Laboratory-5

Question

Generate intermediate code for if, if-else & while loop in C.

Lex File (lex.l):

```
%{
typedef struct B {
    int t;
    int f;
}B;
#include "y.tab.h"
%}
%option header-file="myLexHeader.h"
%%
"#include"                      return INCLUDE;
"<"[a-zA-Z]+\.h">"                  return INCL_FILE;
"#define"                       {strcpy(yylval.string,yytext); return MACRO;}
(int|char|float)                  return TYPE;
"if"                           return IF;
"else"                          return ELSE;
"while"                          return WHILE;
"main"                          return MAIN;
"return"                                  return RETURN;
(<|>|>=|<=|==)                      {strcpy(yylval.string,yytext); return
LOGIC_OPRTR;}
(\+=|\-=|\*=|\/=)                  return OPRTR_ASSGN;
(\+\+|\-\-)                       return UNARY;
([_|a-z]+[0-9]*)*                   {strcpy(yylval.string,yytext); return
VARIABLE;}
[0-9]*                          {strcpy(yylval.string,yytext); return
CONSTANT;}
[\n\r]                          {yylineno++;}
[-;+*/=(),{}]                      {return yytext[0];}
[ \t]+                          ;
.                             return ERROR;
%%
int yywrap(void)
{
    return 1;
}
```

Yacc File (par.y):

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
extern int yylex();
int yyerror();
int temp_var_num = 0;
```

```
int label_num=0;
int else_label_num = 0;
bool else_flag=0;
int bool_ir_index=100;
int bool_ir_start = 100;
int bool_ir_if_top=-1;
int bool_ir_if_stack[10];
int loop_stack_top=-1;
int loop_stack[10];
char bool_code[25][50];
struct three_address_code {
    char instr[4][20];
} *tac_temp,tac[50];
int tac_len, tac_temp_len, tac_temp_tot = 1;
struct tac_package {
  int tac_len;
  struct three_address_code *tac;
};
void addToThreeAddressArthmtc(char* op, char* arg1, char* arg2, char*
result)
{
  if(tac_temp_len >= tac_temp_tot-1)
 {
    tac_temp = reallocarray(tac_temp,sizeof(struct three_address_code),
tac_temp_tot *= 2);
  }
  strcpy(tac_temp[tac_temp_len].instr[0],result);
  strcpy(tac_temp[tac_temp_len].instr[1],arg1);
  strcpy(tac_temp[tac_temp_len].instr[2],arg2);
  strcpy(tac_temp[tac_temp_len].instr[3],op);
  tac_temp_len++;
}
void addToThreeAddressBrnch(char *res)
{
  char *arg1 = "",*arg2 = "",*op = "",*result = "";
  char *init = strtok(res," ");
  if(res[0] == 'i')
  {
    arg1 = strtok(NULL," ");
    op = strtok(NULL, " ");
    arg2 = strtok(NULL, " ");
    strtok(NULL, " ");
    result = strtok(NULL, " ");
  }
  else if(res[0] == 'g')
  {
    op = init;
    result = strtok(NULL, " ");
  }
  else result = strtok(init,":");
  strcpy(tac[tac_len].instr[0],result);
  strcpy(tac[tac_len].instr[1],arg1);
  strcpy(tac[tac_len].instr[2],arg2);
  strcpy(tac[tac_len].instr[3],op);
  tac_len++;
}
void generate_code(char* op, char* arg1, char* arg2, char* result)
{
  //printf("%s = %s %s %s\n", result, arg1, op, arg2);
  addToThreeAddressArthmtc(op,arg1,arg2,result);
```

```
}
void backpatch(int *list,int next_ir) {
  char addr[10];
  sprintf(addr,"L%d ",next_ir-100);
  int i=0;
  while(list[i]!=0) {
      char label[25];
      strcat(bool_code[list[i]-100],addr);
      if(bool_code[next_ir-100][0] != 'L') {
        sprintf(label,"L%d: ",next_ir-100);
        strcat(label,bool_code[next_ir-100]);
        strcpy(bool_code[next_ir-100],label);
      }
    i++;
  }
}
%}
%token INCLUDE INCL_FILE MACRO
%token TYPE IF ELSE VARIABLE CONSTANT
%token WHILE MAIN RETURN
%token LOGIC_OPRTR OPRTR_ASSGN UNARY OR AND
%token ERROR
%left '+' '-'
%left '*' '/'
%left LOGIC_OPRTR
%right '=' UMINUS
%%
program: program_body
|
;
program_body:
  include program_body
| main
;
include: INCLUDE INCL_FILE
main: TYPE MAIN '(' ')' '{' body '}'
body:
  body line
|
;
line: branch
  {
    if(else_flag)
    {
      char label[5] = {0}, buff[5] = {0};
      sprintf(buff,"%d",else_label_num);
      label[0] = 'G';
      strcat(label,buff);
      strcpy(tac[tac_len++].instr[0],label);
      else_flag=0;
      else_label_num++;
    }
  }
|    assignment ';'
%%
#include "myLexHeader.h"
int main(void)
{
    for(int i=0;i<tac_len;i++)
    {
```

```
      char *res = tac[i].instr[0];
      char *arg1 = tac[i].instr[1];
      char *arg2 = tac[i].instr[2];
      char *op = tac[i].instr[3];
      if(res[0] == 'L' || res[0] == 'G')
      {
        if(op[0] == 'g')    printf("%s %s\n", op, res);
        else if(op[0] == '\0')    printf("%s:\n", res);
        else    printf("if %s %s %s goto %s\n", arg1, op, arg2, res);
      }
      else
        printf("%s = %s %s %s\n", res,arg1,op,arg2);
    printf("completed!\n");
else    printf("failed!\n");}
}
int yyerror(char *s)
{
    printf("%d : %s %s\n",yylineno,s,yytext);
    return 1;
}
```

Sample.c:

```
int main(){
int x = 4, count = 1;
if(x < 5)    count = count*2;
else count = count + 2;
return 0;
}
```

Output:

```
kal-el@mos-13:~/Desktop/Compilers/Compilers Lab/lab_5$ ./generator < input.txt
x = 4
count = 1
if x < 5 goto L2
goto L3
L2:
t0 = count * 2
count = t0
goto G0
L3:
t1 = count + 2
count = t1
G0:
completed!
```

Result:

Intermediate code for conditional and looping constructs was generated successfully.