Chapter 9: Compiling and link-editing a program on z/OS
Chapter 9 objectives

Be able to:

- Explain the purpose of a compiler
- Compile a source program
- Create executable code from a compiled program
- Explain the difference between an object module and a load module
- Run a program on z/OS
Key terms in this chapter

• binder
• copybook
• linkage-editor
• load module
• object module
• object-oriented code

• procedure
• procedure library
• program library
• relocatable
• source module
Source, object, and load modules

A source program can be divided into logical units – or *modules* -- that perform specific functions.

Each module is assembled or compiled by one of the language translators.

**Input to a language translator is a source module**

**Output from a language translator is an object module**

Object modules must be processed by the binder before they can be executed.

**Output of the binder is a load module.**
Source code to load module

Source module → Precompiler → Compiler → Object module → Binder → Load module
Object module

An object module:

- Is a collection of one or more compilation units produced by an assembler, compiler, or other language translator and used as input to the binder (or linkage editor)
- Is in relocatable format with machine code that is not executable
- Is stored in an object library
Load module

A load module:

- is also relocatable, but with executable machine code
- is in a format that can be loaded into virtual storage and relocated by program manager, a program that prepares load modules for execution by loading them at specific storage locations.
More about object modules and load modules

Object modules and load modules consist of:

- Information (control dictionaries) to resolve cross-references between sections of different modules
- Text, containing the instructions and data of the program
- An end-of-module indication (END statement)
What are source libraries?

Source programs are stored in a PDS known as **source library**

Source libraries contain the source code to be:
- submitted for a compilation process
- retrieved for modification by an application programmer

A *copybook* is a source library containing prewritten text; it is used to copy text into a source program, at compile time.
Compiling programs on z/OS

Compilers:

- Translate source code into object code
- Assign relative addresses to all instructions, data elements, and labels, starting from zero (required in a virtual storage operating system)

Addresses are in the form of a base address plus a displacement (to allow programs to be relocated)

References to external programs or subroutines are left as unresolved
Compiling programs on z/OS

Various ways to compile programs on z/OS:

- batch job
- under TSO/E through commands, CLISTs, or ISPF panels
- for COBOL programs, you can compile programs in a z/OS UNIX shell by using the cob2 command.
Compilation step

- Define the data sets needed for the compilation
- Specify any compiler options needed by the program
Compilation step – example

Data set (library) that contains the source code is specified on the SYSIN DD statement:

```plaintext
//SYSIN    DD  DSNAME=dsname,  
//             DISP=SHR
```

You can place source code directly in the input stream - use this SYSIN DD statement:

```plaintext
//SYSIN    DD  * 
```
Compiling with cataloged procedures (PROCs)

A common way to compile a program under z/OS is by using a batch job with a cataloged procedure or PROC.

A PROC contains a set of JCL statements placed in a data set called the procedure library (PROCLIB).

The PROC must include the following information:
- Job description
- Execution statement to invoke the compiler
- Definitions for the data sets needed but not supplied by the PROC
Sample JCL for compiling a COBOL source program inline...

```
//COMP     JOB
//COMPILE   EXEC  IGYWC
//SYSIN     DD    *
   IDENTIFICATION DIVISION (source program)
.
.
.
/*
//
```
COBOL compile, link and go PROC

JCL in next slide executes the IGYWCLG PROC

PROC is a three-step procedure for compiling a source program, linking it into a load library and executing the load module

- The first two steps are the same as those in the compile and link example
- We do not need to permanently save the load module in order to execute it

Result of running the JCL is to:

- compile our inline source program
- link-edit the object code
- store the load module (either temporarily or permanently)
- execute the load module
Executing COBOL CLG PROC

```
//CLGO    JOB
//CLG     EXEC IGYWCLG
//COBOL.SYSIN    DD *
    IDENTIFICATION DIVISION (source program)
.
.
./
    [//LKED.SYSLMOD DD DSN=MY.LOADLIB(PROG1),DISP=OLD]
```
COBOL CLG PROC – part 1

//IGYWCLG PROC LNGPRFX='IGY.V2R1M0',SYSBLK=3200,
//        LIBPRFX='CEE',GOPGM=GO
// *
//* COMPILE, LINK EDIT AND RUN A COBOL PROGRAM
// *
//* PARAMETER DEFAULT VALUE USAGE
//* LNGPRFX IGY.V2R1M0
//* SYSLBLK 3200
//* LIBPRFX CEE
//* GOPGM GO
// *
//* CALLER MUST SUPPLY //COBOL.SYSIN DD ...
// *
//COBOL EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB DD DSNAM=&LNGPRFX..SIGYCOMP,
//           DISP=SHR
//SYSPRINT DD SYSOUT=*     
//SYSLIN DD DSNAM=&&LOADSET,UNIT=VIO,
//          DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//          DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1 DD
COBOL CLG PROC – part 2

UNIT=VIO,SPACE=(CYL, (1,1))
//SYSUT2   DD   UNIT=VIO,SPACE=(CYL, (1,1))
//SYSUT3   DD   UNIT=VIO,SPACE=(CYL, (1,1))
//SYSUT4   DD   UNIT=VIO,SPACE=(CYL, (1,1))
//SYSUT5   DD   UNIT=VIO,SPACE=(CYL, (1,1))
//SYSUT6   DD   UNIT=VIO,SPACE=(CYL, (1,1))
//SYSUT7   DD   UNIT=VIO,SPACE=(CYL, (1,1))
COBOL CLG PROC – part 3

//LKED EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB DD DSNAMES=&LIBPRFX..SCEELKED,
    // DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSNAMES=&&LOADSET,DISP=(OLD,DELETE)
    // DD DDNAME=SYSIN
//SYSLMOD DD DSNAMES=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
    // UNIT=VIO,DISP=(MOD,PASS)
//SYSUT1 DD UNIT=VIO,SPACE=(TRK,(10,10))
//GO EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
    // REGION=2048K
//STEPLIB DD DSNAMES=&LIBPRFX..SCEERUN,
    // DISP=SHR
//SYSPRINT DD SYSOUT=*
//CEEDUMP DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*

© Copyright IBM Corp., 2005. All rights reserved.
Load module creation and loading

- Source modules
  - Assembler or compiler
    - Object modules
      - Program management binder
        - Linkage Editor
          - Load modules in PDS program library
            - Program management loader
              - Load modules in virtual storage ready for execution
                - Batch loader
Linkage Editor

The linkage editor prepares a load module to be brought into real storage for execution.

As input, the linkage editor accepts object modules, control statements, and user-specified input.

Output of the linkage editor is:
- A load module placed in a library (a PDS) as a named member
- Diagnostic output produced as a sequential data set
How a load module is created

The linkage editor assigns virtual storage addresses to control sections of the module and resolves references between them.

The output load module contains the input object modules and input load modules processed by the linkage editor.

The load module also contains the text from each input module, and an end-of-module indicator.
Using the binder

Performs all of the functions of the linkage editor
Can also process program objects stored in PDSE data sets
Binder removes restrictions of the linkage editor
Input can include traditional data sets and z/OS UNIX files
Binder sample

Link-edits an object module

The output from the LKED step will be placed in a private library identified by the SYSLMOD DD

The input is passed from a previous job step to a binder job step in the same job (for example, the output from the compiler is direct input to the binder)
EXEC PGM=IEWL,PARM='XREF,LIST', IEWL is IEWBLINK alias
REGION=2M,COND=(5,LT,prior-step)

Define secondary input

//SYSLIB DD DSN=language.library,DISP=SHR optional
//PRIVLIB DD DSN=private.include.library,DISP=SHR optional
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(1,1)) ignored

Define output module library

//SYSLMOD DD DSN=program.library,DISP=SHR required
//SYSPRINT DD SYSOUT=* required
//SYSTEM DD SYSOUT=* optional

Define primary input

//SYSLIN DD DSN=&&OBJECT,DISP=(MOD,PASS) required
// INCLUDE PRIVLIB(membername)
// NAME modname(R)

/*
Creating load modules for executable programs

A load module is an executable program stored in a partitioned data set program library.

Creating a load module to execute only will require that you use a batch loader or program management loader.

Creating a load module that can be stored in a program library requires that you use the binder or linkage editor.
Creating load modules for executable programs (continued)

In all cases, the load module is relocatable

- That is, it can be located at any address in virtual storage within the confines of the residency mode (RMODE)

Relocatable programs allow an identical copy of a program to be loaded in many different address spaces, each being loaded at a different starting address
Batch loader

The batch loader:

- Performs link-editing and loading into one job step
- Accepts object modules and load modules, and loads them into virtual storage for execution
- Prepares the executable program in storage and passes control to it directly

The batch loader is replaced by the binder in later releases of z/OS.
Program management loader

The program management loader:

- can load program objects
- reads both program objects and load modules into virtual storage and prepares them for execution
- resolves any address constants in the program to point to the appropriate areas in virtual storage and supports the 24-bit, 31-bit and 64-bit addressing ranges
Compile, link and execute JCL

//USUAL     JOB     A2317P,'COMPLGO'
//ASM       EXEC    PGM=IEV90,REGION=256K, EXECUTES ASSEMBLER
               PARM=(OBJECT,NODECK,'LINECOUNT=50')
//SYSPRINT   DD    SYSOUT=*,DCB=BLKSIZE=3509    PRINT THE
               ASSEMBLY LISTING
//SYSPUNCH   DD    SYSOUT=B                    PUNCH THE ASSEMBLY LISTING
//SYSLIB     DD    DSNAME=SYS1.MACLIB,DISP=SHR  THE MACRO LIBRARY
//SYSUT1     DD    DSNAME=&&SYSUT1,UNIT=SYSDA, A WORK DATA SET
               SPACE=(CYL,(10,1))
//SYSLIN     DD    DSNAME=&&OBJECT,UNIT=SYSDA, THE OUTPUT
               OBJECT MODULE
               SPACE=(TRK,(10,2)),DCB=BLKSIZE=3120,DISP=(,PASS)
//SYSIN      DD    *                        inline SOURCE
               CODE

. . .

Source Code
/*
//LKED      EXEC    PGM=HEWL, EXECUTES LINKAGE EDITOR
               PARM='XREF,LIST,LET',COND=(8,LE,ASM)
//SYSPRINT   DD    SYSOUT=*
               LINKEDIT MAP

Printout
Compile, link and execute JCL (continued)

//SYSLIN DD DSNAME=&&OBJECT, DISP=(OLD, DELETE) INPUT OBJECT
MODULE

//SYSUT1 DD DSNAME=&&SYSUT1, UNIT=SYSDA, A WORK DATA SET
// SPACE=(CYL,(10,1))

//SYSLMOD DD DSNAME=&&LOADMOD, UNIT=SYSDA, THE OUTPUT LOAD MODULE
// DISP=(MOD, PASS), SPACE=(1024, (50, 20, 1))

//GO EXEC PGM=*.LKED.SYSLMOD, TIME=(, 30), EXECUTES THE PROGRAM
// COND=((8, LE, ASM), (8, LE, LKED))

//SYSUDUMP DD SYSOUT=*
LISTING IF FAILS, DUMP

//SYSPRINT DD SYSOUT=*, OUTPUT LISTING
// DCB=(RECFM=FBA, LRECL=121)

//OUTPUT DD SYSOUT=A, PROGRAM DATA OUTPUT
// DCB=(LRECL=100, BLKSIZE=3000, RECFM=FBA)

//INPUT DD * PROGRAM DATA INPUT

/*
*/
Load library

A load library contains programs ready to be executed

A load library can be one of the following:

- System library
- Private library
- Temporary library
System library

Unless a job or step specifies a private library, the system searches for a program in the system libraries when you code:

```
//stepname EXEC PGM=program-name
```

The system looks in the libraries for a member with a name or alias that is the same as the specified program-name.

The most used system library is SYS1.LINKLIB, which contains executable programs that have been processed by the linkage editor.
Private libraries are useful for storing programs too seldom needed to be stored in a system library.

User-written programs are usually stored as members in a private library.

To indicate that a program is in a private library, you code the DD statement defining the library with JOBLIB or STEPLIB.

Result: The system searches for the program to be executed in the library defined by the JOBLIB or STEPLIB DD statement before searching in the system libraries.
Overview of compilation to execution

After compilation:
- PROGRAM A: 0 bytes
- PROGRAM B: 60 KB

Object Modules

After Link-edit:
- PROGRAM B: 140 KB
- PROGRAM A: 80 KB

Load Module

2-GB Virtual Storage Address Space:
- Common: 16 MB
- MYPROG: 20 KB
- PROGRAM LIBRARY: 0 bytes
- MYPROG: 20 KB
- PROGRAM LIBRARY: 0 bytes
- 2 GB
Using procedures

Saves time and prevents errors

Consist of job control statements placed in a PDS or PDSE known as a procedure library

When stored in the system procedure library SYS1.PROCLIB (or an installation-defined procedure library) is called a cataloged procedure

A procedure in an input stream is called an inline procedure

Limit of 15 inline procedures per job.
Sample definition of a procedure

//DEF PROC STATUS=OLD,LIBRARY=SYSLIB,NUMBER=777777
//NOTIFY EXEC PGM=ACCUM
//DD1 DD DSNAME=MGMT,DISP=(&STATUS,KEEP),UNIT=3400-6,
//     VOLUME=SER=888888
//DD2 DD DSNAME=&LIBRARY,DISP=(OLD,KEEP),UNIT=3350,
//     VOLUME=SER=&NUMBER
Summary

The basic steps for turning a source program into an executable load module are:

1. compile
2. link-edit
3. bind

The compile step translates source code into relocatable machine language, in the form of object code.

Object code must be processed by a binder (or linkage editor or loader) before it can be executed.
Summary (continued)

The binder resolves unresolved references in the object code and ensures that everything needed by the program is included in the final load module.

To execute a load module, the binder loads the module into real storage and then transfers control to it to begin execution.