Lab 3 – Linker/Loader

Linker/Loader User's Guide

CSE 3903

Spring 2023

Group: Worst Name Ever

Sade Ahmed

Jeremy Bogen

Mani Kamali

Giridhar Srikanth

Date of Submission: 04/19/2023

Table of Contents

Introduction

This is the User's Guide for the WNE Linker and Loader. The WNE Linker and Loader takes in one or more input object files (with or without external symbols), and links and loads them into a single object file. This document contains an installation guide, quickstart guide, input file format definitions, output and errors, and so on.

Installation Guide

System Requirements

The user's machine must be run on Java 17 or above to support the assembler. As per the Oracle documentation, the following operating systems will support those versions of Java:

RAM: 128MB
Disk space: 124 MB for JRE; 2 MB for Java Update
Processor: Minimum Pentium 2 266 MHz processor
Browsers: Internet Explorer 9 and above, Microsoft Edge, Firefox, Chrome

Windows

Windows 11 (64 bit only) 8u311 and above
Windows 10 (8u51 and above)
Windows 8 (Modern UI is not supported)
Windows 8 SP1* (No longer supported by MS)
Windows Vista SP2* (No longer supported by MS)
Windows Server 2022
Windows Server 2019
Windows Server 2016
Windows Server 2012 R2
Windows Server 2012
Windows Server 2008 R2 SP

MacOS

macOS 12 (8u311 and above)
maxOS 11 (8u281 and above)
OS X 10.9 and above
OS X 10.8.3 and above
Administrator privileges for installation
64-bit browser (A 64-bit browser (Safari, for example) is required to run Oracle Java on macOS)

Linux

Oracle Linux 8 (8u221 and above)
Oracle Linux 7 (64-bit) (8u20 and above)
Oracle Linux 6.(32-bit and 64-bit)
Oracle Linux 5.5+
Red Hat Enterprise Linux 8 (8u221 and above)
Red Hat Enterprise Linux 7 (64-bit) (8u20 and above)

Red Hat Enterprise Linux 6 (32-bit and 64-bit)
Red Hat Enterprise Linux 5.5+
Suse Linux Enterprise Server 15 (8u201 and above)
Suse Linux Enterprise Server 12 (64-bit)[2] (8u31 and above)
Suse Linux Enterprise Server 11 (32-bit and 64-bit)
Suse Linux Enterprise Server 10 SP2+ (32-bit and 64-bit)
Ubuntu Linux 21.04 (8u291 and above)
Ubuntu Linux 20.10 (8u271 and above)
Ubuntu Linux 20.04 LTS (8u261 and above)
Ubuntu Linux 19.10 (8u241 and above)
Ubuntu Linux 19.04 (8u231 and above)
Ubuntu Linux 18.10 (8u191 and above)
Ubuntu Linux 18.04 LTS (8u171 and above)
Ubuntu Linux 17.10 (8u151 and above)
Ubuntu Linux 17.04 (8u131 and above)
Ubuntu Linux 16.10 (8u131 and above)
Ubuntu Linux 16.04 LTS (8u102 and above)
Ubuntu Linux 15.10 (8u65 and above)
Ubuntu Linux 15.04 (8u45 and above)
Ubuntu Linux 14.10 (8u25 and above)
Ubuntu Linux 14.04 LTS (8u25 and above)
Ubuntu Linux 13
Ubuntu Linux 12.04 LTS

Building and Running

The WNE Assembler uses Gradle to build and run. There are instructions here to both build with the command line and Eclipse. But the command line is the main supported environment for users.

Getting the Code

An archive file of the most recent release of the assembler can be downloaded on the Github repository's releases page. Extracting this archive results in the folder that is used for the following sections on building and running the program.

Running and Building Using the Command Line

Change directories so that the program's root folder is your current working directory. You may want to double-check your java version by running "`java -version`", our program only runs with java 17 or greater.

If your working directory is the program's root directory and you have an up-to-date version of Java, running the program is very simple. To run the program simply run, for example "`./gradlew run --args="../input-file-1.o ../input-file-2.o … ../input-file-n.o output.o`". Replacing "`./input-file-1.o, ../input-file-2.o … ../input-file-n.o'' with` your input object file paths. Note: the program runs from the `app/` directory, which is why the input and output files use `../`.

The project can also be built with a simple "`./gradlew build`".

Quick Start

Before running the program, we first need to define our input file format. This quickstart will give a brief overview of the input object files, and assumes the reader is using another program to generate them. For a more detailed description of the input file format, please check out the Input Object File Definition section.

Input Files

Input files have four types of entries: H-records, N-records, T-records, and E-records, each named after the first character used to denote them. Here are the input files for a quick example run, these can both be found in the `examples/` directory:

examples/Main.o:

```
HMain  0002
NNum=1
T00004000X9Put
T00010007
E0000
```

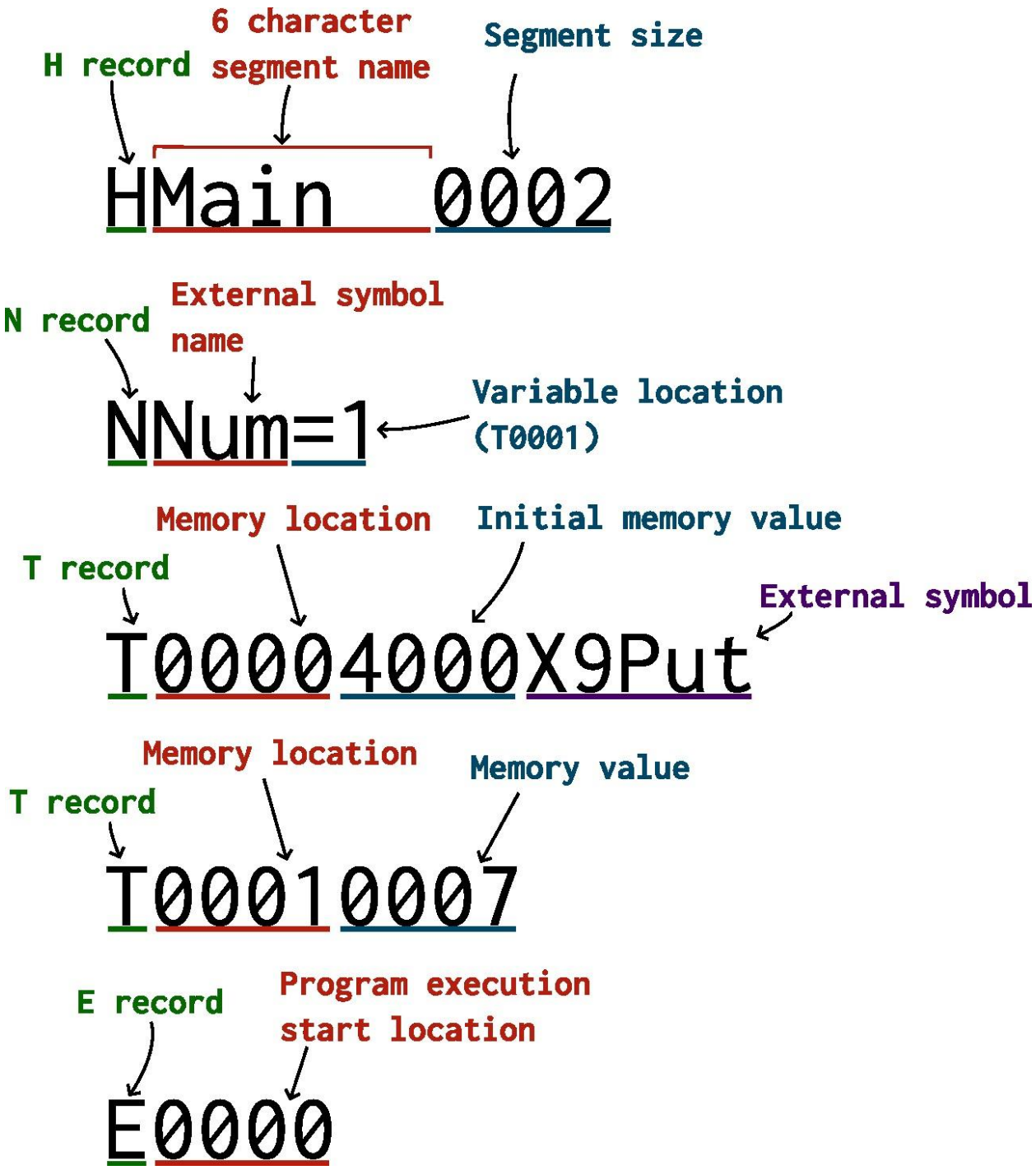
examples/Lib.o:

```
HLib   0003
NPut=0
T00002000X9Num
T0001F031
T0002F025
E0000
```

This is a very simple example of a program with two segments that refer to each other's external symbols. The program will always start executing the "**`Main`** " segment first. In this case the `Main` segment uses the `Put` external symbol to enter the `Lib` segment and continue execution from there. As stated before, each segment has four types of records:

1. H-records denote the segment name and size of the program.
2. N-records denote external symbols defined in the current object and their locations.
3. T-records denote what values go into what memory addresses, and when external symbols are used.
4. E-records denote the end of an object file and where to begin execution, if applicable.

These records are annotated in main.o on the following page.



So, to run this quickstart with the examples files, simply run: "`./gradlew run --args="../examples/main.o ../examples/lib.o ../output.o`" from the project root directory. This will output a linked and loaded "`output.o`" to the project root directory. From there, "`output.o`" can then be run using a compatible simulator (for example, the one provided in the integrated system outlined in the WNE integrated system user's guide).

Input Object File Definition

The input to the WNE Linker and Loader are the several object files produced by the WNE Assembler.

Object file format

The main output file is the file that will become the input for the lab 1 simulator. This includes having a singular header record, any number of text records, and an end record.

The Header Record is split into 4 sections, those being:
Record position 1: H (to indicate Header)
Record positions 2-7: a 6-character segment name (Must be exactly 6 characters)
Record positions 8-11: a set of hex characters denoting the IPLA (initial program load address)
Record positions 12-15: a set of hex characters denoting the full length of the segment

Note that the segment of memory reserved must be within the bounds of the total address space

HLab3EG30B00018
Figure 8: Example of a header record

The Text Records are split into 4 sections, those being:
Record position 1: T (to indicate Text)
Record positions 2-5: a set of hex characters denoting the address at which the information is to be stored.
Record positions 6-9: a set of hex characters that represent the contents of the address specified by positions 2-5
Records positions 10-12: modification records for relocatable programs

T30B0127F
Figure 9: Example of a text record

For each symbol defined by the **.**EXT has unknown values and cannot be determined until all the assembly files are converted to object files. Before passing the relocatable object file to the linker/loader, the program must know which records use external symbols. So, each of those records is attached with an X-record to the end. The X-records include a nine-bit and sixteen-bit variant. The nine-bit X-records are attached for instructions that use page offset as part of their fields (address field). The sixteen-bit X-records are used for instructions where the entire

operand field requires only an address. These records replace the previously used M-records to use the segment name appended to the end of X-records instead.

The format of the X-records is as follows:

1. X
2. Bit-offset (9 or 16)
3. Symbol name/Segment name

Figure 1M below shows examples of text records with X-records with both modifiers.

```
T00000085X9Sym → T 0000 0085 X 9 Sym1
T00000000X16Sym2 → T 0000 0000 X 16 Sym2
```

Figure 1M: X-record examples.

For each symbol given by the **.**ENT op in the operand field is defined in the provided segment. The N-records allow for defining each of those symbols for the linker/loader. The N records are placed after the header record and before any text record. The N-record contains the name of the symbol and the value assigned to it, separated by an equal sign. As the N-records use an equal sign to indicate the value of a symbol, literals are not permitted to be used as entry symbols.

The format of the N-records is as follows:

1. N
2. =
3. Value of the symbol in hex (without the “x”)

Figure 2M below shows examples of text records with N-records with symbols of different character lengths.

```
NSym1=0 → N Sym1 = 0
NLabel1=4 → N Label1 = 4
```

Figure 2M: N-record examples.

The End Records are split into 2 sections, those being:
Record position 1: E (to indicate End)
Record positions 2-5: a set of hex characters denoting the address at which execution is to begin

E30B0

Figure 10: Example of an end record

Linker and Loader Output Definition

The WNE Linker and Loader takes the various input object files and turns them into a singular object file that is able to be executed by the WNE Machine. The definitions for each section of the output object file is identical to the input, with the exception that there are no modification (X or N) records.

Errors

Error	**Common Solution**
Program must fit in one page, it currently does not.	Make sure your object file header records report the correct sizes. If they do, then your program is too large to be relocatable. You will have to either reduce the program space or rewrite the program as a single non-relocatable segment.
Symbol defined outside segment: <segment_name>	An external symbol is defined outside a segment's reported size/space. Double check external symbols and what you're using to generate object files.
No Line found	Either the file is shorter than expected or there is an internal error in the Linker/Loader. Try rewriting or regenerating the object file and trying again.
Symbol <symbol_name> not defined	This object needs an external symbol that is not defined in any of the input object files. This is normally caused by forgetting to include a file path when calling the program.
File not found	One of the input files could not be found. Remember that the program runs relative to the `app/` directory and double check your spelling.
Need "Main " segment	The input files need to include a "Main " segment for the linker-loader to know where execution starts.