LAB TERMINAL

Csc441-sp24-lab terminal

Submitted By : Sharafat Noor

Reg #: Sp21-bcs-036

Submitted To: Sir Bilal Bukhari

Date: 31st-May-2024

Question 1:

Introduction:

This tool is a compiler for the C programming language. It has two main parts: a C lexer, which reads through the C code you give it, and a yacc parser, which understands the structure of the code.

The main job of a compiler is to take code written in a language like C and turn it into instructions that a computer can understand and execute. But computers don't directly understand high-level languages like C; they need something more basic called machine code. So, compilers often translate the C code into a kind of in-between language called intermediate code. This intermediate code is closer to machine code than C, but still keeps some of the higher-level information from the original C code.

Objectives:

Following constructs will be handled by the mini-compiler:

- Data Types: int, char data types with all its sub-types. Syntax : int a=3;
- Comments: Single line and multiline comments,
- Keywords: char, else, for, if, int, long, return, short, signed, struct, unsigned, void, while, main
- Identification of valid identifiers used in the language,

- Looping Constructs: It will support nested for and while loops. Syntax: int i;
- for(i=0;i
- Conditional Constructs: if...else-if...else statements,
- Operators: ADD(+), MULTIPLY(*), DIVIDE(/), MODULO(%) etc.
- Delimiters: SEMICOLON(;), COMMA(,)
- Function construct of the language, Syntax: int func(int x)
- Support of nested conditional statement and nested loops
- Support for a 1-Dimensional array. Syntax : char s[20];

C Program

This section describes the input C program which is fed to the yacc script for parsing. The workflow is explained as under:

1. Compile the script using Yacc tool

```
$ yacc -d c_parser.y
```

2. Compile the flex script using Flex tool

```
$ flex c_lexer.l
```

3. After compiling the lex file, a lex.yy.c file is generated. Also, y.tab.c and y.tab.h files are generated after compiling the yacc script.
4. The three files, lex.yy.c, y.tab.c and y.tab.h are compiled together with the options –ll and –ly

```
$ gcc -o compiler lex.yy.c y.tab.h y.tab.c -ll -ly
```

5. The executable file is generated, which on running parses the C file given as a command line input

```
$ ./compiler test.c
```

The script also has an option to take standard input instead of taking input from a file

---

Question 2:

Sample input and output for your compiler construction project:

Design of Programs:

First I will provide both lexer and parser code for the design of complier.

Code:

Updated Lexer Code:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "y.tab.h"
#define ANSI_COLOR_RED "\x1b[31m"
#define ANSI_COLOR_GREEN "\x1b[32m"
#define ANSI_COLOR_YELLOW "\x1b[33m"
#define ANSI_COLOR_BLUE "\x1b[34m"
#define ANSI_COLOR_MAGENTA "\x1b[35m"
#define ANSI_COLOR_CYAN "\x1b[36m"
#define ANSI_COLOR_RESET "\x1b[0m"
struct symboltable
{
char name[100];
char class[100];
char type[100];
char value[100];
int nestval;
int lineno;
int length;
int params_count;
}ST[1001];
struct constanttable
struct constanttable
{
char name[100];
char type[100];
int length;
}CT[1001];
int currnest = 0;
extern int yylval;
int hash(char *str)
{
int value = 0;
for(int i = 0 ; i < strlen(str) ; i++)
{
value = 10*value + (str[i] - 'A');
value = value % 1001;
while(value < 0)
value = value + 1001;
```

```
41  value = value + 1001;
42  }
43  return value;
44  }
45  int lookupST(char *str)
46  {
47  int value = hash(str);
48  if(ST[value].length == 0)
49  {
50  return 0;
51  }
52  else if(strcmp(ST[value].name,str)==0)
53  {
54  return value;
55  }
56  else
57  {
58  for(int i = value + 1 ; i!=value ; i = (i+1)%1001)
59  {
60  if(strcmp(ST[i].name,str)==0)
```

Parser Code:

```
1   #include <stdio.h>
2    #include <string.h>
3    #include <stdlib.h>
4    void yyerror(char* s);
5    int yylex();
6    void ins();
7    void insV();
8    int flag=0;
9    #define ANSI_COLOR_RED "\x1b[31m"
10   #define ANSI_COLOR_GREEN "\x1b[32m"
11   #define ANSI_COLOR_CYAN "\x1b[36m"
12   #define ANSI_COLOR_RESET "\x1b[0m"
13   extern char curid[20];
14   extern char curtype[20];
15   extern char curval[20];
16   extern int currnest;
17   void deletedata (int );
18   int checkscope(char*);
19   int check_id_is_func(char *);
20   void insertST(char*, char*);
21   void insertSTnest(char*, int);
```

```
21  void insertSTnest(char*, int);
22  void insertSTparamscount(char*, int);
23  int getSTparamscount(char*);
24  int check_duplicate(char*);
25  int check_declaration(char*, char *);
26  int check_params(char*);
27  int duplicate(char *s);
28  int checkarray(char*);
29  char currfunctype[100];
30  char currfunc[100];
31  char currfunccall[100];
32  void insertSTF(char*);
33  char gettype(char*,int);
34  char getfirst(char*);
35  void push(char *s);
36  void codegen();
37  void codeassign();
38  char* itoa(int num, char* str, int base);
39  void reverse(char str[], int length);
40  void swap(char*,char*);
41  void label1();
```

```
36  void codegen();
37  void codeassign();
38  char* itoa(int num, char* str, int base);
39  void reverse(char str[], int length);
40  void swap(char*,char*);
41  void label1();
42  void label2();
43  void label3();
44  void label4();
45  void label5();
46  void label6();
47  void genunary();
48  void codegencon();
49  void funcgen();
50  void funcgenend();
51  void arggen();
52  void callgen();
53  int params_count=0;
54  int call_params_count=0;
55  int top = 0,count=0,ltop=0,lno=0;
56  char temp[3] = "t";
```

Explanation:

The lex code is detecting the tokens from the source code and returning the corresponding token to the parser. In phase 1 we were just printing the token and now we are returning the token so that parser uses it for further computation. We are using the symbol table and constant table of the previous phase only. We added functions like insertSTnest(),insertSTparamscount(),checkscope(),deletedata(),dup licate() etc., in order to check the semantics.In the production rules of the grammar semantic actions are written and these are performed by the functions listed above. Along with semantic actions SDT also included function to generate the 3 address code.

INPUT AND OUTPUT TEST CASES SCREENSHOTS:

1.

```
//Nested if else condition

#include <stdio.h>
void main()
{
    int a,b,c,d;
    if (a<3)
    {
        if(c<d)
        {
            a = 98;
        }
        else
        {
            a = d * b + c;
        }
    }
    else
    {
        a++;
    }
}
```

2.

```
#include <stdio.h>
void main()
{
    int a,b,c,d;
    while(a < 10)
    {
        if (a<3)
        {
            if(c<d)
            {
```

```
                a = 98;
            }
            else
            {
                a = d * b + c;
            }
        }
        else
        {
            a++;
        }
    }
    a = b+c;
}
```

3.

```
#include <stdio.h>

int myfunc(int a,int b)
{
      return a+b;
}

void main()
{
      int a,b,i;

      while(a<3)
      {
            a = a+b;
            for(i=0;i<b;i++)
            {
                  b++;
                  myfunc(a,b);

            }
            a++;
      }
}
```

OUTPUT:

1.

```
func begin main
t0 = 3
t1 = a < t0
IF not t1 GoTo L0
t2 = c < d
IF not t2 GoTo L1
t3 = 98
a = t3
GoTo L2
L1:
t4 = d * b
t5 = t4 + c
a = t5
L2:
GoTo L3
L0:
t6 = a + 1
a = t6
L3:
func end

Status: Parsing Complete - Valid
                        SYMBOL TABLE
                         ------------
    SYMBOL |           CLASS |       TYPE |      VALUE |    LINE NO |         NESTING | PARAMS COUNT |
-----------------------------------------------------------------------------------------------------
         a |      Identifier |        int |          3 |          4 |           99999 |           -1 |
         b |      Identifier |        int |            |          4 |           99999 |           -1 |
         c |      Identifier |        int |            |          4 |           99999 |           -1 |
         d |      Identifier |        int |            |          4 |           99999 |           -1 |
        if |         Keyword |            |            |          5 |            9999 |           -1 |
       int |         Keyword |            |            |          4 |            9999 |           -1 |
      main |        Function |       void |            |          2 |            9999 |            0 |
      else |         Keyword |            |            |         11 |            9999 |           -1 |
      void |         Keyword |            |            |          2 |            9999 |           -1 |


                        CONSTANT TABLE
                         -------------
      NAME |            TYPE
-----------------------------------------------------------------------------
        98 | Number Constant
         3 | Number Constant
```

2. (given below)

```
func begin main
L0:
t0 = 10
t1 = a < t0
IF not t1 GoTo L1
t2 = 3
t3 = a < t2
IF not t3 GoTo L2
t4 = c < d
IF not t4 GoTo L3
t5 = 98
a = t5
GoTo L4
L3:
t6 = d * b
t7 = t6 + c
a = t7
L4:
GoTo L5
L2:
t8 = a + 1
a = t8
L5:
GoTo L0:
L1:
t9 = b + c
a = t9
func end

Status: Parsing Complete - Valid
                        SYMBOL TABLE
                        ------------
   SYMBOL |          CLASS |       TYPE |      VALUE |    LINE NO |        NESTING | PARAMS COUNT |
-------------------------------------------------------------------------------------------------
        a |      Identifier |        int |         10 |          4 |          99999 |           -1 |
        b |      Identifier |        int |            |          4 |          99999 |           -1 |
        c |      Identifier |        int |            |          4 |          99999 |           -1 |
        d |      Identifier |        int |            |          4 |          99999 |           -1 |
       if |         Keyword |            |            |          7 |           9999 |           -1 |
      int |         Keyword |            |            |          4 |           9999 |           -1 |
     main |        Function |       void |            |          2 |           9999 |            0 |
     else |         Keyword |            |            |         13 |           9999 |           -1 |
    while |         Keyword |            |            |          5 |           9999 |           -1 |
     void |         Keyword |            |            |          2 |           9999 |           -1 |


                        CONSTANT TABLE
                        --------------
     NAME |           TYPE
-------------------------------------------------------------------------------
       10 | Number Constant
       98 | Number Constant
        3 | Number Constant
```

3. (given below)

```
func begin myfunc
t0 = a + b
func end

func begin main
L0:
t1 = 3
t2 = a < t1
IF not t2 GoTo L1
t3 = a + b
a = t3
t4 = 0
i = t4
L2:
t5 = i < b
IF not t5 GoTo L3
t6 = i + 1
i = t6
t7 = b + 1
b = t7
refparam a
refparam b
refparam result
call myfunc, 2
GoTo L2:
L3:
t8 = a + 1
a = t8
GoTo L0:
L1:
func end

Status: Parsing Complete - Valid
                        SYMBOL TABLE
                        ------------
    SYMBOL |          CLASS |       TYPE |      VALUE |    LINE NO |        NESTING | PARAMS COUNT |
-----------------------------------------------------------------------------------------------------
         a |      Identifier |        int |            |          3 |          99999 |           -1 |
         b |      Identifier |        int |            |          3 |          99999 |           -1 |
         a |      Identifier |        int |          3 |         10 |          99999 |           -1 |
         b |      Identifier |        int |            |         10 |          99999 |           -1 |
         i |      Identifier |        int |            |         10 |          99999 |           -1 |
       for |         Keyword |            |            |         15 |           9999 |           -1 |
    return |         Keyword |            |            |          5 |           9999 |           -1 |
       int |         Keyword |            |            |          3 |           9999 |           -1 |
      main |        Function |       void |            |          8 |           9999 |            0 |
    myfunc |        Function |        int |            |          3 |           9999 |            2 |
     while |         Keyword |            |            |         12 |           9999 |           -1 |
      void |         Keyword |            |            |          8 |           9999 |           -1 |
```

Results:

We were able to successfully parse the tokens recognized by the flex script for C. The output displays the set of identifiers and constants present in the program with their types,values and line of declaration. Also nesting values changes dynamically as the program ends its made infinite. The parser generates error messages in case of any syntactical errors in the test program or any semantic error. Also we are displaying the 3 address code generated by our yacc script.

Question 3:

Create and implement RE and DFAs for the form below:

RE'S

Regular Expressions for Each Field

First Name:

RE: ^[A-Za-z]+(?:\s[A-Za-z]+)*$

Description: Validates names that may include spaces but not numbers or special characters. It allows for multiple names separated by spaces.

Last Name:

RE: ^[A-Za-z]+(?:\s[A-Za-z]+)*$

Description: Same as the first name to accommodate double-barrelled or multi-part last names.

Username:

RE: ^\w{5,20}$

Description: Validates a username to be between 5 to 20 characters, including letters, digits, and underscores.

Password:

RE: ^(?=.[a-z])(?=.[A-Z])(?=.\d)(?=.[@#$%]).{8,20}$

Description: Ensures the password has 8-20 characters with at least one lowercase letter, one uppercase letter, one digit, and one special character from @#$%.

Email:

RE: ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$

Description: Checks for a valid email format.

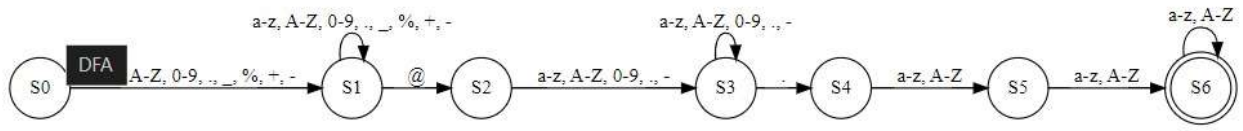
Mobile No:

RE: ^\+?\d{1,3}[-\s]?\d{3}[-\s]?\d{3}[-\s]?\d{4}$

Description: Validates international mobile numbers with an optional country code and spaces or dashes.

See validation using the form:

DFA'S



Form Screenshots:

Information that doesn't match the regex.

With Correct Info:

Html Codes files are attached with it, in directory.

Question 4:

Write a program which generates symbol table for the code you submitted in question 3.

For performing the task the C++ code file are attached with it and its output are below.

Output: