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

Team:
Manuel Loera, cssc3534
Brandon Altamirano Ortega, cssc3740
Shane Moro, cssc3711

Overview & Goals:
Overview: For this Project we will be building a SIC/XE link-editor program that generates executable object file(s) for the XE machines and the ESTAB.

Goals:
1. Open listing file P2sampleAdder.lis and P2sampleWriter.lis
2. Check if memory references are within bounds.
3. Successfully print executable files and create name.st file containing ESTAB.

Project Description:
The purpose of this project is to create a SIC/XE Link-Editor. The program will open SIC/XE listing files and generate executable object file(s) for the XE machine and the ESTAB. SIC/XE assembler listing format files will be read as arguments from the command line. If any memory references are out of bounds the program will print a message regarding the issue. Else the program will create object files and print an ESTAB in a separate file (name.st)

Plan of Action and Milestones:
Phase One: Brainstorm (March 24 - March 31)
- Develop a good understanding of how a link-editor works
- Plan approach to create link-editor
- Discuss periodically goals and tasks to be done
- Question problems or obstacles we might face and do research if necessary
- Our group will set up a github repository to collaborate and work independently on each task

Phase Two: Create Pseudocode- (April 01 - April 06)
- Create Pseudocode for various functions
- Assign Pseudocode to team members to code
- List data necessary
- Comment on functionality of program

Phase Three: Initial Stage Coding- (April 06 - April 13)
- Code source code from Pseudocode
- Combine code created for initial testing of implementation.
- Pinpoint any errors or shortcomings from coding
- Each member will constantly push and commit changes to github branch

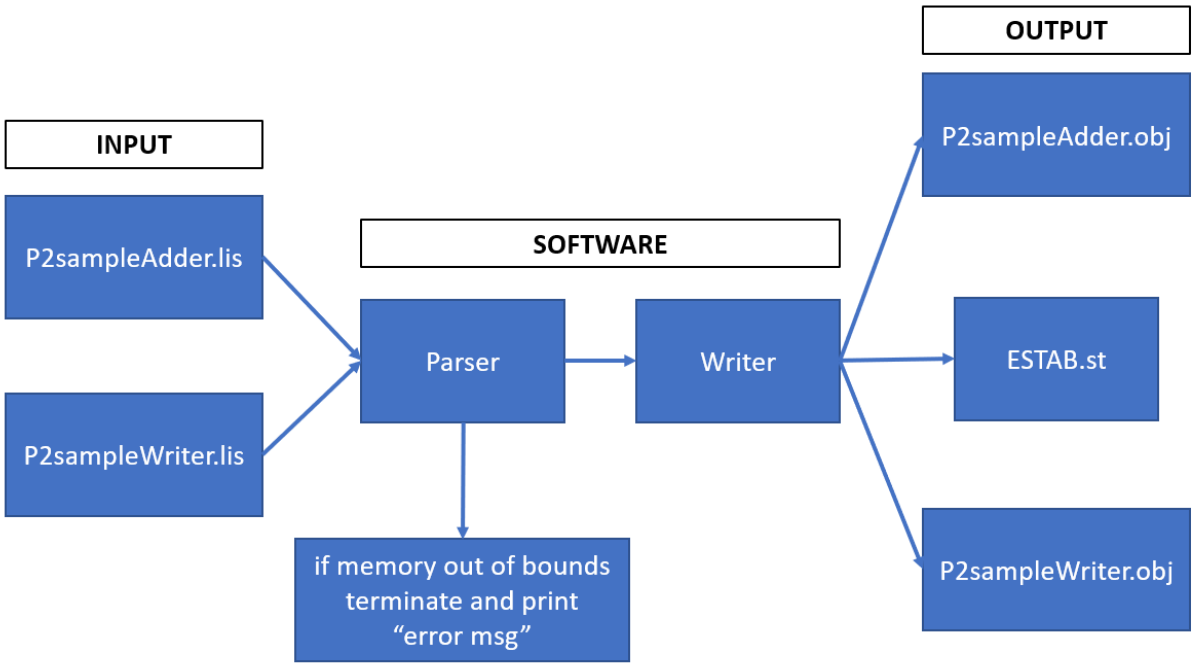
Phase Four: Finalization-(April 13 – April 19)

- Complete all debugging and error finding of the previous phase
- Finishing adding all requirements and features for the program
- Merge github branches and program
- Final submission April 19

Requirements:

- Knowledge of C/C++ Programming Language.
- C++ Programming Language and GCC Compiler Installed.
- MAC or Windows 10 Operating Systems.
- Laptop with Quad Core-core processor recommended.
- 8GB or RAM recommended.

System Design/Specification:



Software Control Flow:

1. **SICXE_Parser::Read()**
    a. Read through .lis line by line. The parser builds individual tokens. Those tokens are then placed into SICXE instructions based on their type. For instance if a token is an address token it would be added into instructions[i].address. After every line the instructions[i] are incremented and the program moves onto parsing the next line in the file.

2. **SICXE_Parser::WriteObjFile()**
    a. After SICXE_Parser::Read() has completed. The program begins to write the object file. This section takes in the instructions created in the Read section and loops through the instructions to get the relevant information to print to that line. This section goes through every instruction line and prints until it reaches the end. This section also separates the writing into different sections. The lines containing info such as NAMEand EXTREF. Then the program builds the text section grabbing object codes from the instructions list. Finally mod records are created. After word all sections are combined into one obj file.

3. **SICXE_Parser::WriteSymTabFile()**
    a. Finally the estab is created by going through all the instructions for the information to build the current line that is being written. Running totals are created for addresses for new program starts. The program also references memory location calls in the instructions when necessary.

Design Reasoning:

Our group decided to build the program this way in order to make it as human readable as possible. This is exactly the steps that we would do if we were to do it by hand. As well as the logic choices made on how we chose what information goes where in the output.

Development Environment:

- Operating System: MAC and Windows OS System.
- IDE: VS Code
- Compiler: GCC Compiler
- EDORAS from SDSU

Run/Test Environment:

Using samples provided on Canvas, we will run and test the environment first, we will run and test the environment on the G++ compiler. We will do our own test cases to ensure the program will successfully run. Finally, we will compile and test our program on the Edoras server to ensure there are no errors.

Code and Test Plan:

1. Code Review:
   - Group members will review the source code to ensure no syntax errors are present.
   - Verify that all coding requirements and guidelines are met.
   - No unused variables or syntax is left.
2. Unit Testing:
   - Each member will test their task individually to make sure each works with no errors.
   - Each method must follow required guidelines stated in the program.
3. Integration Testing:
   - Members of our group will merge the github branches together and combine the code.
   -We will test the program as a whole to analyze the flow of data and final outcome.
   -This is done to ensure that the entire program works as intended and there are no errors.