Semantics: so what does this all mean?

Now that we are able to parse our files, how do we get some meaning from that into something our program can use?

The traditional answers, suitable to compilation of programs, generally fall in the line of building a tree. However, for our purposes, it might make more sense to simply use a list of structures.

Moving data into a C structure

Let's go back and look at our example from last lecture. In that example, we had a configuration file along the lines of:

```
replication "ab-to-xy"
{
   # some sort of general information
   interval = 5 ;
   do_logging = 1 ;        # 1 = yes, 0 = no
   log_identifier = "ab to xy transfer" ;

   # now information about our source of data
   source "ab"
   {
     type = postgresql ;
     authentication_needed = 1 ;
     authentication_type = password ;
     username = username ;
     password = "password" ;
     database = "databasename" ;
     all_tables = 0 ;
     tables = [ table1 table2 table3 table4 ] ;
   }

   # and information about our target
   target "xy"
   {
     type = sqlite ;
     authentication_needed = 0 ;
     database = "databasename" ;
   }
}



replication "cd-to-wx"
```

```
{
   # some sort of general information
   interval = 10 ;
   do_logging = 1 ;        # 1 = yes, 0 = no
   log_identifier = "cd to wx transfer" ;

   # now information about our source of data
   source "cd"
   {
     type = mysql ;
     authentication_needed = 1 ;
     authentication_type = password ;
     username = username ;
     password = "password" ;
     database = "databasename" ;
     all_tables = 0 ;
     tables = [ table1 table2 table3 table4 ] ;
   }

   # and information about our target
   target "wx"
   {
     type = mysql ;
     authentication_needed = 0 ;
```

```
      database = "databasename" ;
    }
}
```

Adding semantics

The place to add semantics is in the bison grammar. We can refer to the value of a terminal or a non-terminal with a simple `$` syntax. For our own value, we use `$$`; for the value of a component of a production rule, we use the syntax `$1, $2, ....`

Putting it all together

Let’s propose a list representation for our stanza data.

```
typedef struct replication *replication;
struct replication
{
  // our list
  replication    next;

  // our data

  // structure-wide data
  char          *replication_name;
  int            interval;
  int            do_logging;
```

```
char          *log_identifier;

// source data
struct source
{
  char        *source_name;
  char        *type;
  int          authentication_needed;
  char        *authentication_type;
  char        *username;
  char        *password;
  char        *database;
  int          all_tables;
  char        *tables;
} source;

// target data
struct target
{
  char        *target_name;
  char        *type;
  int          authentication_needed;
  char        *authentication_type;
  char        *username;
```

```
    char          *password;
    char          *database;
  } target;

};
```

Tying the two together

In order to get the data into our structure, we need to decorate our grammar with some rules telling it how to do this.

There are two types of activities that we should delineate. The first that are simply manipulating our overall list of replications. Those can all go into the `replication_declaration` rule, which is called whenever we go into a state where we are declaring

a new replication.

Simple code to manipulate our list of replications

```
tmp = current;
current = (replication) calloc(1,sizeof(struct replication));
if(replication_list == NULL)
  {
    replication_list = current;
  }
else
  {
    tmp->next = current;
  }
current->next = NULL;
```

Now putting found data into our list

The easiest place to see what we are doing is perhaps where we are declaring the name for the overall replication, right there in the same replication rule:

```
REPLICATION
{yy_private_error_guess = "Expecting string";}
STRING
{yy_private_error_guess = "Expecting '{'";
 current->replication_name = strdup($4);}
```

Other places

While it is easiest to see the rules in action at the points where we name the overall replication and its attendant source and target portions, the most important areas are where we are setting the many different values in the structure. This is done for all three sub-sections in the common production rule `simple_declaration`.

```
simple_declaration :
{yy_private_error_guess = "Expecting unquoted name\n";}
```

```
 NAME
{yy_private_error_guess = "Expecting ’=’\n";}
EQUAL
{yy_private_error_guess = "Expecting value\n";}
value
{yy_private_error_guess = "Expecting semicolon\n";
my_assign(current,state,$2,$6);}
SEMICOLON
;
```

The `state` variable

We added an explicit state variable so that we can distinguish if our `simple_declaration` is in the global, source, or target area. We could as easily have simply split the rules so that were distinguishable versions of `simple_declaration`, or we could of course have created a more complex structure that let us retain the structure of the input in the data (such as a tree.) But this turns out to be something of a win since we also have basically the same variables in both the source and target areas, so we can reasonably easily share code when handling both of those cases.

my_assign();

```
void my_assign(replication current, int state, char *name, char *value)
{

  if(strcmp(name,"interval") == 0)
    {
      if(state != YY_PRIVATE_GLOBAL)
        {
          printf("Complaining about 'interval' being set in non-global co
          return;
        }

      current->interval = atoi(value);
      return;
    }
```

```
if(strcmp(name,"do_logging") == 0)
  {
    if(state != YY_PRIVATE_GLOBAL)
      {
        printf("Complaining about ’do_logging’ being set in non-global
        return;
      }

    current->do_logging = atoi(value);
    return;
  }


if(strcmp(name,"log_identifier") == 0)
  {
    if(state != YY_PRIVATE_GLOBAL)
      {
        printf("Complaining about ’log_identifier’ being set in non-glo
        return;
      }

    current->log_identifier = strdup(value);
    return;
  }

if(strcmp(name,"type") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
      {
        current->source.type = strdup(value);
        return;
      }
    if(state == YY_PRIVATE_TARGET)
      {
        current->target.type = strdup(value);
        return;
      }

    printf("Complaining about ’type’ being set in global context (shoul
    return;
  }

if(strcmp(name,"type") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
      {
        current->source.type = strdup(value);
        return;
      }
    if(state == YY_PRIVATE_TARGET)
      {
        current->target.type = strdup(value);
        return;
      }

    printf("Complaining about ’type’ being set in global context (shoul
    return;
  }
```

```
if(strcmp(name,"authentication_needed") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
      {
        current->source.authentication_needed = atoi(value);
        return;
      }
    if(state == YY_PRIVATE_TARGET)
      {
        current->target.authentication_needed = atoi(value);
        return;
      }

    printf("Complaining about 'authentication_needed' being set in glob
    return;
  }

if(strcmp(name,"authentication_type") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
```

```
        {
          current->source.authentication_type = strdup(value);
          return;
        }
      if(state == YY_PRIVATE_TARGET)
        {
          current->target.authentication_type = strdup(value);
          return;
        }

      printf("Complaining about 'authentication_type' being set in global
      return;
    }


  if(strcmp(name,"username") == 0)
    {
      if(state == YY_PRIVATE_SOURCE)
        {
          current->source.username = strdup(value);
          return;
        }
      if(state == YY_PRIVATE_TARGET)
        {
          current->target.username = strdup(value);
          return;
        }

      printf("Complaining about ’username’ being set in global context (s
      return;
    }



  if(strcmp(name,"password") == 0)
    {
      if(state == YY_PRIVATE_SOURCE)
        {
          current->source.password = strdup(value);
          return;
        }
      if(state == YY_PRIVATE_TARGET)
        {
          current->target.password = strdup(value);
          return;
        }

      printf("Complaining about ’password’ being set in global context (s
      return;
    }



  if(strcmp(name,"database") == 0)
    {
      if(state == YY_PRIVATE_SOURCE)
        {
          current->source.database = strdup(value);
          return;
        }
      if(state == YY_PRIVATE_TARGET)
        {
          current->target.database = strdup(value);
          return;
        }

      printf("Complaining about 'database' being set in global context (s
      return;
    }

  if(strcmp(name,"all_tables") == 0)
    {
      if(state == YY_PRIVATE_SOURCE)
        {
          current->source.all_tables = atoi(value);
          return;
        }

      printf("Complaining about 'all_tables' flag out of context (should
      return;
    }

  if(strcmp(name,"tables") == 0)
    {
      if(state == YY_PRIVATE_SOURCE)
        {
          current->source.tables = strdup(value);
          return;
        }

      printf("Complaining about ’tables’ list out of context (should only
      return;
    }


  printf("Didn’t recognize identifier ’%s’.\n",name);
  return;
}
```

Putting it all together

```
%{

#define YYDEBUG 1

yydebug = 0;

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sys/types.h>
#include <linux/unistd.h>

void my_assign();

typedef struct replication *replication;
struct replication
```

```
{
  // our list
  replication    next;

  // our data

  // structure-wide data
  char          *replication_name;
  int            interval;
  int            do_logging;
  char          *log_identifier;

  // source data
  struct source
  {
    char        *source_name;
    char        *type;
    int          authentication_needed;
    char        *authentication_type;
    char        *username;
    char        *password;
    char        *database;
    int          all_tables;
    char        *tables;
  } source;

  // target data
  struct target
  {
    char        *target_name;
    char        *type;
    int          authentication_needed;
    char        *authentication_type;
    char        *username;
    char        *password;
    char        *database;
  } target;

};

replication tmp;
replication current = NULL;
replication replication_list = NULL;

#define   YY_PRIVATE_GLOBAL   1
#define   YY_PRIVATE_SOURCE   2
#define   YY_PRIVATE_TARGET   3
int state = 0; // 1 = global, 2 = source, 3 = target;
```

```
  char *yy_private_error_guess = "";

%}


%token NAME
%token STRING
%token NUMBER
%token EQUAL
%token SEMICOLON
%token REPLICATION
%token LEFTBRACE
%token RIGHTBRACE
%token SOURCE
%token TARGET
%token RIGHTBRACKET
%token COMMA
%token LEFTBRACKET


%%
```

```
replication_declarations : |
                           replication_declarations replication_declaration
                           ;

replication_declaration :
                          {yy_private_error_guess = "Expecting keyword 'replication'";
                           tmp = current;
                           current = (replication) calloc(1,sizeof(struct replication))
                           if(replication_list == NULL)
                             {
                               replication_list = current;
                             }
                           else
                             {
                               tmp->next = current;
                             }
                           current->next = NULL;
                          }
                          REPLICATION
                          {yy_private_error_guess = "Expecting string";}
                          STRING
                          {yy_private_error_guess = "Expecting '{'";
                           current->replication_name = strdup($4);}
                          LEFTBRACE
```

```
                              {yy_private_error_guess = "Expecting variable declarations";
                               state = YY_PRIVATE_GLOBAL;}
                              variable_declarations
                              {yy_private_error_guess = "Expecting source declaration";
                               state = YY_PRIVATE_SOURCE;}
                              source_declaration
                              {yy_private_error_guess = "Expecting target declaration";
                                state = YY_PRIVATE_TARGET;}
                              target_declaration
                              {yy_private_error_guess = "Expecting '}'";}
                              RIGHTBRACE
                              ;

variable_declarations : | variable_declarations simple_declaration
                        ;

source_declaration :
                     {yy_private_error_guess = "Expecting keyword 'source'";}
                     SOURCE
                     {yy_private_error_guess = "Expecting keyword quoted string";}
                     STRING
                     {yy_private_error_guess = "Expecting '{'";
                      current->source.source_name = strdup($4);}
                     LEFTBRACE
```

```
                    {yy_private_error_guess = "Expecting variable declarations";}
                    variable_declarations
                    {yy_private_error_guess = "Expecting '{'";}
                    RIGHTBRACE
                    ;

target_declaration :
                    {yy_private_error_guess = "Expecting keyword 'target'";}
                    TARGET
                    {yy_private_error_guess = "Expecting keyword quoted string";}
                    STRING
                    {yy_private_error_guess = "Expecting '{'";
                     current->target.target_name = strdup($4);}
                    LEFTBRACE
                    {yy_private_error_guess = "Expecting variable declarations";}
                    variable_declarations
                    {yy_private_error_guess = "Expecting '{'";}
                    RIGHTBRACE
                    ;

simple_declaration :
                    {yy_private_error_guess = "Expecting unquoted name\n";}
                    NAME
                    {yy_private_error_guess = "Expecting '='\n";}
```

```
                     EQUAL
                     {yy_private_error_guess = "Expecting value\n";}
                     value
                     {yy_private_error_guess = "Expecting semicolon\n";
                      my_assign(current,state,$2,$6);}
                     SEMICOLON
                     ;

value : STRING | NAME | NUMBER | list
        ;

list : LEFTBRACKET elements RIGHTBRACKET
       ;

elements : element additional ;

additional : | element additional ;

element : STRING | NAME ;

%%

char *configuration_file = "replication.conf";
int main(int argc, char **argv)
```

```
{
  printf("Found %d arguments...\n",argc);

  // parse config file
  char opt;
  extern char *optarg;

  while((opt = getopt(argc,argv,"c:C:")) != -1)
    {
      switch(opt)
      {
      case 'c':
      case 'C':
        configuration_file = optarg;
        break;
      }
    }

  // read in configuration
  FILE *f = fopen(configuration_file,"r");
  if(f)
    {
      yyrestart(f);
      yyparse();
      printf("Parsed!\n");

      replication r = replication_list;
      int count = 0;
      while(r)
        {
          printf("Replication list item #%d\n",count);
          my_print(r);
          count++;
          r = r->next;
        }
    }
  else
    {
      printf("Couldn't open %s!\n",configuration_file);
      exit(1);
    }

}

void my_assign(replication current, int state, char *name, char *value)
{

  if(strcmp(name,"interval") == 0)
```

```
    {
      if(state != YY_PRIVATE_GLOBAL)
        {
          printf("Complaining about ’interval’ being set in non-global context\n");
          return;
        }

      current->interval = atoi(value);
      return;
    }

  if(strcmp(name,"do_logging") == 0)
    {
      if(state != YY_PRIVATE_GLOBAL)
        {
          printf("Complaining about ’do_logging’ being set in non-global context\n");
          return;
        }

      current->do_logging = atoi(value);
      return;
    }
```

```
if(strcmp(name,"log_identifier") == 0)
  {
    if(state != YY_PRIVATE_GLOBAL)
      {
        printf("Complaining about ’log_identifier’ being set in non-global context\n")
        return;
      }

    current->log_identifier = strdup(value);
    return;
  }

if(strcmp(name,"type") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
      {
        current->source.type = strdup(value);
        return;
      }
    if(state == YY_PRIVATE_TARGET)
      {
        current->target.type = strdup(value);
        return;
      }
    printf("Complaining about ’type’ being set in global context (should be either in
    return;
  }

if(strcmp(name,"type") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
      {
        current->source.type = strdup(value);
        return;
      }
    if(state == YY_PRIVATE_TARGET)
      {
        current->target.type = strdup(value);
        return;
      }

    printf("Complaining about ’type’ being set in global context (should be either in
    return;
  }

if(strcmp(name,"authentication_needed") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
      {
        current->source.authentication_needed = atoi(value);
        return;
      }
    if(state == YY_PRIVATE_TARGET)
      {
        current->target.authentication_needed = atoi(value);
        return;
      }

    printf("Complaining about 'authentication_needed' being set in global context (sho
    return;
  }

if(strcmp(name,"authentication_type") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
      {
        current->source.authentication_type = strdup(value);
        return;
      }
    if(state == YY_PRIVATE_TARGET)
      {
        current->target.authentication_type = strdup(value);
        return;
      }

    printf("Complaining about 'authentication_type' being set in global context (shoul
    return;
  }


if(strcmp(name,"username") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
      {
        current->source.username = strdup(value);
        return;
      }
    if(state == YY_PRIVATE_TARGET)
      {
        current->target.username = strdup(value);
        return;
      }

    printf("Complaining about 'username' being set in global context (should be either
    return;
  }



if(strcmp(name,"password") == 0)
  {
    if(state == YY_PRIVATE_SOURCE)
      {
        current->source.password = strdup(value);
        return;
      }
    if(state == YY_PRIVATE_TARGET)
      {
        current->target.password = strdup(value);
        return;
      }

    printf("Complaining about 'password' being set in global context (should be eithe
    return;
  }



if(strcmp(name,"database") == 0)
```

```
      {
        if(state == YY_PRIVATE_SOURCE)
          {
            current->source.database = strdup(value);
            return;
          }
        if(state == YY_PRIVATE_TARGET)
          {
            current->target.database = strdup(value);
            return;
          }

        printf("Complaining about 'database' being set in global context (should be eithe
        return;
      }

    if(strcmp(name,"all_tables") == 0)
      {
        if(state == YY_PRIVATE_SOURCE)
          {
            current->source.all_tables = atoi(value);
            return;
          }
        printf("Complaining about 'all_tables' flag out of context (should only in be sour
        return;
      }

    if(strcmp(name,"tables") == 0)
      {
        if(state == YY_PRIVATE_SOURCE)
          {
            current->source.tables = strdup(value);
            return;
          }

        printf("Complaining about 'tables' list out of context (should only in be source c
        return;
      }


    printf("Didn't recognize identifier '%s'.\n",name);
    return;
  }

  int my_print(replication r)
  {
    printf("Replication '%s' (0x%x): \n",r->replication_name,r);
    printf(" { \n");
    printf("\t\t interval       = %d\n",r->interval);
    printf("\t\t do_logging     = %d\n",r->do_logging);
    if(r->log_identifier)
      printf("\t\t log_identifier = '%s'\n",r->log_identifier);
    printf("\t\t Source '%s': \n",r->source.source_name);
    printf("\t\t  { \n");
    if(r->source.type)
      printf("\t\t\t\t type = '%s'\n",r->source.type);
    printf("\t\t\t\t authentication_needed = %d\n",r->source.authentication_needed);
    if(r->source.authentication_type)
      printf("\t\t\t\t authentication_type = '%s'\n",r->source.authentication_type);
    if(r->source.username)
      printf("\t\t\t\t username = '%s'\n",r->source.username);
    if(r->source.password)
      printf("\t\t\t\t password = '%s'\n",r->source.password);
    if(r->source.database)
      printf("\t\t\t\t database = '%s'\n",r->source.database);
    printf("\t\t\t\t all_tables = %d\n",r->source.all_tables);
    if(r->source.tables)
      printf("\t\t\t\t tables = '%s'\n",r->source.tables);
    printf("\t\t  } \n");
    printf("\t\t Target '%s': \n",r->target.target_name);
    printf("\t\t  { \n");
```

```
  if(r->target.type)
    printf("\t\t\t\t type = '%s'\n",r->target.type);
  printf("\t\t\t\t authentication_needed = %d\n",r->target.authentication_needed);
  if(r->target.authentication_type)
    printf("\t\t\t\t authentication_type = '%s'\n",r->target.authentication_type);
  if(r->target.username)
    printf("\t\t\t\t username = '%s'\n",r->target.username);
  if(r->target.password)
    printf("\t\t\t\t password = '%s'\n",r->target.password);
  if(r->target.database)
    printf("\t\t\t\t database = '%s'\n",r->target.database);
  printf("\t\t  } \n");
  printf(" } \n");

  printf("\n");
}
```