R.V. COLLEGE OF ENGINEERING
BENGALURU-560059
(Autonomous Institution Affiliated to VTU, Belagavi)

# Simulation of loader in ‘C’

**PROJECT REPORT**

*Submitted by*

| NAME OF THE CANDIDATE | USN |
|---|---|
| Nithish DS | 1RV15IS037 |
| Pranay Garg | 1RV15IS040 |

**Under the Guidance of**

**Faculty Lab Incharge – 1**

**Prof. Swetha S**
**Assistant Professor**
**Department of ISE**
**RVCE, Bengaluru**

**Faculty Lab Incharge – 2**

**Prof. Vanishree**
**Associate Professor**
**Department of ISE**
**RVCE, Bengaluru**

***in partial fulfillment for completion***
***of***
*System Software Laboratory*
***Bachelor of Engineering***
***In***
***Department of Information Science & Engineering***

**R.V. COLLEGE OF ENGINEERING**
**BENGALURU - 560059**
**(Autonomous Institution Affiliated to VTU, Belagavi)**

**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**

## CERTIFICATE

Certified that the project work titled **Simulation of loader in 'C'** is carried out by **Nithish DS (1RV15IS037) and Pranay Garg (1RV15IS040)** who are a bonafide students of R.V College of Engineering, Bengaluru, in partial fulfillment for the completion of **System Software Laboratory (12IS52),** the requirement for the award of degree of **Bachelor of Engineering** in **Department of Information Science & Engineering** of the Visvesvaraya Technological University, Belagavi during the year **2017-18**. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed by the institution for the said degree.

| **Signature of Lab Incharge-1** | **Signature of Lab Incharge-2** | **Signature of HOD** |
|---|---|---|
| Prof. Swetha S<br>Assistant Professor<br>Department of ISE<br>RVCE, Bengaluru | Prof. Vanishree<br>Associate Professor<br>Department of ISE<br>RVCE, Bengaluru | Dr N K Cauvery<br>Head of Department<br>Department of ISE<br>RVCE, Bengaluru |

**Semester End Examination**

| | **Name of Examiners** | **Signature with date** |
|---|---|---|
| **1** | | |
| **2** | | |

**R.VCOLLEGE OF ENGINEERING**

**BENGALURU - 560059**
**(Autonomous Institution, Affiliated to VTU, Belagavi)**

**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**

# DECLARATION

I, **Nithish DS,** student of fifth semester B.E., **Information Science and Engineering,** hereby declare that the project titled **"Simulation of loader in 'C'"** has been carried out and submitted in partial fulfillment for the completion of **System Software Laboratory (12IS52),** the requirement for the award of degree of **Bachelor of Engineering** in **Department of Information Science & Engineering.**

**Place: Bengaluru** **Signature**
**Date:**

**R.VCOLLEGE OF ENGINEERING**
**BENGALURU - 560059**
**(Autonomous Institution, Affiliated to VTU, Belagavi)**

**DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING**

# DECLARATION

I, **Pranay Garg,** student of fifth semester B.E., **Information Science and Engineering,** hereby declare that the project titled **"Simulation of loader in 'C'"** has been carried out and submitted in partial fulfillment for the completion of **System Software Laboratory (12IS52),** the requirement for the award of degree of **Bachelor of Engineering** in **Department of Information Science & Engineering.**

**Place: Bengaluru** **Signature**
**Date:**

## ACKNOWLEDGEMENT

I express my sincere gratitude to Prof. B K Srinivas, Faculty In-charge of System Software Laboratory for guiding me in this project titled “Simulation of loader in ‘C’ ” and providing his valuable inputs for the same. I would also like to thank Prof. Swetha and Prof. Vanishree K, Faculty In-charge of System Software Laboratory for her support and encouragement during the course of this project.

I am grateful to Dr. N K Cauvery, Head of the Department of Information Science and Engineering without whom this project would not have been possible. I would also like to thank Dr. K N Subramanya, Principal, R V College of Engineering, for providing me this wonderful learning opportunity.

# TABLE OF CONTENTS

## ABSTRACT

Loader is a program that loads machine codes of a program into the system memory. In Computing, a loader is the part of an Operating System that is responsible for loading programs. It is one of the essential stages in the process of starting a program. Because it places programs into memory and prepares them for execution. Loading a program involves reading the contents of executable file into memory. Once loading is complete, the operating system starts the program by passing control to the loaded program code. All operating systems that support program loading have loaders.

Our project involves using bash, a command-line based command prompt, loaded with our unique program. The user can choose from a variety of features and options provided, and hence understand loader more effectively and efficiently.

Our implementation involves use of the C language and its libraries to implement the various features. To support software development, the program has relocatable mode in which features such as Auto relocation works efficiently.

This editor is unique since it accurately emulated the SIC/XE relocatable assembly object code.

## LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

Loader is a program that loads machine codes of a program into the system memory. In Computing, a loader is the part of an Operating System that is responsible for loading programs. It is one of the essential stages in the process of starting a program. Because it places programs into memory and prepares them for execution. Loading a program involves reading the contents of executable file into memory. Once loading is complete, the operating system starts the program by passing control to the loaded program code. All operating systems that support program loading have loaders. In many operating systems the loader is permanently resident in memory.

## 1.1 Motivation

The motivation behind choosing this topic as our project is, loader is basically an essential part of any OS as it is the key program which loads the executable file to the primary memory of the machine, thus it is essential to understand it’s working in order to completely understand the functionality of OS.

## 1.2 Problem Statement

The problem statement is Simulation of loader in C.

## 1.3 Objectives

- Defining the structure of the received input file.
- Simulating the allocation of primary memory for the program's execution.
- Copying the address space from secondary to primary memory.
- Copying the .text and .data sections from the executable into primary memory.
- Copying program arguments (e.g., command line arguments) onto the stack.
- Initializing registers, setting up the esp (stack pointer) to point to top of stack, clearing the rest.

- Displaying the final memory allocations in depth (with content of each memory location)

## 1.4 Methodology

- **Defining Input-file:**
  Defining the format of the object file which must be fed to our loader. Parsing the input file to generate required tokens to identify all the variables in the program distinctly. Using those tokens, allocation of memory will be performed.

- **Conversion to machine readable format:**
  The input is read from the source code which is the object code produced by the Assembler. The code is then converted into pure hexadecimal ASCII format so that it can be directly written into a binary file. The binary file interm can be thus directly moved into the RAM, simulation a Loader.

- **Relocating the Code:**
  Now the code is reallocated based on the memory available at the time when the program was loaded, the object code is tweaked to reflect the same.

- **Generating Output File:**
  The Output file is generated simulating the loading of program into the memory, the output is formatted in order to show which instruction was loaded at which particular memory location.

# CHAPTER 2

# REQUIREMENTS SPECIFICATION DETAILS

## 2.1 Hardware requirements

**Recommended:**

Intel® Core™ 2 Duo processor or Intel® Xeon® processor or higher
AMD FX 8350 or higher

**Minimum Requirements:**

One of the following processors
Intel® Pentium® 4 processor family and higher
Intel(R) Xeon Phi(TM) coprocessor
Non Intel® processors compatible with the above processors

## 2.2 Software requirements

One of the following operating systems:
Ubuntu* 12.04 LTS and above
Microsoft* Windows* 7 and above
MAC OS X(El Capitan) and above

Intel(R) Cluster Ready
One of the following compilers:
Intel(R) C++ Compiler 13.1 (Intel(R) Parallel Studio XE 2013 SP1) and higher
For each supported Linux* operating system, the standard gcc version provided with that operating system is supported, including gcc 4.1 through 4.8.2

Recommended:
Intel(R) Parallel Studio XE 2013 SP1

## 2.3 Functional Requirements

**Relocation of source code**
The loader can load the code at the runtime dynamically taking into account the memory available at that point of time. The SIC/XE code has the property that the code addresses are relative to each other, thus our program must be able to handle that dynamic relocation of code.

**Address modification for Extended Instructions**
The loader should be able to handle the records which define the location of the extended instruction in the SIC/XE instruction set. These instruction should contain the absolute address instead of the relative address like the normal instruction. Since the assembler has no idea about the final location of the code, it provides a relative code for this instruction as well, which should be handled and made absolute.

**Formatted Output generation**
The loader should be able to simulate the loading of program into the memory thus it should show the exact location of each instruction at the end of the program. This output file in HEX format should consider each byte of bata that’s written into the memory when the program is actually loaded into the memory.

## 2.4 Performance requirements

The program should be able to produce the output considerably fast and without a huge memory requirement for larger input object codes.

# CHAPTER 3
# DESIGN

## 3.1 Software Architecture

The Figure 3.1 provides an insight to the various features implemented in our loader.

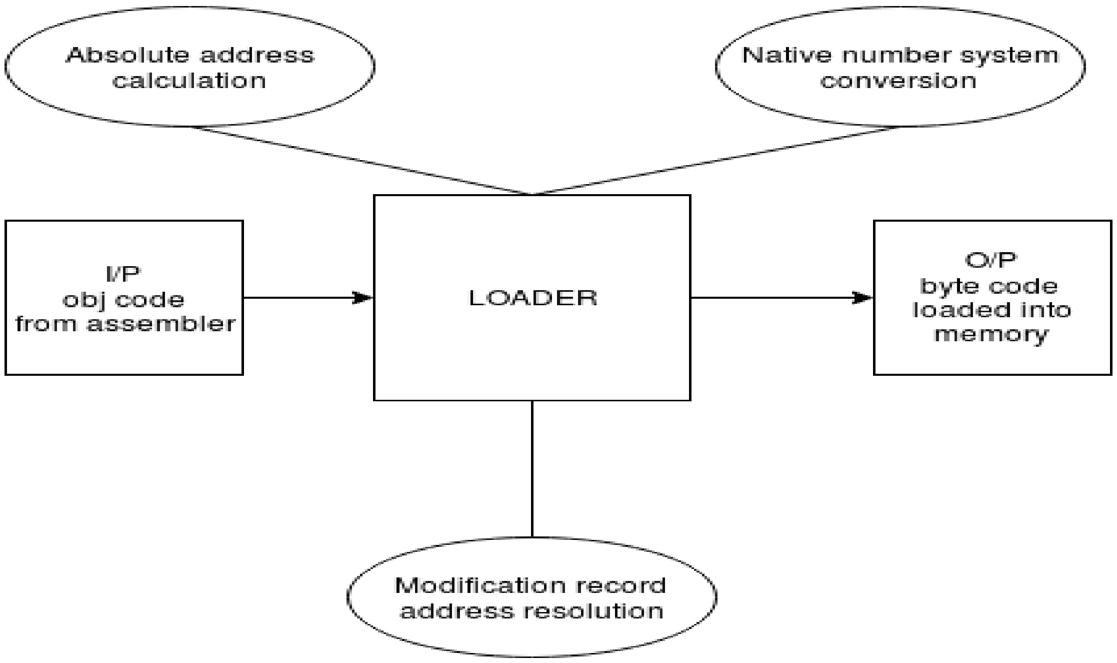

Absolute address calculation
Native number system conversion
I/P obj code from assembler
LOADER
O/P byte code loaded into memory
Modification record address resolution

Figure 3.1: Software architecture of loader

The implemented text editor consists of five menus –

● The SIC/XE Object code rendered from the assembler is taken as the input for the program.

● The program is parsed into a buffer, where each numeric symbol is written into array of bits. The opcode is translated into the binary form for writing into memory.

●Absolute address calculation is done through a code for each text record.

● Modification records have been considered in the address resolution of relocatable loader.

● dynamically, memory is allocated accordingly as addresses are calculated. The same is used to change the offset of return addresses where needed.

● Now to finally complete the simulation, the dump of the program control block is copied into a text file (in binary format) which can be viewed using a hex editor program.

## 3.2 Features

**Absolute address calculation**

The loader should be able to handle the records which define the location of the extended instruction in the SIC/XE instruction set. These instruction should contain the absolute address instead of the relative address like the normal instruction. Since the assembler has no idea about the final location of the code, it provides a relative code for this instruction as well, which should be handled and made absolute.

**Native number system conversion**

Since the ‘C’ programming language doesn't have any in built library for HEX to DEC number conversion, to fo the operations smoothly, we need it to have a library which can convert the addresses to and from HEX to DEC. This will greatly simply our implementation of the Loader.

# CHAPTER 4
# IMPLEMENTATION

## 4.1 Implementation requirements

**Memory Virtualization**

For the idea to work correctly the data should be stored temporarily into an array which would allow us to edit the object code later if the modification instructs us to do so later. This will also allow us to simulate that memory in reality as it would be in the memory.

**Relocation of source code**

The loader can load the code at the runtime dynamically taking into account the memory available at that point of time. The SIC/XE code has the property that the code addresses are relative to each other, thus our program must be able to handle that dynamic relocation of code. To do this we need convert the HEX to DECIMAL and back to efficiently add the required addresses.

**Address modification for Extended Instructions**

The loader should be able to handle the records which define the location of the extended instruction in the SIC/XE instruction set. We shall use the modification record with the correct structure. We shall the hash the values to update correct values.

**Formatted Output generation**

The loader should be able to simulate the loading of program into the memory thus it should show the exact location of each instruction at the end of the program. This output file in HEX format should consider each byte of bata that's written into the memory when the program is actually loaded into the memory. We shall use a normal text file with a binary write operation to simulate the same.

## 4.2 Language Selection

We selected the 'C' programming language because it game is the most fine grain level control over the execution. The even bigger reason was the efficiency. The Loader should be very efficient in cases of large input and take minimal memory in order to execute. C's hardware level support is the best candidate for the same as it doesn't require and VM to run like Java does.

## 4.3 Platform Selection

We selected the Ubuntu platform to develop it, being open-source and very developer friendly.

## 4.4 Code Conventions

- Each function is no longer than 20n lines of code.
- The program has been made modular with a custom header file to make it more extensible and easy to add new functionalities.
- The input is considered to be in the binary format for the compatibility over the system.
- Each function has been commented for it's ease of understanding.
- The command lines input have been taken to make it extensible.

## 4.5 Software Implementation

### Algorithm

```
initiate Object/Bytecode parsing from input file
getline
if  next input character != 'H'
                    print error
                    exit
else
 Read the name of file, length of byte code in file and starting address
  Initialize code section
```

```
While(  next input character==’T’)
          Read the start address ,length  and the bytes of that text record
          Store it in code section
While( next input character==’M’)
         Add the starting address  to the current addresses of bytes specified
         get the new address, rewrite the code section
If  next input character == ‘E’
      Print the location addresses and corresponding bytes there.
end
```

**HEX to INT**

```
Input a hexadecimal number.
get the length
Initialize resultant decimal num to 0
for each character in array
    increment the decimal value equal to the character's significance
return decimal num
```

**Algorithm (intohex)**

```
read the decimal number.
copy decimal number to tempnum.
create an array ans of characters initialized to 0.
i=4 //fixing the length to a normal default value
  while(i!=-1)
         ans[i] =get character form(tempnum%16);
         tempnum = tempnum/16;
```

```
        i--
if tempnum!=0
      print 'Hex number overflow'
return ans
```

## 4.6 Running the Loader

Our project can be run on either Windows or Linux operating system provided C compiler and other relevant packages are installed. Another way to run the editor would be to convert it to an.exe file or .out file.

# CHAPTER 5
# TESTING

Loader is a program that loads machine codes of a program into the system memory. Loader being a system program which indeed loads everything else, thus should be bug free. In Computing, a loader is the part of an Operating System that is responsible for loading programs, thus very critical. It is one of the essential stages in the process of starting a program. Because it places programs into memory and prepares them for execution. Loading a program involves reading the contents of executable file into memory.

## 5.1 Testing Process

The testing process shall have various stages where we shall test various modules individually then combine them to form the combined form which would give us our required output.

## 5.2 Test Plan

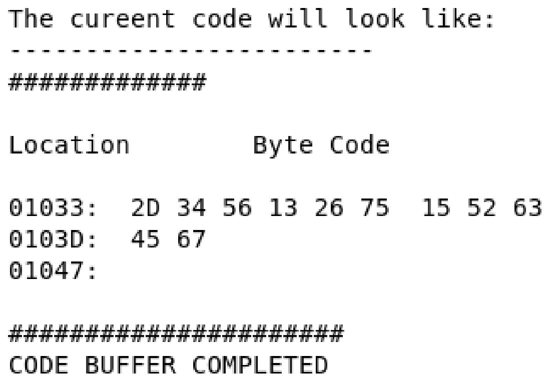
1. Test the Helper functions individually
2. Test the Loader without the relocation ability (Absolute Loader)
3. Test the Loader with the relocation ability.

## 5.3 Integrated Test Cases

**Test case 1:**

```
H COPY 00001A 001033
T 000000 03 2D3456
T 000003 03 122345
T 000007 03 155263
T 00000A 02 4567
M 000004 05
E 000000
```

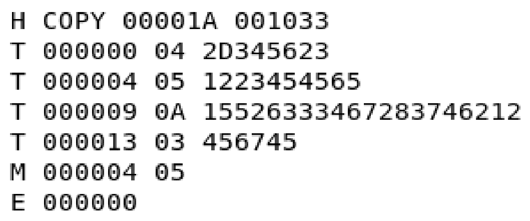
Output:



```
The cureent code will look like:
------------------------
#############

Location        Byte Code

01033:  2D 34 56 13 26 75  15 52 63
0103D:  45 67
01047:

######################
CODE BUFFER COMPLETED
```

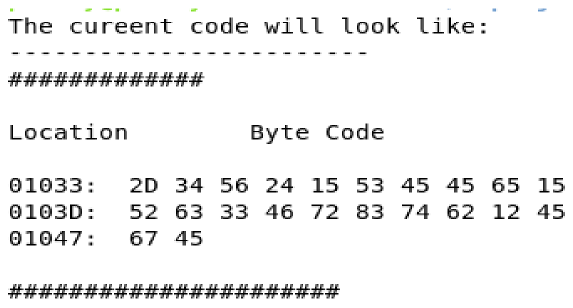
**Test Case 2:**

Input:



```
H COPY 00001A 001033
T 000000 04 2D345623
T 000004 05 1223454565
T 000009 0A 15526333467283746212
T 000013 03 456745
M 000004 05
E 000000
```

Output:



```
The cureent code will look like:
------------------------
#############

Location        Byte Code

01033:  2D 34 56 24 15 53 45 45 65 15
0103D:  52 63 33 46 72 83 74 62 12 45
01047:  67 45

######################
```

As shown in the test cases above, we were successfully able to virtualize the process of loading of the program into the memory.

In the output we can notice that the output is in the required format (as specified in the appendix)

# CHAPTER 6
# RESULTS

Loader being the part of an operating system that is responsible for loading programs and libraries, is one of the essential stages in the process of starting a program, as it places programs into memory and prepares them for execution.

As we have designed our project of simulating a loader in C for both Unix and windows OS, in Unix, the loader is the handler for the system call `execve()`. we have been able to read the object code from an input file (executable) generated by assembler, calculate the absolute and relative addresses and load the byte code into specific memory locations and writing the result into output file.

In our project we have done the simulation of both absolute loader and relocatable loader. Although the logic used in both of them remains almost same, the relocatable loader has been implemented considering the modification records.

Some of the significant results/outputs have been Defining the structure of the received input file, Simulating the allocation of primary memory for the program's execution, Copying the address space from secondary to primary memory, Copying the .text and .data sections from the executable into primary memory, Displaying the final memory allocations in depth (with content of each memory location).

# CHAPTER 7

# CONCLUSIONS AND FUTURE ENHANCEMENTS

All operating systems that support program loading have loaders, apart from highly specialized computer systems that only have a fixed set of specialized programs. Loading a program involves reading the contents of the executable file containing the program instructions into memory, and then carrying out other required preparatory tasks to prepare the executable for running. Once loading is complete, the operating system starts the program by passing control to the loaded program code.

All operating systems that support program loading have loaders, apart from highly specialized computer systems that only have a fixed set of specialized programs. Embedded systems typically do not have loaders, and instead the code executes directly from ROM. In order to load the operating system itself, as part of booting, a specialized boot loader is used. In many operating systems the loader is permanently resident in memory, although some operating systems that support virtual memory may allow the loader to be located in a region of memory that is pageable

## 7.1 Future Enhancements

Dynamic linking loaders are another type of loader that load and link shared libraries (like .so files or .dll files) to already loaded running programs. This could be done in addition to what we have already simulated.

## REFERENCES

[1] Rober C Miller, Brad A Myers, *Relocatable Loader*, School of Computer Science, Carnegie Mellon University, *Comm. ACM*, 1965, pp. 023 – 271.

[2] Allen, R. B.: 1997, M. G. Helander, T. K. Landauer, and P. V. Prabhu (eds.), *Absolute Loader*, *Volume 1*, Slevier Science B.V., New York, pp. 49-73.

[3] Fischer, G. and Girgensohn, A.: 1990, *Multipurpose Loader*, *(CHI'90) (Seattle, WA)*, ACM, New York.

[4] IBM Corporation (1972). IBM OS Linkage Editor and Loader, *System Programming*, *Volume 1*, California, pp. 149-273.

[5] “System Software”, LeLand. A . Beck, Chapter 5,pp. 110-150 3rd edition, Princeton publication(2007).

## APPENDIX

## Input format:

**Start Clause:**

| Label | Column | EXPLANATION |
|---|---|---|
| H | 0-1 | Denotes the starting of the OBJ File |
| Program Name | 2-7 | Contains the name of the program. |
| Program Length | 8-13 | Contains the no of bytes of data in the object code of program (HEX). |
| Program Starting Location | 14-19 | Contains the starting address of the location where the object code is to be placed. |

| Label | Column | EXPLANATION |
|---|---|---|
| T | 0-1 | Denotes the starting of the Text record |
| Starting offset | 2-7 | Contains the offset address of text record from the starting location |
| No. of bytes | 8-9 | Contains the no of bytes of data that record holds. |
| Data | 10-78 | Contains several bytes (1-69) of object code. |

| Label | Column | EXPLANATION |
|---|---|---|
| M | 0-1 | Denotes the modification record |
| Nth byte offset | 2-7 | This contains the offset address of starting of byte where modification  record is needed |
| No. of half bytes to make absolute | 8-9 | Gives the no. of  half bytes starting from the Specified location ,that need to be made absolute |