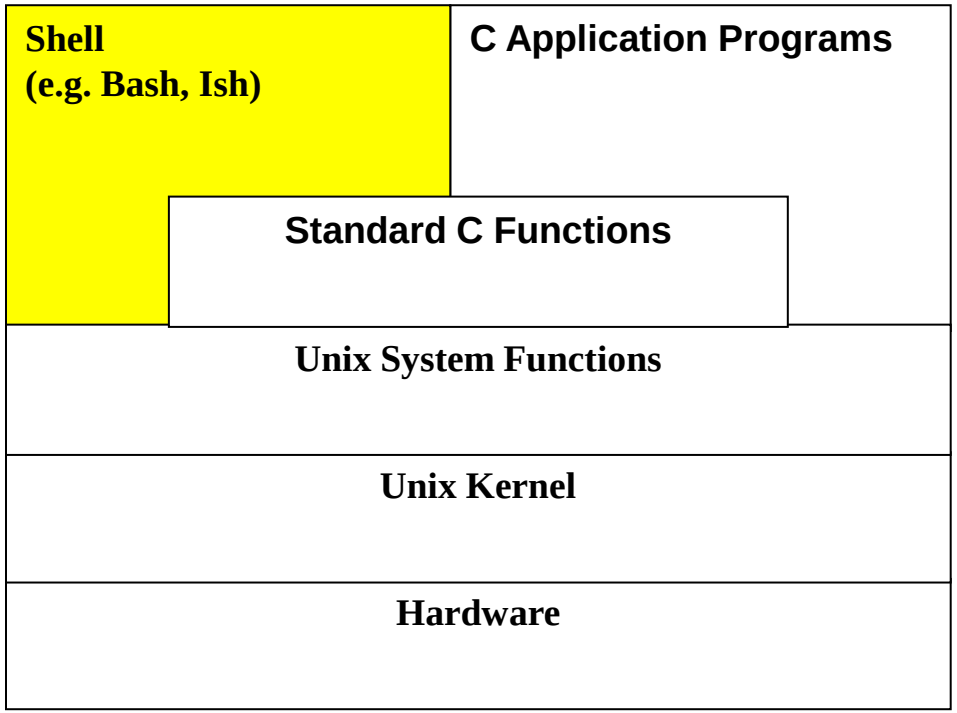
Ish Basic Design

Ish – A Primitive Shell

- Illustration



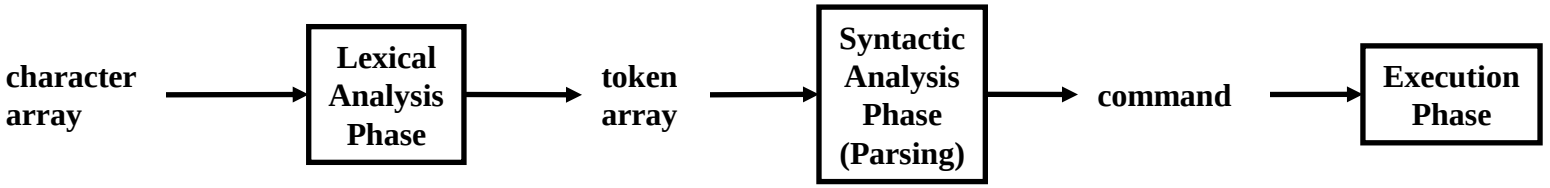
Initialization

- .ishrc
  - in your home directory
  - ish reads and interprets it
- Another usage
  - automatic testing
  - put test commands in the file
- Let's look at some examples

Demo

- Demo

Ish Design



`echo one two three > myfile`	`echo`
	`one`
	`two`
	`three`
	`>`
	`myfile`
		`Command:`
		`  Name:  echo`
		`  Arg 0:  one`
		`  Arg 1:  two`
		`  Arg 2:  three`
		`  Stdinredirect:  NULL`
		`  Stdoutredirect:  myfile`

`echo one "two three" > myfile`	`echo`
	`one`
	`two three`
	`>`
	`myfile`
		`Command:`
		`  Name:  echo`
		`  Arg 0:  one`
		`  Arg 1:  two three`
		`  Stdinredirect:  NULL`
		`  Stdoutredirect:  myfile`

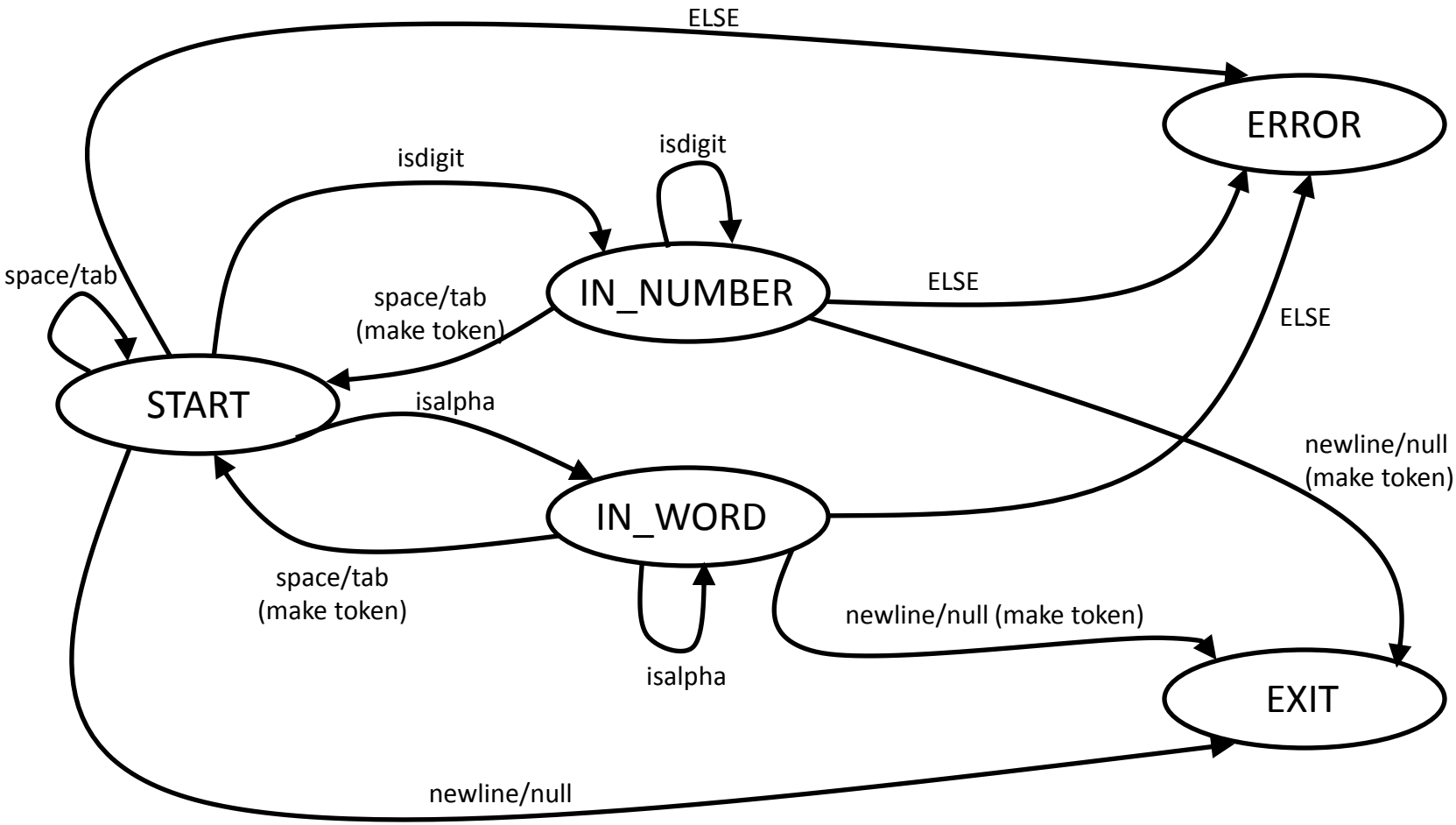
Ish Development Stages

- Stage 0: Preliminaries
  - Learn the overall structures
  - Identify key modules!
    - nouns => ADT, or Abstract Objects
    - verbs => functions

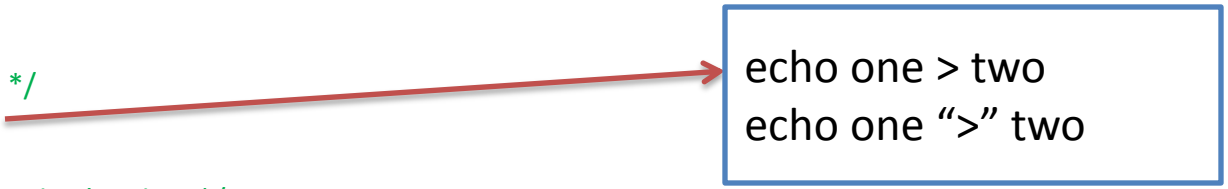
Ish Development Stages

- Stage 1: Lexical Analysis
  - Create a lexical analyzer
    - Input: character array
    - Output: token array
  - Recommendation #1
    - Design DFA (As with decomment program)
    - Test DFA enough before you move to the next stage
  - Recommendation #2
    - The lexical analyzer reads from a char array rather than a file
  - Use DFA implementation in your handouts

A Sample DFA



```
/*---------------------------------------------------------------------*/
/* dfa.c                                                               */
/* Author: Bob Dondero                                                 */
/* Illustrate lexical analysis using a deterministic finite state      */
/* automaton (DFA)                                                     */
/*---------------------------------------------------------------------*/

#include "dynarray.h"
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

enum {MAX_LINE_SIZE = 1024};

enum {FALSE, TRUE};

enum TokenType {TOKEN_NUMBER, TOKEN_WORD};

enum LexState {STATE_START, STATE_IN_NUMBER, STATE_IN_WORD, STATE_ERROR, STATE_EXIT};

/* A Token is either a number or a word, expressed as a string. */
struct Token
{
  /* The type of the token. */
  enum TokenType eType;

  /* The string which is the token's value. */
  char *pcValue;
};
```



```
int main(void)
/* Read a line from stdin, and write to stdout each number and word that it contains.  Repeat until EOF.  Return 0. */
{
  char acLine[MAX_LINE_SIZE];
  DynArray_T oTokens;
  int iSuccessful;

  printf("------------------------------------\n");
  while (fgets(acLine, MAX_LINE_SIZE, stdin) != NULL)
  {
    oTokens = DynArray_new(0);

    iSuccessful = lexLine(acLine, oTokens);
    if (iSuccessful)
    {
       printf("Numbers:  ");
       DynArray_map(oTokens, printNumberToken, NULL);
       printf("\n");

       printf("Words:  ");
       DynArray_map(oTokens, printWordToken, NULL);
       printf("\n");
    }
    else
      printf("Invalid line\n");
    printf("------------------------------------\n");

    DynArray_map(oTokens, freeToken, NULL);
    DynArray_free(oTokens);
  }

  return 0;
}
```

Demo

- Demo

Ish Development Stages

- Syntactic Analysis (Parsing)
  - Create a parser
  - Input: token array
  - Output: command
- Use DFA – A "smart" version
- Do we need a number token for ish?

Recommendations

- Design modules carefully
  - Functions, ADT, Abstract Objects, …
  - Divide modules development amongst each partner (Plan & Pseudo-code)
- Consider various cases
  - The users can make any mistakes
  - Your shell should not crash!

Recommendations

- Detects errors early
  - Lexical analyzer should detect lexical errors

    ex)

    % “ls

    % ls”

    % ls”-al
  - Syntactic Analyzer should detect syntactic errors

    % < one

    % echo one >

    % echo one > f1 > f2