Project Phase 2 Report

SID	Name	Rate
11911202	袁恒宸	1/3
12011543	林洁芳	1/3
12011906	汤奕飞	1/3

Basic

In the basic section, the basic semantic checking requirements are completed. The details processing need to be explained as follows:

1. For type.h file:

   We add a new FUNCTION constant to facilitate the processing and saving of functions.

   ```
   typedef struct Type {
       char* name;
       /*add a new category--FUNCTION*/
       enum { PRIMITIVE, ARRAY, STRUCTURE, FUNCTION} category;
       /*other parts are the same of pdf*/
   } Type;
   ```

2. For operators, including +, -, *, /:

   When there are no non-primitive types on either side of the operator symbol, float type will be returned if there is a float type, and int type will be returned if no float type exists.

   For example:

   ```
   /*
   1. exist a float type: return float type
      3.4 + 9, 'c' + 8.3, 1.2 + 3.2
   2. no float type exists: return int type
      'a' + 'e', 2 + 5, 'c' + 7
   */
   ```

3. For comparison of basic data types:

   According to the characteristics of c language, it allows comparison between basic data types, so SUSTech program language also allows comparison between int, float, char, all will return the legal int type (because C language does not set the boolean constant, in general, 0 is false, non-zero integers are true).

Bonus

Different scopes

Revoke Assumption 6, thus variables in different scopes can share the same identifier, and variables defined in the outer scope will be shadowed by the inner variables with the same identifier when redefining the variables with the same name.

Furthermore, there are some possible situations will arise in codes:

1. It needs to allow that different functions can have the same names of arguments and variables.

```
// codes as below are legical
int add(int m, int n) {
    return m + n;
}
int minus(int m, int n) {
    return m - n;
}
```

2. Within a function scope, it needs to allow that defining variables in if/while scope and being invalidated out of scopes.

```
// codes as below are legical
int foo(int m) {
    if (m > 20) {
        int b = 1;
    }
    else {
        int b = 5 + m;
    }
    return m;
}
```

3. Child scopes can access variables defined in the parents scopes.

```
// codes as below are legical
int test(int m, int n) {
    int a = 0;
    while (a < m) {
        int b = n;
        a = a + b;
    }
    return a;
}
```

Structural equivalence for struct

Revoke Assumption 7, then two struct is equivalent if they have the same number of each attribute.

For instance: struct Apple and struct Orange are equivalent.

```
struct Apple {
    float round;
    int weight;
};

struct Orange {
    int weight;
    float round;
};
```

Recognition of continue and break

In lex.l:

```
"break" {yylval = newNodeTER(BREAk,yylineno); return BREAK;}
"continue" {yylval = newNodeTER(CONTINUe,yylineno); return CONTINUE;}
```

In syntax.y:

```
Stmt: Exp SEMI
    ...
    | BREAK SEMI {...}
    | CONTINUE SEMI {...}
    ...
    ;
```

In semantic.c file, we need to judge whether break and continue are in while scope or its child scopes.

```
...
// the variable "prev" will be passed to determine if break and continue
are in the correct scope.
else if (!strcmp(NDtypes[leftmost->type],"BREAK")){
    //BREAK SEMI
    ...
}else if (!strcmp(NDtypes[leftmost->type],"CONTINUE")){
    //CONTINUE SEMI
    ...
}
...
```

... ? ... : ...

Similar as phase1, we support ternary operator:

```
...
else if(!strcmp(NDtypes[operator->type],"QM")){
    // Exp QM Exp COLON Exp
    ...
}
...
```

Multidimensional nested array

We can deal with more than one dimensional array.

For example,

```
// codes as below are legical
int test()
{
    int arr[2][3][4]...[3];
    return 1;
}
```

Extra

type 16

For grammar "STRUCT ID", if this ID refers to a non-structure type, then cause that using a non-structure variable as struct.

type 17

For grammar "STRUCT ID", if this struct isn't defined, then cause that using struct without defination.

type 18

If using key word break in the wrong scopes, then cause using break without loop.

type 19

If using key word break in the wrong scopes, then cause using continue without loop.