NDHU, CSIE Dept., 2021 Spring

Professor Chung Yung

Programming Languages and Compilers

Programming Assignment #2
Addendum

- Successful Demonstration
- Reference File Set
- Basic Requirement

1.

Successful Demonstration

1. Rebuild Executable

MyTiny2C — -bash — 80×67

```
Chungde-Mac-mini:MyTiny2C yung$ make mytiny2c
bison -d -o mytiny_parse.c mytiny_parse.y
gcc -c -o mytiny.o mytiny.c
gcc -c -o mytiny_tree.o mytiny_tree.c
gcc -c -o mytiny_parse.o mytiny_parse.c
flex -omytiny_lex.c mytiny_lex.l
gcc -c -o mytiny_lex.o mytiny_lex.c
gcc -o mytiny2c mytiny.o mytiny_tree.o mytiny_parse.o mytiny_lex.o
Chungde-Mac-mini:MyTiny2C yung$
```

2. Test on test.t

MyTiny2C — -bash — 80×67

```
Chungde-Mac-mini:MyTiny2C yung$ ./mytiny2c test.t
Chungde-Mac-mini:MyTiny2C yung$ gcc -o test test.c
Chungde-Mac-mini:MyTiny2C yung$ ./test
A41.input 5
A42.input 7
A4.output 0
Chungde-Mac-mini:MyTiny2C yung$
```

3. Test on fac.t

MyTiny2C — -bash — 80×67

```
Chungde-Mac-mini:MyTiny2C yung$ ./mytiny2c fac.t
Chungde-Mac-mini:MyTiny2C yung$ gcc -o fac fac.c
Chungde-Mac-mini:MyTiny2C yung$ ./fac
Input an integer value for x: 7
fac(x) = 5040
Chungde-Mac-mini:MyTiny2C yung$
```

2.

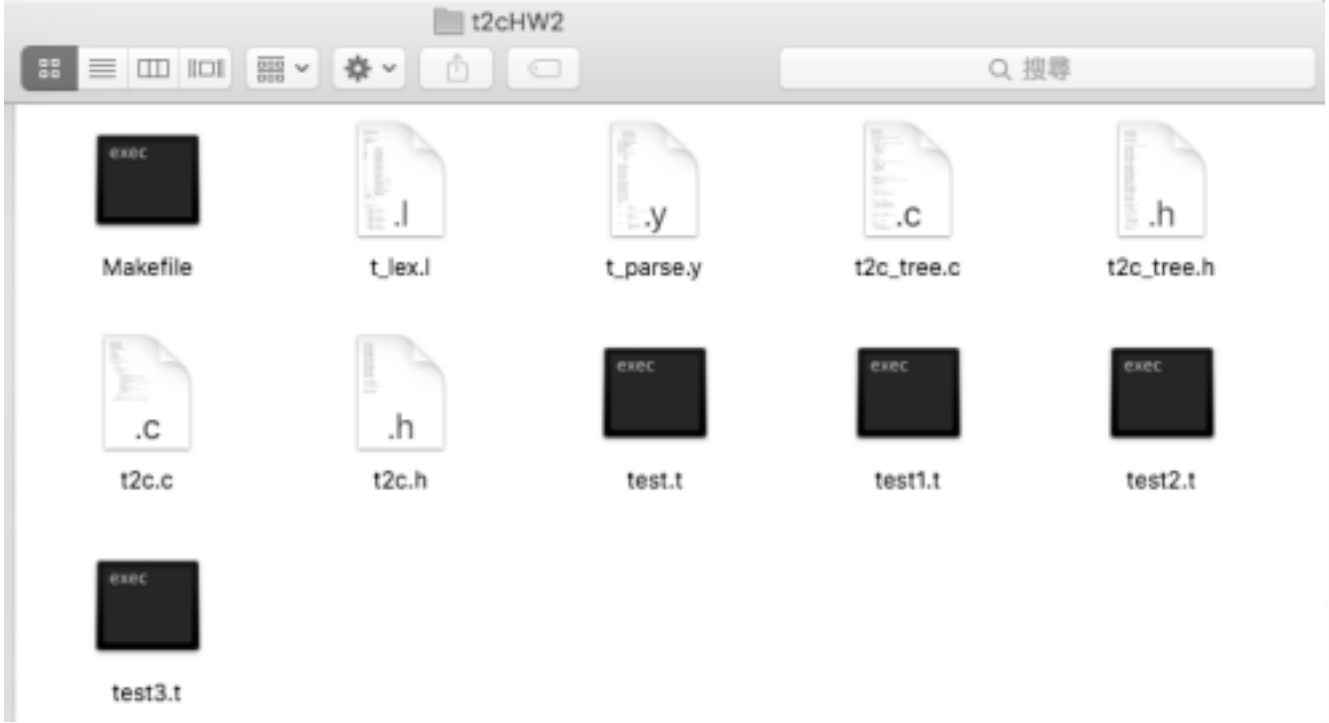
Reference File Set

In Elearn Website

05月 30日 - 06月 5日 (第15週)

Lab Session #4 (Remote classroom)
Lecture for HW#2 will start at 9:30am at the remote classroom.
Programming Assignment #2
HW#2 Reference Files

Unzip it



3.

Basic Requirement

Basic Requirement

- Understand the suggested data structures for parse trees (mytiny_tree.h and mytiny_tree.c).
- Declare the types of grammar symbols (mytiny_parse.y)
- Complete the semantic actions for constructing a parse tree(mytiny_parse.y)
- Complete the C code generator (mytiny_tree.c: gen_code() and gen_exp())

3.1

Data Structures

Data Structures
(t2c_tree.h)

```
typedef struct t_exp {
  int exp_id;
  char name[16];
  int  ival;
  float rval;
  char qstr[80];
  struct t_exp *exp1;
  struct t_exp *next;
} tEXP;
```

```
typedef struct t_stm {
  int stm_id;
  struct t_exp *exp1;
  struct t_exp *exp2;
  struct t_stm *stm1;
  struct t_stm *stm2;
  struct t_stm *next;
} tSTM;

typedef struct sym_node {
  char name[16];
  int type;
  struct sym_node *next;
} symNODE;
```

Related Functions

```
extern tEXP* create_exp();
extern tSTM* create_stm();
extern symNODE* create_symnode( char*, int );
extern void free_exp( tEXP* );
extern void free_stm( tSTM* );
extern void free_symnode( symNODE* );
extern void print_exp( tEXP* );
extern void gen_exp( tEXP* );
extern void print_stm( tSTM* );
extern void gen_code( tSTM* );
extern int lookup( symNODE*, char* );
extern void init_all();
extern void gen_rwcode();
extern int ana_exptype( tEXP* );
```

3.2

Types of Grammar Symbols

Types of Symbols (1/2)

(t_parse.y)

The Union Type

```
%union { tSTM* sm;
         tEXP* ex;
         int iv;
         float rv;
         char* sr;
       }
```

Statement Nodes

```
%type <sm> prog
%type <sm> mthdcls
%type <sm> mthdcl
%type <sm> block
%type <sm> stmts
%type <sm> stmt
%type <sm> vardcl
%type <sm> astm
%type <sm> rstm
%type <sm> istm
%type <sm> wstm
%type <sm> dstm
```

Types of Symbols (2/2)

Expression Nodes

```
%type <ex> type
%type <ex> formals
%type <ex> formal
%type <ex> oformal
%type <ex> expr
%type <ex> mexprs
%type <ex> mexpr
%type <ex> pexprs
%type <ex> pexpr
%type <ex> bexpr
%type <ex> aparams
%type <ex> oparams
```

Token Types

```
%type <sr> lID
%type <iv> lINUM
%type <rv> lRNUM
%type <sr> lQSTR
```

3.3

Values of Grammar Symbols

Values of Tokens
(t_lex.l)

```
{ID}            {sscanf(yytext,"%s", name);
                 yylval.sr = strdup( name );
                 return lID;}
{DIG}           {sscanf(yytext,"%d", &ival);
                 yylval.iv = ival;
                 return lINUM;}
{RNUM}          {sscanf(yytext,"%f", &rval);
                 yylval.rv = rval;
                 return lRNUM;}
\"{NQUO}*\"     {strcpy( qstr, yytext );
                 yylval.sr = strdup( qstr );
                 return lQSTR;}
```

Values of Symbols
(t_parse.y)

$A \rightarrow X_1 \ldots X_n$

- $$ = return value of A
- $k = return value of X_k

```
mthdcls :       mthdcl mthdcls
                { $$ = $1;
                  $$->next = $2; }
```

An Example

```
mthdcl  :       type lMAIN lID lLP formals lRP block
                { $$ = create_stm();
                  $$->stm_id = sMAIN;
                  $$->exp1 = create_exp();
                  $$->exp1->exp_id = eID;
                  strcpy( $$->exp1->name, $3 );
                  $$->exp1->exp1 = $1;
                  $$->exp2 = $5;
                  $$->stm1 = $7;
                  symtab = create_symnode( $3, $1->ival ); }
```

Complete Such Cases
(Statements: 6)

```
vardcl  :       type lID lSEMI
                { // Write your own semantic action here.
                }
        |       type astm
                { // Write your own semantic action here.
                }
        ;
```

Complete Such Cases
(Expressions: 15)

```
mexpr   :       pexpr pexprs
                { // Write your own semantic action here.
                }
        ;

pexprs  :       lTIMES pexpr pexprs
                { // Write your own semantic action here.
                }
        |       lDIVIDE pexpr pexprs
                { // Write your own semantic action here.
                }
        |
                { $$ = NULL; }
        ;
```

3.4

Code Generator

Gen_Code()
(t2c_tree.c)

```
void gen_code ( tSTM* p ) {
  tEXP *te;
  tSTM *ts;
  int t;

  if( p ) {
    switch( p->stm_id ) {
    case sMAIN: gen_exp( p->exp1->exp1 );
                fprintf( yyout, " main ( " );
                gen_exp( p->exp2 );
                fprintf( yyout, ")\n" );
                gen_code( p->stm1 );
                break;
```

Complete Such Cases
(Statements: 4)

```
case sRSTM: // Write your own gen_code here.
            break;
case sISTM: // Write your own gen_code here.
            break;
case sWSTM: // Write your own gen_code here.
            break;
case sDSTM: // Write your own gen_code here.
            break;
```

Gen_exp()
(t2c_tree.c)

```
void gen_exp ( tEXP* p ) {
  tEXP* te;
  if( p ) {
    switch( p->exp_id ) {
    case eTYPE: fprintf( yyout, "%s", c_types1[ p->ival - 1] );
                break;
    case eFORM: gen_exp( p->exp1 );
                fprintf( yyout, " %s", p->name );
                if (p->next) {
                    fprintf( yyout, ", " );
                    gen_exp( p->next );
                }
                break;
```

Complete Such Cases
(Expressions: 12)

```
case eMEXP: // Write your own gen_exp here.
            break;
case eTIMES: // Write your own gen_exp here.
            break;
case eDIVIDE: // Write your own gen_exp here.
            break;
```

That's all!

- Time To Start writing Your own Program!