# CSI 402 – Program III

## Administrative Information:

- **Deadline:** 11 PM, Monday, April 3, 2006.
- Two parts, but just one `makefile`.
- Two or more C source files for each part.
- README file (by 10 PM, Mar. 21, 2006):

  `~csi402/public/prog3/prog3.README`

## Part (a): (Weightage: 60%)

- **Goal:** Linker for TMIPS programs.
- **Command line:**

  % p3a *configfile*
  % p3a *configfile* -o *loadmodule*

## Example of configuration file:

```
3   200
part1.obj
part2.obj
part3.obj
```



## TMIPS assembly source file:

```
      .extdef   x1
      .extref   y1
      .text
m1:   lwa       $5,x1
      swa       $5,y1
      hlt
      .data
x1:   .resw     1
```

## Corresponding object module:

```
   H    m1
   D    x1      3
   T    1745158147
   T    1812267008
   T    0
   T    0
   M    0       m1
   M    1       y1
```

**Important note:** The load module is a *binary* file.

## Format of load module:


## Notes:

- `Z` is a 2-byte unsigned integer which gives the size of the load module.
- `S` is a 2-byte unsigned integer which gives the starting address of the load module.
- `M1` represents the object code for the first module, ..., `Mk` represents the object code for the last module.

**Suggestion:** Use the outline given in Handout 7.2 from the lecture notes to implement Part (a).



## Errors to be detected:

- Not enough room in memory for load module.
- Address resolution leads to an address larger than 65535.
- Undefined or multiply defined symbol.
- Usual command line errors.
- One of the of the object module files can't be opened.

## Additional Notes:

- Assume that the file names mentioned in the configuration file are of length at most 15 characters.
- Assume that the External Symbol Table (EST) to be constructed by the linker has at most 50 entries.
- You may use any data structure to implement tables such as ERT, EDT and EST.
- The H-record does *not* contain the size of the corresponding object module; the program must compute the size.

- Each module name is an external symbol; if two object modules have the same name, your program produce a "multiply defined symbol" error and stop.

**Part (b):** (Weightage: 40%)

**Goal:** To implement a simple version of the Unix `tar` command.

**Command line:**

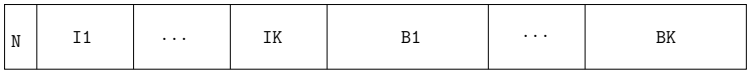
```
% p3b -c archive infile1 ... infilek
% p3b -x archive
```

**Notes:**

- No. of input files may range from 1 to 255.
- Input files may be text files or binary files.
- The archive is a *binary* file.
- The size of each input file is at most $2^{32} - 1$.



**Format of archive:**




**Notes:**

- `N` represents one byte which gives the number of files in the archive.
- `I1` represents information about the first file, ..., `Ik` represents information about the last file.
- `B1` represents the bytes of the first file, ..., `Bk` represents the bytes of the last file.

**Information about each file:**


- `L` is one byte which gives the number of characters in the file name. (This does *not* include the byte for the `'\0'` character.)
- `S` is a string representing the file name. (The length of `S` was specified in `L`.)

- Z represents an unsigned 4-byte integer which gives the size of the file.

**Errors to be detected:**

- Usual command line errors.

**Suggestion:** Study the following material from the text by Haviland, Gray and Salama.

- Chapter 2 (functions such as `open`, `close`, `read`, `write` and `lseek`).
- Pages 53–57 of Section 3.3 (`stat` structure and functions such as `stat` and `fstat`).

**Other notes about Part (b):**

- Be careful about the number of files that are open simultaneously.
- If you create a file in your program and want to delete it before stopping your program, use the `remove` (or `unlink`) system call (see Chapter 2 of Haviland et al.).