Software Design Document (SDD)
Assignment #2 Link-Editor Variant for XE
CS530, Spring 2021

Team:
Joshua Boltz, Username: cssc3745 (Project files are Joshua's edoras account)
James Lee, Username: cssc3719
Ayoub Rammo, Username: cssc3733

Overview & Goals:

With this project, we aim to be consistently communicating with each other and progressing further into the project in a timely fashion. We plan on meeting once a week and discussing any major errors that arise as we are coding. With the goal of wanting to learn the ins and outs of this project, we believe that it is a key factor to talk about these errors and fix them as a group to learn through experience.

Overview: For this project we will be building a link-editor variant to convert listing files to object files. It will return object files as well as the ESTAB state table.

Goals:

1. First create a function to turn a line string into groups of data, one string for each column of data (ie. Address, Symbol, Directive, Operand, Object code)
2. Next create pass 1 function to create the ESTAB and Symbol Table to help with processing everything in pass 2
3. Pass 2 will write the header/external define/external reference/text and will call the memory check function to check for memory access errors.
4. Pass 3 will write modification records and will also write the end record
5. Finally, successfully write out each file to disk and write the ESTAB to file as well

Project Description:

For this program we are building a link-editor variant that works for the XE version of the SIC Family that will take listing files and convert them into object files. It will also check all the format 3 and 4 instructions for memory access errors where an instruction might try to access an out of bounds address outside the program. If an error occurs, it will notify the user of the operand that caused the error and terminate, making sure to remove any previous files written before the error. If successful, it will write these object files out to disk as well as write the External Symbol Table (ESTAB) to file as well.

Plan of Action and Milestones:

Phase One: Outline/Brainstorm – (March 5th – March 10th)

- Decide which functions are needed to create this project
- Figure out the timeframe we want to complete each part of the project in
- Figure out function dependencies so we can do these functions first
- Decide what parts of the program are the most difficult and need the most time to complete
- Identify which development environments to use to effectively create this project

Phase Two: Create Pseudocode/Diagram – (March 5th – March 10th)

- Create a high-level picture with each function, who's calling what, and rough pseudocode for each function
- Address the order in which the project functions need to be created (for dependency issues)
- Create a rough outline on the timing for completing each part of the project

Phase Three: Initial Coding – (March 11th – March 17th)

- Create the function convertLine which is a dependency for most other functions since they require each column of data to be separated so they can use this information for processing of pass 1 and 2 as well as memory checking
- Create test files to use for testing purposes later in the program
- Start working on the pass 2 function that creates header records, external define/reference records (if any), text records and the end record

Phase Four: Main coding – (March 18th – March 28th)

- Complete the pass 1 and 2 functions, as well as the Main function and memory checking functions. Also complete the pass 3 function which is for creating modification records. The reason for a pass 3 rather than being part of pass 2 is to make sure each team member can work on their own without worrying about the other team members work

- Add any additional functions that are needed to help make your functions work properly that may not have been thought of during planning.
- Continue communicating to make sure we are creating our project in a way that combines easily when finished

Phase Five: Finalization – (March 29th – April 10th)
- Combine each group members work together to form a completely functioning link-editor variant
- Debug any issues we have when combining our work like making sure we are using correct data structures
- Test our project against known testing files like that on Page 136 of the book to make sure that everything is working properly
- Try to find any flaws in each others work to ensure everything that everything works perfectly and nothing is missing from each others work.

Phase Six: Completion – (April 10th – April 14th)
- Create our README file
- Create our Makefile
- Update our SDD for a final draft
- Final submission April 14th

Requirements:
- Must have a good understanding of the C language as our project will be written in C rather than C++
- Must be able to compile using the GCC compiler, particularly in edoras, to make sure no errors occur on the system it will be tested on
- The operating systems will help if we use MAC OS/Windows 10
- Laptops with enough processing power to take create a somewhat large program
- 4GB of RAM or more is ideal for this project
- Must be able to convert the files quickly and deal with errors that arise while completing
- Must have at least one input argument to create an output object file, otherwise prints error and terminates.
- Checks to make sure that the format 3 and format 4 instructions are making memory references within the scope of the program's memory space, and if not, it will output an error message and terminate the program.
- The program will also print out the ESTAB in a separate file.

System Design/Specification:
- Main Function:
    - This function will check to see if there are 0 arguments, or more. With the case of 0 arguments, the main will print out an error stating we need at least one listing file and then terminate the program. Otherwise, it will call the pass 1 and pass 2 and pass 3 functions and if successful, write the ESTAB file and return a 0 if successful
- Pass 1 Function:
    - This function will fill up the empty symtab and estab created in the main function. The function will then go on to return the filled in symtab and estab so pass 2 and pass 3 can use it.
- Pass 2 Function:
    - This function will create the Header record, Define record, Reference record, and Text records. It will also call the memory check function whenever a format 3 or 4 instruction is seen.
- Pass 3 Function:
    - This function will be creating the modification records for format 3 and 4 instructions. This function will also be creating the end record.
- Memory Check:
    - This function will check to see if the addresses are in bound and valid. This will then return either a 0 or 1 depending on whether the bounds were met or not. It will also return an error message if a memory access error has occurred.

Main
- if no args
  - return error message and return
- create empty symtab and estab
- for (int i = 1; i < argc; i++)
  - call) Pass 1 (symtab, estab)
- for (int i=1; i<argc; i++)
  - call Pass2 (symtab, estab)
- for (int i=1; i<argc; i++)
  - call Pass3 (symtab, estab)
- if (everything succesful)
  - create ESTAB.sf file
- return

Pass 1 (symtab, estab)
- while (! EOF)
  - Store 5 variables [addr, sym, direc, op, objcode]
  - if (direc == Extdef/Extref)
    - Store symbol w/o location
  - if (sym in symtab/estab)
    - add location to symbol
  - if (op == start)
    - add symbol in control section w/ address
- Add control section length to estab
- return (symtab, estab)

Pass 2 (symtab, estab)
- while (! EOF)
  - Store 5 variables
  - if (direc == start)
    - create header record
  - if (direc == Extdef)
    - create defino record
  - if (direc == Extref)
    - create reference record
  - if (direc == End)
    - if (objcode list ! Empty)
      - create finN text record
  - if (objcode exists)
    - call memcheck (objcode, bounds)
    - if (no error)
      - if (objcode list is empty)
        - Add objcode to list w/ location and increase by length of objcode
      - else if (objcode list + # of half-bytes of objcode > 60)
        - create text record
        - empty objcode list and add object code w/ location and increase by length of objcode
      - else
        - Add objcode to list and increase length

memcheck (objcode, bounds)
- check if address is within bounds
- if (yes)
  - return 0
- if (no)
  - error message
  - return 1

Pass3 (symtab, estab)
- while (! EOF)
  - Store 5 variables
  - if (op includes extref)
    - create a Modification record for each extref in op using location/length
    - if (we have a positive/negative relative local symbol)
      - create modification for the control section length
  - if (op is local relative and format 4)
    - create Modification record for the control section length
  - if (op == End)
    - create End record
  - return

Development Environment:

Operating system: Windows 10
Integrated Development Environment: Visual Studio as well as Vi
Compiler: GCC Compiler
Edoras for the actual testing of the code

Run/Test Environment:

Using a ssh terminal or linux terminal, we will test out our code by running sample input SIC/XE assembler listing files through our code to receive desired output. We will also be adding breakpoints throughout the code in order to get a better picture on where errors will lie and how we can go about fixing them. We will be using the GCC compiler to finalize and run our executable object code. We will also test everything on Edoras to make sure everything works on the environment that it is actually going to be run on.