A Project Report

ON

IMPLEMENTATION OF LINKERS AND LOADERS

BY

Sharath KP - 1PI13CS141
Shashwath.S.Bharadwaj – 1PI13CS144
Shreyas.N.Gandhi – 1PI13CS154
Nithin Jain – 1PI13CS101

Guide
Preet Kanwal
Assistant Professor,
PESIT-CSE
Bangalore

August 2015 – Dec 2015
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
PES INSTITUTE OF TECHNOLOGY
(an autonomous institute under VTU)
100 FEET RING ROAD, BANASHANKARI III STATE
BANGALORE-560085

PES Institute of Technology,
(an autonomous institute under VTU)

100 Feet Ring Road, BSK 3rd Stage, Bangalore-560085

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE (Centre, Bold, Underlined –16size)

Certified that the Special Topics: Mini Project work entitled **Implementation of Linkers and Loaders** is a bonafied work carried out by **Sharath KP(1PI13CS141), Shashwath.S.Bharadwaj(1PI13CS144), Shreyas.N.Gandhi(1PI13CS154), Nithin Jain(1PI13CS101)** in partial fulfillment for the award of degree of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the academic semester August 2015 to December 2015.

Signature of the Guide   Signature of the HOD

Preet Kanwal   **Prof. Nitin V Pujari**

Sharath KP(1PI13CS141)

Shashwath.S.Bharadwaj(1PI13CS144)

Shreyas.N.Gandhi(1PI13CS154)

Nithin Jain(1PI13CS101)

ABSTRACT

A two pass direct linking loader was implemented in C. The input to the first pass of the system was the object file containing the various control sections of the programs to be linked. The output of the first pass, an external symbol table containing the addresses of external symbols, was input to the second to the second pass along with the object file. The output of the second pass was the object program loaded into the memory locations specified.

ACKNOWLEDGEMENT

Several people were responsible for the successful completion of this project. I would like to take this opportunity to thank them for their support and encouragement through the course of this project.
First of all, I would like to thank our course instructor and guide, Preet Kanwal ma'am, for helping us take up this particular project. Her constant support, guidance and encouragement through the various phases of our project was instrumental in its successful completion. I would also like to thank my team mates for their commendable efforts towards the completion of the project. The project couldn't have been completed without their persistent efforts and perseverance.
Lastly, I would also like to thank the department of Computer Science and Engineering for providing us with such opportunities to broaden our skill set. A special mention is to be made of the families of all the team members for their continued help and support without which this project would not have been possible.

TABLE OF CONTENTS

1. Introduction

A linker is a tool that merges the object files produced by separate compilation or assembly of various source code modules and creates an executable file. It primarily has 3 tasks:

- Search the program to find library routines used by it
- Determine the memory locations that code from each module will occupy and relocate its instructions by adjusting absolute references
- Resolve references among files

A loader is a part of the compiler or the OS that brings an executable file residing on disk into memory and starts its execution. The steps involved in theloading process are:

- Read executable file's header to determine the size of text and data segments
- Create a new address space for the program
- Copy the instructions and data into the address space
- Copy arguments passed to the program on to the stack
- Initialise the machine registers including the stack pointer
- Jump to a startup routine that copies the program's arguments from the stack to the registers and calls the program's main routine.

A two pass direct linking loader performs the operations of both linking and loading phases in 2 separate passes. It supports linking, relocation and the loading of the executable program into memory.

2. Problem Definition

To implement a two pass direct linking loader to demonstrate the linking and loading operations of a compiler.
The input to the system is an object code containing control sections of various programs to be loaded into memory. The output is linked object code loaded into the memory locations specified.

3. Project and System requirements

The project requires an input object file containing Header, Data, Text, Modification and end records. It must also contain relocation information required by the linking phase and references to locations in memory where the program is to be loaded for execution, that is required by the loading phase.
Since the implementation of the system is in C and the UI in python, the project requires compilers of both the above languages.
The system has been designed keeping in mind the SIC/XE machine architecture. It can be implemented and run on any machine that supports this architecture and has C and python compilers installed.

4. System Design

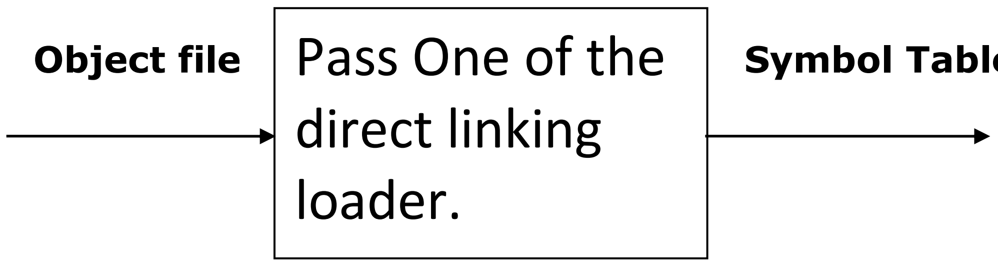
The system has been designed to work in two phases corresponding to the two passes of the direct linking loader.
The functionalities of the two passes along with the inputs and output of each pass is described below:

Pass 1:

Since it is common for separately assembled control sections to make references to sections of code that don't appear until later in the input stream. It is important for the loader to resolve such references to be able to link them and load the code segments into memory. For this purpose, the first pass of the direct linking loader takes the object file containing the various control sections to be linked as the input and produces an external symbol table as the output. The external symbol table contains resolved memory addresses of all external symbols that appear in the object file. This table will be used by the second pass of the loader to perform the actual linking, relocation and loading into memory. The basic functionalities of the first pass of the loader, therefore, are:

- Read the header record of the object file to identify the program to which the control section belongs and identify the length of the control section.
- Identify the starting address(memory location) where the program is to be loaded. This can either be generated by the OS or be input by the user.
- Scan the object file for external references/symbols.
- Resolve all external references i.e assign memory addresses to all external symbols.
- Store all the above information into an external symbol table to be used by the subsequent pass of the loader.

The functionality can be pictorially represented as below:

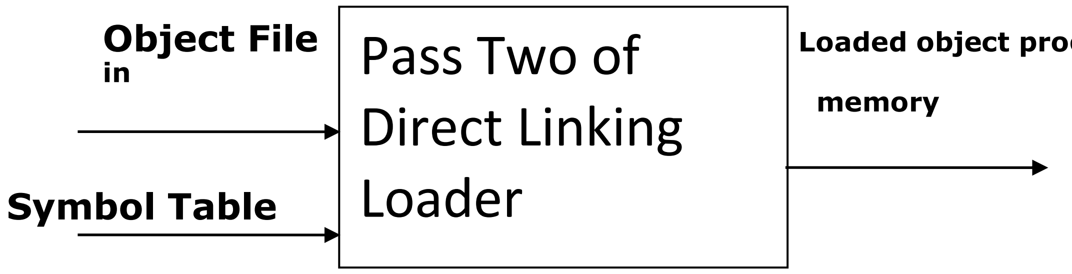


PASS 2:

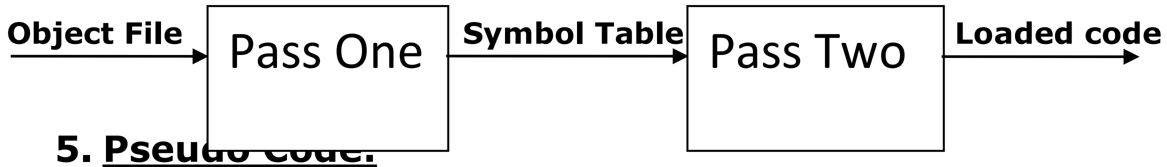
The second pass of the direct linking loader performs the actual linking, relocation and loading of object program into the memory addresses specified. The input to this pass of the loader is the object file and the external symbol table generated by the first pass of the loader. The output is the memory configuration corresponding to the object program loaded into the memory locations specified. The functionality of the second pass can be described as follows:

- Read the text records in the object file and move the data contained, into the memory location specified.
- If a modified record exists for a particular memory address, resolve the actual memory address(to where the data specified in the text record needs to be loaded) using the external symbol table generated using pass one of the loader.
- The above step usually involves adding offsets and relative addresses to the actual address of a particular data record that is specified in the symbol table.
- Load all the data specified in the text records of a control section into their respective memory locations. Repeat this for all control sections present in the input object file.

  The pictorial representation of the above mentioned functionality is given below:

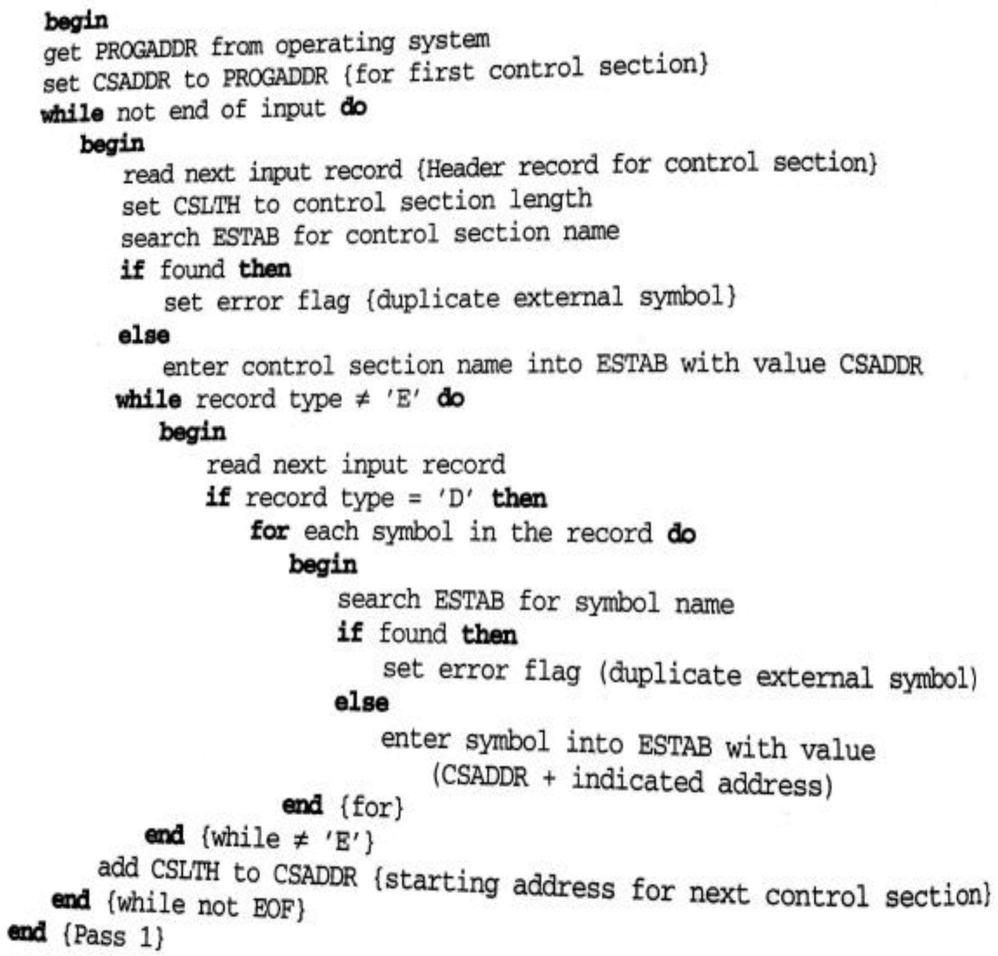

Object File in
Symbol Table
Pass Two of Direct Linking Loader
Loaded object program memory

The combined working of both the passes system can be depicted as follows:



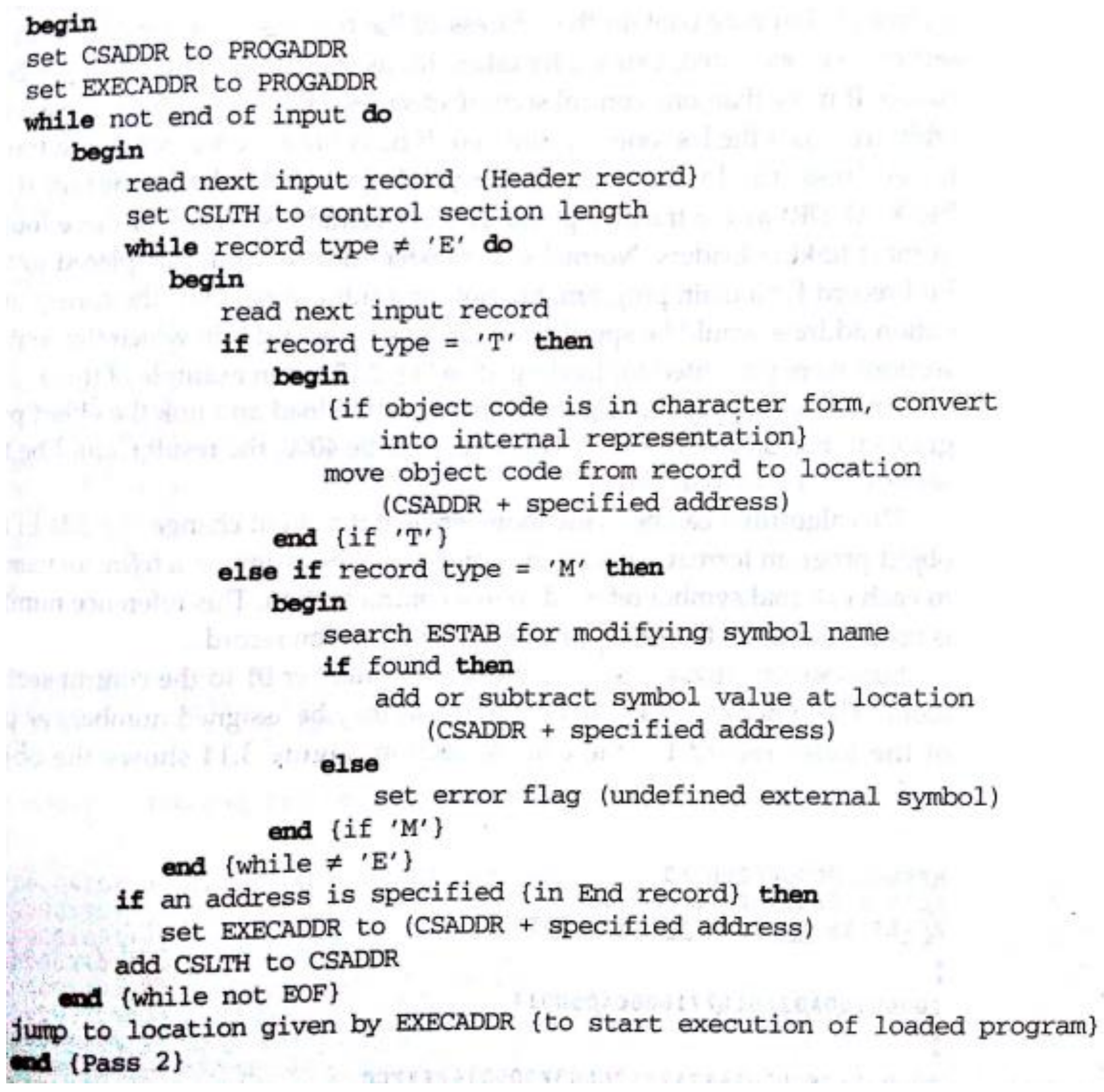
5. Pseudo Code:

Pass one:



```
begin
get PROGADDR from operating system
set CSADDR to PROGADDR {for first control section}
while not end of input do
   begin
      read next input record {Header record for control section}
      set CSLTH to control section length
      search ESTAB for control section name
      if found then
         set error flag {duplicate external symbol}
      else
         enter control section name into ESTAB with value CSADDR
      while record type ≠ 'E' do
         begin
            read next input record
            if record type = 'D' then
               for each symbol in the record do
                  begin
                     search ESTAB for symbol name
                     if found then
                        set error flag (duplicate external symbol)
                     else
                        enter symbol into ESTAB with value
                           (CSADDR + indicated address)
                  end {for}
         end {while ≠ 'E'}
      add CSLTH to CSADDR {starting address for next control section}
   end {while not EOF}
end {Pass 1}
```

Pass two:



```
begin
set CSADDR to PROGADDR
set EXECADDR to PROGADDR
while not end of input do
   begin
      read next input record {Header record}
      set CSLTH to control section length
      while record type ≠ 'E' do
         begin
            read next input record
            if record type = 'T' then
               begin
                  {if object code is in character form, convert
                     into internal representation}
                  move object code from record to location
                     (CSADDR + specified address)
               end {if 'T'}
            else if record type = 'M' then
               begin
                  search ESTAB for modifying symbol name
                  if found then
                     add or subtract symbol value at location
                        (CSADDR + specified address)
                  else
                     set error flag (undefined external symbol)
               end {if 'M'}
         end {while ≠ 'E'}
      if an address is specified {in End record} then
         set EXECADDR to (CSADDR + specified address)
      add CSLTH to CSADDR
   end {while not EOF}
jump to location given by EXECADDR {to start execution of loaded program}
end {Pass 2}
```

6. **<u>Results:</u>**

Thus, a two pass direct linking loader was implemented using C and python. The operations performed during the linking and loading phases of a compiler were clearly demonstrated using a suitable object file as input. A clear understanding of the linking, relocation and loading processes involved was obtained.

7. **Future Enhancements:**

- The system can be made more robust by adding a simulator and converting the code into assembly code so as to obtain a clearer understanding of how linking and loading actually happens in a machine.
- The starting address of loading which is currently input by the user can be auto generated by the OS to make the system more real in terms of its operation.
- The system, during simulation, can be made to show how the control transfers to the starting address of the object program for execution upon encountering an End record.

8. **Bibliography**:

- System Software 3rd edition – Leland Beck and D Manjula
- https://cs.gmu.edu/~setia/cs365-S02/assembler.pdf
- https://www.scribd.com/doc/35787646/11/PASS-TWO-OF-DIRECT-LINKING-LOADER#logout
- http://www.tutorialspoint.com/python/pdf/python_gui_programming.pdf
- https://docs.python.org/2/library/subprocess.html