Project Phase 3 Report

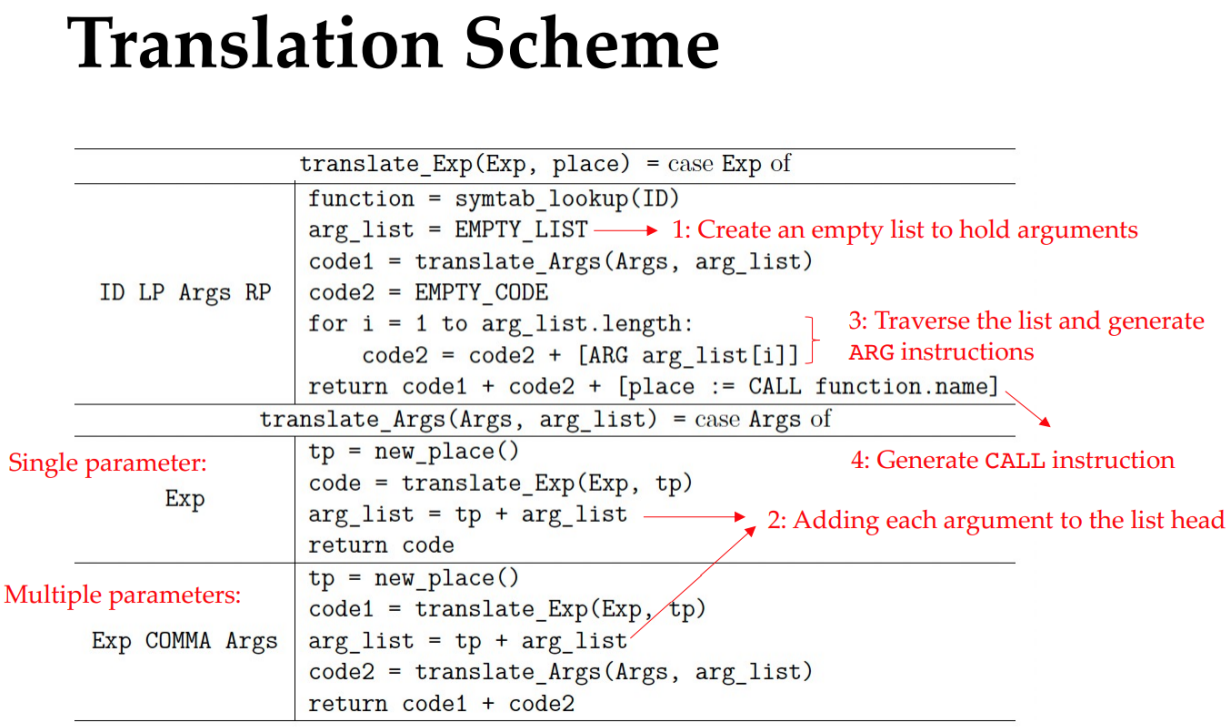
Design Pattern

First we design stuctures to store data:

- Operand: store opertands
- Quadruple: store a piece of IR code
- Arglist: store arguments
- Node: store nodes in a parsing tree

Then we build a parsing tree using bottom-up parsing:

- We used linked list to maintain our parsing table. Then we generate IR code:
- We used post-traveral to traveral the parsing tree and generated the IR code.
- When generating IR code we follow the following slide.



- Linked list was used to store IR code, with each node is a quadruple struct.
- In this way our design has a slopwer speed but a good modularity.

Code exhibition

```
typedef struct _OperandStru // 操作数
{
    char* type;
    union {
        int tempvar; // 临时变量
        int lable;   // 标签
```

```
        int value;   // 常数的值
        char *name;  // 语义值，变量名称、函数名称
    } operand;
    int value;
} OperandStru, *Operand;
```

- Then we define the Quadruple structure to store pieces of IR code

```
typedef struct _InterCodeStru // 中间代码
{
    char* operation;
    union {
        struct
        {
            Operand left, right; // 赋值 取地址 函数调用等
        } assign;
        struct
        {
            Operand result, op1, op2;// 二元运算 + = * /
        } binop;
        struct
        {
            Operand lable, op1, op2;// GOTO 和 IF...GOTO
            char *relop;
        } jump;
        Operand var;// 函数声明、参数声明、标签、传实参、函数返回、读取、打印
    } operands;
    struct _InterCodeStru *prev, *next;
} InterCodeStru, *Quadruple;
```

- Other functions:

```
Quadruple translate_Program(SyntaxTreeNode Program);
Quadruple translate_ExtDefList(SyntaxTreeNode ExtDefList);
Quadruple translate_ExtDef(SyntaxTreeNode ExtDef);
Quadruple translate_FunDec(SyntaxTreeNode FunDec);
Quadruple translate_VarList(SyntaxTreeNode VarList);
Quadruple translate_ParamDec(SyntaxTreeNode ParamDec);
Quadruple translate_CompSt(SyntaxTreeNode ComSt);
Quadruple translate_StmtList(SyntaxTreeNode);
Quadruple translate_Stmt(SyntaxTreeNode Stmt);
Quadruple translate_DefList(SyntaxTreeNode DefList);
Quadruple translate_Def(SyntaxTreeNode Def);
Quadruple translate_DecList(SyntaxTreeNode DecList);
Quadruple translate_Dec(SyntaxTreeNode Dec);
Quadruple translate_Exp(SyntaxTreeNode Exp, Operand place);
Quadruple translate_Cond(SyntaxTreeNode Exp, Operand lable_true, Operand
lable_false);
Quadruple translate_Args(SyntaxTreeNode Args, ArgList arg_list);
```

Optimization

- We created and maintained a symbol table recording svaribles that had been created.
- Each time we encountered a new constant, we first looked it up to check if it had the same value with some varibles in the argument list.

```
typedef struct _ArgListStru
{
    int num;
    Operand list[10];
} ArgListStru, *ArgList;
```

- If they held the same constant value, we would use the existing symbol instead of creating a new one.