Document File for Semantic Analyzer

> 陈睿　Chen, Rui 1809853Z-I011-0017
>
> 李亦初　Li, Yichu 18098533-I011-0103
>
> 王虹博　Wang, Hongbo 18098532-I011-0018

0. Develop environment

Using c++ and Visual Studio 2019 Community.

The c++ version must be higher than c++11, the following is the compile command

> g++ -std=c++11 automata.cpp list.cpp main.cpp token.cpp tools.cpp parse.cpp analyzer.cpp symbolTable.cpp -o main (not recommanded)

A test file is appended with the souce code. If you compile the program with the above command, you can try the Test.pym file with the following command.

> main Test.pym

Due to different defination of **NULL** in different version of c++ compiler, the compiled program sometimes will **collapse** in the process of printing the created parse tree. If the collapse happens, please retry the excute command for several times (for some reason it will work), or create a new project using Visual Studio 2019 Community and run it.

> When I use MinGW with gcc version 4.9.2, the compiled program will sometimes assign the address ***0xbaadf00dbaadf00d*** rather than **NULL** to pointers, when I assign **NULL** to a pointer. Thus, the program will collapse due to ***0xbaadf00dbaadf00d*** is not equal to **NULL**.
>
> It also happens when I change the **NULL** to **nullptr**. However, it is weird that restarting the compiled program will sometimes fix the problem.

I also put a compiled version in moodle, using Visual Studio 2019 Community.

If the printing process still collapses and the compiled version is also not working, please email me at chen107101@163.com. Than you !

1. Features

> The semantic analyzing is divided into two procedure. It will do pre-trasversal using the prepared parse tree and symbol table. Then do post-transversal with the parse tree.

1.1 Pre-transversal

In this procedure, the semantic analyzer will go through all the tree node of given parse tree and record the declaration information in the symbol table, or create a new symbol table for compound statements to implement a local scope for variables.

source code

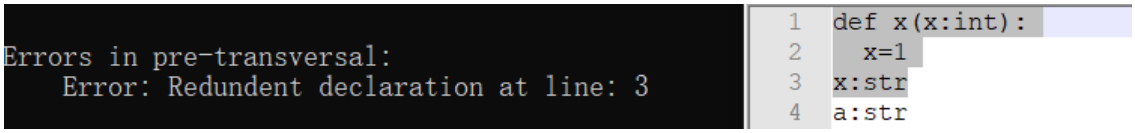
```
void preTransversal(SymbolTable* st,TreeNode* td) {
    SymbolTable* current = st;
    TreeNode* temp = td;
    if (temp->nodeKind == PRAMA_LIST_ND || temp->nodeKind==PROGRAM_NO) {
        while (temp->rSibling!=NULL)
        {
            temp = temp->rSibling;
            preTransversal(current, temp);
        }
    }
    else {
        int i = 0;

        //compound statement, create a new scope
        if (temp->nodeKind == CPD_ND&&temp->parent->dcl!=FUN_DCL) {
            current = new SymbolTable(temp, current);
        }
        //function declaration
        if (temp->nodeKind == DCL_ND && temp->dcl == FUN_DCL) {
            if (current->insert_dcl(temp)) {
                isError = true;
            }
            current = new SymbolTable(temp, current);
            preTransversal(current, temp->child[0]);
            preTransversal(current, temp->child[1]);
            return;
        }
        //insert declaration
        if (temp->nodeKind == DCL_ND) {
            if (current->insert_dcl(temp)) {
                isError = true;
            }
        }
        //insert epression usage
        if (temp->nodeKind == EXPRE_ND) {
            if (current->insert_ref(temp)) {
                isError = true;
            }
        }
```

```
        //tranversal all children
        while (temp->child[i] != NULL) {
            preTransversal(current, temp->child[i]);
            i++;
        }
    }
}
```

Errors that can be detected

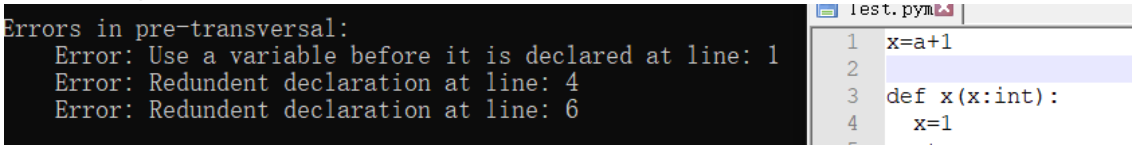
- Redundent declaration

  x is already defined in the current scope.



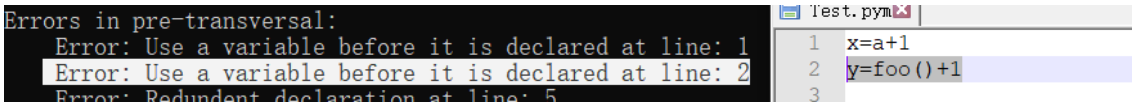
- Undefined variable

  Referencing a variable that is not defined



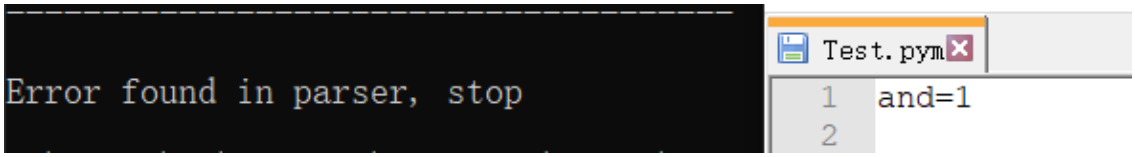
- Undefined function

  Calling a function before it is declared.



- The declared name is same as the keyword

  The semantic analyzer will treat keyword as a pre-declaration.



1.2 Post-transversal

In this procedure, the semantic analyzer will go through all the tree node of given parse tree and check their types.

source code

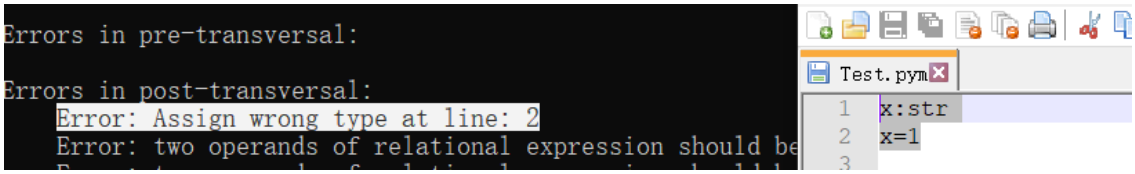
```
void postTransversal(TreeNode* td) {
    TreeNode* temp = td;
    if (temp->nodeKind == PRAMA_LIST_ND || temp->nodeKind == PROGRAM_NO) {
        while (temp->rSibling != NULL)
        {
            temp = temp->rSibling;
            postTransversal(temp);
```

```
        }
    }
    else {
        int i = 0;
        //tranversal all children
        while (temp->child[i] != NULL) {
            postTransversal(temp->child[i]);
            i++;
        }
    //deal with each tree node and annote information
    //ellipse the source code... see detail in the appendix 6.1
    //or in source code file
}
```

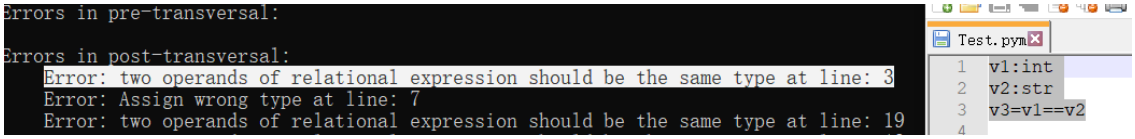
Errors that can be detected

- Assign a wrong type to a declared variable

  x is declared with string type, but assigned with an integer

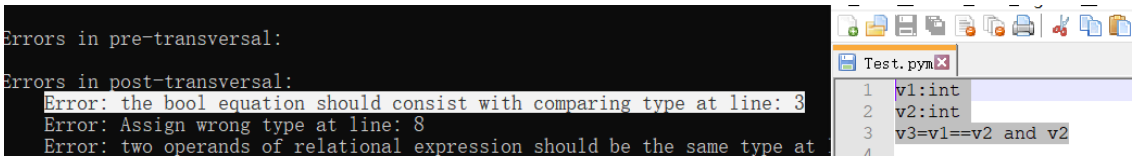


- Operand of relational expression is not the same type

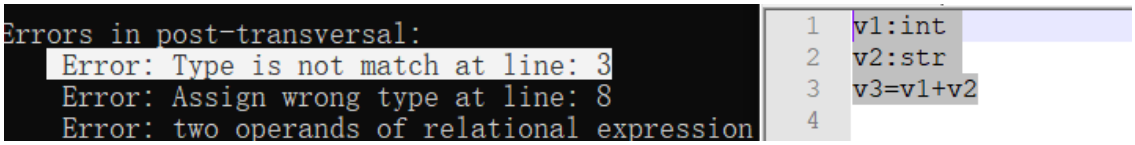


- Operand of bool equation is not relational expression

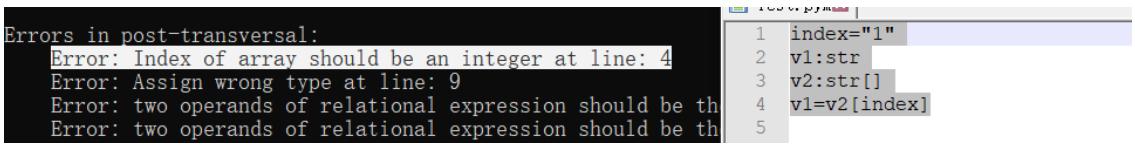
  Due to Pym only support Integer, Number, String, the relational expression's operands have to be relational expression.



- Operand of mathematical expression cannot convert to each other



- The index of array is not interger



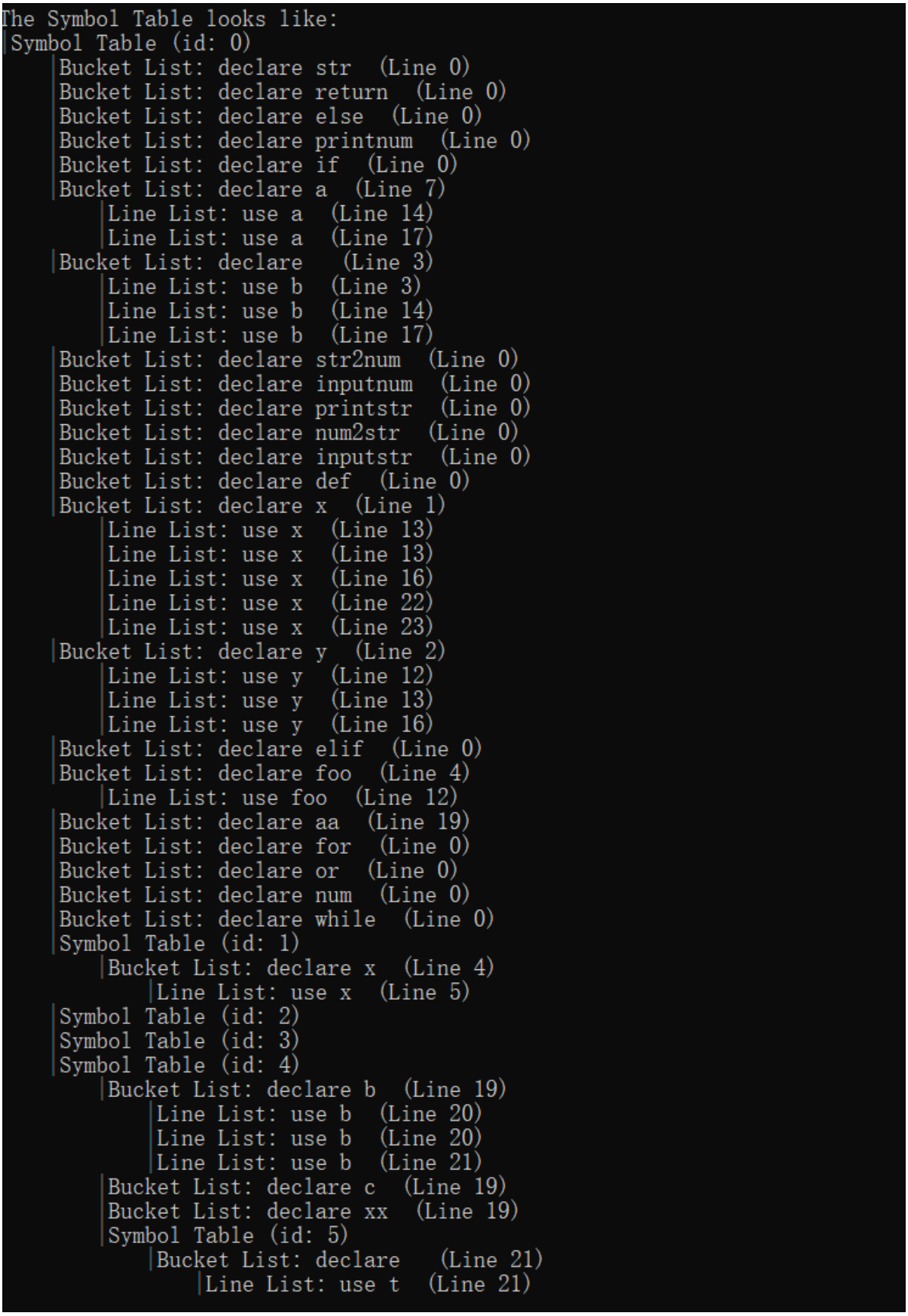
1.3 Symbol Talbe

> Symbol table is implemented by hash table with hash function

Hash function

```
//Hash function
int calHash(string s) {
    int temp = 0;
    int i = 0;
    while (s[i]!='\0')
    {
        temp = ((temp << SHIFT) + s[i]) % ST_SIZE;
        i++;
    }
    return temp;
}
```

Sample output of symbol table



Sample code of pym

```
x:int
y:int
b="1"
def foo(x:int):
  x=1
f:int
a:str
```

```
d=1
d7=2
f=d+d7

y=foo()+1
if x==y and x==1:
  a=b

while x==y:
  a=b

def aa (xx:int,b:str[],c):
  if b[1]==b[1]:
    t=b
  x=2
  return x
```

2. Remaining problem

- The type of function call cannot be annoted.

  Due to the type of a function is not explicit, the semantic analyzer cannot analyze the return type of a function call

- The name of a variable cannot be the same as a declared function

  Due to the data structure of the symbol table, the semantic analyzer cannot distinguish the function name and variable name. As a result, the name of a function and the name of a variable cannot be the same, otherwise it will report an error.

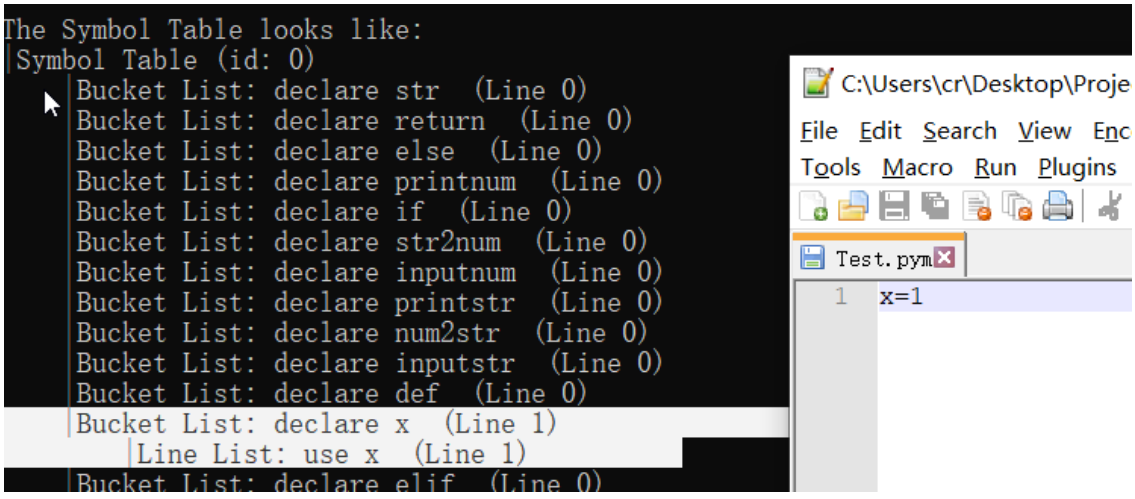
3. Bonus features

- Implicit declaration of a variable

  In the semantic analyzer, it supports to use an assignment to implicitly declare a variable.

  For example, x is not explicitly declared in the function, but exists a statement x=1, then x will be declared.

  Also a use record will be saved in line list because the assignment



- Keywords and built-in function.

  The keywords will be detected by parser and the built-in function will treat as pre-declaration, which will automatically add in the analyzer. If the name of built-in function is declared, the semantic analyzer will report an error.

4. Division of Labor

1. CHEN, RUI
    - Finish the design of semantic analyzer.
- Design the classes and the function, referencing the code provided by professor.
    - Implement the ***preTransversal()*** and ***postTransversal()*** function and the entire project.
- Finish the errors detection mechanism.
    - Test and improve the project and the performance.
    - Write this document.
2. Li, Yichu
    - Test the project and find bugs.
    - List the errors that should be found by semantic analyzer.
3. Wang, Hongbo
    - Test the project and find bugs.
    - List the errors that should be found by semantic analyzer.

5. Difficulties and Solutions

- Data structure

  The main data structure need to be implemented is symbol table. The function **insert_dcl** to insert a bucket list record to a the symbol is easy to implement with the help of code provided by professor and the slides. And the function **insert_ref** is a little bit harder due to the line lists are the children of records and it need a **while** loop to search the last line list.

- Pre-transversal function

  The pre-transversal function is relatively easy if the **insert_dcl** and **insert_ref** function is finish. As long as using a PreEval function as the slides shows, the function can be implemented.

- Post-transveral function

  The post-transversal function will be harder due to the different kind of nodes that I need to classify and deal with them in different action. With mulit-level of switch sentence, I finally make it.

- Debugging

  The process of debugging is monotonous and time-comsuming. However, when the program runs actually the same as I excepted, It makes me fillfuling and satisfying. The effort that I put on it worth it.

6. Appendix

6.1 source code for postTransversal

```
void postTransversal(TreeNode* td) {
    TreeNode* temp = td;
    if (temp->nodeKind == PRAMA_LIST_ND || temp->nodeKind == PROGRAM_NO) {
        while (temp->rSibling != NULL)
        {temp = temp->rSibling;
            postTransversal(temp);}}
    else {int i = 0;
        //tranversal all children
        while (temp->child[i] != NULL) {
            postTransversal(temp->child[i]);
            i++;}
         if (temp->nodeKind == EXPRE_ND) {
         //useless node
            if (temp->child[1] == NULL&&temp->child[0]!=NULL&&temp-
>epx!=ARRAY_EXPR&&temp->epx!=CALL_EXPR) {temp->epx_type = temp->child[0]-
>epx_type;
                return;}
          //array expression and index checking
            if (temp->epx == ARRAY_EXPR) {
                if (temp->child[0] != NULL) {
                    if (temp->child[0]->epx_type == INT_TYPE) {
                        temp->epx_type = temp->dcl_type;}
                    else {semantic_error_report(temp, "Index of array should be
an integer");}}
                else {temp->epx_type = ARRAY_TYPE;}}
            switch (temp->epx)
            {//uncompleted call function
            case CALL_EXPR:
            case CALL_EXPR_ARGS:
            {break;}
            case CONST_EXPR: {break;}
            //identifier referencess
            case ID_EXPR:
            case ARRAY_EXPR:
            {BucketList* bucket = ((BucketList*)temp->something);
                if (bucket != NULL) {
                    temp->epx_type = bucket->td->epx_type;
                    if (bucket->td->dcl_type != DEFAULT_TYPE) {
                        temp->epx_type = bucket->td->dcl_type;//set as
declaration type}}
                break;}
            case ASSIGN_EXPR: {
                if (temp->child[0]->epx_type == DEFAULT_TYPE) {//dynamic type
                    temp->child[0]->epx_type = temp->child[1]->epx_type;}
                else if (temp->child[0]->epx_type != temp->child[1]->epx_type)
{//declared type
                    semantic_error_report(temp, "Assign wrong type");}
                temp->epx_type = temp->child[0]->epx_type;
                break;}
            case OP_EXPR: {
                switch (temp->epx_op.getType())
                {//boolean equation, and or
                    case AND:
                    case OR:
                    case NOT: {
```

```
        if (temp->child[0]->epx_type == BOOL_TYPE&& temp->child[1]->epx_type==
BOOL_TYPE) {
                            temp->epx_type = BOOL_TYPE;}
        else {semantic_error_report(temp, "the bool equation should consist with
comparing type");}
                        break;}
                    //relational equation, > <
                    case LT:case LTE:case GT:case GTE:case EQUAL:
                    case NOT_EQUAL: {
                        if (temp->child[0]->epx_type == temp->child[1]-
>epx_type) {
                            temp->epx_type = BOOL_TYPE;}
                        else {
    semantic_error_report(temp, "two operands of relational expression should be
the same type");}
                        break;}
                    //calculating equation, + -
                    default:
                    {   int i = 0;
                        bool isStr = false;
                        bool isNum = false;
                        bool isInt = false;
                        while (temp->child[i] != NULL)
                        {switch (temp->child[i]->epx_type)
                            {case INT_TYPE:isInt = true;
                                break;
                            case NUM_TYPE:isNum = true;
                                break;
                            case STR_TYPE:isStr = true;
                                break;
                            default:
                                break;};
                            i++;}
                        if (isNum && !isStr) {temp->epx_type = NUM_TYPE;}
                        else if (isInt && !isStr) {temp->epx_type = INT_TYPE;}
                        else if (isStr && !isInt && !isNum) {temp->epx_type =
INT_TYPE;}
                        else {semantic_error_report(temp, "Type is not match");}
                        break;} }}
                break;
            default: break;}}}}
```

7. Reference

- codes in code.zip provided by professor.
- TINY_compiler from the TextBook