defaults
```

IBM provides a C compiler that conforms to ANSI C standards. This compiler can be executed from TSO or batch. As with all other UNIX operating systems, the C compiler can be executed as command from a z/OS UNIX shell.

The IBM C compiler is called XL C/C++.

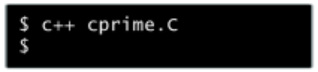


With IBM Z, C programs created on other platforms can be ported to z/OS. Some software vendors, including IBM, have done this for some of their software products.

IBM XL C supports all the normal C features but also includes z/OS-specific functions and features to do things such as access traditional z/OS data sets, and work with JES spool.

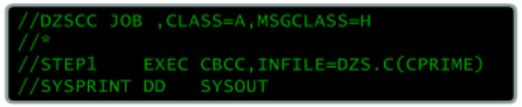
Executing C++ compiler on z/OS

UNIX command



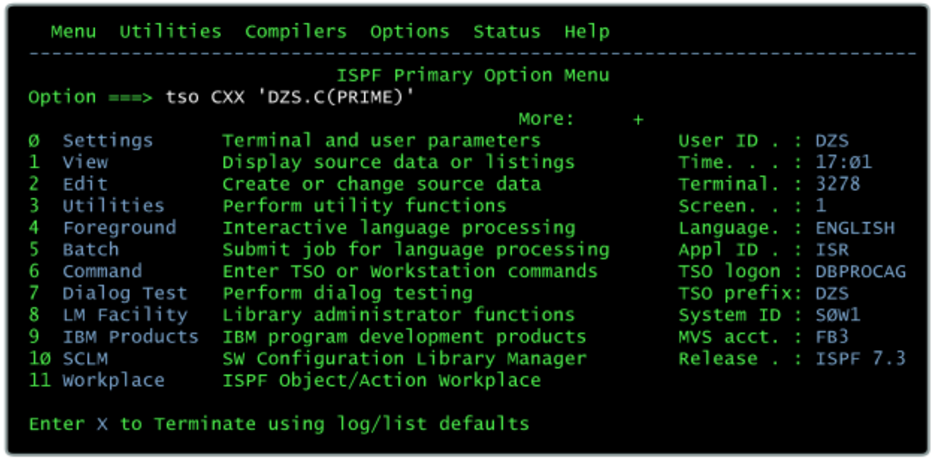
```
$ c++ cprime.C
$
```

Batch



```
//DZSCC JOB ,CLASS=A,MSGCLASS=H
//*
//STEP1    EXEC CBCC,INFILE=DZS.C(CPRIME)
//SYSPRINT DD   SYSOUT
```

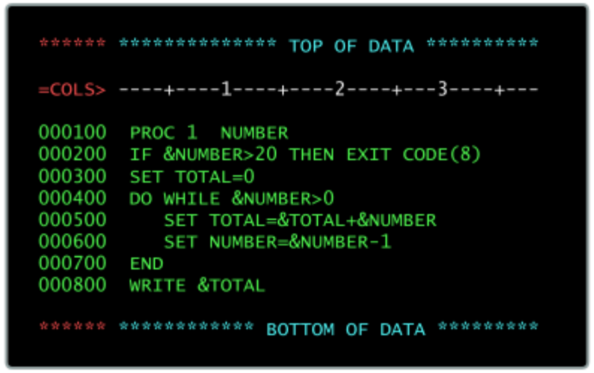
TSO command



```
  Menu  Utilities  Compilers  Options  Status  Help
 ------------------------------------------------------------------------------
                          ISPF Primary Option Menu
 Option ===> tso CXX 'DZS.C(PRIME)'
                                              More:     +
 0  Settings      Terminal and user parameters             User ID . : DZS
 1  View          Display source data or listings          Time. . . : 17:01
 2  Edit          Create or change source data              Terminal. : 3278
 3  Utilities     Perform utility functions                Screen. . : 1
 4  Foreground    Interactive language processing          Language. : ENGLISH
 5  Batch         Submit job for language processing       Appl ID . : ISR
 6  Command       Enter TSO or Workstation commands        TSO logon : DBPROCAG
 7  Dialog Test   Perform dialog testing                   TSO prefix: DZS
 8  LM Facility   Library administrator functions          System ID : S0W1
 9  IBM Products  IBM program development products         MVS acct. : FB3
 10 SCLM          SW Configuration Library Manager         Release . : ISPF 7.3
 11 Workplace     ISPF Object/Action Workplace

 Enter X to Terminate using log/list defaults
```

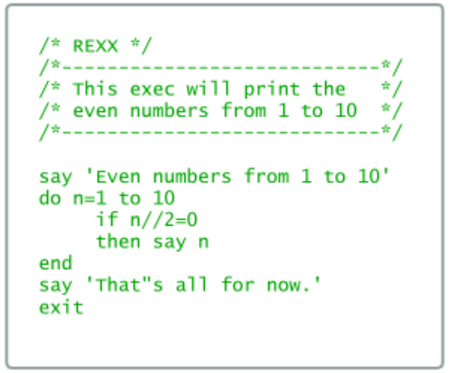
As the name suggests, XL C/C++ also provides a C++ compiler with the same portability and features of the C compiler. As the name suggests, XL C/C++ also provides a C++ compiler with the same portability and features of the C compiler.

CLIST



```
****** ************** TOP OF DATA **********

=COLS> ----+----1----+----2----+---3----+---

PROC 1  NUMBER
IF &NUMBER>20 THEN EXIT CODE(8)
SET TOTAL=0
DO WHILE &NUMBER>0
   SET TOTAL=&TOTAL+&NUMBER
   SET NUMBER=&NUMBER-1
END
WRITE &TOTAL

****** ************ BOTTOM OF DATA *********
```

REXX

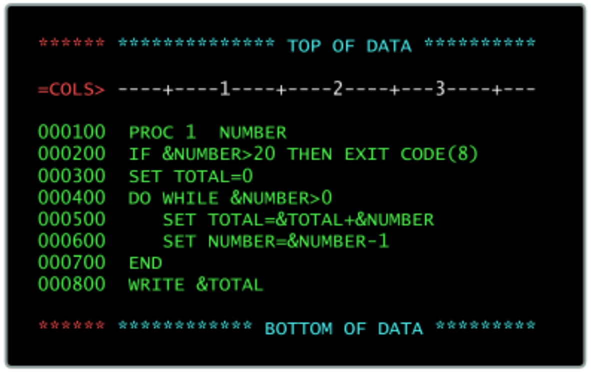


```
/* REXX */
/*---------------------------*/
/* This exec will print the   */
/* even numbers from 1 to 10  */
/*---------------------------*/

say 'Even numbers from 1 to 10'
do n=1 to 10
     if n//2=0
     then say n
end
say 'That''s all for now.'
exit
```

CLIST and REXX are two scripting languages available in the IBM enterprise environment. CLIST is the original language, while REXX is seen as its successor with more extensive capabilities.

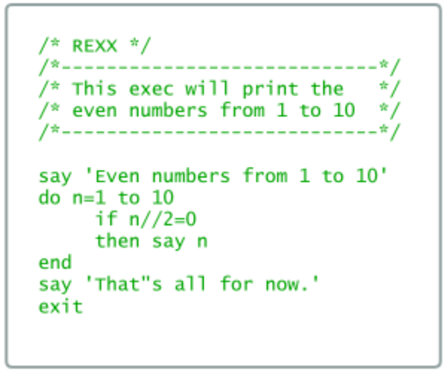
These languages are often used to enhance and extend the TSO environment and system tools. REXX is also often used to create automated operations scripts for products such as IBM Syste
Automation and CA OPS/MVS.

CLIST



```
****** ************** TOP OF DATA **********

=COLS> ----+----1----+----2----+---3----+---

PROC 1  NUMBER
IF &NUMBER>20 THEN EXIT CODE(8)
SET TOTAL=0
DO WHILE &NUMBER>0
   SET TOTAL=&TOTAL+&NUMBER
   SET NUMBER=&NUMBER-1
END
WRITE &TOTAL

****** ************ BOTTOM OF DATA *********
```

REXX



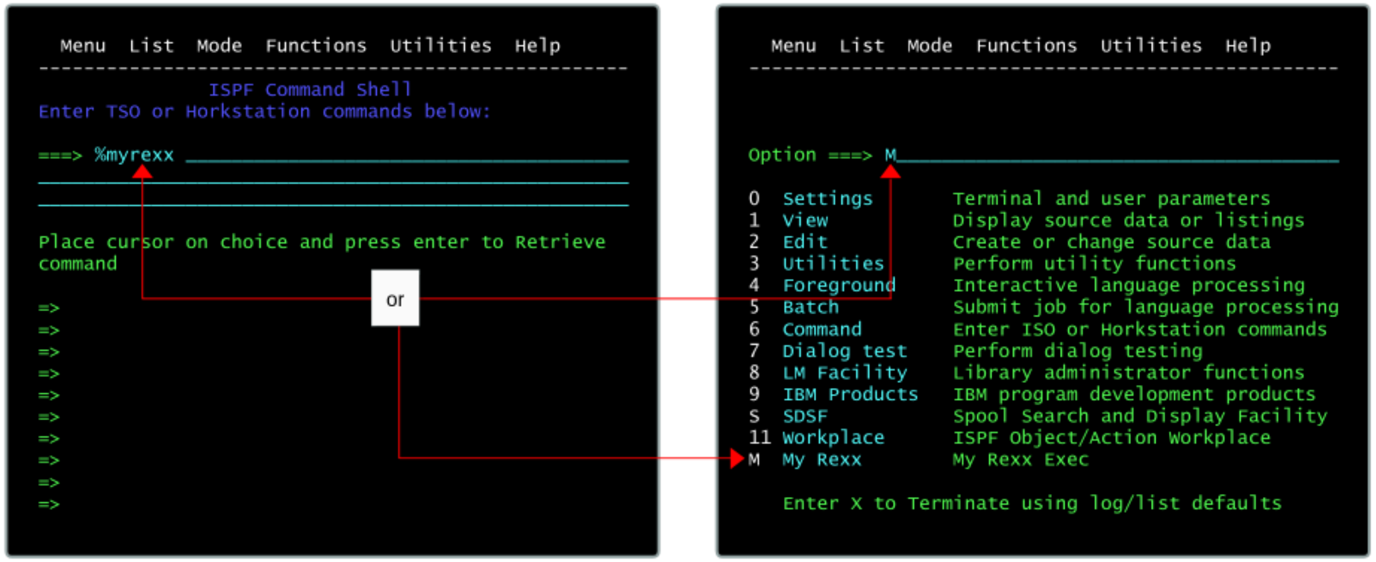
```
/* REXX */
/*----------------------------*/
/* This exec will print the   */
/* even numbers from 1 to 10  */
/*----------------------------*/

say 'Even numbers from 1 to 10'
do n=1 to 10
     if n//2=0
     then say n
end
say 'That"s all for now.'
exit
```

CLIST and REXX languages are normally run interpreted and not compiled, meaning that the source and executable are the same; however, a compiler is available for REXX.

REXX for CICS is also available and REXX can be used in the batch environment.

Both languages can consist of text statements that have very similar syntax and structure, but all REXX programs begin with a comment with the word REXX in it, for example:

```
/* REXX */
```

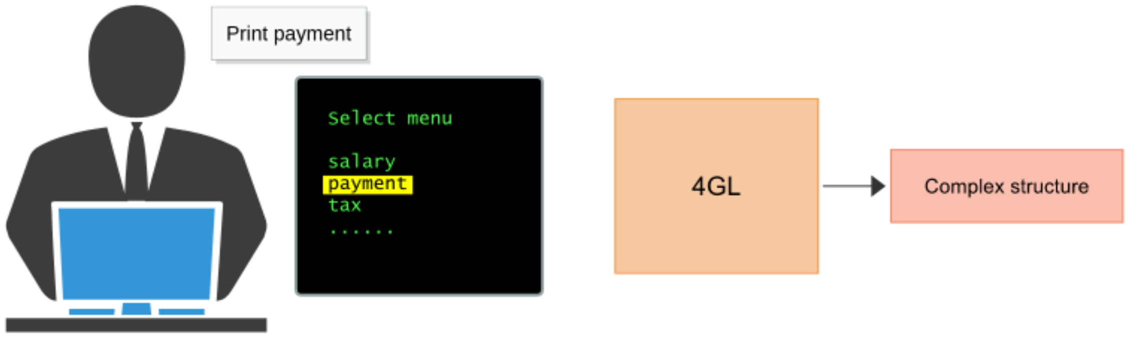


```
 Menu  List  Mode  Functions  Utilities  Help
---------------------------------------------------
              ISPF Command Shell
Enter TSO or Workstation commands below:

===> %myrexx ______________________________________
___________________________________________________
___________________________________________________

Place cursor on choice and press enter to Retrieve
command

=>
=>
=>
=>
=>
=>
=>
=>
=>
=>
```

```
 Menu  List  Mode  Functions  Utilities  Help
---------------------------------------------------



Option ===> M______________________________________

0  Settings       Terminal and user parameters
1  View           Display source data or listings
2  Edit           Create or change source data
3  Utilities      Perform utility functions
4  Foreground     Interactive language processing
5  Batch          Submit job for language processing
6  Command        Enter TSO or Workstation commands
7  Dialog test    Perform dialog testing
8  LM Facility    Library administrator functions
9  IBM Products   IBM program development products
S  SDSF           Spool Search and Display Facility
11 Workplace      ISPF Object/Action Workplace
M  My Rexx        My Rexx Exec

   Enter X to Terminate using log/list defaults
```

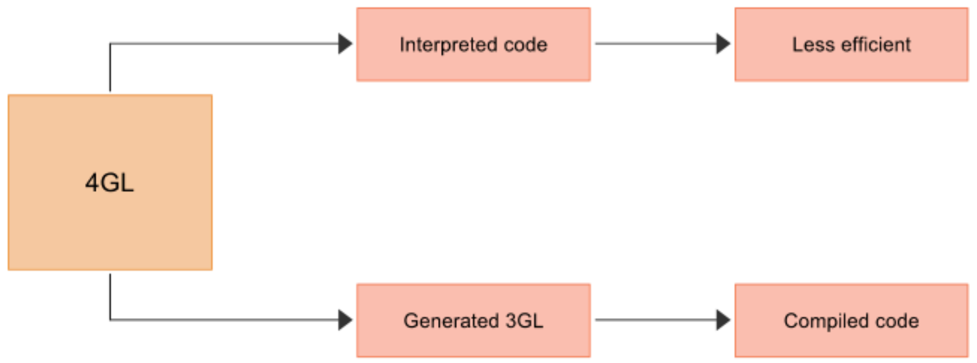
REXX and CLIST execs can both be run from the TSO command line or be called by other programs, such as ISPF.

Along with the ISPF definition elements, such as panels, skeletons, tables, and messages, this allows REXX and CLIST to be used extensively in the construction and modification of TSO/ISPF-based systems.



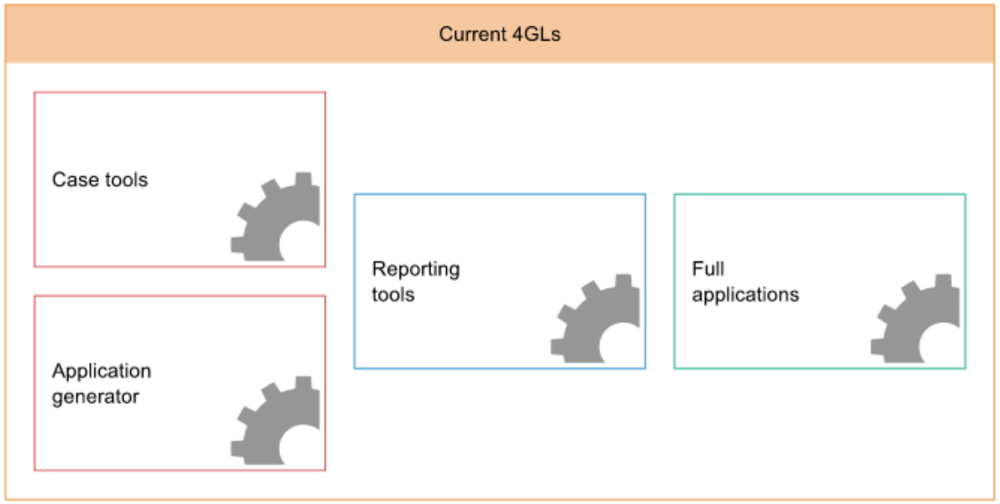
In the mid-1980s, many fourth-generation languages were produced. These were seen as an improvement over third-generation languages because they handled much of the lower-level contrc and structure.

They also enabled users to program by using either natural language-like statements or menu-driven interfaces.



For many reasons, however, 4GLs did not replace 3GLs. Some of these reasons were:

- Many 4GLs produced code that had to be interpreted at runtime, which is much less efficient than compiled code.
- One or two languages did not become universal standards like COBOL and C, so skilled practitioners were not widely available.
- Many 4GLs had limited flexibility that made it difficult to use them for tasks not foreseen by the language creator.
- Some languages were better supported by their creators than others, and some were not kept up to date with the progress of the enterprise environment.



In the current environment, 4GLs and their descendants remain in use as computer-aided software engineering (CASE) tools, often with application generators for COBOL or C.

They are also employed as reporting tools as reporting is a very time-consuming task in a 3GL like COBOL but straightforward in many 4GLs.

4GLs

- Focus
- Easytrieve
- SAS
- Others

There are still many organizations with applications that were built when 4GLs were most popular, where they continue to perform their function.

FOCUS and Easytrieve have been used to create complex systems, but their strength lies in their ability to quickly create complex reports on all types of data. Many installations write update processes in a language like COBOL and pass the output to Easytrieve or FOCUS to produce reports.

SAS is a language environment that is designed for statistical analysis and reporting. It is very good at its designed task but is not suitable for general systems.

The most modern design philosophy in programming is object orientation (OO).

OO is provided in the IBM enterprise environment by extensions to many existing 3GLs and the support of the new OO languages, Java, and C++.

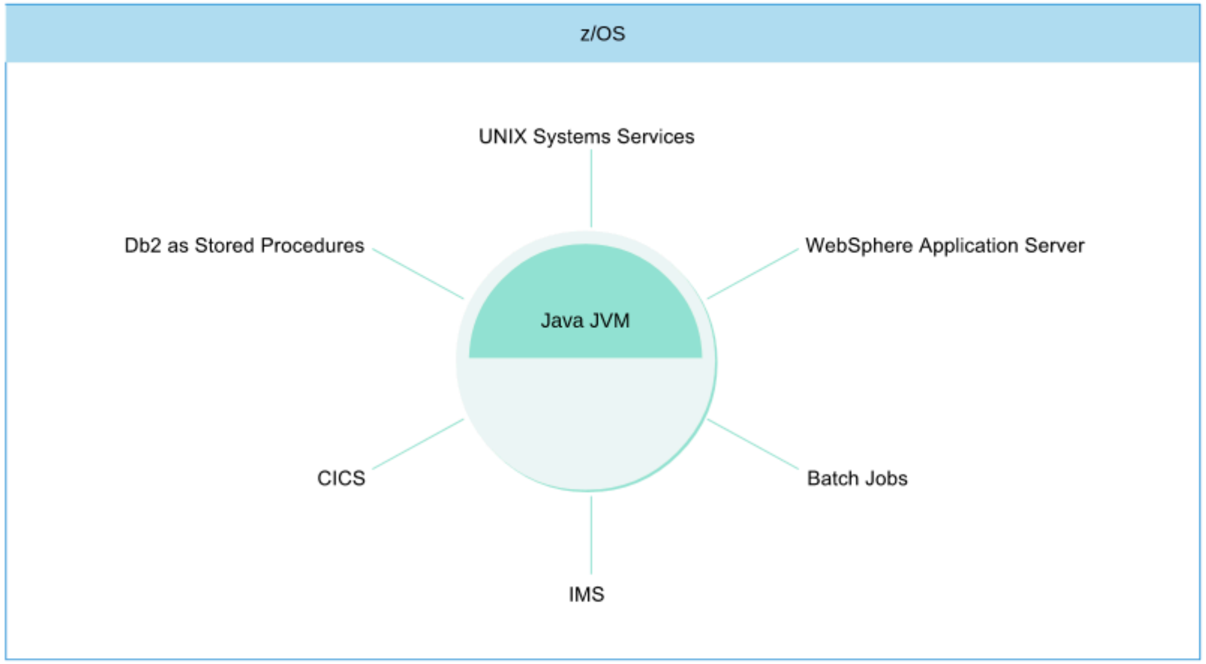
It is possible to use OO design and programming techniques with COBOL, PL/1, and REXX through the use of additional language facilities and options, and by structuring code in an OO way.

The additional facilities have led to the labeling of the languages as OO COBOL and OO REXX, even though they are the same language and are predominantly used to create programs in a procedural way.

It is also possible to provide OO interfaces to legacy programs in these languages by using CORBA through the facilities provided by CICS and WebSphere Application Server.

You have seen how C++ is supported under z/OS. A compiler is provided and executable modules are produced and run in a similar fashion to C.

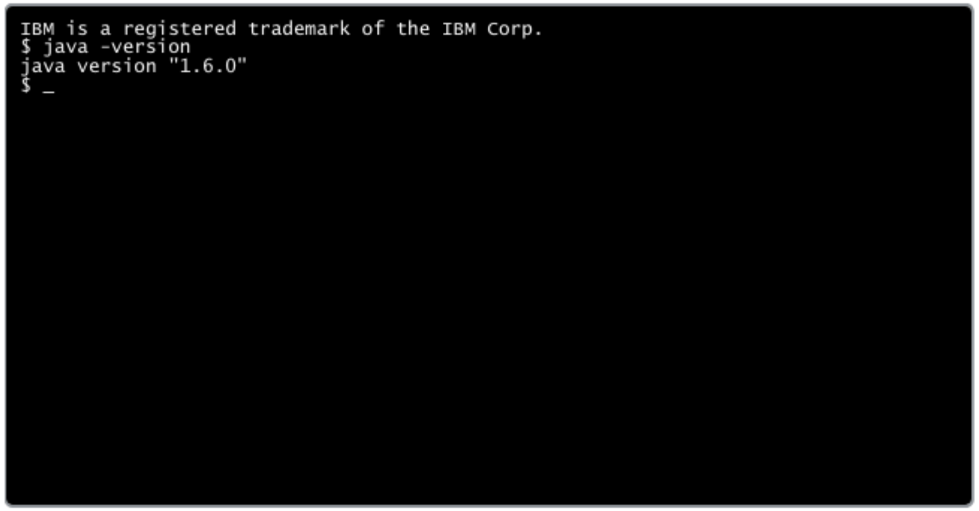
Java is also supported but its implementation is very different and it does not produce machine-level executable modules that can be run like those compiled and linked from other languages.



Java, a language that is owned and defined by Sun Microsystems, is designed to be completely portable between all systems on which it is implemented.

This portability requires the implementation of a standard Java Virtual Machine (JVM) for Java-compiled byte code to run on. The implementation of the JVM is the responsibility of the host ope system's builder.

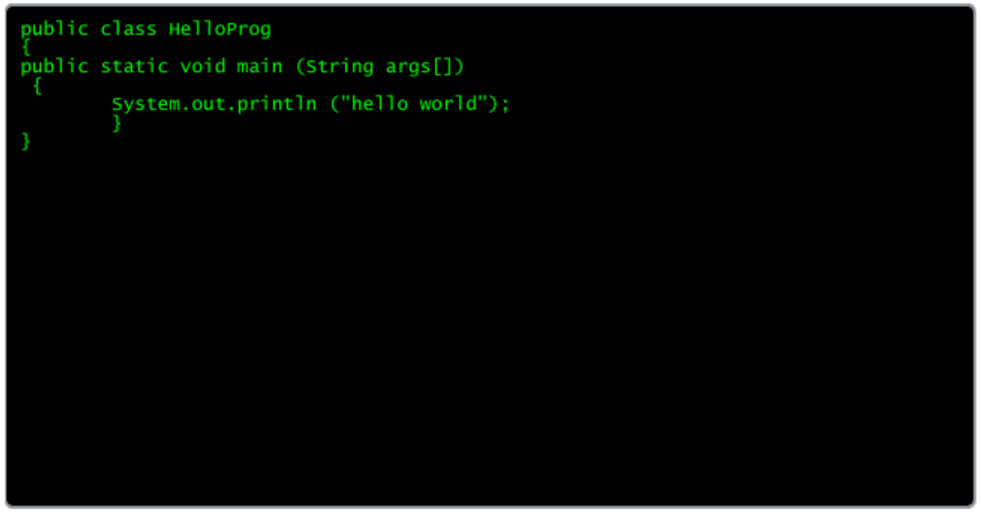
Java has become a strategic programming language on z/OS. IBM supplied JVMs and Toolkits on z/OS allow Java programs to run in the containers above. There are other software systems t are also capable of running Java.



```
IBM is a registered trademark of the IBM Corp.
$ java -version
java version "1.6.0"
$ _
```

A version of the Java System Development Kit (SDK) is available for UNIX Systems Services, giving access to all the Java development tools.

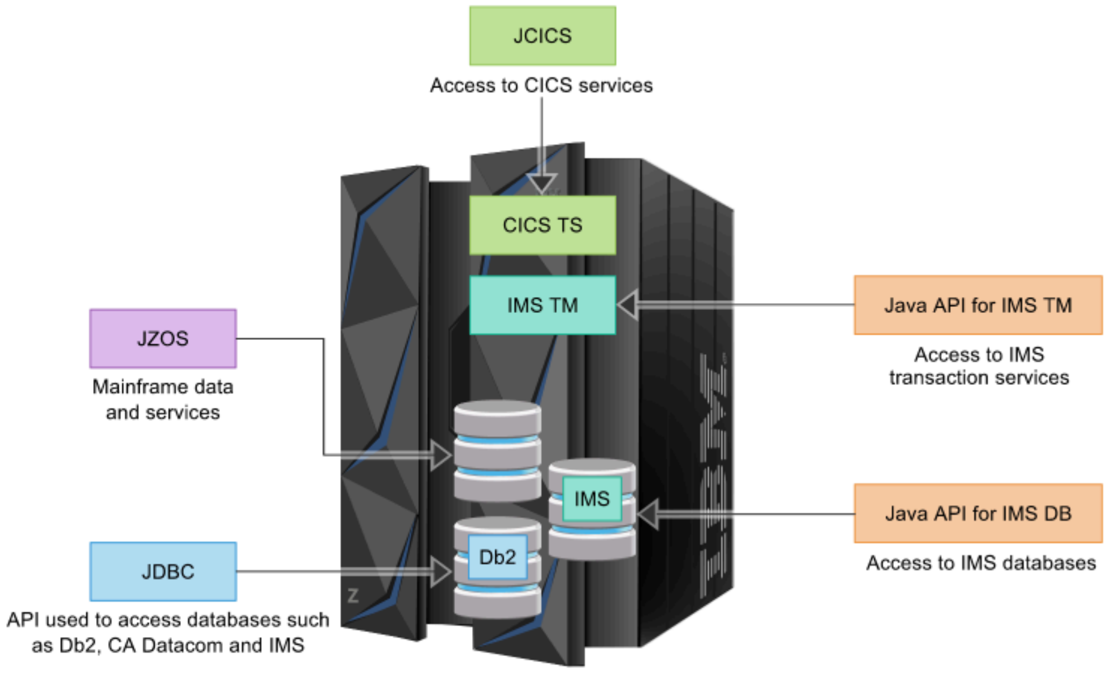
With the portability of Java, development can take place on another platform and be moved to the mainframe for execution.



```
public class HelloProg
{
public static void main (String args[])
 {
        System.out.println ("hello world");
        }
}
```

The adoption of Java as a strategic language by IBM opens up IBM enterprise systems for use in mainstream Internet processing. The implementation is standard with all standard-class librari
available, and the ability to create, compile, or run any Java process.

This simple "hello world" example can be compiled and run in exactly the same way under UNIX Systems Services as it would under any Windows or UNIX system.


JCICS
Access to CICS services
CICS TS
IMS TM
Java API for IMS TM
Access to IMS transaction services
JZOS
Mainframe data and services
IMS
Java API for IMS DB
Access to IMS databases
Db2
JDBC
API used to access databases such as Db2, CA Datacom and IMS

A number of standard and specific java interfaces and classes are available so that java applications can access the following:

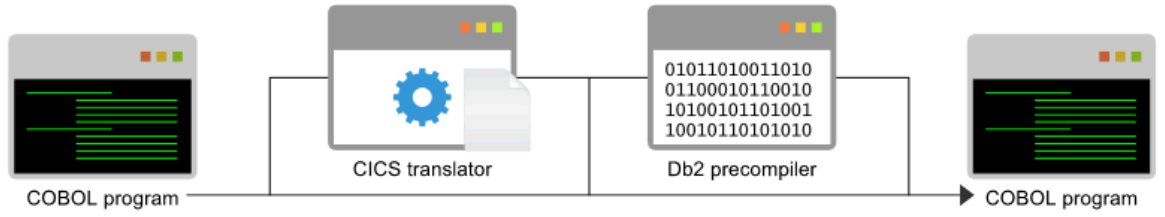
- z/OS data and services
- CICS services
- Db2 database content
- IMS database content

You have now looked at the following programming languages and methodologies:

- Assembler
- COBOL
- PL/I
- C and C++
- CLIST and REXX
- 4GLs
- Object-orientation
- Java

You will now explore the process of compiling a source program, binding it and creating an executable module.

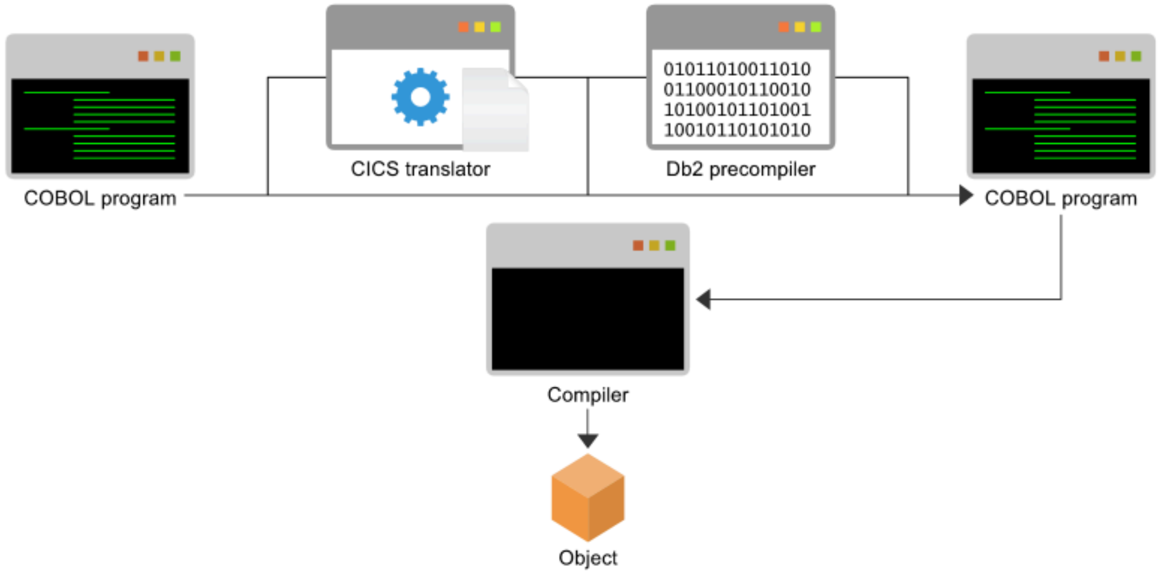
You will also examine the integrated language environment and stored procedures.



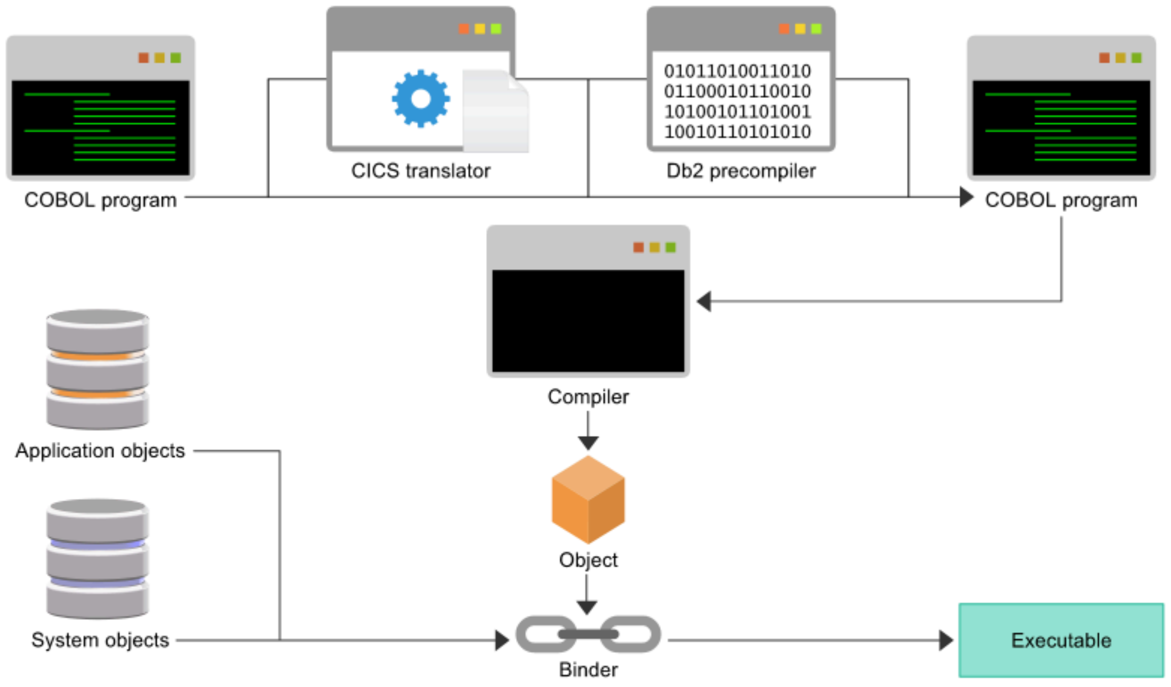
Starting with a data set containing source, for example, a COBOL program, the first steps may involve one or more translators or a precompiler. These programs change the structure and langu
used by subsystems like CICS or Db2 into the source language's call syntax. This produces a source file that contains only standard language statements. For example, CICS:

```
EXEC CICS
....
END EXEC.
```

is translated to `CALL "DFH..... ........"`



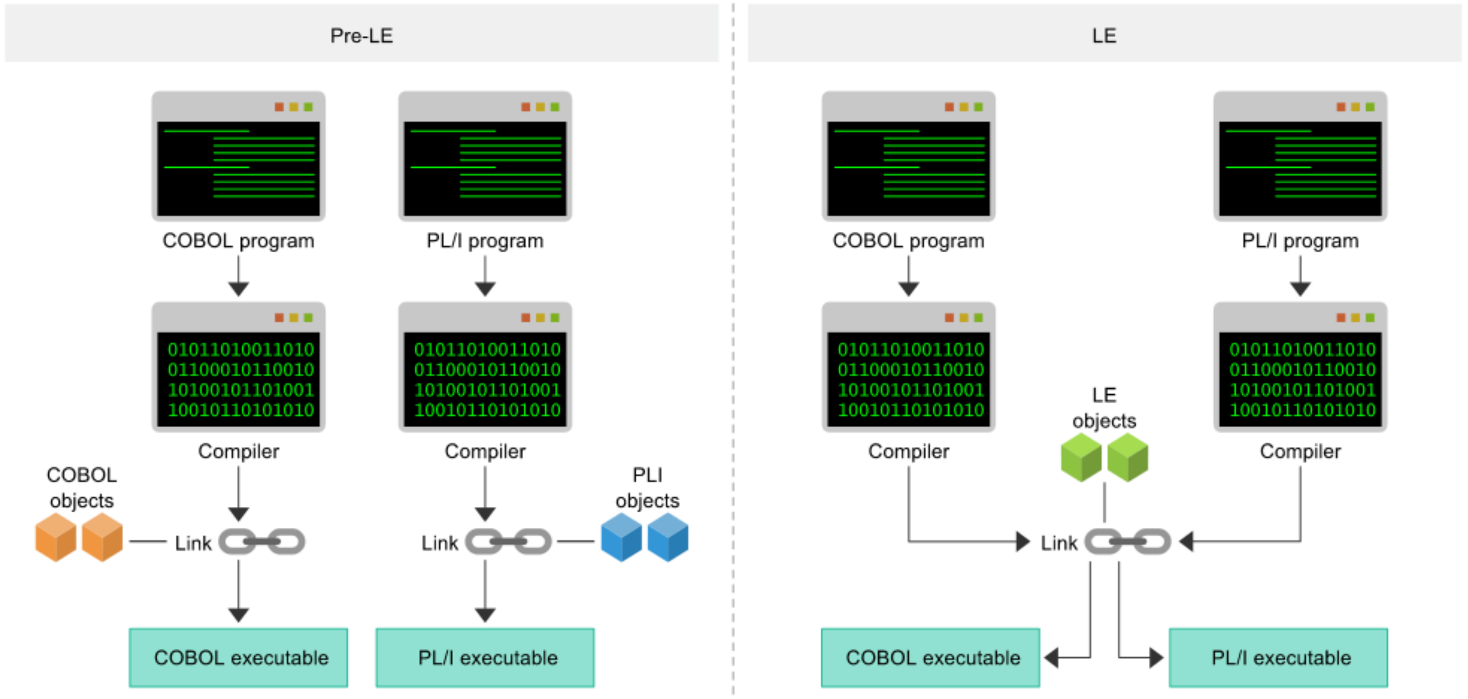
The second step is the compiler accepting a source program, checking it for syntax, and producing an object module that contains machine-level code for all of the statements in the source pro



The third step is to bind the object module into an executable module - a load module, or program object. This process is sometimes called link editing.

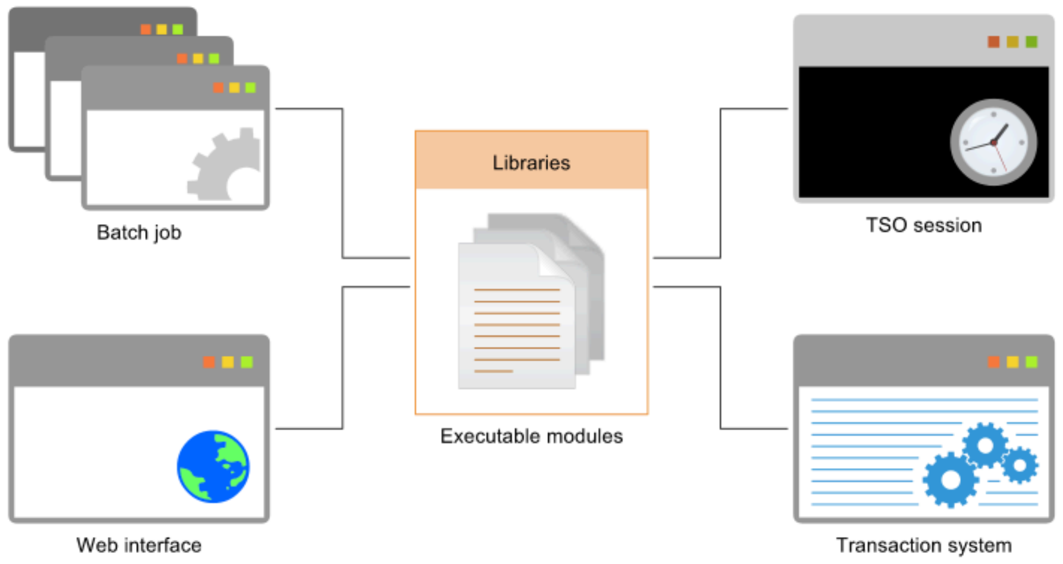
In the bind process, the z/OS binder locates and includes any references to external modules, such as other applications, language, and system modules.

These references may be resolve during the bind process, and external module may be statically bound into the executable module. Alternatively, these references could be resolved at executi time, and external modules loaded when needed. Compiler and binder options determine which.



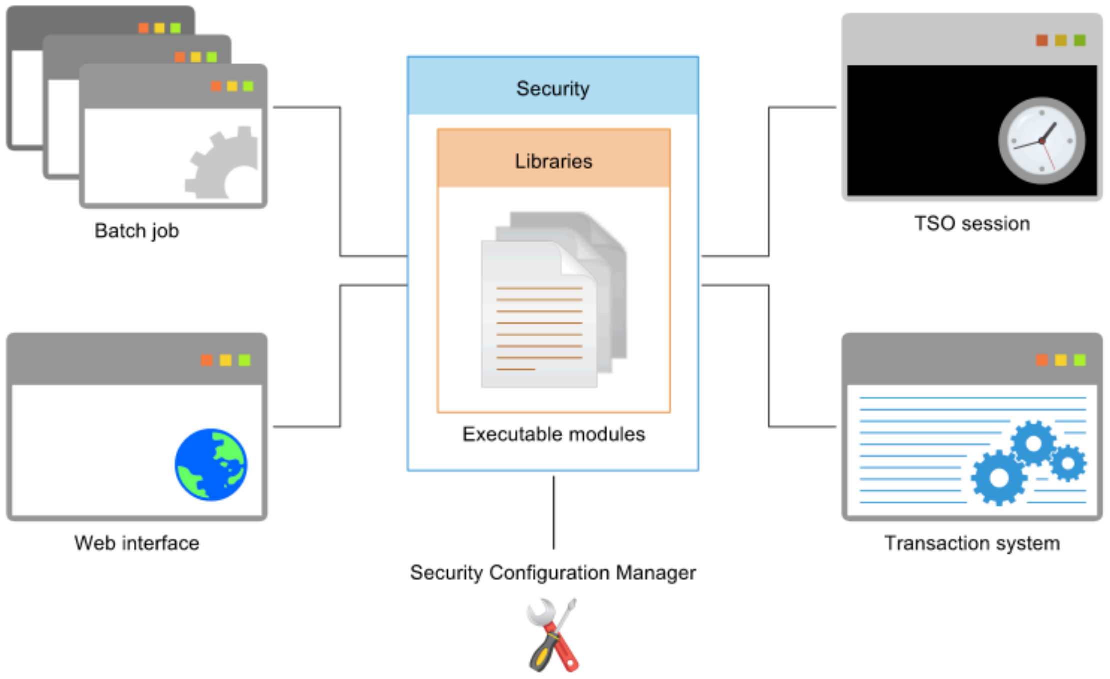
The Language Environment (LE) contains standard objects and interfaces for other languages to refer to. Before the implementation of the LE in the 1990s, each language provided its own set low-level modules for interface-to-systems resources and other subsystems, like CICS and Db2.

With LE, the languages now refer to these common interfaces and only LE needs to be maintained and enhanced for changes to the underlying system and subsystems.



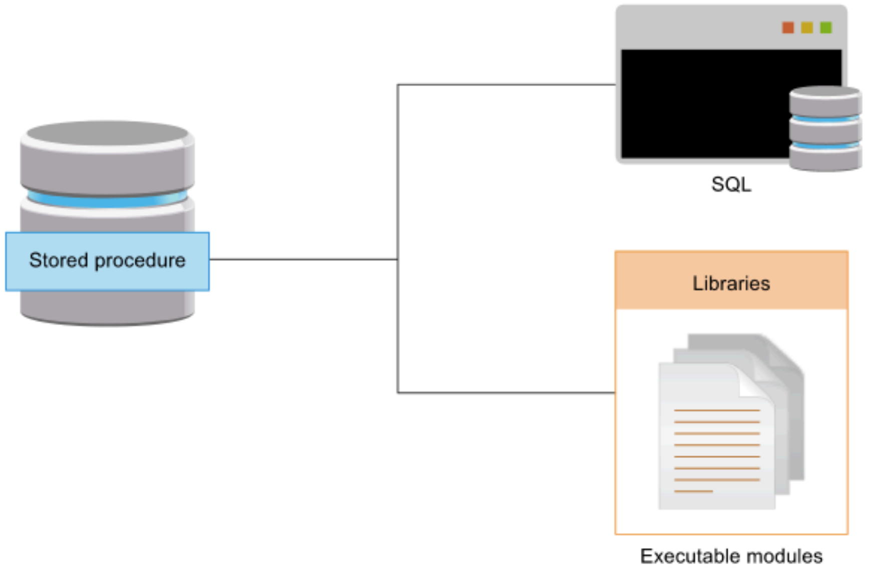
After producing a program in one of the available languages, and performing the compile and link required, you will have an executable module.

This module lives in a data set commonly called a library, which may not be on the same machine on which it was created. Due to the uniformity of the IBM enterprise environment, it may have distributed to one or more test or production environments. The module then becomes available for execution by batch jobs, transaction systems like CICS, TSO sessions, or through one of the interfaces.



For most executable modules, a security system such as RACF controls the processes and users that can access or execute the module.

The use of a Software Configuration Manager (SCM), such as CA Librarian or SCLM, controls update and maintenance of the modules, and may provide distribution and installation functions a multiple systems.



One type of module that has a higher degree of control and integration is the Db2 stored procedure. These modules are created like normal modules by using compiles and links, but they are al
defined to Db2. They are then stored in Db2-controlled libraries and accessed through Db2 via the SQL interface.

Although any process can use them in this way, stored procedures are mostly used for processes that update or summarize data in the Db2 database, or functions commonly used with the Db2 database.

Summary

Programming Capabilities

In this module, you explored the most commonly used programming languages in the IBM enterprise environment.

You should now be able to:

- Recognize Highlights in the Timeline of Programming Language Development
- Identify Commonly Used Programming Languages in the IBM Enterprise Environment
- Recognize Fourth Generation (4GL) Languages
- Define the Integrated Language Environment
- Identify Stored Procedures