Project Title: Finding variable dependencies using Lex and Yacc.

Team Members: Pradeep Reddy Y10UC221
K Raj Koushik Reddy Y10UC138

Course Name: Operating Systems and Principles
Course Instructor: Gaurav Somani

Abstract:

Running a code sequentially is not the optimal way of using computer resources and time. For the purpose of auto parallelizing a sequential code we need to first find the variable dependencies. The code developed in this project helps us in detecting the lines of code that can run independently from a given code.

The program takes a file containing lines of code as input, extracts all the variables used in the code and gives us line by line usage of all the variables. In Output file we get whether the variable is read, written or unused in each and every line of the input code file. This can lead us to develop auto parallelizing program by running these independent lines concurrently.

Introduction:

Problem Definition: Detecting the independent sections of a given code
.
For detecting the patterns in the input code file and for taking necessary actions we use lex and yacc. Lex divides a stream of input characters into meaningful units (lexemes), identifies them (token) and pass the token to a parser generator, yacc. Lex will read your patterns and generate C code for a lexical analyzer.

Lex rules for detecting the patterns from the given input file.

Meta character	Matches
.	any character except newline
\n	newline
*	zero or more copies of the preceding expression
+	one or more copies of the preceding expression
?	zero or one copy of the preceding expression
^	beginning of line
$	end of line
a|b	a or b
(ab)+	one or more copies of ab (grouping)
"a+b"	literal "a+b" (C escapes still wor
[ ]	character class

Tokens that we used in our lex code.

Metacharacters	Token name
[a-zA-Z0-9]+\.[a-zA-Z0-9]+	FILEN
"main"	MAIN
\(	SOBRACE
\)	SEBRACE
int|float|char	TYPE
,	COMMA
[0-9]+	NUMBER
[a-zA-Z0-9]+	WORD
[+*-/]	OP
\"	QUOTE
\{	OBRACE
\}	EBRACE
\]	SQEBRACE
\[	SQOBRACE
;	SEMICOLON
\>	GT
\<	LT
\!	NOT
=	EQ

Working:

Lex code detects the patterns in the input file and returns the respective tokens to another C file.
The yacc file detects the order of tokens and performs some action.

Algorithm

- If the Syntax in the input file does not match with the defined rules in Lex the Yacc file returns standard syntax error message.
- Whenever lex code encounters a variable, it passes the token named WORD to yacc program.. IT checks that WORD in 3-D array of variables, where one dimension is used for storing variables and the second dimension is used to store whether the variable is READ OR WRITTEN in a line, where each line represents a row in the second dimension.
- The SEMICOLON token is used to know the end of the line. Whenever it encounter "=" it passes EQ token.
- The WORDS which are right side of EQ token are stored as READ and WORDS on the left side of EQ are stored as WRITTEN.
- In conditional statements the variables on left and right side of "=" are stored as READ.
- Whenever it FORTOK ,MAIN token or WORD token followed by SOBRACE there we

consider it as the starting point of FOR loop ,Main ,User defined functions and EBRACE as ending point of them.

Critical view:

- We were not able to know the dependencies of array variables in looping.
- We tried to store the values of variable in the 3rd dimension of array but we failed.
- We further plan to implement these concepts to make an Auto parallelizing compiler by using the concepts of threading to thread independent sections of the code.

Conclusion:

- We found out the dependencies of variables from the given user code.
- We also found out where the MAIN function,User defined function, FOR loop begins and ends.

References:

http://epaperpress.com/lexandyacc/
http://tldp.org/HOWTO/pdf/Lex-YACC-HOWTO.pdf

Code:

Sample Input file: inputfile

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

int a;
int a,b,c;
main(){
int a;
a++;
b[i]=b[i+1]+j;
for(i=0;;){
k++;
}
int a;
}

int root(){
kpkr=1;
}
```

Sample output file: outputfile

main at line : 5
for loop started at line : 8
for loop ended at line : 12
main ended at line : 13
root started at line : 13
root ended at line : 14

14 6

a	j	b	i	k	kpkr
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
b	0	0	0	0	0
0	r	b	0	0	0
0	0	0	w	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	b	0
0	0	0	0	0	0
0	0	0	0	0	w

Filename: myproject.lex

```
%{
#include <stdio.h>
#include <string.h>
#include "y.tab.h"
%}
%%
for      return FORTOK;
"#include"            return INCLUDE;
[a-zA-Z0-9]+\.[a-zA-Z0-9]+ return FILEN;
"main"          return MAIN;
\(              return SOBRACE;
\)              return SEBRACE;
int|float|char   return TYPE;
,                       return COMMA;
[0-9]+              yylval.number=atoi(yytext);return NUMBER ;
[a-zA-Z0-9]+     yylval.string=strdup(yytext); return WORD;
```

```
[+*-/]                  return OP;
\"                     return QUOTE;
\{                      return OBRACE;
\}                      return EBRACE;
\]                     return SQEBRACE;
\[                     return SQOBRACE;
;                      return SEMICOLON;
\>                       return GT;
\<                       return LT;
\!                      return NOT;
=                       return EQ;
\n                    ;
[ \t]+                  ;

%%
```

File name: myproject.yacc

```
%{
#include <stdio.h>
#include <string.h>
char* data[100][1000][2],*arr1[10][100];
int count=0,i,j,line=1,flag=0,p,counta=0,linea=1,arru,aflag=1;
void yyerror(const char *str)
{
	fprintf(stderr,"error: %s\n",str);
}

int yywrap()
{
	return 1;
}

main(){
for(i=0;i<100;i++){
for(j=0;j<1000;j++){
data[i][j][0]="0";
}
}
	yyparse();
printf("%d %d\n",line,count);
for(j=0;j<line;j++){
for(i=0;i<count;i++){
```

```
printf("%5s",data[i][j][0]);
}
printf("\n");
}
/*
printf("%d %d\n",linea,counta);

for(j=0;j<linea;j++){
for(i=0;i<counta;i++){
printf("%5s",arr1[i][j]);
}
printf("\n");
}
*/
}



%}
%token FORTOK  QUOTE OBRACE EBRACE SEMICOLON EQ   SOBRACE SEBRACE LT GT
NOT SQOBRACE SQEBRACE INCLUDE FILEN MAIN TYPE COMMA
%union
{
	int number;
	char *string;
}

%token <string> WORD
%token <number> NUMBER
%token <string> OP
%%
commands:
|commands command
;

command:INCLUDE LT FILEN GT     {line++;}
|part
|MAIN {printf(" main at line : %d\n",line);} SOBRACE SEBRACE OBRACE parts EBRACE
{printf("main ended at line : %d\n",line);}
|TYPE WORD {printf("%s started at line : %d\n",$2,line);} SOBRACE SEBRACE OBRACE parts
EBRACE {printf("%s ended at line : %d\n",$2,line);}
;
decl:TYPE WORD dvar
;
dvar:
|SEMICOLON
| COMMA WORD dvar
;
parts:
```

```
|parts part
;

part:decl  {line++;}
|FORTOK {printf("for loop started at line : %d\n",line);} SOBRACE  forequ SEMICOLON cond
SEMICOLON forequ SEBRACE zonecontent      {printf("for loop ended at line : %d\n",line);}
|equ SEMICOLON         {line++;}
;
forequ:       {line++;}
|equ          {line++;}
;
cond:        {line++;}
|WORD condright1
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){data[i][line][0]="r";flag=1;}}if(flag==0){da
ta[i][0][0]=$1;count++;flag=0;data[i][line][0]="r";}line++;}
;
condright1:LT condright2
|GT condright2
|EQ condright2
|NOT EQ NUMBER

|NOT EQ WORD
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$3)){data[i][line][0]="r";flag=1;}}if(flag==0){da
ta[i][0][0]=$3;count++;flag=0;data[i][line][0]="r";}}
;
condright2:
|WORD
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){data[i][line][0]="r";flag=1;}}if(flag==0){da
ta[i][0][0]=$1;count++;flag=0;data[i][line][0]="r";}}

|NUMBER
|EQ NUMBER

|EQ WORD
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$2)){data[i][line][0]="r";flag=1;}}if(flag==0){da
ta[i][0][0]=$2;count++;flag=0;data[i][line][0]="r";}}
;

zonecontent:
OBRACE zonestatements EBRACE      {}


zonestatements:                          {}
|zonestatements statements          {}
;

fordec:
|WORD EQ NUMBER
;
```

```
for3equ:
|WORD EQ WORD OP NUMBER      {line++;}
|WORD OP OP                  {line++;}
;

block:
FORTOK SOBRACE fordec SEMICOLON forequ SEMICOLON for3equ SEBRACE OBRACE
zonestatements EBRACE  {}
;

statements:
| statements statement
;

statement: equ SEMICOLON       {line++;}
|block                        {}
;

equ:WORD EQ right
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){if(!strcmp(data[i][line][0],"r")){data[i][line]
[0]="b";}else{data[i][line][0]="w";}flag=1;}}if(flag==0){data[i][0][0]=$1;count++;flag=0;data[i][line]
[0]="w";}}


|WORD OP OP
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){data[i][line][0]="b";flag=1;}}if(flag==0){d
ata[i][0][0]=$1;count++;flag=0;data[i][line][0]="b";}}

|WORD SQOBRACE sib SQEBRACE EQ aright
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){if(!strcmp(data[i][line][0],"w")){data[i][line
][0]="b";}else{data[i][line][0]="r";}flag=1;}}if(flag==0){data[i][0][0]=$1;count++;flag=0;data[i][line]
[0]="r";}}

|WORD SQOBRACE sib SQEBRACE OP OP
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){data[i][line][0]="b";flag=1;}}if(flag==0){d
ata[i][0][0]=$1;count++;flag=0;data[i][line][0]="b";}}

|WORD  EQ aright
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){if(!strcmp(data[i][line][0],"r")){data[i][line]
[0]="b";}else{data[i][line][0]="w";}flag=1;}}if(flag==0){data[i][0][0]=$1;count++;flag=0;data[i][line]
[0]="w";}}


right:NUMBER

|WORD OP right
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){if(!strcmp(data[i][line][0],"w")){data[i][line
```

][0]="b";}else{data[i][line][0]="r";}flag=1;}}if(flag==0){data[i][0][0]=$1;count++;flag=0;data[i][line]
[0]="r";}}

|WORD
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){if(!strcmp(data[i][line][0],"w")){data[i][line
][0]="b";}else{data[i][line][0]="r";}flag=1;}}if(flag==0){data[i][0][0]=$1;count++;flag=0;data[i][line]
[0]="r";}}

|NUMBER OP right
;

sib:WORD OP NUMBER //printf("%d\n",$3);}
|WORD
//flag=0;if(aflag==1){for(i=1;i<linea;i++){if(!strcmp(arr1[arru][i],$1)){flag=1;}}if(flag==0){arr1[arru]
[i]=$1;linea++;}}}
|NUMBER
;

aright:NUMBER
|WORD OP aright
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){if(!strcmp(data[i][line][0],"w")){data[i][line
][0]="b";}else{data[i][line][0]="r";}flag=1;}}if(flag==0){data[i][0][0]=$1;count++;flag=0;data[i][line]
[0]="r";}}

|WORD
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){if(!strcmp(data[i][line][0],"w")){data[i][line
][0]="b";}else{data[i][line][0]="r";}flag=1;}}if(flag==0){data[i][0][0]=$1;count++;flag=0;data[i][line]
[0]="r";}}

|NUMBER OP aright

|WORD SQOBRACE sib SQEBRACE
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){if(!strcmp(data[i][line][0],"r")){data[i][line]
[0]="b";}else{data[i][line][0]="w";}flag=1;}}if(flag==0){data[i][0][0]=$1;count++;flag=0;data[i][line]
[0]="w";}//flag=0;for(i=0;i<counta;i++){if(!strcmp(arr1[i][0],$1)){flag=1;}}if(flag==0){arr1[i][0]=$1;
counta++;}arru=i;}
}

|WORD SQOBRACE sib SQEBRACE OP aright
{flag=0;for(i=0;i<count;i++){if(!strcmp(data[i][0][0],$1)){if(!strcmp(data[i][line][0],"r")){data[i][line]
[0]="b";}else{data[i][line][0]="w";}flag=1;}}if(flag==0){data[i][0][0]=$1;count++;flag=0;data[i][line]
[0]="w";}//flag=0;for(i=0;i<counta;i++){if(!strcmp(arr1[i][0],$1)){flag=1;}}if(flag==0){arr1[i][0]=$1;
counta++;}arru=i;}
}
;