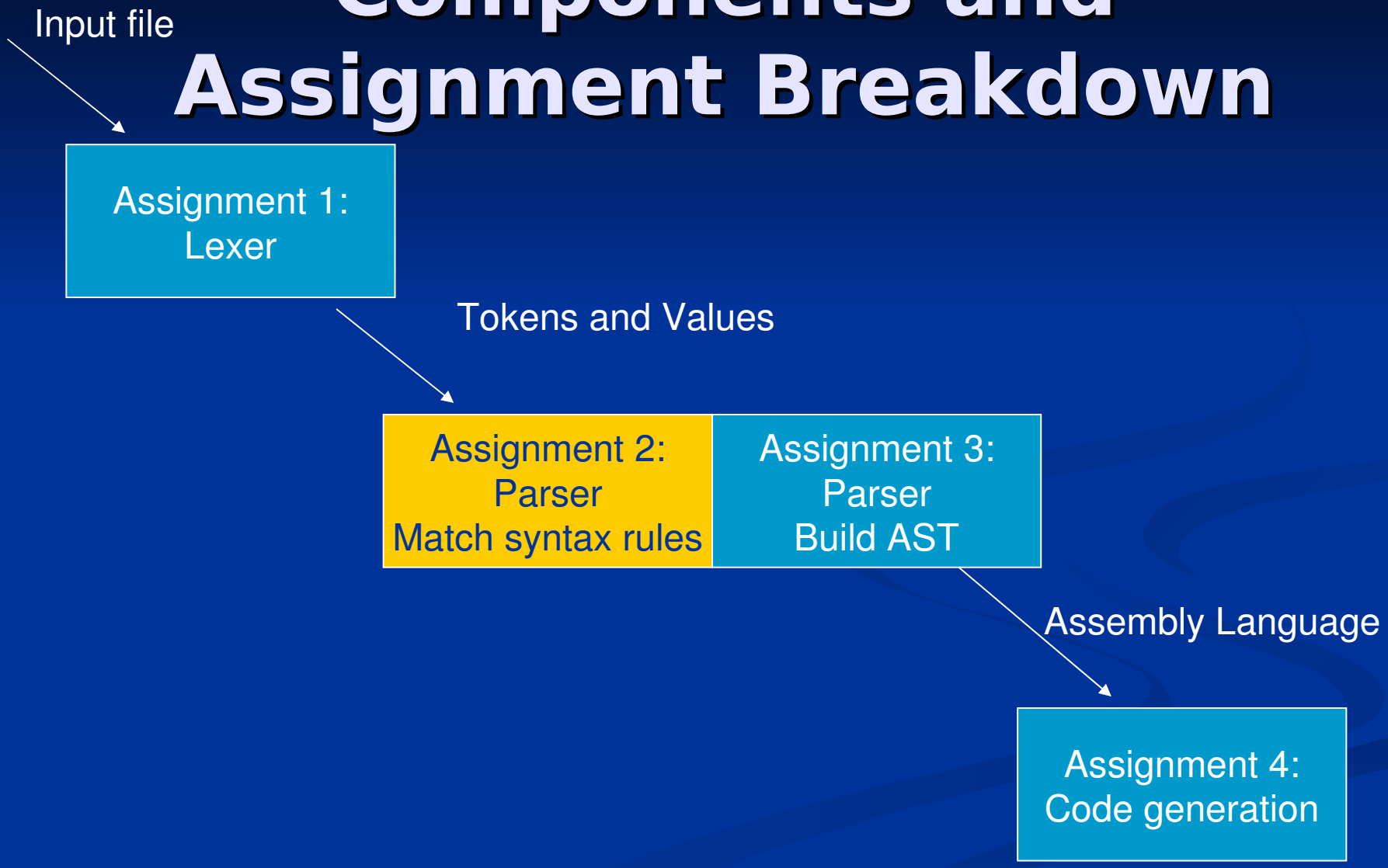
Bison Tutorial

Plus a Quick Look at A2

Recall: Compiler Components and Assignment Breakdown



Note

- Not everything discussed today will be required for assignment #2

Description of Bison

- LALR(1) parser generator under the GNU license

Layout of Bison File (Look familiar?)

%{
Prologue
%}
Bison declarations
%%
Grammar Rules
%%
Epilogue

Calculator Example

- Recall: We implemented a simple number lexer in flex
- Now we can implement a parser which will take actions on this file

Bison Prologue

- Between “%{” and “%}”
- Content between those is copied verbatim into output file

```
%{
  #define YYSTYPE double
  #include <math.h>
  #include <stdio.h>
  int yylex(void);
  void yyerror(char const*);
%}
```

Bison Declarations

- Specify tokens and precedence
- Lower the rule, the higher the precendence
- Left associatitivity vs. right associativity

```
%token NUM
%left ‘-’ ‘+’
%left ‘*’ ‘/’
%left NEG
```

Grammar Rules

- Grammar rules have the form

  result: components

  e.g.: exp: exp ‘+’ exp

- Can be followed by braces to indicates actions to take

```
e.g.: exp:      exp ‘+’ exp {
        printf(“summing two numbers\n”);
      }
```

Grammar Rules

- Can return a value

```
exp:NUM                    { $$ = $1;
    }
    | exp ‘+’ exp          { $$ = $1 + $3;
    }
```

- $$ is the result
- $n is the n-th term in the syntax rule

Grammar Rules for Calculator (minus, division, exp omitted)

```
input: /* empty */
       | input line
;
line: '\n'
       | exp '\n'   { printf("\t %.10g\n", $1); }
;
exp: NUM                { $$ = $1;                 }
       | exp '-' exp          { $$ = $1 - $3;}
       | exp '*' exp          { $$ = $1 * $3;
       }
       | '-' exp %prec NEG  { $$ = -$2      }
```

Precedence

- %prec indicates that the unary minus has the same precedence as NEG
  - Or second highest (recall our declarations)

Epilogue

- Place helper functions here or the main() function

```
int main (void)
{
  return yyparse();
}
```

Compilation

- > bison parser.y
  - Outputs to parser.tab.c
- > bison parser.y –o othername.c
  - Outputs to othername.c
- > bison parser.y –d
  - -d flag tells bison to also create a header file with macro definitions
  - Ouputs to parser.tab.c and parser.tab.h

See Bison Run…

Data Types

- Sometimes the result (i.e. $$) is a float
  - E.g. the calculator
- Sometimes the result can be different values
  - E.g.
  - str: str ‘+’ str { $$ = plusStr($1, $3); };
  - num: num ‘+’ num {$$ = plusNum($1,$3); }

Data Types

- To accommodate this
- In the declarations

```
%union {
  int val;
  char* str;
}
```

  - States that $$ can be either an int or a char *

Data Types

- Declare the type of each token

  %token <val> NUM

  - NUM token is the same type as val (which is int)

  %token <str> STRING

  - STRING token is the same type as str (which is char*)

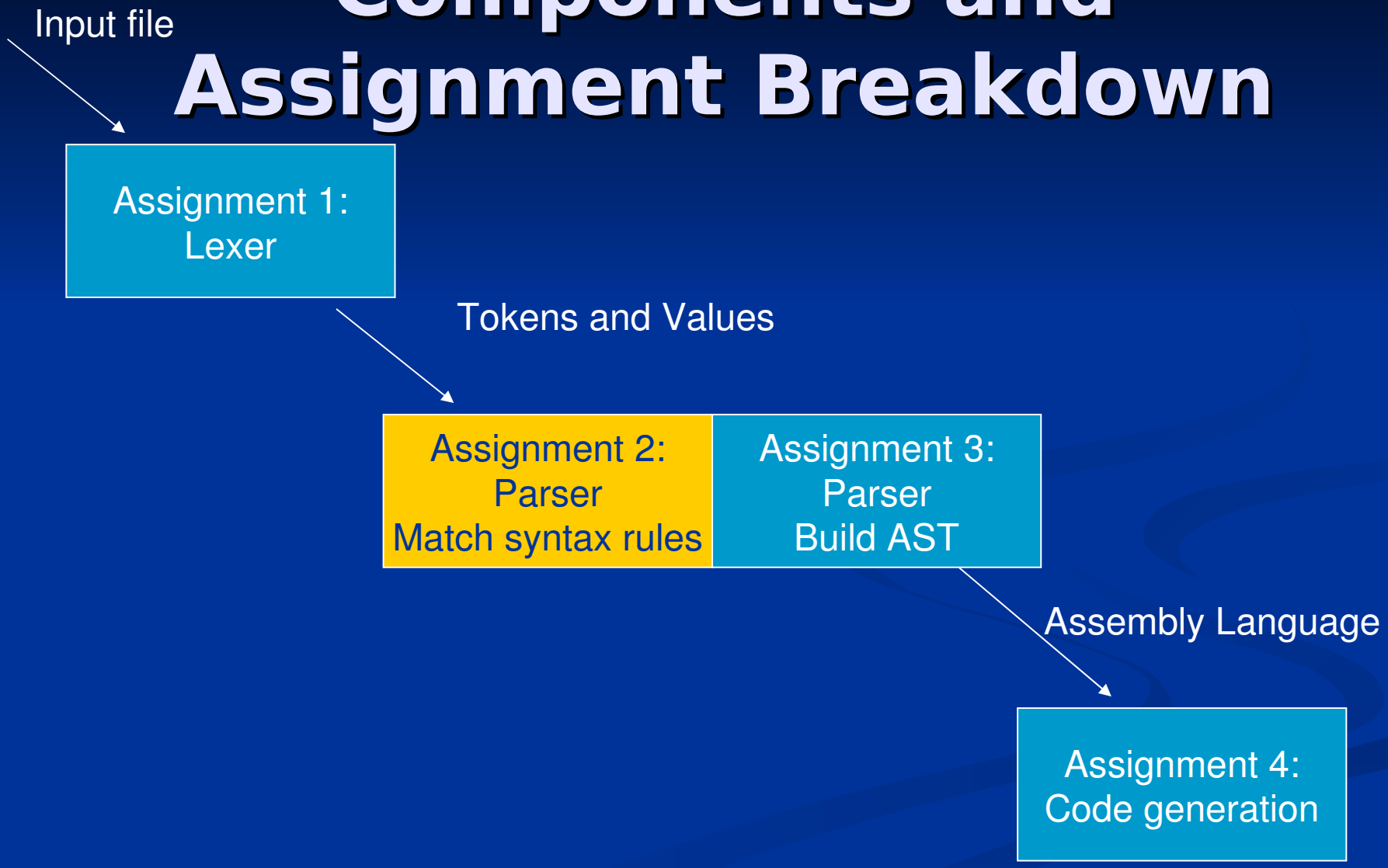
Error Handling

- Whenever yyparse() detect an error yyerror() is called
- Example of yyerror()

```
void yyerror (char *const s) {
  fprintf(stderr, “ERROR on line %d\n”,
  lineno);
}
```

On to Assignment 2

Recall: Compiler Components and Assignment Breakdown



Assignment Two Options

- Use Bison to create a parser for LR(1) grammar
- Implement a parser in C/C++ with for an LL(k) grammar

Example Run…

Expectations

- Documentation
  - Discuss: Design, implementation, testing
- Testing
  - Cover: Normal, tricky and error cases
  - The more rigorous the better
- Implementation
  - Create the parser as described
  - Every failed test case cause marks to be deducted

Advice

- Use a macro so that parsing information is only output when –Tp is set

Submission

- Directory for starter2
  - doc/
    - Documentation notes
  - cases/
    - Test cases
  - other files: scanner.l, parser.y, etc.

- tar –zcvf starter2.tar.gz starter2
- submitcsc467f 1 starter2.tar.gz