Lab 11: Parsing

Goals

Demonstrate proficiency in parsing sentences in a simple language used to specify operations on image files.

Background

Parsing is the process of consuming elements of some input language and mapping the information contained therein to a useful data structure. We will define an input language for specifying operations on images and build a parser that can digest the language.

An example of a "sentence" in our language is:

```
fade
{
   in0  dive.ppm
   factor 0.4
   out faded_dive.ppm
}
```

This alternative variant specifies the same information.

```
fade
{
   factor      0.4
   in0         dive.ppm
   out         faded_dive.ppm
}
```

The following rules define our language

- Words in a sentence must be separated by one or more whitespace characters.
- Each sentence starts with the name of an image operation (fade, gray, mirror, blend, etc...)
- The operation must be followed by the { character.
- Following the { are a variable number of clauses of the form *parameter-name parameter-values*
- The parameter names (like image operations) are predefined reserved words in the language.
- For each parameter name, the type and number of parameter values is always the same.
- A parameter-name parameter-values clause may be followed by either another parameter-name parameter-values clause or the } character which indicates the end of a sentence.

Implications of the rule set

- The fact that words in the language *must* be separated by white space makes it possible to use the *fscanf(stdin,...)* function with the %s, %d, %lf format codes to consume text, integers and double precision values respectively.
- We cannot assume anything about the order in which the p*arameter-name parameter-value* sets appear.

- We have a potential need to deal with missing parameters (e.g. *factor* missing in a *fade* operation) or
- Extraneous parameters (e.g. *factor* specified) in a *gray* operation. We will not deal with the missing/extraneous parameter issue in this lab.

The target data structure

For consistency and ease of remembering we use the *same names* for each parameter in both our language and its corresponding element in the following structure.

```
typedef struct imageop_type
{
   int    opcode;    // index of operation name
   char   in0[64];   // primary input image
   char   in1[64];   // secondary input image
   char   out[64];   // output image
   int    dims[2];   // dims of output for resize
   double factor;    // factor for gray or blend
}  imageop_t;
```

Table lookups

Table lookups will comprise an important element of the processing. A table of addresses of character strings can be constructed as follows:

```
char *op_names[] =
{
   "mirror",
   "gray",
   "fade",
   “resize”,
};


char *param_names[] =
{
   "in0",
   "in1",
   "out",
   "dims",
   "factor",
};
```

The *table_lookup()* function

You will build a table lookup function whose mission is to return the index of the parameter string name in the parameter table. It should return (-1) if the string is not in the table.

The *strcmp()* function from the standard string library should be used to compare the string pointed to by *table[i]* with the string *name* for i = 0 to count - 1.

```
int table_lookup(char *table[], int count, char name[])
{

}
```

Invoking table lookup:

```
int count = sizeof(first_names)/sizeof(first_names[0]);

int ndx = table_lookup(op_names, count, "gray");

int ndx = table_lookup(op_names, count, "blend");
```

- In the first case the table_lookup function should return the value 1.
- In the second case it should return -1.
- Setting count as shown is important.. It makes it easy to add or remove table entries without having to change the code that invokes table_lookup.

Building the complete parser

An invocation of the parser will attempt to read *exactly one sentence* in our input language. The parser will read from the standard input (*stdin*). Later we will want to be able to perform multiple image operations in a single run of our program, but that will require multiple invocations of the parser!

```
int parser(imageop_t *op)
{
```

read a string from *stdin* into into character array *opname;*

if (no data is read) // end of file reached

return(-1);

attempt to look up *opame* in the *op_names[]* table

if look up fails print the invalid *opname* and exit.

save the table index that was returned in *op->opcode*.

invoke `consume_parameters(op)` to consume the delimiters { }, parameter names, and parameter values.

return(0);

```
}
```

Consuming the parameters of a particular operation

```
int consume_parameters(imageop_t *op)
{
  read a string from stdin into into  character array pname;
  ensure that pname[0] is the { character.
  If not print pname in an error message and exit

  read a string from stdin into into  character array pname;
  while (pname[0] is not the } character and not end-of-file)
  {
    attempt to look up pname in the param_names[] table
    if that fails print the invalid pname in an error message and exit.
     consume_parameter_values(op, table_index);
     read next string from stdin into into  character array pname;

  }
  return(0);
}
```

Consuming the values associated with a parameter

There exist far more elegant ways to consume the parameter values, but the simplest way is via a big switch statement:

```
int consume_parameter_values(imageop_t *op, int ndx)
{
   int howmany;

   switch (ndx)
   {
   case 0:   // param is in0
      howmany = fscanf(stdin, "%s", op->in0);
      if (howmany != 1)
          Print error message and exit.
      break;
   case 1:   // param is in1
      howmany = fscanf(stdin, "%s" op->in1);
     if (howmany != 1)
          Print error message and exit.
      break;

    etc
   }

}
```

Assignment:

In this assignment you will create and submit a file named parser.c that should accept sentences using the operations found in the *op_names()* table and parameters found in the *param_names()* table.

The directory 1ab11 contains a main.c and image.h file that you *must* use. It also contains a parser.c that you are free to use as a starting point. Be sure that your parser function returns the correct value and aborts with reasonable error messages when it encounters invalid input.

Turn In Work

Show your TA that you completed the assignment. Then turn in your parser.c using the command:

sendlab.101.*section_number* 11 parser.c